Beyond binary hyperparameters in deep transfer learning for image classification

Jo Plested

A thesis submitted for the degree of
Doctor of Philosophy
The Australian National University

May 2023

Except where otherwise indicated, this thesis is my own original work.

Jo Plested
May 1, 2023

Acknowledgments

I would like to thank my supervisor Tom Gedeon for all his support, encouragement and enthusiastic discussion of ideas. Also for giving me the opportunity and encouragement to develop as an academic while completing my PhD by lecturing and supervising project students in my field. My mother Dawn Plested for her unwavering support and tireless editing. My father Ian Plested for his support and constant encouragement to hurry up and finish. My partner Peter Horvath for his support and understanding. And all my friends and family for loving me for who I am despite my occasional desire to work myself to an early grave and still being there when I come up for air occasionally. I couldn't have done this without you all.

Abstract

Convolutional neural networks (CNNs) have achieved many successes in image classification in recent years. It has been consistently demonstrated that CNNs work best when there is abundant labelled data available for the task and large deep models can be trained. However, there are many real world scenarios where the requirement for large amounts of training data to get the best performance using modern very deep neural networks cannot be met. In these scenarios transfer learning can help improve performance.

This thesis defines deep transfer learning and the problem it attempts to solve in relation to image classification. It provides a new taxonomy of the applications of transfer learning for image classification. This taxonomy makes it easier to see overarching patterns where transfer learning has been effective and where it has failed to fulfill its potential. This thesis also provides the first comprehensive review of deep transfer learning as it relates to image classification overall and suggestions for future research directions in the field. While there have been recent general surveys of deep transfer learning, and ones that relate to particular specialised target image classification tasks, there have been none that review the area as a whole. It is important for future progress in the field that all current knowledge on deep transfer learning in image classification is collated and the overarching patterns are analysed and discussed.

The aim of this thesis is to advance the state-of-the-art in deep transfer learning as it relates to image classification. This applies particularly for datasets where the number of training examples is small. Its main contributions are 1) improvements to best practice that go beyond the standard binary transfer learning practices, 2) heuristics that can be used to predict optimal non-binary transfer learning hyperparameters, and 3) a model that can learn optimal non-binary transfer learning hyperparameters. Experimental results are provided on a variety of image classification datasets with few training examples. These datasets range from very closely related to the source dataset to far less related. The benefits of the non-binary approach presented are supported by final results that come close to or exceed state of the art performance on a variety of target datasets that traditionally transfer learning has not performed well on.

Contents

List of Figures

List of Tables

PhD Research Papers, Presentations and Awards

Papers - first author

Plested, J.; Gedeon, T. D.; Zhu, X.; Dhall, A.; and Geocke, R. R., 2017. Detection of universal cross-cultural depression indicators from the physiological signals of observers. In2017 Seventh International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW), 185–192. IEEE

Plested, J. and Gedeon, T., 2019. An analysis of the interaction between transfer learning protocols in deep neural networks. In International Conference on Neural Information Processing, 312–323. Springer. (cited on pages 6, 66, 71, 74, 83, 88, 104,133, and 140)

Plested, J.; Shen, X.; and Gedeon, T., 2021. Non-binary deep transfer learning for image classification, (Jul. 2021), arXiv:2107.08585. (cited on pages 71,72, 76, 80, 82, 84, and 89) accepted in International Conference on Neural Information Processing 2021

Papers - I was project supervisor and created the research topic

Shen, X.; Plested, J.; Caldwell, S.; and Gedeon, T., 2021a. Exploring biases and prejudice of facial synthesis via semantic latent space. In 2021 International Joint Conference on Neural Networks (IJCNN), 1–8. IEEE.

Evans, N.; Plested, J.; and Gedeon, T., 2020. Exploring the correlation between random convolutional architectures and the trained equivalent. In 2020 International Joint Conference on Neural Networks (IJCNN), 1–6. IEEE

Wang, Y.; Plested, J.; and Gedeon, T., 2020. Multitune: Adaptive integration of multiple fine-tuning models for image classification. In International Conference on Neural Information Processing, 488–496. Springer.

Papers - I supervised but did not create the topic

Shen, X.; Plested, J.; and Gedeon, T., 2021b. Feature selection on thermal-stress dataset. arXiv preprint arXiv:2109.03755, (2021)

Shen, X.; Plested, J.; Yao, Y.; and Gedeon, T., 2020. Pairwise-gan: Pose-based view synthesis through pair-wise training. In International Conference on Neural Information Processing, 507–515. Springer.

Yao, Y.; Plested, J.; and Gedeon, T., 2020. Information-preserving feature filter for short-term eeg signals. Neurocomputing, 408 (2020), 91–99

Yao, Y.; Plested, J.; and Gedeon, T., 2018. Deep feature learning and visualization for eeg recording using autoencoders. In International Conference on Neural Informa- tion Processing, 554–566. Springer

Yao, Y.; Plested, J.; and Gedeon, T., 2018b. A feature filter for eeg using cycle- gan structure. In International Conference on Neural Information Processing, 567–576. Springer.

Yao, Y.; Plested, J.; Gedeon, T.; Liu, Y.; and Wang, Z., 2019. Improved techniques for building eeg feature filters. In 2019 International Joint Conference on Neural Networks (IJCNN), 1–6. IEEE.

Liu, Y.; Yao, Y.; Wang, Z.; Plested, J.; and Gedeon, T., 2019b. Generalized alignment for multimodal physiological signal learning. In 2019 International Joint Conference on Neural Networks (IJCNN), 1–10. IEEE.

Grants and awards

2021 2 x AU$20,000 Defence Science and Technology Group of the Department of Defence AI for Decision Making Program grant. I was lead investigator for one and researcher for the other.

2021 AU$70,000 “Data Science for an Unmanned Ground Vehicle” part of the team sponsored by Defence CRC TAS Ltd.

2020 2 x AU$20,000 Defence Science and Technology Group of the Department of Defence AI for Decision Making Program grant. I was lead investigator for one and researcher for the other.

2018 US$1,500 Women in Machine Learning (WIML) travel grant to present at WIML workshop at the Neural Information Processing Systems conference in Montreal.

2018 Nominated for best student paper at the International Conference on Neural Information Processing Systems (ICONIP)

Chapter 1

Introduction

Researchers in convolutional neural networks (CNNs) have achieved many successes in image classification in recent years [Krizhevsky et al., 2012; Girshick et al., 2014; Li and Deng, 2020; Masi et al., 2018; Mazurowski et al., 2019]. It has been consistently demonstrated that CNNs work best when there is abundant labelled data available for the task and large models can be trained [Ngiam et al., 2018; Mahajan et al., 2018; Kolesnikov et al., 2019]. However, there are many real world scenarios where the requirement for large amounts of training data to get the best performance using modern very deep neural networks cannot be met. Some of these are:

1. Insufficient data because the data is very rare or there are issues with privacy etc. For example new disease and rare disease diagnosis tasks in the medical domain have limited training data due to both the examples themselves being rare and privacy concerns.

2. It is prohibitively expensive to collect and/or label data. For example in settings where labelling can only be done by highly qualified experts in the field.

3. The long tail distribution where a small number of objects/words/classes are very frequent and thus easy to model, while many many more are rare and thus hard to model [Bengio, 2015]. For example most language generation problems.

There are several other reasons why we may want to learn from a small number of training examples:

- It is interesting from a cognitive science perspective to attempt to mimic the human ability to learn general concepts from a small number of examples.

- There may be restraints on compute resources that limit training a large model from random initialisation with large amounts of data. For example environmental concerns [Strubell et al., 2019].

In all these scenarios transfer learning can often greatly improve performance. In this paradigm the CNN model is trained on a related dataset and task for which more data is available and weights are used to initialise a model for the target task. In order for this process to improve rather than harm performance the dataset must be sufficiently closely related and best practice methods used. The aim of this thesis is to advance the state-of-the-art in deep transfer learning as it relates to image classification.

1.1 Contributions

I make the following contributions in this thesis:

1. Formally define deep transfer learning and the problem it attempts to solve in relation to image classification in Chapter 5.

2. Provide a new taxonomy of the applications of transfer learning for image classification in Chapter 6. This taxonomy makes it easier to see overarching patterns of where transfer learning has been effective and, where it has failed to fulfill its potential. It also allows me to make suggestions as to where the problems lie and how it could be used more effectively in the latter areas. I show that under this new taxonomy, many of the applications where transfer learning has been shown to be ineffective or even hinder performance are to be expected when taking into account the source and target datasets and the techniques used. In many of these cases, the key problem is that methods and hyperparameter settings designed for large and very similar target datasets are used for smaller and much less similar target datasets, and alternative choices would lead to better outcomes.

3. Provide the first comprehensive review of deep transfer learning as it relates to image classification overall in Chapter 6. While there have been recent general surveys of deep transfer learning, and ones that relate to particular specialised target image classification tasks, there have been none that review the area as a whole. It is important for future progress in the field that all current knowledge is collated and the overarching patterns are analysed and discussed. I summarize current knowledge in the area as well as identifying knowledge gaps and suggest directions for future research.

4. Present a detailed summary of source and target datasets commonly used in the area in Chapter 5 to provide an easy reference for the reader looking to

understand relationships between where transfer learning has performed best and where results have been less consistent.

5. Develop methods for non-binary hyperparameters in transfer learning including combining L2SP and L2 regularization and performing non-traditional fine-tuning hyperparameter searches in Chapters 7 and 9. I show that the optimal settings result in significantly better performance than binary settings for all datasets except the most closely related. I also recommend heuristics for determining the optimal transfer learning hyperparameters. Extensive experiments are performed across four different datasets to find the optimal settings when binary assumptions about hyperparameters are discarded. I show that these optimal settings often outperform binary hyperparameters particularly when the target dataset is small and less similar to the source dataset. The results come close to or exceed state of the art performance on a variety of tasks that have traditionally been more difficult for transfer learning. The changes I suggest are much more accessible than previous state of the art works, for difficult transfer learning tasks, that have had to rely on more complex and compute intensive methods.

6. Develop a model that can learn high performing non-binary transfer learning hyperparameters in Chapter 8.

Chapters 7 and 9 are published as [Plested and Gedeon, 2019] and [**?**] respectively. Chapter 6 along with parts of Chapter 5 are under review as a journal paper and Chapter 8 is under review as a conference paper.

1.2 Thesis Outline

In Chapters 2 to 5 I present background material needed to understand this thesis. Chapter 5 introduces the problem domain and formalises the difficulties with learning from small datasets that transfer learning attempts to solve. This chapter includes terminology and definitions that are used throughout this thesis.

Chapter 6 reviews transfer learning for image classification and related areas. Within that chapter, Section 6.3 provides a detailed analysis of recent advances and improvements to transfer learning and specific application areas and highlights gaps in current knowledge. In 6.7 I give an overview of other problem domains that are closely related to deep transfer learning for image classification including the similarities and differences in each. Finally Section 6.8 summarises all current knowledge, gaps and problems and recommends directions for future work in the area.

Chapters 7 to 9 describe methods, experiments and results for moving beyond standard binary deep transfer learning hyperparameters. Finally Chapter 10 summarises my thesis and discusses current and future directions to extend this field of research.

Chapter 2

Machine learning basics

Table 2.1: Notation and terminology summary

terminology	notation	detailed description
Task	T	The problem or application the algorithm is applied to solve
Performance measure	P	A formal calculation of how well an algorithm has performed
Experience	E	A dataset from which the model is expected to learn
Source Dataset	D_s	Data related to the target data and used for pretraining a model
Target Dataset	D_T	Data belonging to the target task
Model output	Y	Output predictions made by the algorithm
Target labels	t	The designated output values for a particular input
Input	x	Values used by an algorithm to calculate an output
Parameters	θ	The model values that are learned from the data
Hyperparameters		The model values that determine how the Parameters are learned and need to be set prior to training

Deep learning is a modern name for neural networks with more than one hidden layer. Neural networks are themselves a sub-area of machine learning, so to un-

derstand deep learning some basics of machine learning are needed. Mitchell et al. [1997] provide a succinct definition of machine learning:

Definition 2.1. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

For example, consider an image classification task (T). The aim is to increase classification accuracy (P) by training on labelled examples of each image class (E).

2.1 Types of machine learning

Machine learning algorithms can be divided into supervised and unsupervised depending on whether or not the training dataset (E) contains labelled examples or other target information and the format of the information. The types of machine learning are shown in Figure 2.1 and described below.

Supervised: In supervised machine learning we have a training set with known labels. An example of this is the image classification task mentioned above.

Unsupervised: In unsupervised learning we have a training set without known labels and the algorithm is designed to learn about the input distribution in general. An example of this is the K-means algorithm which divides a set of training examples into clusters where the within cluster distances between input features vectors are smaller and the between cluster distances are larger.

Self-supervised learning is a subsection of unsupervised learning where algorithms use a part of the input data as the training target. It is usually used when target values are unavailable or limited. An example would be a video classification task where the dataset had limited labelled examples and a large number of unlabelled examples. We could pretrain an algorithm to predict the next frames in the video sequence in the hope that it would learn features in the input distribution that were helpful in the final task of mapping from input to output labels.

Reinforcement learning In the reinforcement learning paradigm, instead of having target labels for each training example the model supervision comes in the form of occasional positive or negative rewards for achieving particular goals. An example would be the problem of learning to play chess where the model is given a positive

reward for winning and a negative reward for losing, with no other labels for interim moves.

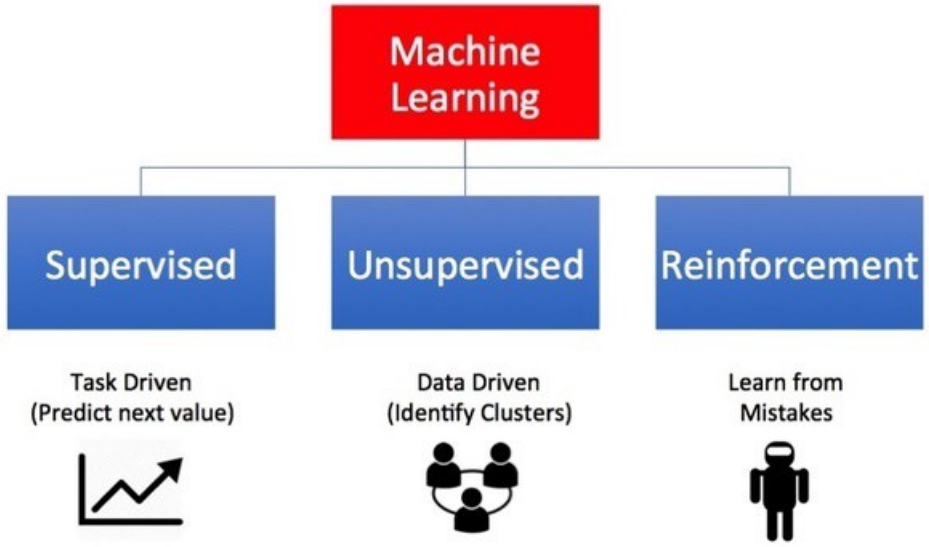


Figure 2.1: Types of machine learning [Heidenreich, 2018]

2.2 Types of tasks

Most machine learning tasks can be defined as either regression or classification.

Regression: The goal in regression is to take an input vector $x_n \in R^D$ and predict the value of a real number. The target value $t_n \in R$. It is used to model continuous output values. The right image of Figure 2.2 illustrates a linear regression task.

Classification: The goal in classification is to take an input vector $x_n \in R^D$ and assign it to one of K discrete classes C_k, $k = 1, ..., K$. The target value $t_n \in$ a set of K discrete numbers. It is used to model discrete output values. The left image of Figure 2.2 illustrates a two class classification task.

It could be argued that generation does not fall into the above two categories, however the final prediction does. For example generation of images can be modelled by predicting a real number or numbers for each pixel for greyscale or coloured images, or as a binary classification task for each pixel for black and white images.

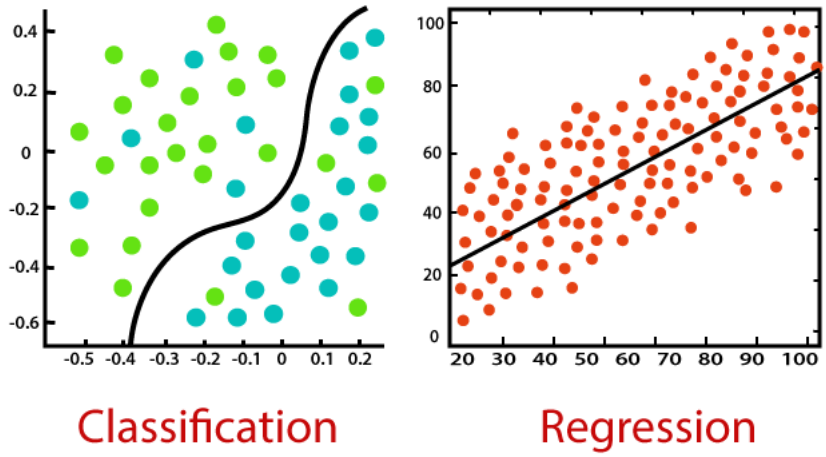


Figure 2.2: Two class classification versus linear regression [Ngoma et al., 2017]. Two class classification is attempting to split two classes - light green and dark green dots based on the two dimensional inputs. Regression is attempting to predict the y axis output from the x axis input

2.3 Hyperparameters

Hyperparameters are settings that determine how a machine learning algorithm learns and need to be set before learning begins. In terms of the discussion to follow in Section 5.1 hyperparameters determine the family of candidate functions $\mathcal{F}$ that the final model will be selected from.

They include settings that are difficult to optimise and those that cannot be optimised well on the training set. An example of the latter is settings that control model capacity. If these settings are optimised on the training set the largest possible capacity values will always fit the training set best, but will overfit and not generalise well to new data.

Hyperparameters can be set by an expert or optimised with an algorithm. A class of hyperparameter optimisation algorithms are covered in Section 4.5.1.

2.4 Splitting the dataset

With supervised, self-supervised and reinforcement learning the dataset must be split into appropriate subsets prior to training.

Training set This is generally the largest subset and is used to train the learning parameters of the model.

Validation set This subset is used to tune all the hyperparameters that determine how the model is trained to get best performance.

Test set This is used to do final testing to prove the efficacy of the model on unseen data. It is important that no training or hyperparameter tuning is performed on the test set as this will result in fitting the model to the test set and invalidate the final results.

Cross-validation If the test set is too small it will be difficult to show the generalisation of the results. If the data is too limited k-fold cross-validation can be used to make better use of the available data. With k-fold cross-validation the entire dataset is divided into k equal sized non-overlapping sets. Each one of the k sets is then used as the test set in turn and a model trained from initialization each time. This means that all data can be used as the test set while still making sure that the model is not trained on the test data. An example of cross-validation is shown in Figure 2.3. The extreme end of k-fold cross-validation is leave-one-out where each data point is used as the test set in turn with the model being trained on all the remaining data points each time.

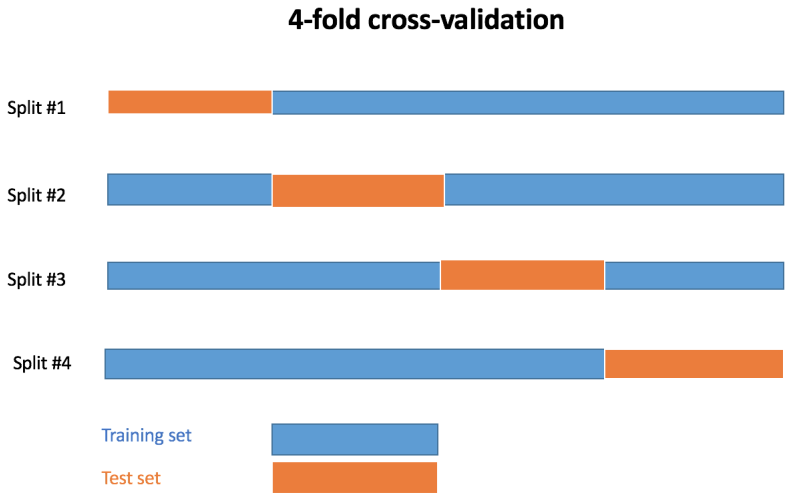


Figure 2.3: Cross-validation [Moudiki, 2020]

2.5 Overfitting and Underfitting

A model's performance depends both on its ability to fit the training data, and to generalise to new data. If it does not fit the training data well it is called underfitting. The opposite problem, overfitting, occurs when the model fits the training data too well and does not generalise well to new data. Underfitting can occur when the model class is too inflexible to learn the training data well or the model has not been trained sufficiently or well on the training data. Overfitting occurs when the model class is very flexible and the model has been trained to fit the training data too well.

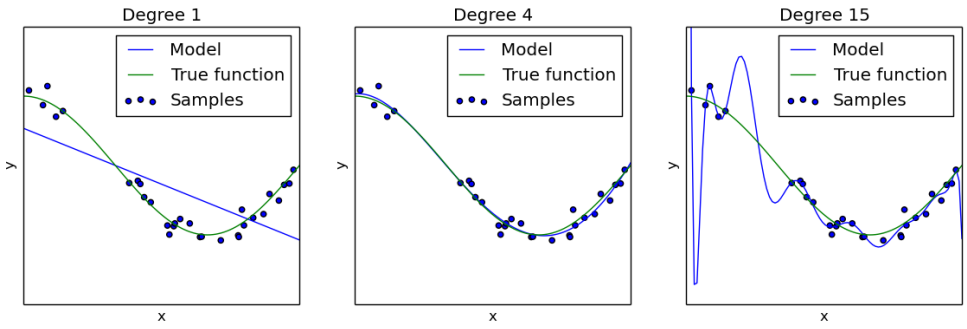


Figure 2.4: Overfitting and Underfitting example [Pedregosa et al., 2011]. Degrees refer to polynomial degrees. Degree 1 shows underfitting, degree 4 a good fit and degree 15 overfitting.

2.6 Gradient Descent

When the gradient information cannot be used to calculate the minimum of the parameters of a function directly it can be used to inform the direction to change the parameters. For a function $f(w)$ the partial derivative $\frac{\partial}{\partial w_i} f(w)$ tells us how $f(w)$ changes as only the variable w_i changes at point w. This tells us in which direction to change w_i to minimise $f(w)$. Decrease $f(w)$ by moving w_i in the opposite direction (downhill) of the partial derivative as shown in Figure 2.5. The gradient descent algorithm updates the parameter vector w, using the partial derivatives of the error function:

$$w \leftarrow w - \eta \nabla E\left(f\left(x, w\right), t\right) \tag{2.1}$$

where t is the target, and $0 < \eta < 1$ is the learning rate.

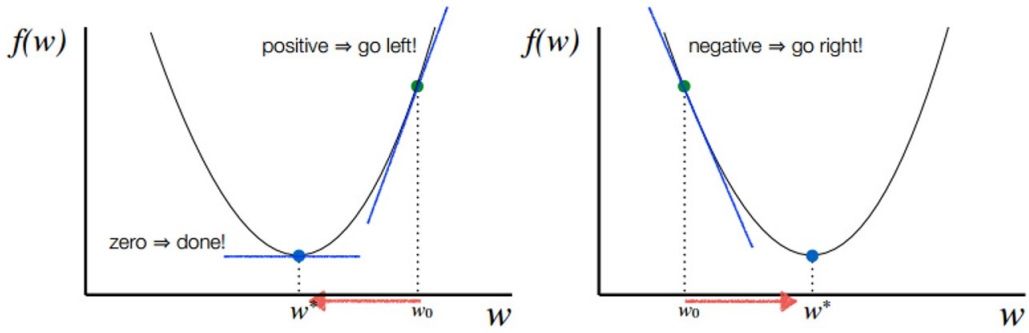


Figure 2.5: Gradient descent [Dowling, 2017].

Chapter 3

Deep Feedforward Neural Networks

Deep learning is a modern name for neural networks with more than one hidden layer. Neural networks are themselves a sub-area of machine learning.

The term neural networks gets its name from early attempts to build computational models based on human and animal brains. However biological realism puts unnecessary constraints on computational models and so today's neural networks are based on statistical theory of pattern recognition. The specific class of neural network that has proven to be most valuable in practice is the multilayer perceptron (MLP) [Bishop, 2006].

3.1 Single layer perceptron

The single layer perceptron was created by Rosenblatt [1962] and is shown in Figure 3.1. It is a two-class classification model where the input vector x is transformed via a linear model of the form:

$$y(x) = f\left(w^T x\right) \tag{3.1}$$

$f(.)$ is a step function of the form:

$$f(w^T x) = \begin{cases} +1 & w^T x \geq 0 \\ \\ -1 & w^T x < 0 \end{cases} \tag{3.2}$$

So +1 is the output for class 1 and -1 for class 2. Generally a learnable bias is included by adding another weight to the weight vector with input x always equal to 1. This allows the point where the decision boundary crosses the y axis to be learned, as shown in Figure 3.1.

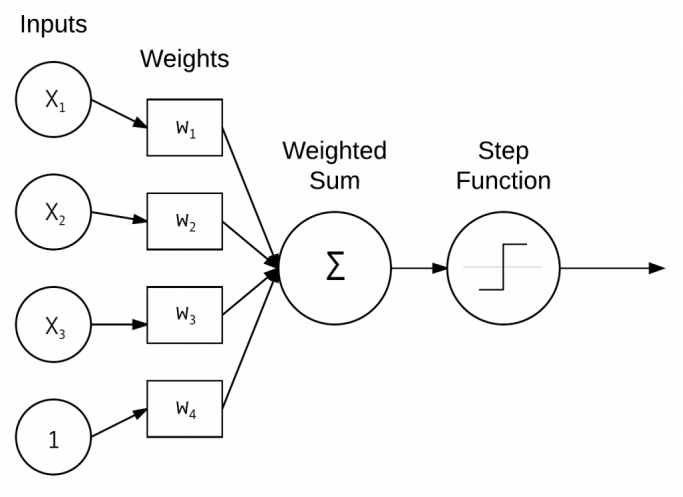


Figure 3.1: A perceptron [Rosenblatt, 1962].

Often a fixed non-linear transformation is performed on input vector x to give feature vector $\phi(x)$ so we have:

$$y\left(\phi(x)\right) = f\left(w^T\phi(x)\right) \tag{3.3}$$

In this case $\phi\left(.\right)$ is called a basis function and is generally hand designed by an expert in the application area.

The loss function is based on misclassified patterns only to eliminate the non-differentiable step in the threshold function:

$$E(\mathbf{x}, \mathbf{w}) = -\left(\sum_{i=1}^{N}(w_i x_i) - \theta\right) t \tag{3.4}$$

$$\frac{\partial}{\partial w_i} E(\mathbf{x}, \mathbf{w}) = -x_i t \tag{3.5}$$

The final gradient descent learning algorithm for the perceptron is:

1. Set $w(\tau)$ at time $\tau = 0$, to small random values
2. Present input vector $X = x_1, x_2, ..., x_N$ and target output t
3. Calculate actual output $y = \sum_{i=1}^{N}\left(w_i x_i\right) - \theta$
4. Calculate error $E(\mathbf{x}, \mathbf{w}) = -\left(\sum_{i=1}^{N}(w_i x_i) - \theta\right) t$

5. Adapt weights using gradient descent $w_i^{\tau+1} = w_i^{\tau} - \eta\,(-x_i t)$ $\eta < 1$ (η, called the learning rate, is usually small, e.g. 0.01)
6. Go to step 2. Repeat until convergence or a maximum number of updates.

3.2 Hidden units

The function that combines inputs with weights and performs a non-linear activation on them is called a neuron in keeping with the idea of imitating the human brain. Many neurons together is called a neural network.

A function that just depends linearly on its inputs has severe limitations. An example of the limitations of the simple perceptron algorithm is that it cannot compute the XOR function. This was pointed out by Minsky and Papert [1969] and caused the perceptron algorithm to fall out of favour. The limitations of the simple perceptron algorithm can be overcome by adding more layers of computation to allow for more complex combinations of the input features and must include more complex (non-linear) activation functions. In a typical fully connected feed forward neural network weight connections go from each neuron in one layer to each neurons in the next layer, but not between neurons in the same layer or back to a previous layer.

A hidden layer in a neural network is a layer that is not directly connected to the input or the output. Figure 3.2 shows a feed forward neural network with one hidden layer and four hidden layers. A linear combination of a linear combination of the input can be reduced to just one linear combination i.e. two layers can be reduced to one. This means that each layer must include a non-linear activation function after the linear combination of inputs and weights $\mathbf{W^T x}$. So a perceptron function with one hidden layer becomes:

$$\begin{aligned} h(\mathbf{x}) &= f^1\left(\mathbf{W^T x}\right) \\ y(\mathbf{x}) &= f^2\left(\mathbf{W^T h}\right) \end{aligned} \tag{3.6}$$

where $\mathbf{h}$ is the hidden layer, f^1 is the hidden layer activation function and f^2 is the output layer activation function.

While it has been shown that neural networks with one hidden layer are universal approximators [Hornik et al., 1989], in practice because the loss function is non-convex with respect to the weights it is difficult to optimise. For this reason modern networks are often arranged in very deep networks and task specific architectures.

To add more hidden layers the output from the previous hidden layer becomes the input to the next hidden layer and an activation function is applied after each

linear combination of weights and inputs. A perceptron with one or more hidden layers is called a multi layer perceptron (MLP) or neural network.

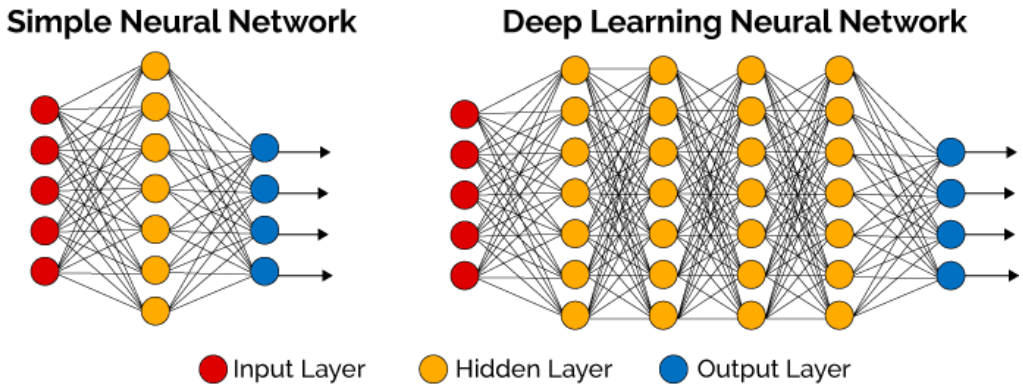


Figure 3.2: Multi-layer perceptrons [Thanaki, 2017]. A multi-layer perceptron with one hidden layer left and four hidden layers right.

3.3 Activation functions

Activation functions are divided into output layer and hidden layer activation functions.

3.3.1 Output layer activation functions

These are set by the prediction task.

- For a linear regression task with a real number target output no activation function is used as the output of the final layer is already in the correct format.
- For a logistic regression task (output is a real number between 0 and 1) the Sigmoid activation function $f(x) = \frac{1}{1-e^{-x}}$ is used as per Figure 3.3. This function squashes the full range of real number outputs to the range 0-1 so the final output can be interpreted as the probability that the input belongs to class 2.
- For classification tasks the softmax activation function is used:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{3.7}$$

  This function takes N real number outputs (logits) and normalises them so the final outputs are between 0 and 1 and sum to 1. The Sigmoid function can also

be used for 2 class classification problems by creating a network with only one output and setting the output $y(\mathbf{x})$ as the probability of class 1 and $1 - y(\mathbf{x})$ as the probability of class 2.

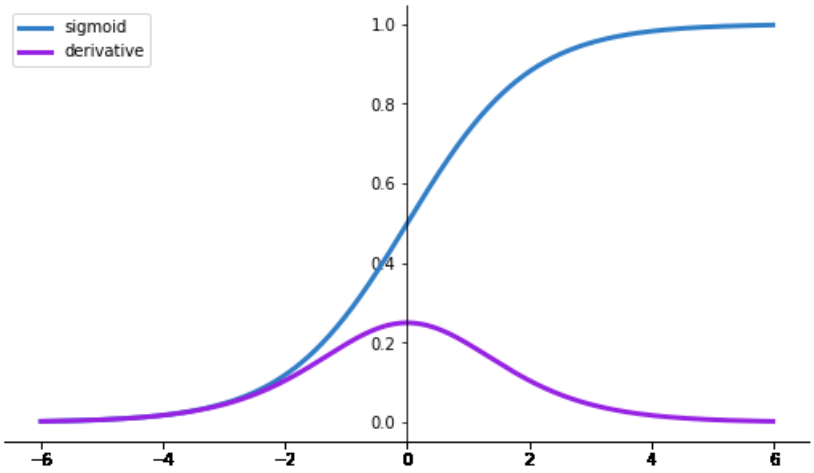


Figure 3.3: Sigmoid activation function and derivative [Shaw].

3.3.2 Hidden layer activation functions

Sigmoid Traditionally the Sigmoid Function was used as the hidden layer activation function as it was thought to most closely resemble the firing pattern of neurons in the brain [LeCun et al., 2015]. The major drawback of the Sigmoid function is that its maximum derivative is 0.25 as shown in Figure 3.3. This means that when its derivatives are multiplied together during the backpropagation algorithm detailed in Section 3.5 the values quickly become small and disappear. This is called the disappearing gradients problem and is one of the reasons that deep neural networks were traditionally pretrained a layer at a time using the self-supervised learning algorithm Greedy Layerwise Unsupervised Pretraining [Bengio et al., 2007].

Rectified linear (ReLU) The vanishing gradients problem was overcome with the introduction of the Rectified Linear Unit (ReLU) activation function as shown in Figure 3.4:

$$f(x) = max\,(0, x) \tag{3.8}$$

The ReLU activation function was shown to speed up training considerably compared to the Sigmoid function [Krizhevsky et al., 2012].

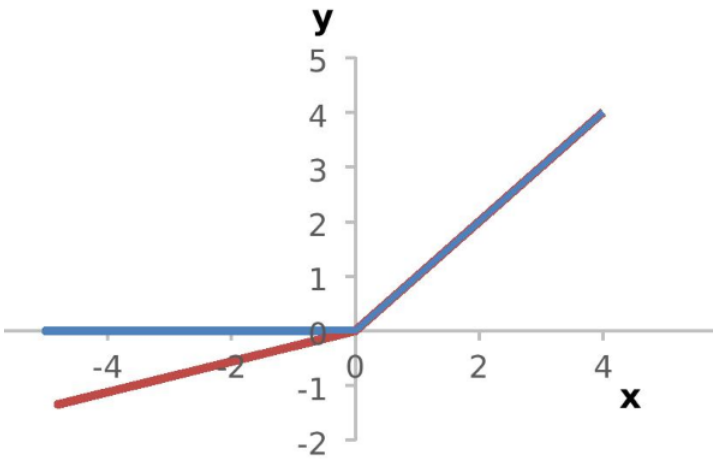


Figure 3.4: ReLU (blue) and leaky ReLu (red) activation functions [Nayef et al., 2022].

Leaky ReLU A problem with the ReLU activation function is that for any negative value of $\mathbf{w}^T\mathbf{x}$ the output is set to 0 which also makes the gradient 0. This means that no gradient information is passed through the neuron to update the weights. If the weights happen to start at or become values such that $\mathbf{w}^T\mathbf{x} \leq 0 \forall \mathbf{x}$ then the weights can no longer be updated and the neuron "dies". Leaky ReLU [Maas et al., 2013] solves this problem by having a small gradient for $\mathbf{w}^T\mathbf{x} < 0$ as per Figure 3.4.

There are many other activation functions that have been shown to be effective in particular applications, but the above are the most commonly used and are generally effective [Goodfellow et al., 2016].

3.4 Loss functions

In order to update the weights using the gradient descent algorithm the error or loss must first be calculated. The loss function is the calculation of the empirical risk equation from the data. Traditionally the loss function was just the mean squared error (MSE) between the actual output and target output for each training example [Rumelhart et al., 1995] as shown in Figure 3.5:

$$L(y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - t_i)^2 \tag{3.9}$$

where y_i is the ith training example, t_i is the ith target value and n is the number of training examples.

The MSE is a simple way of ensuring that positive and negative differences in output and target do not cancel out and that large mistakes are weighted more heavily than small mistakes. For this reason it can be sensitive to outliers.

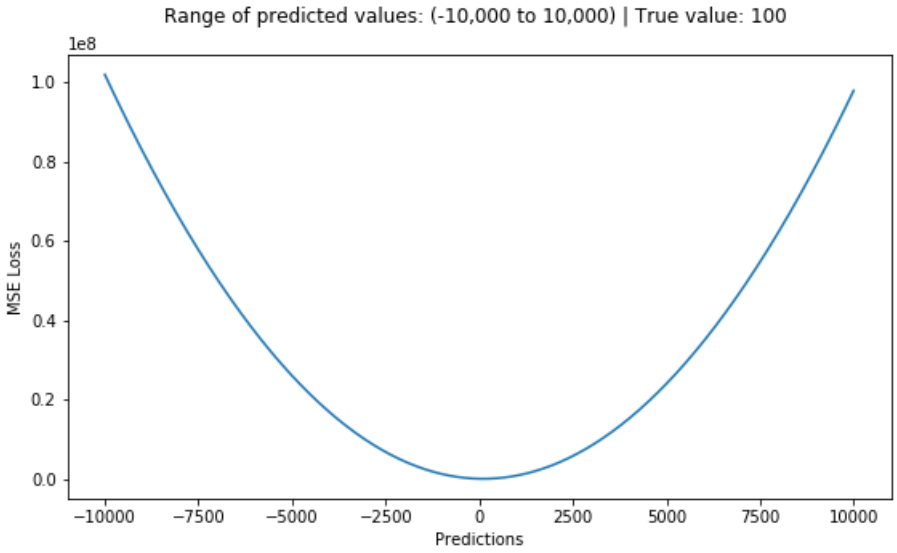


Figure 3.5: Mean Squared Error [Feng, 2021]

In modern algorithms the loss function chosen tends to be matched to the output, target and task, leading to better results. The MSE loss function is still often used for linear regression. For classification tasks where the softmax activation function is used and all outputs are less than 1, the MSE will exacerbate the problem of very small gradients when the output is very far from the target. In this case the cross-entropy loss improves learning as it increases exponentially as the output gets further away from the target and so can counteract the small gradients in the tails of the softmax. The cross-entropy loss is:

$$L(y,t) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} log(y_{nk}) \tag{3.10}$$

where t_{nk} is the target value, which is a binary indicator for whether class example n, belongs to class k. y_{nk} is the predicted probability of example n belonging to class k. N is the number of training examples and K is the number of classes in the classification problem.

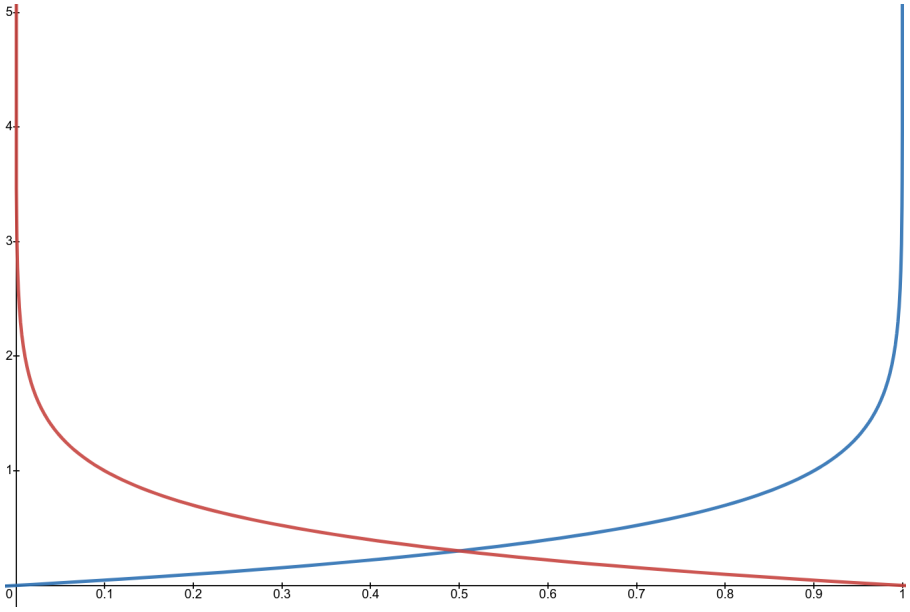


Figure 3.6: Binary cross entropy loss. Red line is target = 1, blue is target = 0

Reinforcement learning uses specific loss functions that are designed for the task. These are covered in Section 4.4. There are also extra terms which can be added to the loss functions that improve regularisation and learning efficiency. These are covered in Sections 3.6 and 4.2.

3.5 Backpropagation

The aim of a neural network is to learn to improve its performance at the task from the training data by finding the weights that minimise the loss function.

The analytic solution relies on it being linear and having a squared error measure. Iterative methods are usually less efficient but they are much easier to generalise, so generally gradient descent is used.

Backpropagation is an application of the chain rule of differentiation applied to propagate derivatives from final layers of the neural network to the hidden and input weights.

$$
\begin{aligned}
y &= f(g(x)) \\
u &= g(x) \\
y &= f(u) \\
\frac{dy}{dx} &= \frac{du}{dx} \times \frac{dy}{du}
\end{aligned}
\tag{3.11}
$$

3.5.1 Learning rate

The optimal learning rate is not too big and not too small. As shown in Figure 3.7 a learning rate that is too big will cause the loss to oscillate back and forth across the minimum. A learning rate that is too small will cause training to progress too slowly and will also be more likely to get stuck in small local minima. The learning rate is a hyperparameter that needs to be carefully tuned for each learning task to achieve best performance. In practice it is common to decay the learning rate throughout training.

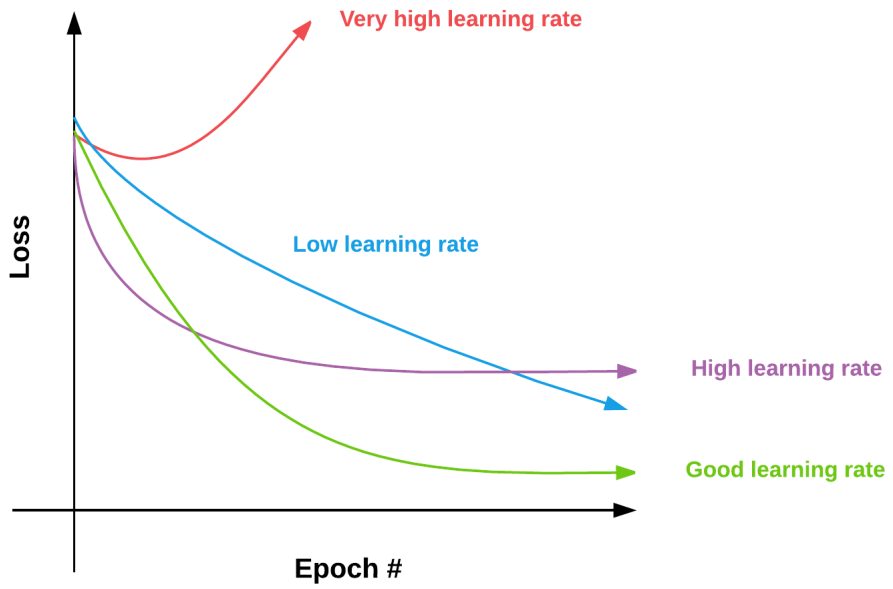


Figure 3.7: Effect of learning rate [Kaur, 2018].

3.5.2 Batch training

Calculating the gradients based on the entire training set for each update is called full batch gradient descent. This method gives the most accurate gradient information. However there are several reasons why using this may not give best performance:

1. We expect the loss surface to be non-convex and extremely complex with many local minima, so a noisy gradient estimate may allow the model to get out of local minima.

2. For large training datasets, training will progress extremely slowly if we wait to calculate the loss based on all training examples before backpropagating the gradients and updating the weights.

Stochastic gradient descent is the name used when updating after every training example. Calculating gradients based on just one training example can cause estimates to be too noisy and again lead to training that is too slow as shown in Figure 3.8. Usually a compromise between full batch and stochastic gradient descent is used, being mini-batch gradient descent. This is where a small portion of the training data is used to calculate the loss and gradients for each update to the weights. Commonly used batch sizes are in the 10s or 100s depending on the dataset, task and compute resources. It is important to note that batch size is a hyperparameter and interacts with other hyperparameters. For example a larger batch size generally requires a larger learning rate as less updates are made for each loop through the training set (epoch) and makes it harder to avoid local minima.

3.6 Optimizers

The optimizer is the function that calculates the updates to weights at each training step. The simplest optimizer is the plain gradient descent from equation 2.1.

3.6.1 Momentum

A common addition to gradient descent is momentum [Polyak, 1964]. Momentum adds a fraction of the previous update to the current update. This dampens large swings in the direction of the weight updates, especially when the error surface is steep in one dimension and not in others. The update rule is given by

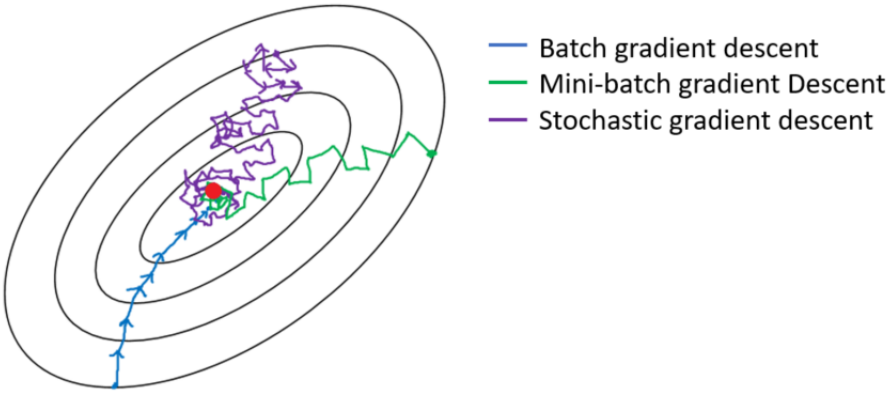


Figure 3.8: Gradient descent compared to stochastic and batch updates [Dabbura, 2017]

$$\begin{aligned} \mathbf{v} &\leftarrow \alpha \mathbf{v} - \eta \nabla L\left(f\left(\mathbf{x}, \mathbf{w}\right), \mathbf{t}\right) \\ \mathbf{w} &\leftarrow \mathbf{w} + \mathbf{v} \\ 0 < \eta \leq 1, &\quad 0 < \alpha \leq 1 \end{aligned} \tag{3.12}$$

where α is a hyperparameter that determines how quickly the contributions of the previous gradients decay.

An example of momentum applied to a loss surface where one dimension is steep and one dimension is not is shown in Figure 3.9.

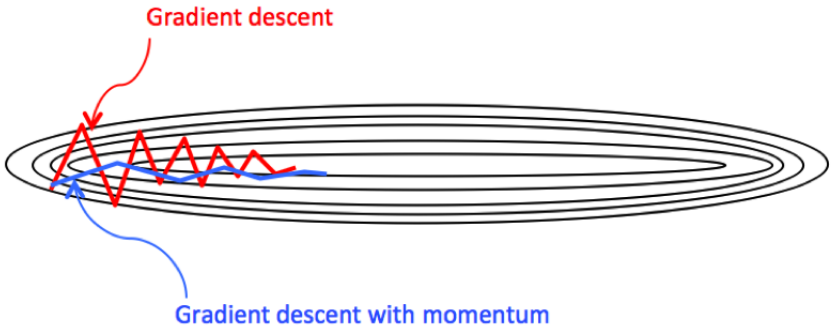


Figure 3.9: The effect of momentum on an error surface that is steep in one dimension and not in the other [Ng, 2022]

3.6.2 Adaptive learning rates

Momentum goes some way to addressing the case where the loss function of a neural network is more sensitive to updates in some directions in parameter space and insensitive to others. Adaptive learning rates are another way of addressing this by adapting learning rates for each individual parameter throughout training based on the accumulation of historical gradients.

AdaGrad [Duchi et al., 2011] adapts the learning rate of each model parameter by scaling them proportionally to the inverse of the sum of the squared values of all the historical gradients. So the scaled learning rate is:

$$\frac{\eta}{\sqrt{G_t + \epsilon}} \tag{3.13}$$

where G_t is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. w_i up to time step t

This means that the parameters with the largest partial derivatives have a large decrease in learning rate and those with the smallest have a small decrease. Similar to momentum this can help the algorithm get out of saddle points or similar where the gradient is large in one dimension and not in others. Adagrad is designed to perform well with a convex loss function, but in a nonconvex space the sum of all historical gradients may make the learning rate too low by the time a good locally convex space is found.

RMSProp [Hinton et al., 2012a] This algorithm modifies Adagrad by using an exponentially decaying moving average instead of a sum of all the gradients. That way learning rates are not decayed too far and it can still converge rapidly after reaching a good locally convex space. RMSProp adapts the learning rate as follows:

$$\begin{aligned} E[g^2]_t =& \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \\ & \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \end{aligned} \tag{3.14}$$

where:

$$g = \nabla L\left(f\left(\mathbf{x}, \mathbf{w}\right), \mathbf{t}\right)$$

and 0.9 is suggested for γ.

Adam [Kingma and Ba, 2014] Adam adds the momentum term back into RMSProp by applying it to the gradient scaling term. It also incorporates bias corrections to the estimates of both the momentum and gradient scaling terms.

formula:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\, g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \end{aligned} \tag{3.15}$$

The best optimization algorithm There is no such thing as an optimisation algorithm that performs best in every circumstance. The modern algorithms outlined above have been designed to perform well in specific cases discussed in their descriptions. However it has been shown that vanilla stochastic gradient descent with momentum performs well in a wide variety of applications [Goodfellow et al., 2016], so it is frequently used. This is the algorithm used in this work, as the focus is on generality across many different tasks as opposed to the best optimisation algorithm for a specific task.

3.7 Other training methods

There are a few other methods that are less frequently used to optimize neural networks but have been shown to be effective in particular scenarios. These include:

- second order methods that make use of second derivatives to improve optimization,
- genetic algorithms that optimize weights without use of any gradient information.

These methods are beyond the scope of this thesis.

3.8 Weight initialisation

The optimisation algorithms that are used in this work are iterative and require the user to specify the starting point of the weights to begin iteration. The loss surface for deep neural networks is usually extremely complex and non-convex, so back-propagation optimization algorithms that are designed to rely on local gradient information for learning are strongly affected by the choice of initialization point. The

initialization point chosen can determine how quickly the optimization algorithm converges, how close the point it converges to is to the global minimum and sometimes even whether it converges at all. It can also determine whether the convergence point overfits or generalises well.

The most important point to consider when initialising weights is breaking symmetry. If two neurons in the same layer of a fully connected feed forward neural network are initialised with the same weights to begin with then they will continue to have the same weights throughout training and a redundant neuron will be created [Goodfellow et al., 2016], unless different updates for different weights are created through dropout or similar.

Weights are generally initialised from either a uniform or gaussian distribution. The size of the hyperparameters for each distribution is more important than which one is chosen. This is because choosing the scale requires balancing two competing factors:

1. larger values produce more differences in the weights and less likelihood of symmetry.

2. larger values also produce a strong prior as they are more difficult to change without the algorithm getting stuck in local minima. So the final optimisation point is likely to be not too far from the initialisation point.

In the end, point two has a stronger influence and modern initialisation algorithms tend to be scaled by the number of inputs m and outputs n to a layer. For example Xavier Uniform [Glorot and Bengio, 2010]:

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right) \tag{3.16}$$

Kaiming Initialization [He et al., 2015] is an initialisation method for neural networks that takes into account the non-linearity of activation functions, such as ReLU activations. A proper initialization method should avoid reducing or magnifying the magnitudes of input signals exponentially. Using a derivation the authors work out that the condition to stop this happening is:

$$\frac{1}{2} n_l Var[w_l] = 1$$

where $n_l = k_l^2 d_l$, k_l is the kernel size, and d_l is the number of channels. This leads to an initialization of:

$$w_l \sim \mathcal{N}\left(0, \frac{2}{n_l}\right) \tag{3.17}$$

We use Kaiming initialization in this work as it is specifically designed for ReLU activation functions and it performs well in general in a variety of optimisation spaces.

Chapter 4

Specialised Deep Neural Network Architectures and Training Strategies

To improve learning in the complex non-convex weight optimization space, modern deep neural networks tend to be organised in task specific architectures. These depend on the structure of both the input data and the patterns that need to be learned in order to solve the task.

4.1 Traditional CNN architecture

Convolutional Neural Networks (CNNs) [LeCun et al., 1989] were originally designed for image data, but have been shown to work well on a variety of grid like data with feature equivariance. For example sound waves and language [Oord et al., 2016; Palaz et al., 2013; Hu et al., 2014; Kalchbrenner et al., 2014].

4.1.1 Convolution layers

CNNs use convolution layers with sparse weights and parameter sharing to achieve equivariance. A convolution layer consists of a set learnable kernels that are convolved with the input data to produce a feature map. One dimensional discrete convolution for neural networks is defined as follows:

$$feature\ map_i = weights \bigotimes input + bias_i \tag{4.1}$$

$$= \sum_f (\sum_m (w_{f,m} \times x_{i+m})) + bias_i \tag{4.2}$$

where each filter **w**, has predefined length m less than the input length and there are f filters. More dimensions can be added by summing over them as well. For example two dimensions:

$$feature\ map_{ij} = weights \bigotimes input + bias_{ij} \tag{4.3}$$

$$= \sum_{f} (\sum_{n} (\sum_{m} (w_{f,m,n} \times x_{i+m,j+n}))) + bias_{i,j} \tag{4.4}$$

Convolving the kernel with one section of input produces one neuron, so one convolution operation can be thought of as a fully connected neural network in that section of data. The kernel moves across the input in each dimension in turn as shown in Figure 4.1.

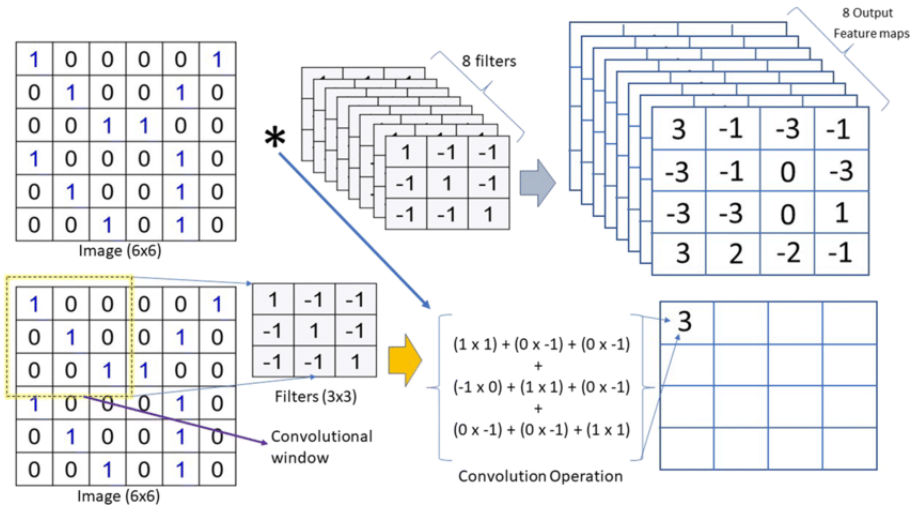


Figure 4.1: Convolution [Singh et al., 2020].

The output of a convolution layer is called a feature map as it maps the location of the output of each application of the kernel, see Figure 4.1.

The number of kernels in a layer is a hyperparameter set by the user, as is the size of the kernels. The number of kernels essentially defines the number of different features that can be learned in a particular layer and the size of the kernel determines the size of the features to be learned and the efficiency of the calculations. The number of kernels generally increases from lower layers to higher layers throughout the network as lower layers tend to learn simple general features and higher layers tend to learn more complex task specific features, as shown in Figure 4.2.

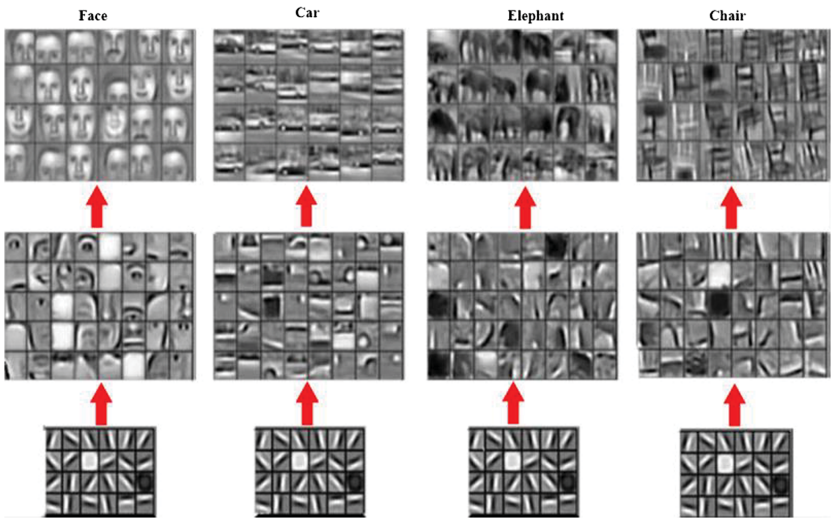


Figure 4.2: CNN layers feature responses [Lee et al., 2009].

All the neurons in a feature map share the same weights but a different part of the input as shown in Figure 4.1. The sparsity makes the calculations more efficient and the weight sharing reduces the number of free parameters and helps prevent overfitting.

This sparsity and weight sharing makes creates a property called equivariance to translation:

$$f\left(T\left(x\right)\right) = T\left(f\left(x\right)\right) \tag{4.5}$$

where T is a translation. In the case of the convolution operation in neural network this means that the kernel applied to one area of the input creates the same output when applied to a different area of the input, just in a different location in the feature map output. This means that they perform well on data that contains the same features in different locations. For example in image classification a cat is still a cat whether it is on the ground or on a table, and in speech to text a word still needs to create the same output whether it's said at a higher or lower frequency or at the start or end of a sentence.

Another hyperparameter that determines the behaviour of a convolution layer is stride length. The stride length determines how the kernel moves across the input space. If the stride length is one it will move one space across each dimension of the

input in turn. If the stride length is two the kernel will move two spaces across each time and so on. Figure 4.3 shows a stride length of one and two respectively.

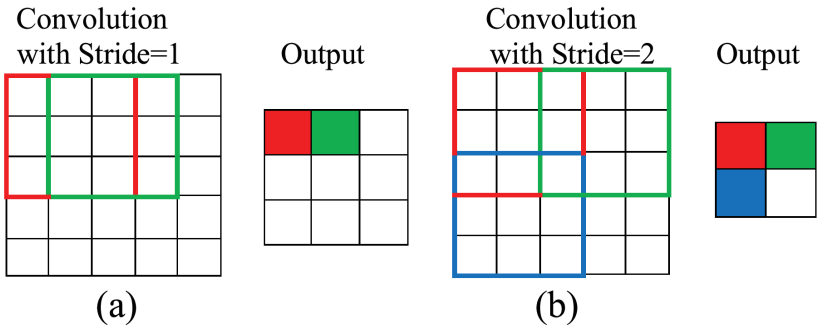


Figure 4.3: Stride lengths [Yepez and Ko, 2020].

An activation function is applied to each neuron in the feature map output from the convolution layer as per a fully connected neural network.

4.1.2 Pooling

Traditionally CNNs generally included pooling layers at regular intervals throughout the network. A pooling layer replaces the outputs from the feature map with summary statistics to smooth and/or reduce the feature map size and improve the efficiency of the CNN. A pooling function has predefined dimensions and is applied to each section of the input like a kernel in a convolution layer as shown in Figure 4.4.

There are several different pooling functions. The most common pooling operations are average pooling which takes the average over each section of input, and max pooling takes the maximum, see Figure 4.4. Others include weighted average and L^2 norm.

Pooling also adds a small amount of translation invariance.

Modern CNNs often use stride lengths of two to decrease the feature map size and a clever combination of two convolution layers with kernel size of one and stride of two to create a pooling type function with learned parameters rather than having to choose the type of pooling in advance. However global average pooling which averages over the full final feature map for each kernel has been shown to be effective [Lin et al., 2013].

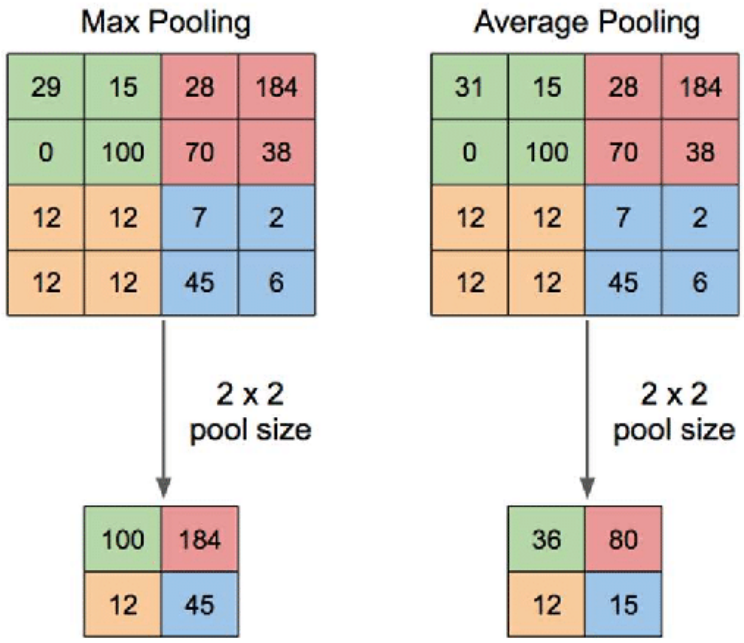


Figure 4.4: Pooling [Yani et al., 2019]

4.1.3 Fully connected layers

Traditionally CNN architectures had one or more fully connected layers as described in Section 3, that took the features output from the final CNN or pooling layer and made the final prediction.

4.1.4 Examples of early CNN architectures

Figure 4.5 shows the first CNN architecture LeNet [LeCun et al., 1989].

AlexNet, [Krizhevsky et al., 2012] shown in Figure 4.6, was the first neural network to win the ImageNet 1k [Deng et al., 2009] image classification challenge resulting in a sudden increase in interest in the application of and improvement of CNN models.

4.2 Modern CNN designs

CNNs and other deep neural network architectures have exploded in popularity over the past decade [Krizhevsky et al., 2012; Girshick et al., 2014; Li and Deng, 2020; Masi

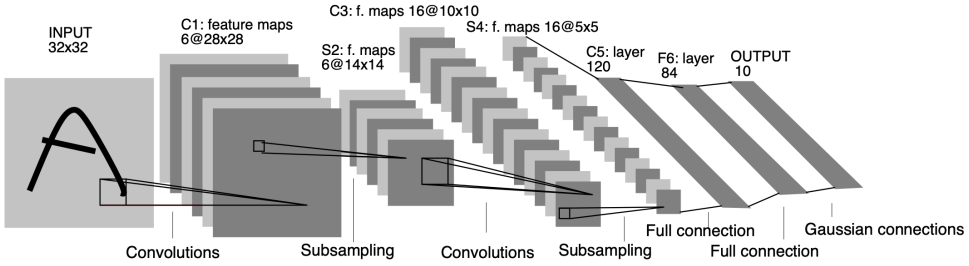


Figure 4.5: LeNet [LeCun et al., 1989]

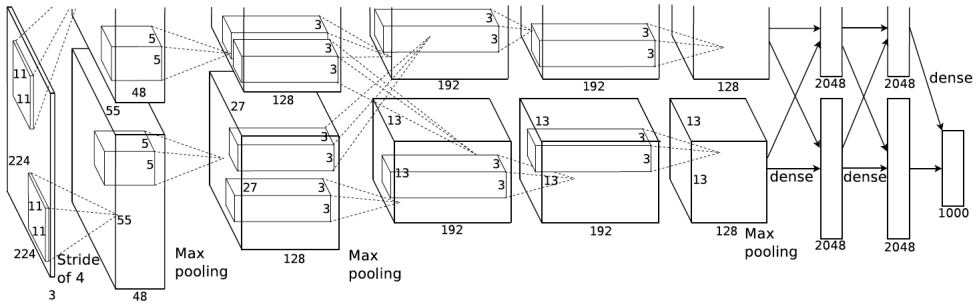


Figure 4.6: AlexNet

Large 3D blocks are feature maps output from convolutional layers and small 3D blocks within them are kernels. 2D blocks are fully connected layer output vectors [Krizhevsky et al., 2012].

et al., 2018; Mazurowski et al., 2019]. With this increased interest has come a large number of improvements to architectures and learning algorithms.

4.2.1 Residual Connections

Residual connections address the problems of getting gradients back to the lowest layers with very deep modern CNNs. As shown in Figure 4.7 with a residual connection the output from one or more layers is concatenated together with the original input to create the final output and input to the next layer. This creates "short-cuts" through the network when backpropagating the gradients.

Modern style residual connections were first introduced in the ResNet family of architectures [He et al., 2016]. This is still a popular architecture today as it can be scaled depending on available compute resources and delivers good results for all

model sizes.

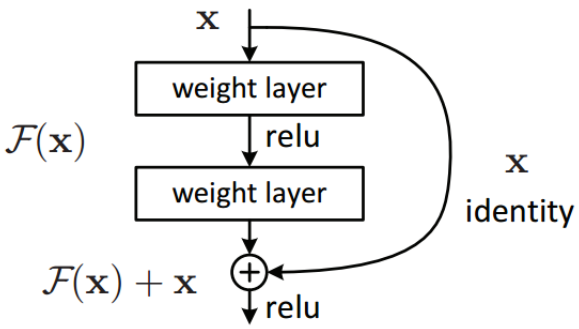
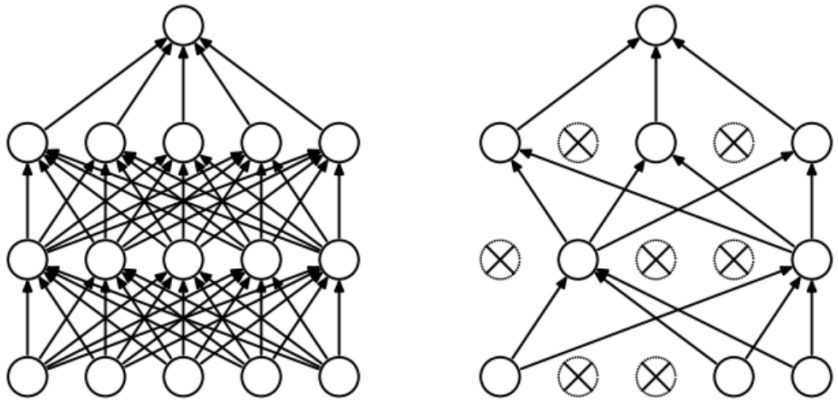


Figure 4.7: Residual block [He et al., 2016].

4.2.2 Dropout

If dropout [Hinton et al., 2012b] is used, it is usually applied to a fully connected layer after the convolution layers. It "drops", sets the output to zero, for each neuron in the fully connected layer with a set probability for each training batch. At test time each neuron is used but is scaled by the dropout probability. For example if the dropout probability is 50% then roughly 50% of neurons will be dropped each time during training, then at test time all neurons will be used but the outputs will be scaled by 50%. The dropout probability is a hyperparameter set by the user. This trains neurons to be more self-sufficient and have a unique function as they cannot rely on the presence of other features to create output. An example of dropout is shown in Figure 4.8.

Dropout can also be thought of as a way of augmenting the features from the convolutional layers. Only a certain percentage of the features produced by the convolutional layers are used for training batch and the features that are used are different each time. This means for a model that outputs a large number of features and has a reasonably large dropout rate, the features used for prediction are likely to be different most if not every time a particular training example is used. For example the Inception_v4 model [Szegedy et al., 2017] with global average pooling after the final convolution layers outputs 1,536 features. If the dropout probability is 0.5 then there are $2^{1,536}$ combinations of features that can be produced by applying dropout and the probability of a particular training example being trained with the exact same features active twice during training would be small.



(a) Standard Neural Net (b) After applying dropout.

Figure 4.8: Dropout applied to three fully connected layers [Hinton et al., 2012b]

4.2.3 Normalisation

It is important to normalise the initial inputs to the network so all input features have the same mean and standard deviation, as otherwise large input features with large variance are likely to have a bigger initial influence on training than smaller features with smaller variance. This effect can be undone through training as the model should learn the appropriate weights, but it makes training slower and more likely to get stuck in a sub-optimal local minima.

Internal covariate shift In deep neural networks outputs to lower layers become inputs to the layers above. As the weights for each layer are trained the outputs change, thus without some sort of normalisation higher layers will be training on a continuously changing input. This has been shown to slow down training and make it more sensitive to the learning rate [Ioffe and Szegedy, 2015].

To counteract internal covariate shift the outputs from each convolution layer should be normalised. There are several ways to do this, with each one performing the following calculation:

$$\hat{x}_i = \frac{1}{\sigma_i}(x_i - \mu_i) \tag{4.6}$$

where μ_i and σ_i are the mean and standard deviation respectively calculated over a particular set of feature map outputs x_i as shown in Figure 4.9. All methods listed

below then learn a per channel linear transform to compensate for the loss of representation ability:

$$yi = \gamma \hat{x}_i + \beta \tag{4.7}$$

Batch normalisation performs normalisation with mean and variances calculated along the batch dimension N. This means that it can be strongly affected by batch sizes. It also creates issues as the concept of a batch is not always present or may change.

Several normalisation methods have been proposed to address the problem of batches in batch normalisation. *Layer normalisation* acts along one instance in all channels and *instance normalisation* works along one instance in one channel C. However none of the proposed methods have been able to consistently achieve the same performance as batch normalisation across a variety of image recognition tasks. [Wu and He, 2018].

Group normalisation [Wu and He, 2018] is in between layer and instance normalisation and works on one instance across multiple channels. The number of instances in a group is defined by a hyperparameter number of groups G, with the number of instances being C/G. Group normalisation has been shown to have similar performance to batch normalisation for moderately sized batches across a variety of image recognition tasks, with improved performance as the batch size is reduced.

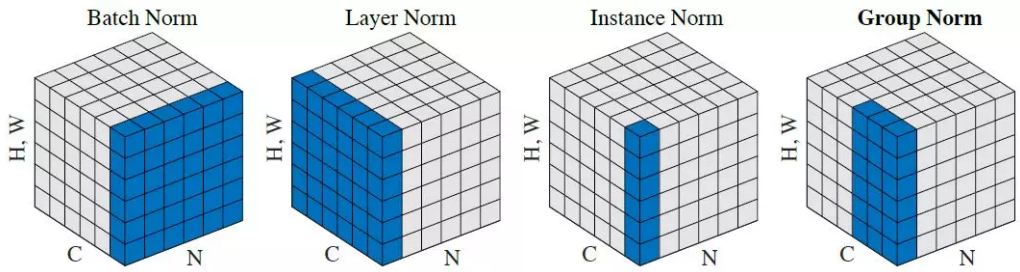


Figure 4.9: Types of normalisation

Each subplot shows a feature map. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels. N is the number of instances in a batch, C is the number of channels in this layer and H and W are the height and width of one instance. In this case W is one so the fourth dimension has been removed [Wu and He, 2018]

4.2.4 Regularisation

Modern deep models with millions of parameters are generally trained with some form of weights regularisation to address overfitting. Standard weights regularisation takes the form of an *Lnorm* term applied to the difference between the weights and 0 added to the loss function.

$$min_w L(w) = \left\{ \frac{1}{n} \sum_{i=1}^{n} L(z(x_i, w), y_i) + \frac{\alpha}{2} \|w - 0\|_p \right\} \tag{4.8}$$

The smaller the value of p, the more the weights are encouraged to be sparse. In Figure 4.10 it can be seen that the L_1regularizer, also known as the lasso, will drive some weights to 0 with sufficiently large α, whereas the L_2 term gives no preference for 0 weights.

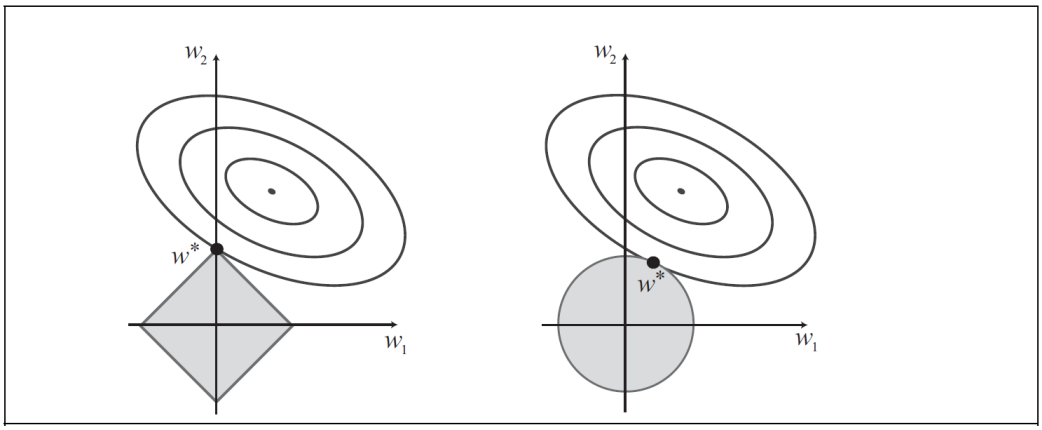


Figure 4.10: L1 (left) compared to L2 regularisation (right) [Stuart and Peter, 2016].

4.2.5 Mixup

Mixup [Zhang et al., 2017a] performs data augmentation throughout training by mixing two randomly draw training examples as follows:

$$x^* = \lambda x_i + (1 - \lambda) x_j \tag{4.9}$$

where x_i, and x_j are input vectors

$$y^* = \lambda y_i + (1 - \lambda) y_j \tag{4.10}$$

where y_i, and y_j are one-hot label encodings

Mixup works on the assumption that linear interpolations of feature vectors should result in linear interpolations of the associated targets. This can help prevent large models from overfitting idiosyncrasies of the training data. Mixup has been shown to improve state of the art performance on a variety of image classification datasets.

4.2.6 Modern CNN architectures

An early application of residual connections was the ResNet family of architectures Figure 4.11. Scaled up versions of ResNet architectures by both depth and width, plus hybrids with other architectures are still comparable with the state of the art.

Another successful family of architectures is Inception. Inception v4 performs well on ImageNet 1K and was shown to perform better than all other available architectures at the time for transferring from ImageNet 1K to a broad range of target tasks in [Kornblith et al., 2019]. As shown in Figure 4.12 after an initial stem, the Inception models consist of repeated blocks of alternate paths of convolutional and other layers. The blocks consist of simple paths with one convolution layer for standard blocks or one max pooling layer for blocks that reduce the size of the feature maps. These simple paths are similar to the residual paths in ResNet models and they are concatenated with the output of more complex paths with multiple convolutional and/or pooling layers that provide the complex hierarchical representation power of the models.

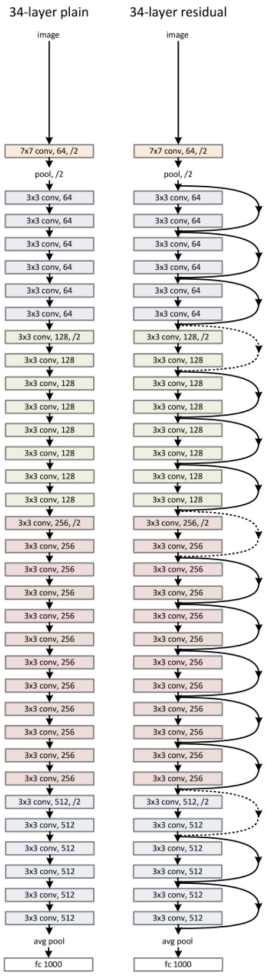


Figure 4.11: ResNet versus plain 34 layer CNN [He et al., 2016].

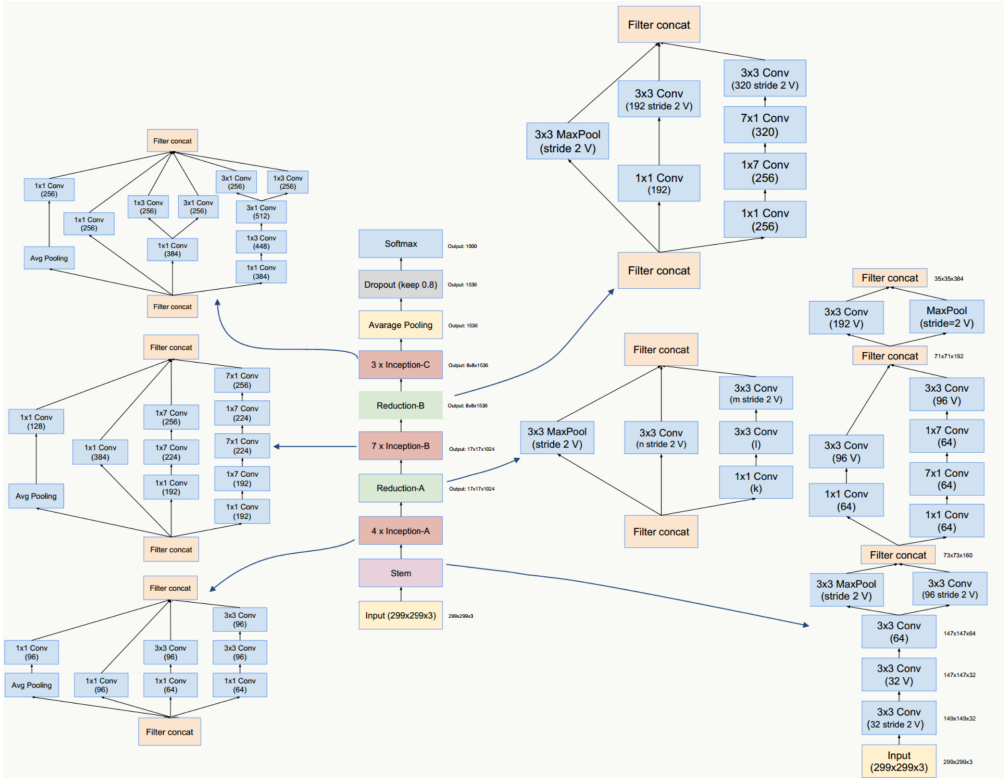
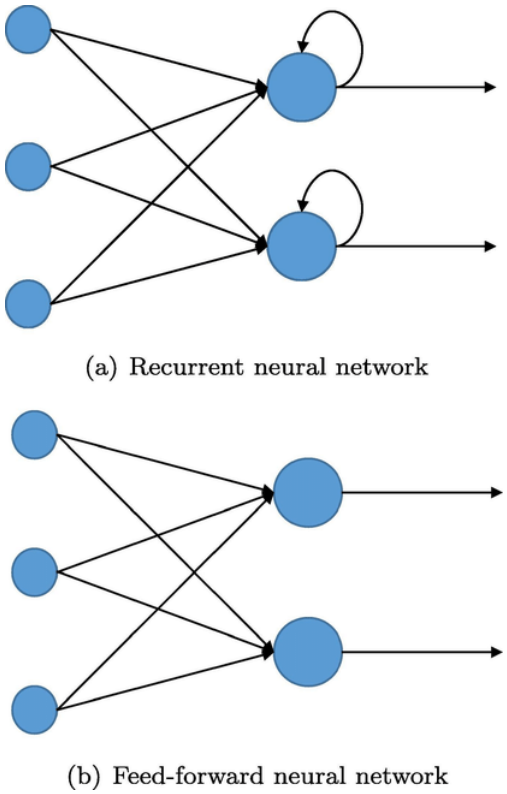


Figure 4.12: Inception v4 [Szegedy et al., 2017]

4.3 Recurrent neural networks

In the previous Chapters we considered feed forward neural networks where weights connect a neuron in one layer to a neuron in a subsequent layer. Recurrent neural networks (RNNs) relax this assumption to deal with data that is in the form of a sequence of vectors. RNNs are explicitly designed to model time sequences by introducing a weight that connects the output of each hidden neuron at one time step to become the input of the same neuron at the next time step as shown in Figure 4.13.



(a) Recurrent neural network

(b) Feed-forward neural network

Figure 4.13: Feedforward connection compared to recurrent connection [Do et al., 2019].

RNNs can be used to model sequences in both the input data and or the output data. Figure 4.14 shows the many different forms of sequence data that can be modelled by an RNN. A standard feedforward neural network has no way of modelling a relationship between a previous input and a current output, whereas in a RNN the recurrent connections mean that information persists across time steps and the model contains a memory.

As for a convolutional neural network and other specialised architectures, a final

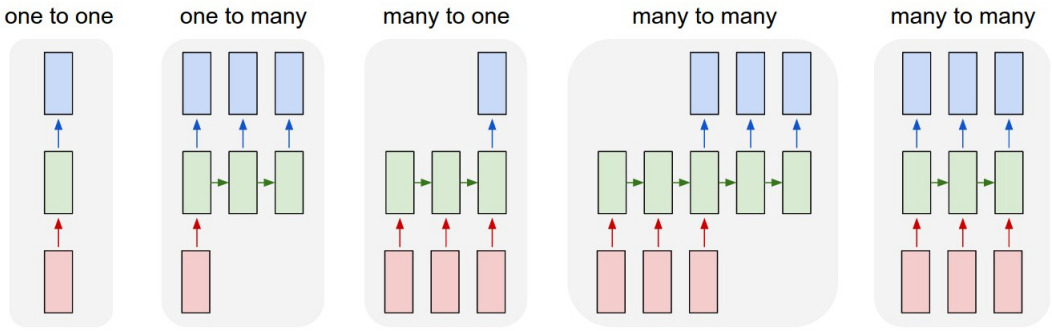


Figure 4.14: Types of sequences [Karpathy, 2015].

feed forward fully connected layer is generally added on top of one or more recurrent layers to perform the final prediction task using the features produced by the recurrent layers.

4.3.1 Forward pass

The forward pass is similar to a standard feed forward neural network with the only difference being that the hidden layer now has an input from the previous time step as well as a new input vector:

$$\mathbf{h}_t = f(\mathbf{w}_{hh}\mathbf{h}_{t-1} + \mathbf{w}_{xh}\mathbf{x}_t) \tag{4.11}$$

where $\mathbf{w}_{hh}$ are the hidden to hidden weights, $\mathbf{w}_{xh}$ are the input to hidden weights , $\mathbf{h}_t$ is the output of the hidden layer at time t and f is the activation function.

The hidden state at time 0 (h_0) needs to be initialised for each new sequence. This is often done as a random initialisation or fixed constant such as 0, but can also be learned, or transferred from the last time step of the previous sequence if sequences are in order. From there the sequence is looped through in order with the output of each hidden state becoming the input to the next one as shown in Algorithm 1.

4.3.2 Backwards pass

A recurrent neural network can be unrolled through time to take a form similar to a deep feed forward neural network, see Figure 4.15. Then backpropagation through time [Pineda, 1987] can be applied via repeated applications of the chain rule. The

Algorithm 1 Forward pass for a RNN with one hidden layer and inputs and outputs at each time step

initialise $\mathbf{W_{hh}}, \mathbf{W_{xh}}, \mathbf{h}$
loop through for each step in sequence
for $t = 1$ to T:
  #update the hidden state
  $\mathbf{h} = f((\mathbf{W_{hh}} \cdot \mathbf{h}) + (\mathbf{W_{xh}} \cdot \mathbf{x}))$
  # compute the output vector
  $\mathbf{y} = (\mathbf{W_{hy}} \cdot \mathbf{h})$

difference being that all the weights $\mathbf{w}_{hh}, \mathbf{w}_{xh}, \mathbf{w}_{hy}$ are constrained to be the same value at each timestep. We sum over the whole sequence of gradients to get the final value for each of the network weights to perform the weight update.

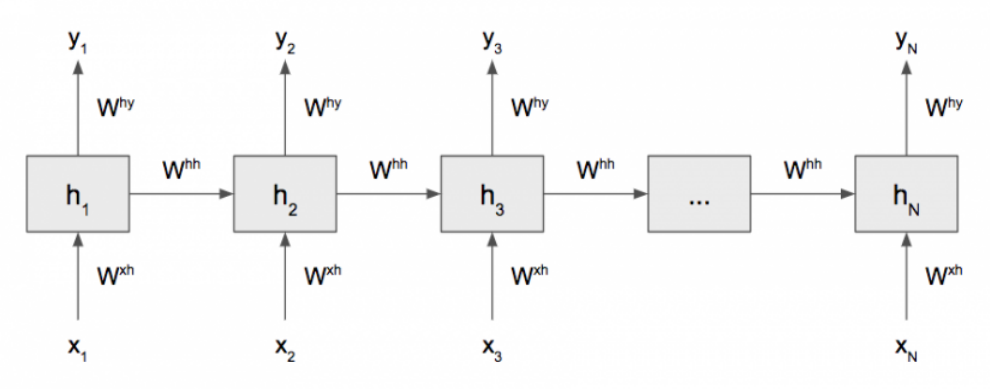


Figure 4.15: RNN unrolled through time [Deshpande, 2017].

4.3.3 Long Short-Term Memory (LSTM) recurrent neural networks

With just one state vector and weight vector learning the relationship over time and being constantly overwritten, long term dependencies are difficult to model so the range of contexts that can be learned is limited. In theory the backpropagation through time algorithm allows us to train RNNs using gradient descent. In practice RNNs are unstable and when backpropagating gradients through long time windows, the gradients can explode or vanish. This is part of the vanishing gradient problem as shown in Figure **??**.

An effective solution to the vanishing gradients problem is the Long Short Term Memory (LSTM) RNN [Hochreiter and Schmidhuber, 1997] and other similar variants. LSTMs have been shown to be effective at modelling sequences that are 100s or

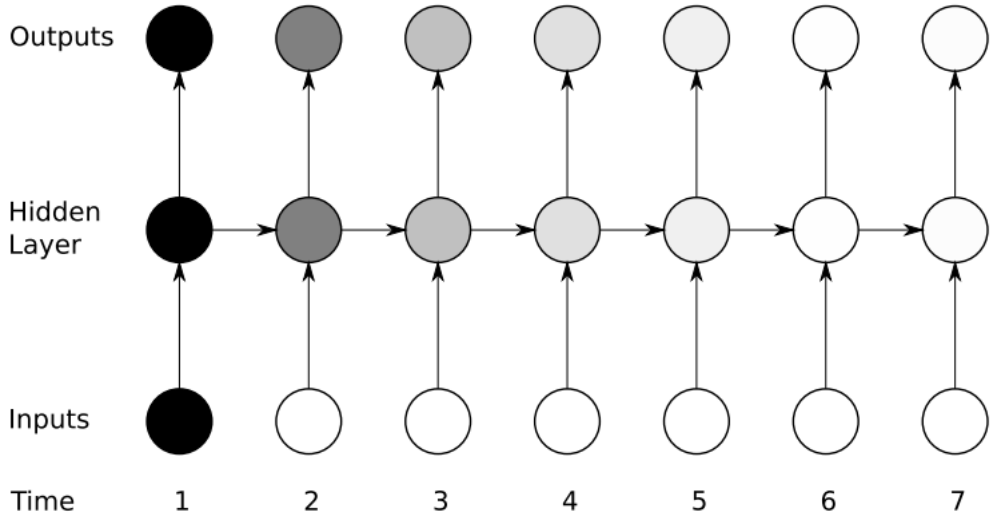


Figure 4.16: The sensitivity of an RNN to old inputs decays exponentially over time [Graves, 2012]. The shading of the nodes indicates the sensitivity over time of the network nodes to the input at time one (the darker the shade, the greater the sensitivity)

even 1,000s of steps long [Graves, 2012].

LSTMs follow the same basic structure as RNNs except that the nonlinear units in the hidden layer are replaced by memory blocks. Each memory block contains one or more memory cells along with input, output and forget gates which control the flow of information into and out of the memory cell as shown in Figure 4.17. The forward pass for the gates, cell input and output, and hidden layer output are as follows:

Input gate: $i = \sigma(W_{ii}x + W_{hi}h^{t-1})$
Forget gate $f(t) = \sigma(W_{if}x + W_{hf}h^{t-1})$
Output gate $o(t) = \sigma(W_{io}x + W_{ho}h^{t-1})$
Input modulation $g = tanh(W_{ig}x + W_{hg}h^{t-1})$
Memory cell $c^t = f * c + i * g$
Hidden layer output $h^t = o * tanh(c^t)$

4.4 Reinforcement learning

The reinforcement learning environment is defined by [Sutton and Barto, 2018] as

Definition 4.1. Reinforcement learning is learning what to do—how to map situa-

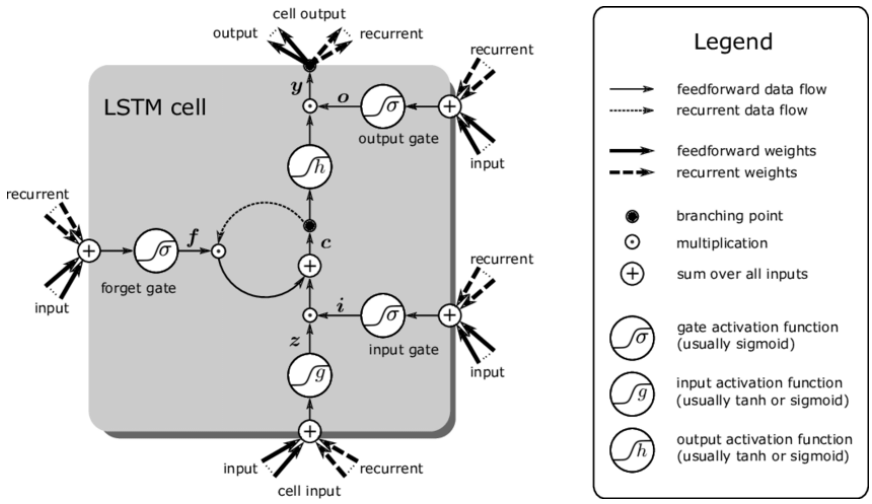


Figure 4.17: LSTM memory cell [Arras et al., 2019].

tions to actions—so as to maximize a numerical reward signal.

In the reinforcement learning environment a model is not given a training set with the correct action to take for each training example, but instead must discover which actions yield the most reward by trying them. It is an interaction between an active decision making agent and its environment, within which the agent seeks to achieve a goal despite uncertainty about its environment.

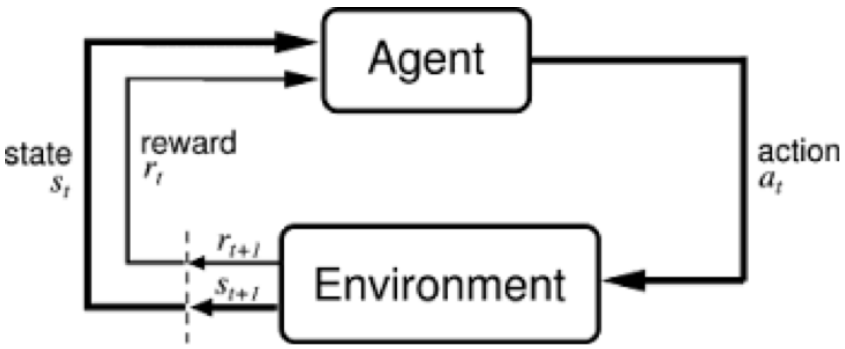


Figure 4.18: Reinforcement learning environment [Sutton and Barto, 2018].

The reinforcement learning environment is summarised in Figure 4.18. The terms for the elements of the set up are:

Action ($a \in A$): A is the set of all possible moves the agent can make. A can be

discrete or continuous. a is a particular action taken by an agent.

State ($s \in \mathbb{S}$): A state s is a representation of information the agent receives about its current environment. The entire state space $\mathbb{S}$ can be finite or infinite and discrete or continuous.

Reward ($r \in \mathbb{R}$): A reward is the feedback by which we measure the success or failure of an agent's actions and the training signal, they are usually sparse.

Discount factor ($0 < \gamma \leq 1$): The discount factor is multiplied with future rewards in order to dampen their effect on the agent's choice of action. It makes future rewards worth less than more immediate rewards

Policy ($\pi(a|s)$): The policy is the strategy the agent employs to determine the next action based on the current state. It is the mapping of states to the probabilities of selecting each possible action.

Value $V_{\pi(s)}$: The expected long-term return of the current state under policy π.

Rewards can be:

1. Sparse, in that the model only receives occasional rewards after a number of actions.

2. Affected by past actions for an indeterminate amount of time.

3. Non-deterministic in that the same action in the same state may lead to different rewards from the environment.

An example of an environment with all three of the challenges listed above is the game of Chess. The model would generally only get a reward for a win and there is no way of knowing whether an interim move is right or wrong, a bad move early on may affect the whole game, and making one bad move against a poor player may still result in a win, whereas against a strong player it will likely result in a loss.

In this environment the agent seeks to maximise the long term reward, which for a finite number of actions T is just the discounted sum of each reward received:

$$R_{t_0} = r_t + \gamma r_{t_0+1} + \gamma^2 r_{t_0+2} + ... + \gamma^{T-t} r_T = \sum_{t=t_0}^{T} \gamma^{t-t_0} r_t \tag{4.12}$$

The value function is then the expected return of starting in state s and following policy p from then on [Sutton and Barto, 2018]:

$$V_{\pi(s)} = E_\pi[\sum_{t=t_0}^{T} \gamma^{t-t_0} r_t | S_t = s] \forall s \in \mathbb{S} \tag{4.13}$$

An environment with a finite number of actions is called episodic, and one chain of actions an episode.

4.4.1 Deep reinforcement learning

In this thesis, experiments are performed in an extremely large action space. For this reason we use deep reinforcement learning where the optimal action in a state is predicted by a deep neural network as shown in Figure 4.19.

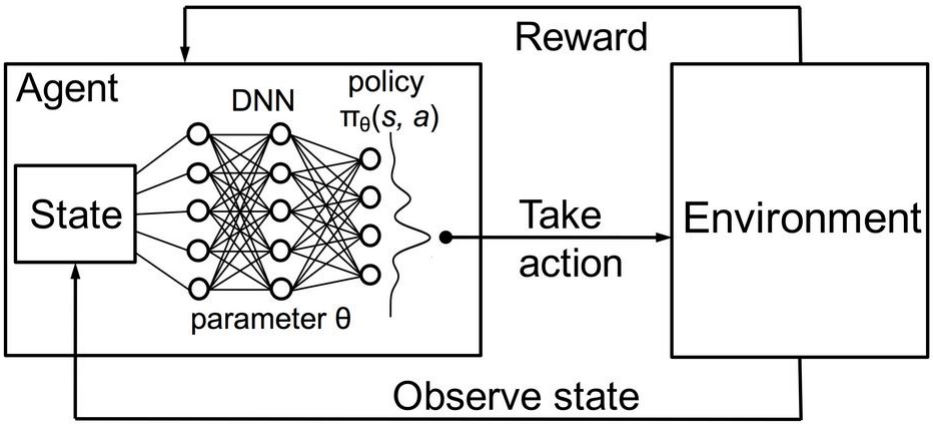


Figure 4.19: Deep reinforcement learning [Mao et al., 2016].

4.4.1.1 REINFORCE algorithm

One way of dealing with the challenges of sparse and non-deterministic rewards in an episodic reinforcement learning environment is through directly optimising in the policy space via the REINFORCE algorithm [Williams, 1992; Williams and Peng, 1991]. We are trying to maximise the expected value of the reward defined by equation 4.12 given the current weights weights w. This is a special case of the general formula for the expectation of some scalar valued score function $f(x)$ (the reward function) under some probability distribution $p(x;w)$ parameterized by w:

$$\mathbb{E}[f|w] = \sum_x p(x \mid w) f(x)$$

where in this case $p(x \mid w)$ is the probability of taking a particular action in a particular state and $f(x)$ is the reward from taking that action. We can find the gradient of this value as follows:

$$\nabla_w \mathbb{E}[f|w] = \nabla_w \sum_x p(x \mid w) f(x)$$

$$= \sum_x f(x) \nabla_w p(x \mid w)$$

substituting in $\nabla_w p(x \mid w) = p(x \mid w) \frac{\nabla_w p(x|w)}{p(x|w)} = p(x \mid w) \nabla_w \log p(x \mid w)$

$$= \sum_x f(x) p(x \mid w) \nabla_w \log p(x \mid w)$$

$$= \mathbb{E}[f|w \nabla_w \log p(x \mid w)]$$

for an episodic case with N steps this is approximated by:

$$\approx \sum_{i=1}^{N} f(x_i) \nabla_w \log p(x_i \mid w)$$

In English, the term $\nabla_w log p(x \mid w)$ gives us the direction we would need to move the parameters w to increase the probability of action x, and $f(x_i)$ is the score of x (the total reward). So if the total reward after taking action x is positive this will move the parameters in the direction to make x more likely and if it is negative it will move the parameters in the opposite direction.

Using reinforcement learning terms our objective function to maximise becomes:

$$E[Rt] = \sum_{i=1}^{N} R_i log \pi(a_i|s_i, w) \tag{4.14}$$

and the loss function we want to minimise is:

$$L(w) = \sum_{i=1}^{N} R_i \left(-log \pi(a_i|s_i, w)\right) \tag{4.15}$$

with a discount factor included this is:

$$L(w) = \sum_{i=1}^{N} \gamma^i R_i \left(-log \pi(a_i|s_i, w)\right) \tag{4.16}$$

From here the gradients of the weights can be calculated as usual by applying the

chain rule of differentiation recursively to the loss function.

4.4.1.2 REINFORCE extensions

There are many extensions to the basic REINFORCE algorithm described above. Two are relevant to this work.

One drawback of the basic REINFORCE algorithm is the high variance in estimating the gradient of $E[Rt]$. Each time we perform a gradient update, we are using an estimation of gradient generated by a series of data points <s,a,r,s´> accumulated through a single episode of game play. This is known as the Monte Carlo method. This estimate can be very noisy, and a bad gradient estimate can adversely impact the stability of the learning algorithm. For example if R is always positive then every update will be in the direction of making the action more likely, no matter how it compares to other actions.

A simple way to reduce the variance in Policy Gradients is to subtract a baseline from the reward in the Policy Gradients formula:

$$L\left(w\right) = \sum_{i=1}^{N} \gamma^{i} \left(R_i - B\right) \left(-log\pi(a_i|s_i, w)\right) \tag{4.17}$$

Subtracting a proper baseline ensures that the scaling factor (Rt−B) is only positive if the action is better than average and negative if the action is worse than average. Empirically, this trick reduces the variance of the gradient estimate quite drastically. The baseline can be any function, even a random variable. As long as it does not vary with the action a. The baseline should vary with state. In some states all actions have high values and we need a high baseline to only have positive updates for the better actions; in other states all actions will have low values and a low baseline is appropriate.

Another extension to the REINFORCE algorithm is REINFORCE/MENT [Williams, 1992; Williams and Peng, 1991] which adds an entropy regularisation term:

$$L\left(w\right) = \sum_{i=1}^{N} \left(\left(R_i - B - \tau H\left(s, \pi\right)\right) \left(-log\pi(a|s, w)\right)\right) \tag{4.18}$$

where H is the entropy of the policy distribution:

$$log\pi(a|s, w)$$

and $0 < \tau < 1$ is a hyperparameter that controls the degree of entropy regularisation.

Entropy regularisation encourages exploration and helps prevent early convergence to sub-optimal policies [Williams and Peng, 1991; Williams, 1992].

4.5 Deep Learning Hyperparameters

There are many hyperparameter decisions that need to be made in advance when creating and training a deep learning model for a particular task. The decisions to be made when creating a model include:

1. The general type of architecture, for example, a CNN or a RNN.
2. The specific details of the architecture, for example, for a CNN how many layers should be used, how many filters should be in each layer, what shape should those filters be, should skip connections or residual connections be used etc.
3. The hidden and final layer activation functions.

The decisions to be made when training a model, discussed in the section numbers given, include:

1. the loss function to represent performance on the task, Section 3.4,
2. the optimizer, Section 3.6,
3. the learning rate and how to decay it over the length of training, Section 3.5.1
4. the type of weight initialisation, Section 3.8
5. the type of regularisation if any, Section 4.2.4,
6. the type of normalisation, Section 4.2.3,
7. the type of data augmentation if any, Section 4.2.5.

It can be extremely challenging to find the combination of these hyperparameters that will achieve optimal performance on a particular task, especially considering that all the hyperparameters interact with each other. For some of the more simple ones such as learning rate, regularisation etc, a grid search within a sensible range can be used. However, for others such as specific architectural decisions the space is too large and even in the simple cases a grid search can be out of reach for many people without large compute resources. Other strategies have recently been created to automate these decisions in particular categories, they are also computationally

expensive, but progress is being made to reduce the reliance on massive compute. At this stage though it is common for many researchers and users to ignore most of the hyperparameters and their interaction and use existing tested designs. In some cases, especially when the training dataset is small this can result in significantly suboptimal performance. This will be expanded on in Chapters 5 and 6.

4.5.1 Neural Architecture Search

Neural architecture search (NAS) is a hyperparameter optimisation strategy that was originally designed to make the architectural decisions for designing a CNN. It has since been extended to many related hyperparameter decision tasks, including loss function [Li et al., 2019a] and activation function [Ramachandran et al., 2017] design, and data augmentation strategies [Cubuk et al., 2019].

The first NAS model [Zoph and Le, 2016] consisted of a controller and child model as shown in Figure 4.20. The controller model was an LSTM model trained with REINFORCE with baseline to predict a CNN architecture consisting of number of filters, filter height, filter width, stride height, stride width for each layer. The child model implemented and trained the specified CNN and the reward for the controller model was the final validation set accuracy for the child model.

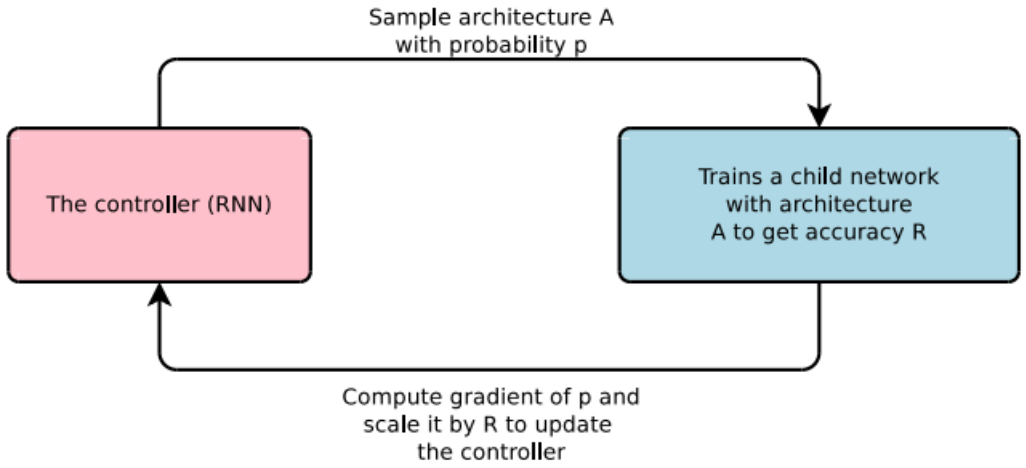


Figure 4.20: Neural architecture search [Zoph and Le, 2016]

Since 2016 there have been many advances in the field of neural architecture search Ren et al. [2021]. The major NAS algorithms use reinforcement learning Zoph and Le [2016]; Zoph et al. [2018], evolutionary algorithms Real et al. [2019],

or gradient-based optimisation Liu et al. [2018b]; Shin et al. [2018] as the optimization method, however there are some other less common categories as shown in Figure 4.21. The original NAS model used 800 GPUs for 30 days for training on the ImageNet 1K, which puts it out of reach for most researchers and users [Zoph and Le, 2016]. More recent advances have focused on reducing the huge cost in training a NAS model. This is done via various methods, including:

- jointly training the model hyperparameters and parameters by relaxing the hyperparameters to be continuous differentiable Liu et al. [2018b]; Shin et al. [2018],
- restricting the search space by putting constraints on the architecture Zoph et al. [2018]; Pham et al. [2018],
- using transfer learning to train subsequent architectures more quickly after the first or fine-tuning existing architectures Fang et al. [2020]; Cai et al. [2018],
- speeding up training time for candidate architectures via, searching for fast architectures by explicitly using the inference speed as a constraint **citation**, parameter sharing Pham et al. [2018], predicting the learning curve to allow for early stopping Baker et al. [2017]; Liu et al. [2018a].

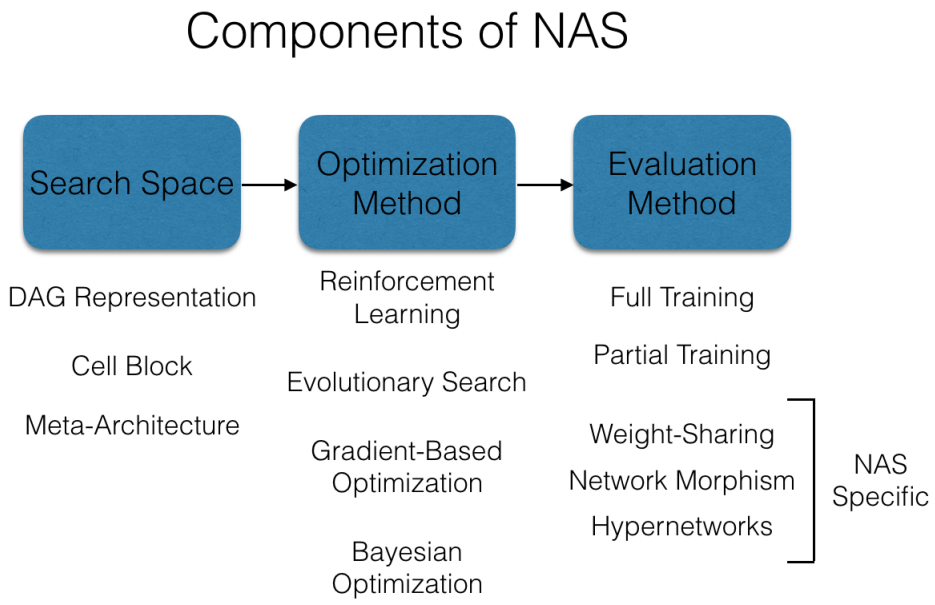


Figure 4.21: Advances in NAS models [Li and Talwalkar, 2018].

Chapter 5

Transfer Learning

The hierarchical structure of these networks allows for ever more complex patterns to be learned. This is one of the things that has allowed deep learning to be successful at many different tasks in recent years, when compared to other machine learning algorithms. However, this only applies if there is enough data to train them. Figure 5.1 shows the increase in ImageNet 1K performance with the number of model parameters. Figure 5.2 shows that for large modern CNN models in general the performance on ImageNet 1K increases with the number of training examples in the source dataset. This suggests that large modern CNNS are likely overfitting when trained from random initialization on ImageNet 1K. Of course there are some outliers as the increase in performance from additional source data also depends on how related the source data is to the target data. This is discussed further in in Section 6.4.6. These two results combine to show the stated effect that deep learning performance scales with the size of the dataset and model.

As noted in Section 1 there are many real world scenarios where large amounts of data are unavailable or we are interested in training a model on a small amount of data for other reasons.

5.1 Learning from small vs large datasets

Wang et al. give a thorough treatment of the problems of learning from a small number of training examples in [Wang et al., 2020b].

5.1.1 Empirical Risk Minimization.

We are interested in finding a function f that minimises the expected risk:

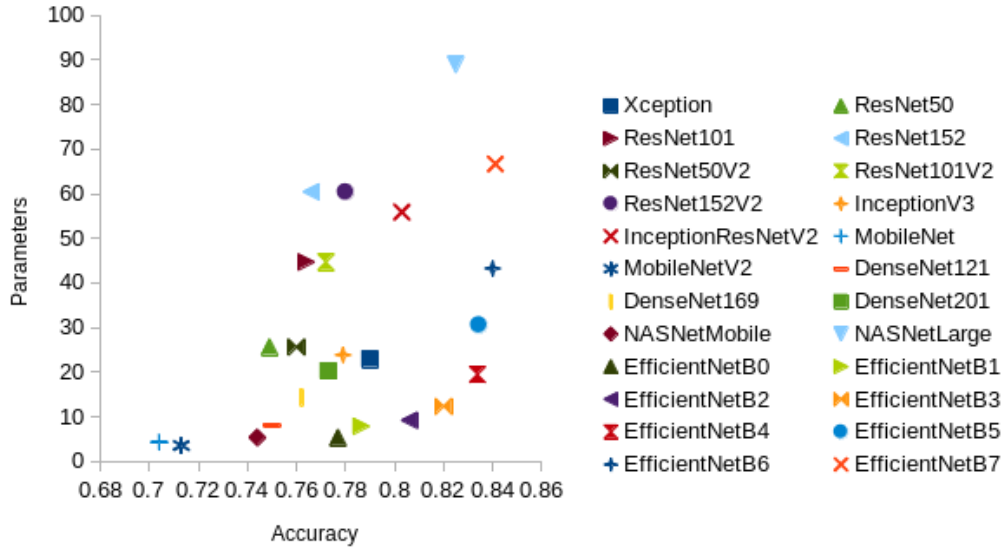


Figure 5.1: Increase in performance on ImageNet 1K due to model size, measured by number of parameters in millions

$$R_{TRUE}(f) = E[\ell(f(x), y)] = \int \ell(f(x), y)\, dp(x, y) \tag{5.1}$$

with

$$f^* = arg\, min_f R_{TRUE}(f)$$

$R_{TRUE}(f)$ is the true risk if we have access to an infinite set of all possible data and labels, with $\hat{f}$ being the function that minimizes the true risk. In practical applications however the joint probability distribution $P(x, y) = P(y|x)P(x)$ is unknown and the only available information is contained in the training set. For this reason the true risk is replaced by the empirical risk, which is the average of sample losses over the training set D

$$R_n(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i), \tag{5.2}$$

leading to empirical risk minimisation [Vapnik, 1992].

Before we begin training our model we must choose a family of candidate func-

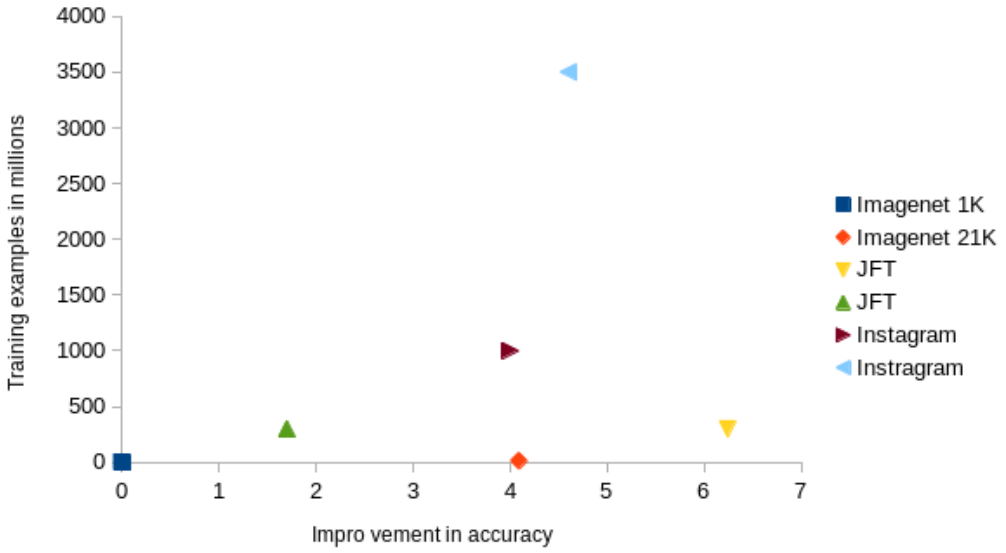


Figure 5.2: Percentage increase in performance on ImageNet 1K due to increased source dataset size

tions $\mathcal{F}$. In the case of CNNs this involves choosing the relevant hyperparameters that determine our model architecture, including number of layers, the number and shape of filters in each convolutional layer, whether and where to include features like residual connections and normalization layers, and many more. This constrains our final function to the family of candidate functions defined by the free parameters that make up the given architecture. We are then attempting to find a function in $\mathcal{F}$ which minimises the empirical risk:

$$f_n = arg\,min_f R_n\left(f\right) \tag{5.3}$$

Since the optimal function f^* is unlikely to be in $\mathcal{F}$ we also define:

$$f_{\mathcal{F}}^* = arg\,min_{f \epsilon \mathcal{F}} R_{TRUE}\left(f\right) \tag{5.4}$$

to be the function in $\mathcal{F}$ that minimises the true risk. We can then decompose the excess error that comes from choosing the function in $\mathcal{F}$ that minimizes $R_n\left(f\right)$:

$$E[R(f_n) - R(f^*)] = E[R(f^*_{\mathcal{F}}) - R(f^*)] + E[R(f_n) - R(f^*_{\mathcal{F}})] = \varepsilon_{app} + \varepsilon_{est} \quad (5.5)$$

The approximation error ε_{app} measures how closely functions in $\mathcal{F}$ can approximate the optimal solution f^*. The estimation error ε_{est} measures the effect of minimizing the empirical risk $R(f_n)$ instead of the expected risk $R(f^*)$ [Bottou and Bousquet, 2008]. So finding a function that is as close as possible to f^* can be broken down into:

1. choosing a class of models that is more likely to contain the optimal function, and
2. having a large and broad range of training examples in D to better approximate an infinite set of all possible data and labels.

5.1.2 Unreliable Empirical Risk Minimiser

In general, ε_{est} can be reduced by having a larger number of examples [Wang et al., 2020b]. Thus, when there are sufficient and varied labelled training examples in D , the empirical risk $R(f_n)$ can provide a good approximation to $R(f^*_{\mathcal{F}})$ the optimal f in $\mathcal{F}$. When n the number of training examples in D is small the empirical risk $R(f_n)$ may be further from being a good approximation of the expected risk $R(f^*_{\mathcal{F}})$. The resultant empirical risk minimizer overfits.

To alleviate the problem of having an unreliable empirical risk minimizer when D_{train} is not sufficient, prior knowledge must be used. Prior knowledge can be used to augment the data in D_{train}, constrain the candidate functions $\mathcal{F}$, or constrain the parameters of f via initialization or regularization [Wang et al., 2020b]. Task specific deep neural network architectures such as CNNs and Recurrent Neural Networks (RNNs) are examples of constraining the candidate functions $\mathcal{F}$ through prior knowledge of what the optimal function form may be.

In this thesis I focus on transfer learning as a form of constraining the parameters of f to address the unreliable empirical risk minimizer problem. Section 6.7 discusses how deep transfer learning relates to other techniques that use prior knowledge to solve the small dataset problem.

5.1.3 Deep transfer learning

Deep transfer learning is transfer learning applied to deep neural networks. Pan and Yang [Pan and Yang, 2009] define transfer learning as:

Definition 5.1. Given a source domain $\mathcal{D}_S$ and learning task T_S, a target domain $\mathcal{D}_T$ and learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f_T(.)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and T_S, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $T_S \neq T_T$.

For the purposes of this paper we define deep transfer learning as follows:

Definition 5.2. Given a source domain $\mathcal{D}_S$ and learning task T_S, a target domain $\mathcal{D}_T$ and learning task T_T deep transfer learning aims to improve the learning of the target model M by initialising it with weights W that are trained on source task T_S using source dataset D_S (pretraining), where $\mathcal{D}_S \neq \mathcal{D}_T$, or $T_S \neq T_T$.

Some or all of W are retained when the model is "transferred" to the target task T_T and dataset D_T. The model is used for prediction on T_T after fully training any reinitialised weights and with or without continuing training on the pretrained weights (fine-tuning).

Figure 5.3 shows the pretraining and fine-tuning pipeline when applying transfer learning with a deep neural network.

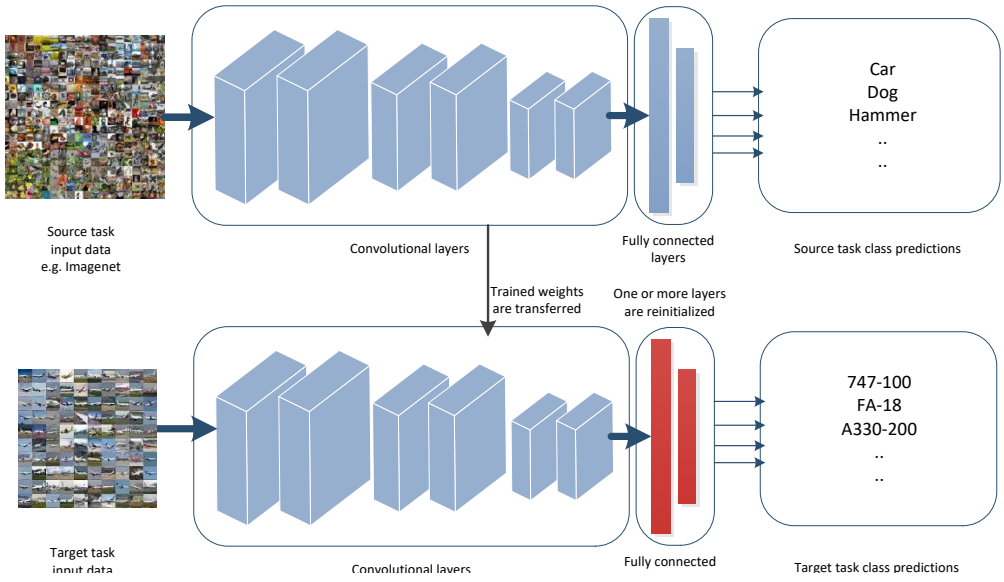


Figure 5.3: Deep transfer learning

Combining the discussion from Section 5.1.2 with definition 5.2 using deep transfer learning techniques to pretrain weights W, can be thought of as regularising W. Initialising W with weights that have been well trained on a large source dataset rather than with very small random values limits the distance that the weights move

from their pretrained values [Neyshabur et al., 2020; Liu et al., 2019a]. In the classic transfer learning setting the source dataset is many orders of magnitude larger than the target dataset. One example is pretraining on ImageNet 1K with 1.3 million training images and transferring to medical imaging tasks which often only have 100s of labelled examples. So even with the same learning rate and number of epochs, the number of updates to the weights while training on the target dataset will be orders of magnitude less than for pretraining. This prevents the model from creating large weights that are based on noise or idiosyncrasies in the small target dataset.

Advances in transfer learning can be categorized based on ways of constraining the parameters of W as follows:

1. *Initialization.* Answering questions like:
    - how much pretraining should be done?
    - is more source data or more closely related source data better?
    - which pretrained parameters should be transferred vs reinitialized?
2. *Parameter regularization.* Regularizing weights, with the assumption that if the parameters are constrained to be close to a set point, they will be less likely to overfit.
3. *Feature regularization.* Regularizing the features for each training example that are produced by the weights. This is based on the assumption that if the features stay close to those trained by the large source data set the model will be less likely to overfit.

We describe progress and problems with deep transfer learning under these categories as well as based on the relationship between source and target dataset in Section 4. Then, in Section 5, we describe how deep transfer learning relates to other methods.

5.1.4 Negative Transfer

The stated goal of transfer learning as per Definition 5.1 is to improve the learning of the target predictive function $f_T(.)$ in $\mathcal{D}_\mathrm{T}$ using the knowledge in $\mathcal{D}_\mathrm{S}$ and T_S. To achieve this goal the source dataset must be similar enough to the target dataset to ensure that the features learned in pretraining are relevant to the target task. If the source dataset is not well related to the target dataset the target model can be negatively impacted by pretraining. This is negative transfer [Rosenstein et al., 2005;

Pan and Yang, 2009]. Wang et al. [Wang et al., 2019b,a] define the negative transfer gap (NTG) as follows:

Definition 5.3. Let τ represent the test error on the target domain, θ a specific transfer learning algorithm under which the negative transfer gap is defined and $\varnothing$ is used to represent the case where the source domain data/information are not used by the target domain learner. Then, negative transfer happens when error using the source data is larger than the error without using the source data: $\epsilon_\tau(\theta(S,\tau)) > \epsilon_\tau(\theta(\varnothing,\tau))$, and the degree of negative transfer can be evaluated by the negative transfer gap

$$NTG = \epsilon_\tau(\theta(S,\tau)) - \epsilon_\tau(\theta(\varnothing,\tau)) \tag{5.6}$$

From this definition we see that negative transfer occurs when the negative transfer gap is positive. Wang et al. elaborate on factors that affect negative transfer [Wang et al., 2019b,a]:

- Divergence between the source and target domains. Transfer learning makes the assumption that there is some similarity between joint distributions in source domain $P_S(X,Y)$ and target domain $P_T(X,Y)$. The higher the divergence between these values the less information there is in the source domain that can be exploited to improve performance in the target domain. In the extreme case if there is no similarity it is not possible for transfer learning to improve performance.

- Negative transfer is relative to the size and quality of the source and target datasets. For example, if labelled target data is abundant enough, a model trained on this data only may perform well. In this example, transfer learning methods are more likely to impair the target learning performance. Conversely, if there is no labelled target data, a bad transfer learning method may perform better than the random guess, which means negative transfer would not happen.

With deep neural networks, once the weights have been pretrained to respond to particular features in a large source dataset there will be limits as to how far these weights change from their pretrained values [Neyshabur et al., 2020]. This is particularly so if the target dataset is orders of magnitude smaller as is often the case. This premise allows transfer learning to improve performance and also allows for negative transfer. If the weights transferred are pretrained to respond to unsuitable features then this training will not be fully reversed during the fine-tuning phase

and the model could be more likely to overfit to these inappropriate features. For example:

- pretrain a large deep neural network on a natural object dataset like iNaturalist [Van Horn et al., 2018],
- transfer all but the final layer of weights to a man-made object dataset like Stanford Cars [Krause et al., 2013].

The pretrained model could be learning to identify car makes and models with features like types of leaves, flowers etc. that were useful to the source task. Scenarios such as this usually lead to overfitting the idiosyncrasies of the target training set [Plested and Gedeon, 2019].

In image classification models, features learned through lower layers are more general, and those learned in higher layers are more task specific [Yosinski et al., 2014]. It is likely that if less layers are transferred negative transfer should be less prevalent, with training all layers from random initialization being the extreme end of this. However there has been limited prior work to test this.

5.2 Datasets commonly used in transfer learning for image classification

5.2.1 Source

ImageNet 1K, 5K, 9K ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. At present there are an average of over five hundred images per node [Deng et al., 2009]. ImageNet 1K or ILSVRC2012 is a well known subset of ImageNet that is used for an annual challenge. ImageNet 1K consists of 1,000 common image classes with at least 1,000 total images in each class for a total of just over 1.3 million images in the training set. ImageNet 5K and 9K are larger subsets of the full ImageNet dataset containing the most common 5,000 and 9,000 image classes respectively. All three ImageNet datasets have been used as both source and target datasets, depending on the type of experiments being performed. They are most commonly used as a source dataset because of their large sizes and general classes.

JFT dataset JFT is an internal Google dataset for large-scale image classification, which comprises over 300 million high-resolution images [Hinton et al., 2015]. Im-

ages are annotated with labels from a set of 18,291 categories. For example, 1,165 type of animals and 5,720 types of vehicles are labelled in the dataset. These categories form a rich hierarchy with the maximum depth of hierarchy being 12 and maximum number of children for a parent node being 2,876 [Sun et al., 2017]. There are 375M labels and on average each image has 1.26 labels.

Instagram hashtag datasets [Mahajan et al., 2018] collected a weakly labelled image dataset with a maximum size of 3.5 billion labelled images from Instagram, being over 3,000 times larger than the commonly used large source dataset ImageNet 1K. They collected the dataset with the following data collection pipeline:

1. Select a set of hashtags.
2. Download images that are tagged with at least one of these hashtags.
3. Because multiple hashtags may refer to the same underlying concept, apply a simple process that utilizes WordNet synsets [Miller, 1995] to merge some hashtags into a single canonical form.
4. Finally, for each downloaded image, replace each hashtag with its canonical form and discard any hashtags that were not in the selected set.

The canonical hashtags were used as labels for training and evaluation. By varying the selected hashtags and the number of images to sample, a variety of datasets of different sizes and visual distributions were created. One of the datasets created contained 1,500 hashtags that closely matched the 1,000 ImageNet 1K classes.

Places365 Places365 (Places) [Zhou et al., 2017] contains 365 categories of scenes collected by counting all the entries that corresponded to names of scenes, places and environments in the WordNet English dictionary. They included any concrete noun which could reasonably complete the phrase "I am in a place", or "Let's go to the place". Images belonging to each scene category were found using online image search engines by querying for each scene category term. There are two datasets:

- Places365-standard has 1.8 million training examples total with a minimum of 3,068 images per class.
- Places365-challenge has 8 million training examples.

Places365 is generally used as a source dataset when the target dataset is scene based such as SUN.

iNaturalist The iNaturalist [Van Horn et al., 2018] dataset consists of 859,000 images from over 5,000 different species of plants and animals. iNaturalist is generally used as a source dataset when the target dataset contains fine-grained plants or animal classes.

5.2.2 Target

General General object classification datasets contain a variety of classes with a mixture of superordinate and subordinate classes from many different categories in WordNet [Miller, 1995]. ImageNet is a canonical example of a general image classification dataset.

Examples of general object classification datasets commonly used as target datasets are:

- CIFAR-10 and CIFAR-100 [Krizhevsky et al., 2009]: they have a total of 50,000 training and 10,000 test images of 32x32 colour images from 10 and 100 classes respectively.
- PASCAL VOC 2007 [Everingham et al.]: Has 20 classes belonging to the superordinate categories of person, animal, vehicle, and indoor objects. It contains 9,963 images with 24,640 annotated objects and a 50/50 train test split. The size of each image is roughly 501×375.
- Caltech-101 [Fei-Fei et al., 2004]. Pictures of objects belonging to 101 categories. About 40 to 800 images per category, with most categories having around 50 images. The size of each image is roughly 300×200 pixels.
- Caltech-256 [Griffin et al., 2007]. An extension of Caltech-101 with 256 categories and a minimum of 80 images per category. It includes a large clutter category for testing background rejection.

Fine-grained Fine-grained object classification datasets contain subordinate classes from one particular superordinate class. examples are:

- Food-101 (Food) [Bossard et al., 2014]: Contains 101 different classes of food objects with 75,750 training examples and 25,250 test examples.
- Birdsnap (Birds) [Berg et al., 2014]: Contains 500 different species of birds, with 47,386 training examples and 2,443 test examples.

- Stanford Cars (Cars) [Krause et al., 2013]: Contains 196 different makes and models of cars with 8,144 training examples and 8,041 test examples.
- FGVC Aircraft (Aircraft) [Maji et al., 2013]: Contains 100 different makes and models of aircraft with 6,667 training examples and 3,333 test examples.
- Oxford-IIIT Pets (Pets)[Parkhi et al., 2012]: Contains 37 different breeds of cats and dogs with 3,680 training examples and 3,369 test examples.
- Oxford 102 Flowers (Flowers) [Nilsback and Zisserman, 2008]: Contains 102 different types of flowers with 2,040 training examples and 6,149 test examples.
- Caltech-uscd Birds 200 (CUB) [Wah et al., 2011]: Contains 200 different species of birds with around 60 training examples per class.
- Stanford Dogs (Dogs) [Khosla et al., 2011]: Contains 20,580 images of 120 breeds of dogs.

Scenes Scene datasets contain examples of different indoor and/or outdoor scene settings. Examples are:

- SUN397 (SUN) [Xiao et al., 2010]: Contains 397 categories of scenes. This dataset preceded Places-365 and used the same techniques for data collection. The scenes with at least 100 training examples were included in the final dataset.
- MIT 67 Indoor Scenes [Quattoni and Torralba, 2009]: Contains 67 Indoor categories, and a total of 15620 images. The number of images varies across categories, but there are at least 100 images per category.

Others There are a number of other datasets that have less of an overarching theme and are less related to the common source datasets. These are often used in conjunction with deep transfer learning for image classification to show models and techniques are widely applicable. Examples of these are:

- Describable Textures (DTD) [Cimpoi et al., 2014]: Consists of 3,760 training examples of texture images jointly annotated with 47 attributes. The texture images are collected "in the wild" by searching for texture adjectives on the web.

- Daimler pedestrian classification [Munder and Gavrila, 2006]: Contains 23,520 training images with two classes, being contains pedestrians and does not contain pedestrians.
- German road signs (GTSRB) [Stallkamp et al., 2012]: Contains 39,209 training images of German road signs in 43 classes.
- Omniglot [Lake et al., 2015]: Contains over 1.2 million training examples of 1,623 different handwritten characters from 50 writing systems.
- SVHN digits in the wild (SVHN) [Netzer et al., 2011]: Contains 73,257 training examples of labelled digits cropped from Street View images.
- UCF101 Dynamic Images (UCF101) [Soomro et al., 2012]: Contains 9,537 static frames of 101 classes of actions cropped from action videos.
- Visual Decathlon Challenge (Decathlon) [Rebuffi et al., 2017]: A challenge designed to simultaneously solve 10 image classification problems representative of very different visual domains. The data for each domain is obtained from the following image classification benchmarks already described above with all images resized to have a shorter side of 72 pixels: ImageNet, CIFAR-100, Aircraft, Daimler pedestrian, Describable textures, German traffic signs, Omniglot, SVHN, UCF101, VGG-Flowers.

Chapter 6

Literature Review

In the past decade, the successes of deep CNNs on image classification tasks have inspired many researchers to apply them to an increasingly wide range of domains. Model performance is strongly affected by the relationship between the amount of training data and the number of trainable parameters in a model. As a result there has been ever growing interest in using transfer learning to allow large CNN models to be trained in domains where there is only limited training data available or other constraints exist.

As deep learning gained popularity in 2012 to 2016, transferability of features and best practices for performing deep transfer learning was thoroughly explored [Agrawal et al., 2014; Azizpour et al., 2015; Huh et al., 2016; Sharif Razavian et al., 2014; Yosinski et al., 2014]. While there are some recent works that have introduced improvements to transfer learning techniques and insights, there are many more that have focused on best practice for either general [Kornblith et al., 2019; Mahajan et al., 2018; Li et al., 2020a; Plested et al., 2021] or specific [Raghu et al., 2019; Heker and Greenspan, 2020] application domains rather than techniques. We fully review both.

When reviewing the application of deep transfer learning for images we divide applications into categories. We split tasks in two ways, being small versus large target datasets and closely versus loosely related source and target datasets. For example using ImageNet [Deng et al., 2009] as a source dataset to pretrain a model for classifying tumours on medical images is a loosely related transfer and is likely to be a small target dataset due to privacy and scarcity of disease. This category division aligns with the factors that affect negative transfer outlined in [Wang et al., 2019b,a].

The distinction between target dataset sizes is useful as it has been shown that small target datasets are much more sensitive to changes in transfer learning settings [Plested and Gedeon, 2019]. It has also been shown that standard transfer learning protocols do not perform as well when transferring to a less related target task [He

et al., 2018; Kornblith et al., 2019; Plested et al., 2021], with negative transfer being an extreme example of this [Wang et al., 2019b,a], and that the similarity between datasets should be considered when deciding on hyperparameters [Li et al., 2020a; Plested et al., 2021]. These distinctions go some way to explaining conflicting performance of deep transfer learning methods in recent years [He et al., 2018; Li et al., 2020a; Wan et al., 2019; Zoph et al., 2020].

6.1 Review papers

Many reviews related to deep transfer learning have been published in the past decade and the pace has only increased in the last few years. However, they differ from ours in two main ways. The first group consists of more general reviews, that provide a high level overview of transfer learning and attempt to include all machine learning sub-fields and all task sub-fields. Reviews in this group are covered in Section 6.1.1. The second group is more specific with reviews providing a comprehensive breakdown of the progress on a particular narrow domain specific task. They are discussed in the relevant parts of Section 6.4.5. Our review is designed to sit in between these two groups and provide a broad review of deep transfer learning as it relates to image classification so that overarching patterns can be identified, analysed and discussed. This allows us to make recommendation for best practice when using deep transfer learning applied to image classification as well as identify knowledge gaps and suggest directions for future research. There are a few surveys that are more closely related to ours with differences discussed in Section 6.1.2.

6.1.1 General transfer learning surveys

The most recent general transfer learning survey [Zhuang et al., 2020a] offers a thorough theoretical analysis of general transfer learning techniques. Transfer learning techniques are split into data-based and model-based, then further divided into subcategories. Deep learning models are explicitly discussed as a sub-Section of model-based categorisation. The focus is entirely on generative models such as autoencoders and Generative Adversarial Networks (GANs) and several papers are reviewed. Neural networks are also mentioned briefly under the Parameter Control Strategy and Feature Transformation Strategy Sections. However, the focus is on unsupervised pretraining strategies, rather than best practice for transferring from source to target domains under any pretraining strategy.

Weiss et al. [Weiss et al., 2016] divide general transfer learning into homogeneous,

where the source and target dataset distributions are the same, and heterogeneous, where they are not, and give a thorough description of each. They review many different approaches in each category, but few of them are related to deep neural networks. Convolutional neural networks are only mentioned to state that it has proved to be possible to transfer CNNs.

6.1.2 Closely related review papers

There are some recent review papers that based on their title seem to be more closely related. However, they are short summary papers containing limited details on the subject matter rather than full review papers.

A Survey on Deep Transfer Learning [Tan et al., 2018] defines deep transfer learning and separates it into four categories based on the subset of techniques used, being instances-based deep transfer learning, mapping-based deep transfer learning, network-based deep transfer learning, and adversarial-based deep transfer learning. A few examples are given of each category, but these are extremely brief. The focus is more on showing a broad selection of methods rather than providing much detail or focusing particularly on deep transfer learning methods. Most major works in the area from the past decade are missing.

Deep Learning and Transfer Learning Approaches for Image Classification [Krishna and Kalluri, 2019] focuses on defining CNNs along with some of the major architectures and results from the past decade. The paper includes a few brief paragraphs defining transfer learning and some of the image classification results incorporate transfer learning, but no review of the topic is performed.

A survey of transfer learning for convolutional neural networks [Ribani and Marengoni, 2019] is a short seven page paper which briefly introduces the transfer learning task and settings, and introduces general categories of approaches and applications. It does not review any specific approaches or applications.

Transfer learning for visual categorization: A survey [Shao et al., 2014]. Is a full review paper, but is older with no deep learning techniques included.

In Small Sample Learning in Big Data Era [Shu et al., 2018] deep transfer learning is a large part of the work, but not the focus. Methods for machine learning with small samples of labelled training data are divided into experience learning and concept learning. Experience learning includes models that perform data augmentation and models that learn from related data, such as transfer learning. Concept learning includes methods that create a concept embedding space, including prototype models and matching models. Some examples of deep learning applied to image

classification domains are mentioned, but there is no discussion of methods for improving deep transfer learning as it relates to image classification.

6.2 General deep transfer learning for image classification

Early work on deep transfer learning showed that:

1. Deep transfer learning results in comparable or better performance in many different tasks, when compared to training from random intialization, and particularly when compared to shallow machine learning methods [Sharif Razavian et al., 2014; Azizpour et al., 2015].

2. More pretraining both in terms of the number of training examples and the number of iterations tends to result in better performance [Agrawal et al., 2014; Azizpour et al., 2015; Huh et al., 2016].

3. Fine-tuning the weights on the target task tends to result in better performance particularly when the target dataset is larger and less similar to the source dataset [Agrawal et al., 2014; Yosinski et al., 2014; Azizpour et al., 2015].

4. Transferring more layers tends to result in better performance when the source and target dataset and task are closely matched, but less layers are better when they are less related [Agrawal et al., 2014; Yosinski et al., 2014; Azizpour et al., 2015; Chu et al., 2016].

5. Deeper networks result in better performance [Azizpour et al., 2015].

It should be noted that all the studies referenced above were completed prior to advances in residual networks [He et al., 2016]. It has been argued that residual networks when combined with fine-tuning makes features more transferable [He et al., 2016]. As many of the above studies were carried out within a similar time period, some results have not been combined. For instance, most were done with AlexNet, a relatively shallow network, as a base and many did not perform fine tuning and/or simply used a deep neural network as a feature detector at whatever layer it was transferred. It has since been shown that when fine-tuning is used effectively, transferring less than the maximum number of layers tends to result in better performance. This applies even when the source and target datasets are highly related, particularly with smaller target datasets [Plested and Gedeon, 2019].

More recently it has been shown that the performance of models on ImageNet 1K correlates well with performance when the pretrained model is transferred to other

tasks [Kornblith et al., 2019]. They additionally demonstrate that the increase in performance of deep transfer learning over random initialisation is highly dependent on both the target dataset size and the relationship between the classes in the source and target datasets. This will be discussed in more detail in the following sections.

6.3 Recent advances in transfer learning for image classification

Recent advances in the body of knowledge related to deep transfer learning for image classification can be divided into advances in techniques, and general insights on best practice. We describe advances in transfer learning techniques here and insights on best practice in Section 6.3.4. Recent advances in techniques are divided into regularization, hyperparameter based, matching the source domain to the target domain, and a few others that do not fit the previous categories. We discuss matching the source domain to the target domain under the relevant source versus task domains in Sections 6.4.6 and 6.4.6.1 and the rest below. In our reviews of recent work we attempt to present a balanced view of the evidence for the improvements offered by newer models compared to prior ones and the limitations of those improvements. However, in some of the more recent cases this is difficult as the original papers provide limited evidence and newer work showing the limitations of the methods has not yet been done.

6.3.1 Regularization based technique advances

Most regularization based techniques aim to solve the problem of the unreliable empirical risk minimizer 5.1.2 by restricting the model weights or the features produced by them so they will not fit small idiosyncrasies in the data. They achieve this by adding a regularization term $\lambda \cdot \Omega(.)$ to the loss function:

$$min_w L(w) = \left\{ \frac{1}{n} \sum_{i=1}^{n} L(z(x_i, w), y_i) + \lambda \cdot \Omega(.) \right\} \tag{6.1}$$

with the first term $\frac{1}{n} \sum_{i=1}^{n} L(z(x_i, w), y_i)$ being the empirical loss and the second term being the regularization term. The tuning parameter $\lambda > 0$ balances the trade-off between the two.

Weight regularization directly restricts how much the model weights can move.

Knowledge distillation or feature based regularization uses the distance between the feature maps output from one or more layers of the source and target networks to regularize the model:

$$\Omega(w, w_s) = \frac{1}{n} \sum_{j=1}^{N} \sum_{i=1}^{n} d\left(F_j\left(w_t, x_i\right), F_j\left(w_s, x_i\right)\right) \tag{6.2}$$

where $F_j\left(w_t, x_i\right)$ is the feature map output by the jth filter in the target network defined by weights w_t for input value x_i, and $d\left(.\right)$ is a measure of dissimilarity between two feature maps.

The success of regularization based techniques for deep transfer learning rely heavily on the assumption that the source and target datasets are closely related. This is required to ensure that the optimal weights or features for the target dataset are not far from those trained on the source dataset.

There have been many new regularization based techniques introduced in the last three years. We review major new techniques in chronological order.

1. L2-SP [Li et al., 2018, 2020b] is a form of weight regularization. The aim of transfer learning is to create models that are regularized by keeping features that are reasonably close to those trained on a source dataset for which overfitting is not as much of a problem. The authors argue that because of this, during the target dataset training phase the fine tuned weights should be decayed towards the pretrained weights, not zero. Several regularizers that decay weights towards their starting point, denoted SP regularizers, were tested in the original papers. The L2-SP regularizer $\Omega(w) = \frac{\alpha}{2} \left\|w - w^0\right\|_2^2$ which is the L2 loss between the source weights and the current weights is shown to significantly outperform the standard L2 loss on the four target dataset shown in the paper with a Resent-101 model. The original paper showed results for transferring to four small target datasets that were very similar to the two source datasets used for pretraining. It has since been shown that the L2-SP regularizer can result in minimal improvement or even worse performance when the source and target datasets are less related [Li et al., 2020a; Wan et al., 2019; Plested et al., 2021; Chen et al., 2019]. More recent work has showed that in some cases using L2-SP regularization for lower layers and L2 regularization for higher layers can improve performance [Plested et al., 2021].

2. DELTA [Li et al., 2019b] is an example of knowledge distillation or feature map based regularization. It is based on the idea of re-using CNN channels that

are not useful to the target task while not changing channels that are useful. Training on the target task is regularized by the attention weighted L2 loss between the final layer feature maps of the source and target models:

$$\Omega(w, w^0, x_i, y_i) = \sum_{j=1}^{N} (W_j(w^0, x_i, y_i) \cdot \left\| FM_j(w, x_i) - FM_j(w^0, x_i)) \right\|_2^2 \qquad (6.3)$$

where $FM_j(w, x_i)$ is the output from the *jth* filter applied to the *ith* input. The attention weights W_j for each filter are calculated by removing the model's filters one by one (setting its output weights to 0), calculating the increase in loss. Filters resulting in a high increase in loss are then set with a higher weight for regularization, encouraging them to stay similar to those trained on the source task. Others that are not as useful in the target task are less regularized and can change more. This regularization resulted in performance that was slightly better than the L2-SP regularization in most cases with ResNet-101 and Inceptionv3 models, ImageNet 1K as the source dataset and a variety of target datasets. The original paper showed state of the art performance for DELTA on Caltech 256-30, however they used mostly the same datasets as the original L2-SP paper [Li et al., 2018] and for the two additional datasets used they showed that L2-SP outperformed the baseline L2 regularization. Because of this it seems likely that this method would suffer from the same limitations as L2-SP and only improve performance on closely related datasets. This has been partially confirmed in one recent study [Kou et al., 2020].

3. Wan et al. [Wan et al., 2019] propose decomposing the transfer learning gradient update into the empirical loss and regularization loss gradient vectors. Then when the angle between the two vectors is greater than 90 degrees they further decompose the regularization loss gradient vector into the portion perpendicular to the empirical loss gradient and the remaining vector in the opposite direction of the empirical loss gradient. They remove the latter term, in the hopes that not allowing the regularization term to move the weights in the opposite direction of the empirical loss term will stop negative transfer. They show that their proposal improves performance slightly with a ResNet-18 on four different datasets. However, their results are poor compared to state of the art as they do not test on modern deep models. For this reason, it is difficult to judge how well their regularization method performs in general.

4. Batch spectral shrinkage (BSS) [Chen et al., 2019] introduces a loss penalty ap-

plied to smaller singular values of channelwise features in each batch update during fine-tuning so that untransferable spectral components are suppressed. They test this method using a ResNet-50 pretrained on ImageNet 1K and fine-tuned on a range of different target datasets. The results show that their method never hurts performance on the given datasets and often produces significant performance gains over L2, L2-SP and DELTA regularization for smaller target datasets. They also show that BSS can improve performance for less similar target datasets where L2-SP hinders performance. All results are poor compared to state of the art for identical model sizes, source and target datasets so it is difficult to be certain of how their method performs in general.

5. Sample-based regularization [Jeon et al., 2020] proposes regularization using the distance between feature maps of pairs of inputs in the same class, as well as weight regularization. The model was tested using a ResNet-50 and transferring from ImageNet 1K and Places365 to a number of different, fine grained classification tasks. The authors report an improvement over L2-SP, DELTA and BSS in all tests. However, the final results reported are significantly below state of the art for the same model and datasets, so it is difficult to confirm the results. Their results reconfirm that BSS performs better than DELTA and L2SP in most cases and in some cases DELTA and L2SP decrease performance compared to the standard L2 regularization baseline.

6.3.2 Normalization based technique advances

Further to regularization based methods, there are several recent techniques that attempt to better align fine-tuning in the target domain with the source domain. This is achieved by making adjustments to the standard batch normalization or other forms of normalization that are used between layers in modern CNNs.

1. Sharing batch normalization hyperparameters across source and target domains has been shown to be more effective than having separate ones across many domain adaptation tasks [Wang et al., 2019a; Maria Carlucci et al., 2017]. Wang et al. [Wang et al., 2019a] introduce an additional batch normalization hyperparameter called domain adaptive α. This takes standard batch normalization with γ and β shared across source and target domain and scales them based on the transferability value of each channel calculated using the mean and variance statistics prior to normalization. As far as we are aware these techniques have not been applied to the general supervised transfer learning case.

2. Stochastic normalization [Kou et al., 2020] samples batch normalisation based on mini-batch statistics or based on moving statistics for each filter with probability hyperparameter p. At the start of fine-tuning on the target dataset the moving statistics are initialised with those calculated during pretraining in order to act as a regularizer. This is designed to overcome problems with small batch sizes resulting in noisy batch-statistics or the collapse in training associated with using moving statistics to normalize all feature maps [Ioffe, 2017; Ioffe and Szegedy, 2015]. The results show that their methods improve over BSS, DELTA and L2-SP for low sampling versions of three standard target datasets and improve over all but BSS for larger versions of the same datasets. Their results again show that BSS performs better than DELTA and L2SP in most cases and in many cases DELTA and L2-SP decrease performance compared to the standard L2 regularization baseline.

6.3.3 Other recent new techniques

Guo et al. [Guo et al., 2019] make two copies of their ResNet models pretrained on ImageNet 1K. One model is used as a fixed feature selector with the pretrained layers frozen and the other model is fine-tuned. They reinitialize the final classification layer in both. A policy net trained with reinforcement learning is then used to create a mask to combine layers from each model together in a unique way for each target example. They report state of the art results on the decathlon 10 task dataset, but their results are less competitive on other more standard datasets like Stanford Cars and Flowers.

Co-tuning for transfer learning [You et al., 2020] uses a probabilistic mapping of hard labels in the source dataset to soft labels in the target dataset. This mapping allows them to keep the final classification layer in a ResNet-50 and train it using both the target data and soft labels from the source dataset. As with many other recent results, they show that their algorithm improves on all others, including BSS, DELTA and L2SP, but their results are significantly below state of the art for identical model sizes, source and target dataset. They do show the same ordering for the target datasets, using BSS improves on DELTA which improves on L2SP.

6.3.4 Insights on best practice

Further to advances in techniques and models, there has been a large body of recent research that extends the early work on best practice for deep transfer learning for image classification described in Section 6.2. These studies give insights on the fol-

lowing decisions that need to be made when performing deep transfer learning for image classification:

- *Selecting the best model for the task*. Models that perform better on ImageNet were found to perform better on a range of target datasets in [Kornblith et al., 2019].
- *Choosing the best data for pretraining*. In many cases pretraining with smaller more closely related source datasets was found to produce better results on target datasets than with larger less closely related source datasets [Ngiam et al., 2018; Mahajan et al., 2018; Cui et al., 2016, 2018; Puigcerver et al., 2020]. There are various measures of similarity used to define closely related that are outlined in Section 6.4.6.
- *Finding the best hyperparameters for fine-tuning*. Several studies include extensive hyperparameter searches over learning rate, learning rate decay, weight decay, and momentum [Kornblith et al., 2019; Li et al., 2018; Mahajan et al., 2018; Li et al., 2020a; Plested et al., 2021]. These studies show the relationship between the size of the target dataset and its similarity to the source dataset with fine-tuning hyperparameter settings. Optimal learning rate and momentum, are both shown to be lower for more related source and target datasets [Li et al., 2020a; Plested et al., 2021]. Also the number of layers to reinitialise from random weights is strongly related to the optimal learning rate. This is because using pretrained weights for the maximum number of layers acts as a regularizer by limiting the distance between pretrained weights and their fine-tuned values [Neyshabur et al., 2020; Plested et al., 2021].
- *Whether a multi-step transfer process is better than a single step process*. A multi-step pretraining process, where the interim dataset is smaller and more closely related to the target dataset, can outperform a single step pretraining process when originating from a very different, large source dataset [Ng et al., 2015; Puigcerver et al., 2020; Gonthier et al., 2020]. Related to this, using a self-supervised learning technique for pretraining on a more closely related source dataset can outperform using a supervised learning technique on a less closely related dataset [Zoph et al., 2020].
- *Which type of regularization to use*. L2-SP or other more recent transfer learning specific regularization techniques like DELTA, BSS, stochastic normalization, etc, improve performance when the source and target dataset are closely related, but often hinder it when they are less related [Li et al., 2018, 2019b; Wan

et al., 2019; Plested et al., 2021]. These regularization techniques are discussed in more detail in Section 6.3.1.

Big Transfer (BiT) was defined as pretraining on source datasets larger than ImageNet 1K in [Kolesnikov et al., 2019]. They pretrain various sizes of ResNet on ImageNet 1k and 21K and JFT. They call pretraining on larger source datasets Big Transfer. They make a number of claims about deep transfer learning when pretraining on very large datasets including:

1. Batch normalization (BN) [Ioffe and Szegedy, 2015] can be detrimental to Big Transfer, and Group Normalization [Wu and He, 2018] combined with Weight Standardization performs well with large batches.

2. MixUp [Zhang et al., 2017a] is not useful for pretraining on large source datasets and is only useful during fine-tuning for mid-sized target datasets (20-500K)

3. Regularization (L2, L2-SP, dropout) does not enhance performance in the fine-tuning phase, even with very large models (the largest model used for experiments has 928 million parameters). Adjusting the training and learning rate decay time based on the size of the target dataset, longer for larger datasets, provides sufficient regularization.

The authors use general fine-tuning hyperparameters for learning rate scheduling, training time and amount/usage of MixUp that are only adjusted based on the target dataset size. They are not adjusted for individual target datasets. They achieve performance that is comparable to models with selectively tuned hyperparameters for their model pretrained on ImageNet and state of the art, or close to in many cases, for their model pretrained on the 300 times larger source dataset JFT. However their target datasets, ImageNet, CIFAR 10 & 100, and Pets are very closely related to their source datasets making them easier to transfer to. Their final target dataset Flowers is also known to be better suited to transfer to, from their source datasets. See Section 6.4.4 for further discussion of which target datasets are easier to transfer to.

We expect that best practice recommendations developed for closely related datasets will not be applicable to less closely related target datasets as has been shown for many other methods and recommendations [Li et al., 2018, 2019b; Wan et al., 2019; Plested et al., 2021; Li et al., 2020a; Chen et al., 2019; Kou et al., 2020]. To test this hypothesis we reran a selection of the experiments in BiT using Stanford Cars as the target dataset which is very different from the source dataset ImageNet 21K and known to be more difficult to transfer to [Kornblith et al., 2019; Plested et al., 2021].

Using BiT-M-R152x4 we first confirmed that we could reproduce their state of the art results for the datasets listed in the paper, then produced the results in Table 6.1 using Stanford Cars. These results show that BiT produces results well below standard fine-tuning results for this less related dataset. The first column shows the results with all the recommended hyperparameters from the paper. While the performance can be improved with increases in learning rates and number of epochs before the learning rate is decayed, final results are still well below results using standard fine-tuning for the same or smaller model, source and target dataset. The fine grained classification task in Stanford Cars is known to be less similar to the more general ImageNet and JFT datasets. Because of this it is not surprising that recommendations developed for more closely related target datasets do not apply.

Table 6.1: Big transfer (BiT): General visual representation learning [Kolesnikov et al., 2019] extended results for BiT-M-R152x4 pretrained on ImageNet 21K and transferred to a less closely related target dataset - Stanford Cars. Best comparable means results using best known fine-tuning for the same or smaller model, pretrained on either ImageNet 1K or 21K with Cars as the target dataset. Default is the learning rate decay schedule specified by the paper for this size target dataset and x2 is two times the number of batches before decaying the learning rate compared to the default. The left most column is the fine-tuning settings recommended in [Kolesnikov et al., 2019] which result in significantly worse performance than the best performance for a comparable model and pretraining dataset as shown in the right most column.

Dataset	default lr 0.003		lr 0.01		lr 0.03		lr 0.1		Best comparable
	default	x2	default	x2	default	x2	default	x2	[Plested et al., 2021]
Cars	86.20	86.15	85.81	87.49	81.41	88.96	27.51	5.22	94.86

6.3.5 Insights on transferability

Here we review works that give more general insight as to what is happening with model weights, representations and the loss landscape when transfer learning is performed, as well as measures of transferability of pretrained weights to target tasks.

Several methods for analysing the feature space were used in [Neyshabur et al., 2020]. They found that models trained from pretrained weights make similar mistakes on the target domain, have similar features and are surprisingly close in ℓ_2 distance in the parameter space. They are in the same basins of the loss landscape. Models trained from random initialization do not live in the same basin, make different mistakes, have different features and are farther away in ℓ_2 distance in the

parameter space.

A flatter and easier to navigate loss landscape for pretrained models compared to their randomly initialized counterparts was also shown in [Liu et al., 2019a]. They showed improved Lipschitzness and that this accelerates and stabilizes training substantially. Particularly that the singular vectors of the weight gradient with large singular values are shrunk in the weight matrices. Thus, the magnitude of gradient back-propagated through a pretrained layer is controlled and pretrained weight matrices stabilize the magnitude of gradient especially in lower layers, leading to more stable training.

Several recent techniques have been proposed for measuring the transferability of pretrained weights:

1. H-score [Bao et al., 2019] is a measure of how well a pretrained model f is likely to perform on a new task with input space X and output space Y based on the inter-class covariance $cov(\mathbb{E}_{P_{X|Y}}[f(X)|Y])$ and the feature redundancy $tr(cov(f(X)))$

$$\mathcal{H}(f) = tr(cov(f(X))^{-1}cov(\mathbb{E}_{P_{X|Y}}[f(X)|Y]) \tag{6.4}$$

   the H-score increases as inter-class covariance increases and feature redundancy decreases. The authors show that H-score has a strong correlation with target task performance. They also show that it can be used to rank transferability and create minimum spanning trees of task transferability. The latter may be useful in guiding multi-step transfer learning for less related tasks as discussed in Section 6.3.4.

2. Transferability and negative conditional entropy (NCE) for transfer learning tasks where the source and target datasets are the same, but the tasks differ, are defined in [Tran et al., 2019]. The authors define transferability as the log-likelihood $l_Y(w_Z, k_Y)$, where w_z is the weights of the model backbone pretrained on the source task Z and k_Y is the weights of the classifier trained on the target task. They then define conditional cross-entropy (NCE) as another measure of transferability, defined as being the empirical cross entropy of label $\bar{y}$ from the target domain given a label $\bar{z}$ from the source domain. To empirically demonstrate the effectiveness of the NCE measure a ResNet-18 model as the backbone was paired with an SVM classifier. NCE was demonstrated to have strong correlation with accuracy on the target tasks for combinations of 437 source and target tasks.

3. LEEP [Nguyen et al., 2020] is introduced as a measure of transferability. Using the pretrained model, the joint distribution over labels in the source dataset and the target dataset labels is estimated to construct an empirical predictor. LEEP is the log expectation of the empirical predictor. LEEP is defined mathematically as:

$$T(\theta, D) = \frac{1}{n} \sum_{i=1}^{n} log \left(\sum_{z \epsilon Z} \hat{P}\left(y_i|z\right) \theta\left(x_i\right)_z \right) \tag{6.5}$$

where $\theta\left(x_i\right)_z$ is is the probability of the source label z for target input data x_i predicted using the pretrained weights θ, and $\hat{P}\left(y_i|z\right)$ is the empirical conditional probability of target label y_i given source label z. LEEP is shown to have good theoretical properties and empirically it is demonstrated to have strong correlation with performance gain from pretraining weights on the source tasks. This is shown on source tasks ImageNet 1K and CIFAR10 and 200 random target tasks taken from the closely related CIFAR100 and less closely related FashionMNIST. The authors expand NCE to the case where the source and target datasets are different by creating dummy labels for the target data based on the source task using the pretrained model θ. They show that LEEP has a stronger correlation with performance gain than the expanded NCE measure and H-score.

6.4 Taxonomy of source and target dataset relationships for transfer learning application to image classification

In this section we present a new taxonomy for applications of transfer learning for image classification. This taxonomy makes it easier to see overarching patterns of where transfer learning has been effective and, where it has failed to fulfill its potential. The taxonomy is laid out in Figure 6.1. Subsections 6.4.1, 6.4.2, 6.4.4, 6.4.5 each deal with a quadrant from the taxonomy shown in Figure 6.1 in clockwise order from the top right ending at the top left. The differences between best practice for each of these quadrants and recent developments in each are fully covered. Final recommendations for each are summarized in Section 6.8.2.

6.4.1 Large closely related target datasets

An early, systematic and extremely thorough treatment of deep transfer learning for large closely related image datasets was performed by Yosinski et al. [Yosinski et al.,

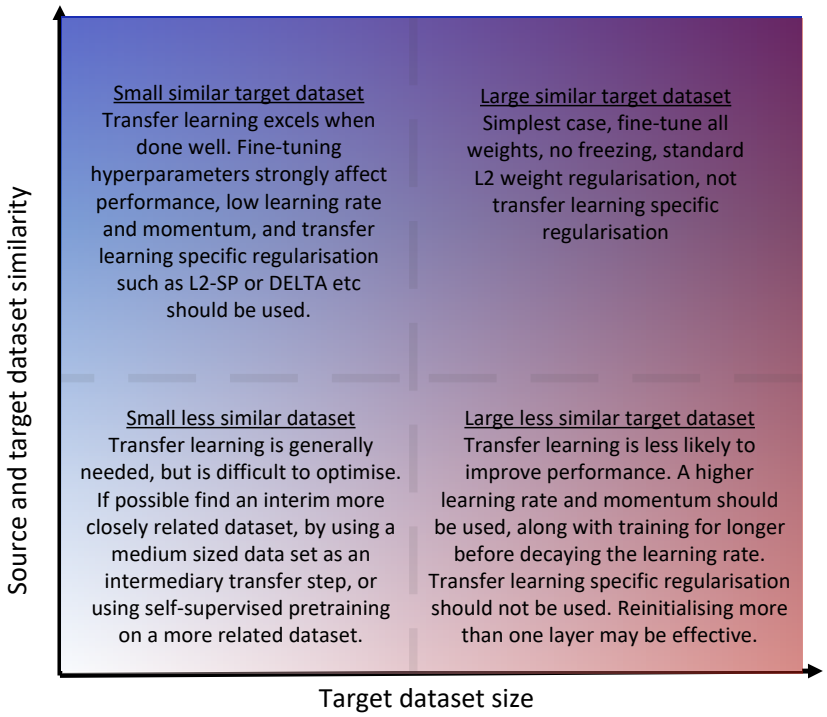


Figure 6.1: Transfer learning taxonomy. Blue is a measure of source and target dataset similarity, where the most similar would be transferring from one set of ImageNet classes to another such as in Yosinski et al. [2014]; Plested and Gedeon [2019]. Red measures the target dataset size, with small being around 10,000 labelled examples and large being a million plus labelled examples. The lines between the quadrants are gray and dotted and the colours gradually merge to indicate that there is not a clear cutoff point from one quadrant to the next, rather as the target dataset increases in size between the range of around 50,000 to 200,000 more strategies for large target datasets should be tried .

2014]. They did two experiments using AlexNet [Krizhevsky et al., 2012]. One split ImageNet 1K [Deng et al., 2009] into two randomly chosen sets of 500 object classes. The other split ImageNet into two sets of classes that were as different as possible to make up the source and target dataset. These two sets of classes were natural object classes and man-made object classes. In both experiments they used the full 1,000 plus training examples per class, so both are examples of a very large target dataset. After creating the source and target datasets they pretrained an AlexNet [Krizhevsky et al., 2012] on the source dataset, then transferred varying numbers of layers of weights to the target dataset reinitializing the remaining layers with small random weights. They then trained either the entire model or just the reinitialised

layers applying the standard training hyperparameters of the day to fine tuning, these being a learning rate of 0.1 decayed by multiplying by 0.1 every 30 epochs.

Yosinski et al. [Yosinski et al., 2014] concluded that weights from the lower layers of a CNN trained on an image task appear to be general and easily transferable between tasks, while the upper layers are more task specific. They showed that fine-tuning is important in transfer learning to allow fragile co-adapted features from the middle layers to retrain together for the new task. Finally, they demonstrated that for the more related source and target datasets (the randomly chosen classes), the more layers transferred the better the final performance, whereas for the less related datasets the upper layers were less transferable. It should be noted that experiments with fine-tuning were not done for the less related source and target datasets. The only experiments done froze the weights that were transferred to show the raw transferability of the weights.

The finding that more layers transferred results in better performance [Yosinski et al., 2014] has heavily influenced transfer learning protocols. It is important to note that their results were only shown to hold for large and closely related target datasets and their specific fine-tuning hyperparameters. In fact they demonstrated that higher layers are less transferable when the source and target datasets are less related. Despite this latter finding, their results and training protocols have been used as a template to inform transfer learning procedures on a wide variety of datasets and tasks [He et al., 2018; Scott et al., 2018; Wu et al., 2018; Tan and Le, 2019; Mormont et al., 2018; Kornblith et al., 2019].

The original experiments performed by Yosinski et al. on closely related datasets (the randomly chosen classes from ImageNet 1K) have since been repeated while varying the size of the target dataset and fine-tuning hyperparameters [Plested and Gedeon, 2019]. These experiments established that transferring more layers did not generally result in better performance when optimal fine-tuning hyperparameters were used. They also demonstrated that transferring the correct number of layers and using optimal fine-tuning hyperparameters had a much larger impact on performance as the size of the target dataset was reduced. Recent work [Plested et al., 2021] has expanded these experiments to more current and deeper models that are known to perform better for transfer learning [He et al., 2016] and less closely related source and target datasets. This work has confirmed that for some datasets transferring more layers is better. However, for others, reinitialising with random weights, multiple layers, or even whole blocks of layers of an Inception v4 model, improves performance over the baseline of transferring all but the final classification layer.

6.4.2 Large target datasets with less similar tasks

There are very few large image datasets that are not closely related to the object classification tasks that are usually used for pretraining. The Places365 with 1.8 million training images was used as a source task in [Mahajan et al., 2018]. It was shown that with less related source and target datasets the bigger and more diverse the source training dataset the better the results on the target dataset.

Until recently the largest object detection dataset was COCO [Lin et al., 2014] with 328,000 training images across 80 classes. There have been mixed results shown in whether pretraining with ImageNet 1K and other common source datasets improve object detection performance on the COCO dataset. These results are discussed in detail in Section 6.4.5.5. In the last few years several larger object detection datasets have been introduced Kuznetsova et al. [2020]; Shao et al. [2019]; Cai et al. [2022]. Shao et al. [2019] introduced the object detection dataset Objects365 which contains 365 object categories, 638 thousand images, and 10 million bounding boxes. The authors found that for the target task COCO, pretraining on Objects365 was more effective than pretraining on ImageNet 1K and resulted in a significantly shorter fine tuning time. Additionally the multi-step pretraining process of pretraining on ImageNet 1K, then Objects365 before transferring to the COCO target dataset slightly improved results over a single pretraining step.

6.4.3 Smaller target datasets

It is often not possible to train deep neural networks on small datasets from random initialization [Mazurowski et al., 2019; Mormont et al., 2018; Kraus et al., 2017; Heker and Greenspan, 2020]. This means that transfer learning becomes more heavily relied on as the size of the target dataset reduces. It has also been shown that the transfer learning protocols including the number of layers to transfer and fine-tuning hyper-parameters have a much great impact on performance as the size of the target dataset reduces [Plested and Gedeon, 2019]. As the size of the target dataset decreases, two competing factors affect transfer learning performance:

1. The empirical risk estimate becomes less reliable 5.1.2 making overfitting idiosyncrasies in the target dataset more likely.

2. The pretrained weights implicitly regularize the fine-tuned model and the final weights do not move far from their pretrained values [Neyshabur et al., 2020; Liu et al., 2019a; Raghu et al., 2019]

The outcome of the first point is an increasing need to use transfer learning and other methods to reduce overfitting. The implicit regularization noted in the second point can have a positive impact in reducing overfitting the empirical risk estimate as per the first point. It can also have a negative impact (negative transfer) if the weights transferred from the source dataset are far from optimal for the target dataset. When the weights, and thus the features produced, are restricted to being far from optimal, then the negative effect on performance can be compounded by the first point [Plested and Gedeon, 2019].

6.4.4 Smaller target datasets with similar tasks

The most well known study on transfer learning [Yosinski et al., 2014] was performed with an AlexNet [Krizhevsky et al., 2012] on strongly related and very large source and target datasets. The same experiment was repeated in [Plested and Gedeon, 2019], but included a range of smaller target dataset sizes. These experiments demonstrated a significant improvement in performance using optimal protocols for transfer learning compared to common protocols from [Yosinski et al., 2014]. This improvement increased considerably as the size of the dataset decreased. By using *optimal* compared to *commonly used* protocols, the average accuracy was found to increase from 20.86 to 30.12% for the smallest target dataset of just 10 examples for each of the 500 classes. The commonly used practice of transferring all layers except the final classification layer was shown to *not* be optimal in any of the experiments.

When pretraining CNN models on ImageNet 1K [Deng et al., 2009] and transferring to significantly smaller target datasets, the improvement of deep transfer learning over random initialization correlates positively with how closely the target dataset relates to the source dataset. The improvement correlates negatively with the size of the target dataset. Both correlations are seen strongly in [Kornblith et al., 2019], with the positive correlation of performance improvement using transfer learning with the similarity between source and target datasets being most obvious. The authors state that on Stanford Cars and FGVC Aircraft, the improvement was unexpectedly small. The improvement over equivalent models trained from scratch is marginal for these two datasets, at 0.6% and 0.2%. Stanford Cars and FGVC Aircraft both contain fine-grained makes and models of cars and aircraft respectively, whereas ImageNet 1K contains no makes and models, just a few coarse categories for both. This means that the similarity between the source and target datasets for these two tasks is much lower than for example the more general CIFAR-10/100 or Oxford-IIIT Pets for which there are actually slightly more fine-grained classes in the

source than the target dataset.

It is interesting to note that the improvement in using fine-tuning over fixed pretrained features for Stanford Cars and FGVC Aircraft is significantly larger, at 25.4% and 25.7%, than for any of the 10 other datasets used in the experiments. This indicates that the pretrained features are not well suited to the task. For comparison, at the other end of the similarity scale, Oxford-IIIT Pets shows one of the largest improvements from 83.2% accuracy for training from random initialisation to 94.5% accuracy for fine tuning a pretrained model. For this dataset the increase in performance using fine-tuning compared to fixed pretrained weights is marginal at 1.1%. This shows that the pretrained features are well suited to the task without fine-tuning. This difference in performance is likely compounded by the fact that Oxford-IIIT Pets is around half the size of both Stanford Cars and FGVC Aircraft. More recent work has shown that the difference between performance on the target task using fixed pretrained weights versus fine-tuning with even very poor hyperparameters can be used to predict the optimal number of layers to transfer and fine-tuning hyperparameters for a given source and target task and model [Plested et al., 2021].

The negative correlation between the size of the target dataset and the improvement over the baselines in [Kornblith et al., 2019] can be seen clearly when the increase in performance is compared to the target training set size as presented in Table 6.2. Another obvious correlation is that target datasets with lower baseline accuracy figures increase by a larger amount as there is more room for improvement. If we remove all the datasets where the baseline performance was above 88% which would likely limit the performance increase, the increases are close to reverse order based on the size of the target dataset. This highlights the negative correlation between target dataset size and transfer learning performance. The only discrepancies in the ordering come from general and scene tasks getting larger performance increases than fine-grained and texture tasks, see Tables 6.3 and 6.4. We would expect the former to be more closely related to the ImageNet 1K source dataset than the latter.

Both Flowers and Food-101 are interesting outliers when looking at the performance increase compared to the baseline performance, and the size of the target datasets. They are both fine-grained tasks, and ImageNet 1K has very few classes of each. A future research direction could involve looking at the reliance on colours in features for both ImageNet 1K pretrained models and models fine-tuned on these two target datasets as we would expect colour to be useful in classifying both.

Table 6.2: Performance increase compared to target dataset size for experiment results from [Kornblith et al., 2019]

Dataset	Type	Size in thousands	Performance increase / baseline performance	Performance increase rank
Food-101	fine-grained	78	3 / 87	8
CIFAR-10	general	50	1.98 / 96.06	10
CIFAR-100	general	50	6.7 / 81	6
Birdsnap	fine-grained	47	2.5 / 75.9	9
SUN397	scenes	20	11.4 / 55	3
Stanford Cars	fine-grained	8	0.6 / 92.7	11
FGVC Aircraft	fine-grained	7	0.2 / 88.8	12
PASCAL VOC 2007	general	5	16.5 / 70.9	2
Describable Textures	textures	4	11.3 / 66.8	4
Oxford-IIIT Pets	fine-grained	4	11.3 / 83.2	4
Caltech-101	general	3	17.9 / 77	1
Oxford 102 Flowers	fine-grained	2	4.55 / 93.9	7

6.4.5 Smaller target datasets with less similar tasks

Transfer learning has been shown to be effective in many areas where the target datasets are small and less related to ImageNet 1K and other common source datasets. However, there have also been several recent results that fit into this category where deep transfer learning has shown little or no improvement over random initialization [Zoph et al., 2020; He et al., 2018; Raghu et al., 2019]. Transfer learning shows better performance on smaller target datasets that are more closely related to the source dataset than larger less related datasets in general see Section 6.4.4 and [Kornblith et al., 2019]. Task specific self-supervised learning methods applied to source datasets that are more closely related but unlabelled, often perform better than supervised learning methods applied to less closely related source datasets [Zoph et al., 2020; Azizi et al., 2021]. Recent work has shown that even when the target dataset is very dissimilar to the source dataset and transfer learning brings no performance gain it can accelerate the convergence speed [He et al., 2018; Raghu et al., 2019].

Table 6.3: Performance increase compared to target dataset size for general and scene target datasets. Experiment results from [Kornblith et al., 2019]

Dataset	Type	Size in thousands	Performance increase / baseline performance	Performance increase rank
CIFAR-100	general	50	6.7 / 81	4
SUN397	scenes	20	11.4 / 55	3
PASCAL VOC 2007	general	5	16.5 / 70.9	2
Caltech-101	general	3	17.9 / 77	1

Table 6.4: Performance increase compared to target dataset size for fine-grained classification target datasets. Experiment results from [Kornblith et al., 2019]

Dataset	Type	Size in thousands	Performance increase / baseline performance	Performance increase rank
Food-101	fine-grained	78	3 / 87	3
Birdsnap	fine-grained	47	2.5 / 75.9	4
Describable Textures	textures	4	11.3 / 66.8	2
Oxford-IIIT Pets	fine-grained	4	11.3 / 83.2	1

6.4.5.1 Face recognition

Face recognition often relies on pretraining of deep CNN architectures on general object classification datasets like ImageNet 1K. However this area of research presents its own unique challenges [Masi et al., 2018]:

1. With each class representing only one individual there are often only slight differences between classes and a small number of training images per class.

2. There can be many more classes than is common for an object classification task with common face recognition datasets having 10s or 100s of thousands or even millions of different subjects.

Related to Point 1, a common challenge for facial recognition models is to pretrain them on a large number of publicly available faces of celebrities then use them to quickly learn to classify new faces with limited examples via transfer learning techniques. Several additions to the typically used cross-entropy loss have been proposed to help with this challenge they:

- explicitly minimize the intraclass variance [Wen et al., 2016],
- increase the margin between classes [Liu et al., 2016, 2017], and
- encourage features to lie on a hypersphere [Ranjan et al., 2017; Zheng et al., 2018].

These methods are all designed to produce a well defined feature space during pretraining so that there is likely to be a good separation between subjects for both source and target subjects. When the number of subjects in face recognition datasets get very large, deep metric learning losses are generally used instead [Schroff et al., 2015; Wang et al., 2018; Deng et al., 2019]. It has recently been shown that metrics designed specifically for face recognition such as CosFace [Wang et al., 2018] and ArcFace [Deng et al., 2019] can be more successful when applied to common deep metric learning benchmark datasets or small fine-grained object classification tasks [Musgrave et al., 2020]. Deep metric learning is discussed further in Section 6.7.3.

6.4.5.2 Facial expression recognition

Like many fine-grained classification problems facial expression recognition (FER) datasets are often challenging because they are small. Most well known FER datasets have less than 10,000 images or videos. Even the larger ones often have only around 100 different subjects, making the individual images highly correlated. An additional challenge unique to facial expression recognition is that high intersubject (intraclass) variations exist due to different personal attributes, such as age, gender, ethnic backgrounds and level of expressiveness [Li and Deng, 2020].

Again, for this task it has been shown that pretraining with source data more closely matched to the target dataset results in better performance. Pretraining on a large facial recognition dataset has been shown to perform better than pretraining on the more general and less closely related ImageNet 1K [Deng et al., 2009]. A multi-stage pretraining pipeline, using a larger FER dataset for interim fine-tuning prior to the final fine-tuning on the smaller target dataset, was shown to improve performance in [Ng et al., 2015].

6.4.5.3 Pretraining with object classification datasets for medical imaging classification tasks

The two major problems presented when using deep neural networks for medical imaging tasks that are most relevant to this review are:

1. Scarcity of data. Training data in medical image datasets often numbers in just the 100s or 1,000s of examples as opposed to the 100,000s, millions or even billions of examples often available in more general image datasets [Mazurowski et al., 2019].

2. Severely unbalanced classes. There are often many more examples of healthy images as opposed to those with a rare disease [Mazurowski et al., 2019].

Training very deep networks from scratch is problematic in many medical imaging cases due to the problems listed above. Deep transfer learning is adopted in almost every modality of medical imaging, including X-rays, CT scans, pathological images, positron emission tomography (PET), and MRI [Mazurowski et al., 2019]. Despite this there has been limited work addressing best practice for deep transfer learning in the medical imaging classification domain.

Early experiments [Tajbakhsh et al., 2016] compared an AlexNet pretrained on ImageNet 1K with and without fine-tuning, an AlexNet trained from scratch, and traditional models with hand crafted features. They demonstrated that:

1. The use of a pretrained AlexNet CNN with adequate fine-tuning consistently improved on or matched training from random initialisation and traditional methods.

2. While the increase in performance using a pretrained and fine-tuned AlexNet was marginal for relatively larger target datasets the performance improvement was much more significant as the size of the target datasets was reduced.

More recent experiments in deep transfer learning for digital pathology classification using a range of modern models with residual connections and four different target datasets were performed by Mormont et al. [Mormont et al., 2018]. The models were pretrained on ImageNet 1K. When using the pretrained models as fixed feature extractors, the last layer features were always outperformed by features taken from an inner layer of the network. In keeping with previous results in the field they found that fine tuning improves on fixed extraction. However, they did not combine fine tuning with testing different layers in the network for optimal performance, which we suggest for future work.

6.4.5.4 More vs better matched pretraining data in the medical image domain

There have been a number of studies in the last two years examining whether a very large less related source dataset is better, for pretraining for image classification

in the medical domain, when compared to a within domain dataset that is orders of magnitude smaller [Raghu et al., 2019; Heker and Greenspan, 2020]. Both these scenarios have also been compared to self-supervised pretraining on a medium sized unlabelled within domain dataset [Azizi et al., 2021].

Previous results showing that pretraining with much larger source datasets increases performance on the target task were extended to the medical image domain in [Mustafa et al., 2021]. Performance on three well known small medical image classification target tasks was shown to improve as the size of the source dataset increased from ImageNet 1K with 1.3 million training examples to JFT with 300 million training examples. All pretraining produced better performance than random initialization. There has also been some work showing that performing supervised pretraining on a moderately sized medical image dataset and transferring to a smaller one can increase performance compared to training from random initialisation [Kraus et al., 2017; Heker and Greenspan, 2020], other shallow machine learning methods [Kraus et al., 2017], and even pretraining with ImageNet 1K [Heker and Greenspan, 2020].

Raghu et al. [Raghu et al., 2019] showed that when medical image datasets are large (around two hundred thousand training examples), pretraining on ImageNet 1K results in limited improvements over random initialisation. This applies to both the large CNN models large ResNet-50 and Inception v3 and small CNN models designed for better performance on medical datasets. When the number of training examples was reduced to a small dataset size of 5,000, pretraining with ImageNet 1K resulted in small improvements over random initialisation. They also reaffirm that lower layers are more transferable than higher more task specific layers as per [Yosinski et al., 2014], even when the source and target tasks are very different.

A two step self-supervised pretraining process was used on two different medical imaging classification target tasks in [Azizi et al., 2021]. This involved:

1. self-supervised pretraining on ImageNet 1K,

2. followed by self-supervised fine-tuning on a large unlabelled medical dataset from same source as the target task, and

3. then fine-tuning on the final labelled medical image classification task.

This was found to produce significantly better results on the target task compared to self-supervised pretraining on only ImageNet and slightly better results than pretraining on the unlabelled medical dataset only or pretraining on ImageNet 1K with supervised methods.

Using a multi-stage supervised pretraining pipeline such as in [Ng et al., 2015] does not appear to have been applied to classifying medical images with deep CNNs. This could be a useful area of further research.

6.4.5.5 Pretraining with image classification datasets for object detection tasks

Although it is somewhat beyond the scope of this thesis, we briefly review the evidence on whether pretraining on object classification datasets is beneficial for object detection tasks.

Starting with [Girshick et al., 2014] many studies have shown improved performance on object detection tasks when ImageNet pretraining is used [Ren et al., 2015; Redmon et al., 2016]. It has become the conventional wisdom that pretraining is needed to achieve top results on object detection tasks as the datasets tend to be smaller than object classification datasets like ImageNet [Deng et al., 2009]. However, a number of recent results have challenged this idea.

Comparable results on the COCO object detection task [Lin et al., 2014] are shown with random initialisation of weights compared to pretraining on ImageNet 1K [Deng et al., 2009] when training protocols are adjusted to be optimal for training from random initialization [He et al., 2018]. This result is repeated even when the target dataset is reduced to be on 10K images total, or 10% of the full COCO size. A number of other experiments show similar results with and without pretraining [Shen et al., 2017a,b; Zhu et al., 2019b] and that performance is often better without pretraining when more exact measures of bounding box overlap are used [He et al., 2018; Zhu et al., 2019b].

Mixed results from pretraining on the 300 $\times$ larger Instagram hashtag dataset for the same COCO task are reported in [Mahajan et al., 2018]. The improvements are marginal at best, whereas the improvements reported on the ImageNet and CUB2011 classification tasks are larger. However in [Sun et al., 2017], performance on COCO was significantly improved when pretraining on the large JFT source dataset with 300 million training examples, compared to ImageNet 1K. Again, these mixed results seem to indicate that deep transfer learning training protocols can have a large influence on results when target datasets are smaller and less similar to source datasets.

A multi-stage pretraining pipeline [Ng et al., 2015] may again be useful to consider in this domain. Developing more domain specific self-supervised pretraining techniques could also be considered.

6.4.5.6 Semantic Image segmentation

Due to the inherent difficulty of gathering and creating per pixel labelled segmentation datasets, their scale is not as large as the size of classification datasets such as ImageNet. For this reason semantic image segmentation models have traditionally been pretrained on ImageNet 1K and other image classification datasets such as JFT [Garcia-Garcia et al., 2018; Ghosh et al., 2019; Minaee et al., 2020]. The same pattern is noted here as with other transfer learning applications in that more training data tends to produce better results [Chen et al., 2018]. Recently it has been shown that self-training on unlabelled but more closely related data consistently improves performance over training the same model from scratch. Whereas pretraining on ImageNet 1K can reduce performance (negative transfer), particularly when the target dataset is larger [Zoph et al., 2020].

6.4.6 More vs better matched pretraining data

Recent works have tested the limits of the effectiveness of transfer learning with large source and target datasets by pretraining on datasets that that are 6×[He et al., 2017], 300× [Sun et al., 2017; Ngiam et al., 2018; Xie et al., 2020], and even 3000× [Mahajan et al., 2018] larger than ImageNet 1K and target datasets that are up to 9× larger [Mahajan et al., 2018].

In general, increasing the size of the source dataset increases the performance on the target dataset. This occurs even when the target dataset is large such as ImageNet 1K (1K classes, 1.3M training images) and ImageNet 5k (5K classes, 6.6M training images) or 9k (9K classes 10.5M training images) [Ngiam et al., 2018; Mahajan et al., 2018; Kolesnikov et al., 2019]. However, source data that is carefully curated to match target data more closely can perform better than pretraining on larger more general source datasets [Ngiam et al., 2018; Mahajan et al., 2018].

Early work in [Huh et al., 2016] showed that additional pretraining data is useful only if it is well correlated to the target task. In some cases adding additional unrelated training data can actually hinder performance. In more recent work a ResNeXt-101 32×16d was pretrained on various large Instagram source datasets. When using ImageNet 1K as the target dataset, the model was found to have the same performance when pretrained on a source dataset of 960M images with hashtags that most closely matched the ImageNet 1K classes as when the model that was pretrained with 3.5B images with 17K different hashtags [Mahajan et al., 2018]. However, pretraining with the smaller source dataset produced significantly worse performance when the target dataset was ImageNet 5K or 9K, or the more task specific Caltech

Birds [Wah et al., 2011] and Places365 [Zhou et al., 2017]. A similar result was found in [Ngiam et al., 2018]. Pretraining using data from subgroups of classes that closely matched the target classes, consistently achieved better performance than using the entire JFT dataset with 300 million images and 18,291 classes. Similar performance was achieved using their technique of reweighting classes during source pretraining so that the class distribution statistics more closely matched the target class distribution. Interestingly though, applying the same technique to pretraining with the much smaller ImageNet 1K dataset resulted in minimal performance gains in most cases and a significant decrease in performance for one target dataset. This suggests a minimum threshold for the number of training examples where better matched training data is better than more training data. In contradiction to [Mahajan et al., 2018], these results showed that pretraining with the entire JFT dataset resulted in worse performance than pretraining with just ImageNet 1K for most target classes. In some cases performance was worse with pretraining on the entire JFT dataset rather than initialising with random weights without pretraining. This may be an indication of issues with the suitability of the JFT dataset as a source task for a mutually exclusive target classification task, since the image labels are non-mutually exclusive and on average, each image has 1.26 labels.

With a similar set up to [Ngiam et al., 2018], Puigcerver et al. [Puigcerver et al., 2020] produced a large number of different "expert" models by of JFT starting from one model pretrained on all of JFT then fine-tuning copies of this model on different subclasses. Performance proxies such as K nearest neighbours were then used to select the relevant expert for each target task. They found that selecting the best expert pretrained on the whole of JFT then fine-tuned on just a subset of classes resulted in better performance than just pretraining on the whole of JFT. These results are consistent with those from [Ng et al., 2015] outlined in Section 6.4.5.2 in showing that a multi-stage fine tuning pipeline with an intermediate dataset that is more closely matched to the target dataset can produce better performance than transferring straight from a larger, less related source dataset.

Sun et al. [Sun et al., 2017]found that pretraining a ResNet-101 [He et al., 2016] with the 300 million images from the full JFT dataset resulted in significantly better performance on ImageNet 1K classification compared to random initialisation. When the target task was object detection on the COCO dataset [Lin et al., 2014], pretraining with the full JFT dataset or JFT plus ImageNet 1K produced a much larger increase in performance than pretraining with only ImageNet 1K as the source dataset. Both sets of results seem to indicate that the large ResNet-101 model generally overfits to the ImageNet 1K classification task when trained from random initialization, de-

spite the dataset having over 1 million labelled training examples. They found that larger ResNet models produced greater performance gains than smaller models on the COCO object detection task with ResNet-152 outperforming both ResNet-101 and 50. Given that the difference between the mAP@[0.5,0.95] of the ResNet-101 and 152 was not significant when they were both pretrained on ImageNet 1K, but was significant when they were pretrained on the 300 times large JFT dataset, it again indicates that larger models may overfit to ImageNet 1K.

Yalniz et al. [Yalniz et al., 2019] created a multi-step semi-supervised training procedure that consisted of:

1. training a model on ImageNet 1K,
2. then using that model to label a much larger dataset of up to one billion social media images with hashtags related to the ImageNet 1K classes,
3. using these weak self-labelled examples to train a new model, and
4. finally, fine-tuning the new model with the ImageNet 1K training set.

They showed a significant increase in ImageNet 1K performance across a range of smaller ResNet architectures.

Xie et al. [Xie et al., 2020] extended the training procedure from [Yalniz et al., 2019] to iterate between training a large EfficientNet [Tan and Le, 2019] model on ImageNet 1K, then using that model to label the 300 million images from the full JFT dataset and training the model using both the weak self-labelled images from JFT and real labelled images from ImageNet 1K. This resulted in an significant improvement in performance on ImageNet 1K. They also found that testing this model on more difficult ImageNet test sets being ImageNet-A (particularly difficult ImageNet 1K test examples) [Hendrycks and Dietterich, 2019], as well as ImageNet-C and ImageNet-P (test images with corruptions and perturbations such as blurring, fogging, rotation and scaling) [Hendrycks et al., 2021] resulted in large increases in state of the art performance. State of the art accuracy was doubled on two out of the three more difficult ImageNet test sets.

6.4.6.1 More versus better matched pretraining data for small target datasets

Similarly to large target tasks, better matched pretraining data has been shown to produce better performance on the target task than more pretraining data.

Pretraining on the scene classification task Places365 achieved considerably better performance on the scene classification dataset MIT Indoor 67 as compared to pretraining with the smaller and less related ImageNet 1K in [Li et al., 2018]. However

the reverse was true for the target tasks that were more related to ImageNet 1K, being Stanford Dogs and Caltech256-30 and 60.

Source and target data classes were matched using the Earth Mover's Distance [Peleg et al., 1989; Rubner et al., 2000] in [Cui et al., 2018]. Pretraining with well matched subsets was shown to perform better than with the largest source dataset on a variety of fine-grained visual classification (FGVC) tasks. They also showed a strong correlation between source and target domain similarity as calculated by the Earth Mover's Distance and performance on the target task for all but one target task.

Ge and Yu [Ge and Yu, 2017] found model performance was improved by fine-tuning a model on the target task along with the most related data from the source task. Related data was calculated using nearest neighbour on the activations of Gabor filters. Sabatelli et al. [Sabatelli et al., 2018] found that performance was improved when pretraining on a smaller art classification source dataset rather than the larger unrelated ImageNet 1K when the target task was art classification. In the same domain Gonthier et al. [Gonthier et al., 2020] used a two step fine-tuning process consisting of:

1. pretraining on ImageNet 1K,
2. fine tuning on an art classification dataset that was an order of magnitude larger than the target task, and
3. fine-tuning on the final target art classification dataset.

They found that this improved performance over pretraining on only ImageNet 1K or only the interim art classification dataset.

6.4.6.2 Similarity measures for matching source and target datasets

In light of the findings in the previous Section, the question of how to measure similarity between source and target datasets is important. In some cases source and target domain similarity can be effectively estimated through human intuition and/or similarity between class labels [Mahajan et al., 2018; Ngiam et al., 2018; Puigcerver et al., 2020]. Further to this, there are many methods for using calculations of domain similarity in the hopes of finding the best source data for pretraining:

- Cui et al. [Cui et al., 2018] showed that performance on the target dataset increases as a measure of similarity based on the earth mover distance between the source and target domains increases.

- Domain adaptive transfer learning (DATL) [Ngiam et al., 2018] uses importance weights based on the ratio $P_t(y)/P_s(y)$ to reweight classes during source pretraining so that the class distribution statistics match the target statistic $P_t(y)$. Where $P_t(y)$ and $P_s(y)$ describe the distribution of labels in the target and source datasets respectively. They show that using adaptive transfer or a subset of the source dataset that more closely matches the target dataset improves performance over using the entire unweighted 300 million JFT training examples.

- Puigcerver et al. [Puigcerver et al., 2020] train a large selection of expert models on particular subsets of JFT source data based on class labels. They then determine the best model to use for each prediction based on performance proxies such as k nearest neighbours. This technique was shown to increase performance on several target datasets compared to training on the whole JFT dataset.

- Using reinforcement learning to learn a weight for each class in the source dataset so as to rely more heavily on more effective training examples during pretraining [Zhu et al., 2019a]. Also using reinforcement learning to value individual training samples for a particular task and rely more heavily on more valuable examples during training [Yoon et al., 2020].

- Ge et al [Ge and Yu, 2017] create histograms for images from the source and target domain using the activation maps from the first two layers of filters in a CNN. They then use nearest neighbour ranking to find the most similar images in the source dataset to those in the target dataset. Samples in the source domain are then weighted based on their KL-divergence from a given target sample.

6.5 Comparison to label based taxonomy

Transfer learning is often categorised based on the labels available for the source and target task as well as whether the source domain and target domain are from the same distribution as follows [Pan and Yang, 2009].

1. **Inductive transfer learning:** In this case the label information of the target-domain instances is available. The source and target tasks are different but related and the data can be from the same or a different but related domain.

This is the broader category and is generally the case with image classification. For this reason, it is the primary focus of this review.

2. **Transductive transfer learning:** In transductive transfer learning the label information only comes from the source domain. The tasks are the same, but the source and target domains are different. A common example of this is domain adaptation which is covered in Section 6.7.1. Transductive transfer learning is not a focus of this review as it is rare in image classification that the source and target domain have exactly the same class labels. This situation is more common in natural language processing where, for example, a sentiment analysis model is transferred between two slightly different product review domains.

3. **Unsupervised or semi-supervised transfer learning:** Pure unsupervised learning where both the source and target domains have no labels is rarely useful in transfer learning. However, domain adaptation with no target task labels is sometimes referred to as unsupervised domain adaptation. Semi-supervised learning is the more common scenario and refers to when the source domain does not have labels but the target domain does. Semi-supervised learning is most commonly used when there are orders of magnitude more unlabelled data for the target domain or a closely related domain [Srivastava et al., 2015; Pathak et al., 2016; Jing and Tian, 2020; Radford et al., 2015; Ledig et al., 2017]. In this situation an unsupervised learning algorithm is used to train on the unlabelled data before fine-tuning on the labelled data. This technique tends to result in negative transfer if the amount of unlabelled data is not significantly larger than the labelled data [Paine et al., 2014].

We argue that for modern very deep CNNs with millions of parameters in 100s of layers used for image classification, the size of and similarity between source and target domains are much more important than the labels used in pretraining. The robustness and generality of the features learned are similarly much more important than the final classification task they are trained on. For example a source domain that is very large and identical or similar to the target domain but without labels, or with weak labels, will produce better results than a fully labelled source dataset that is very different to the target domain [Zoph et al., 2020; Azizi et al., 2021].

6.6 Discussion

There are two overarching themes throughout this review of transfer learning techniques and applications:

1. More source data is better in general, but more closely related source data for pretraining will often produce better performance on the target task than a larger source dataset.

2. The size of the target dataset and how closely related it is to the source dataset strongly impacts the performance of transfer learning. Using sub-optimal transfer learning hyperparameters can result in negative transfer when the target dataset is less related and large enough to be trained from random initialisation.

Currently, there tends to be an all or nothing approach to transfer learning. Either transferring all layers improves performance or it does not and possibly decreases performance (negative transfer). The same approach is often taken to freezing layers, either layers are frozen or they are fine-tuned at the same learning rate as the rest of the model, and again with weight regularization, either all transferred weights are decayed towards their pretrained values (L2SP) or towards zero (or sometimes not at all). We advocate that this all or nothing approach should be discarded, and instead all decisions made about how to perform transfer learning should be thought of as sliding scales. Transferring all layers, freezing layers and decaying all weights towards pretrained values are at one extreme of the scale and likely to be optimal only for source and target datasets that are extremely similar. Training from scratch is at the other end of the scale and is likely to only be optimal if the source and target domain have no similarities. This lines up with the observations in [Yosinski et al., 2014] that "first-layer features appear not to be specific to a particular dataset or task, but general in that they are applicable to many datasets and tasks". More work is needed to show how much transfer learning is optimal for a given source and target dataset relationship and target dataset size, rather than focusing on whether transfer learning is effective.

6.7 Areas related to deep transfer learning for image classification

There are several different problem domains that are either closely related to deep transfer learning or that use it as a standard part of state of the art models. We briefly examine how they relate to deep transfer learning for image classification below.

6.7.1 Domain adaptation

Domain adaptation (DA) could be considered the most closely related problem domain. In domain adaptation the task T remains the same, but the Domain D differs between source and target task. According to the definition 5.1 by Pan and Yang [Pan and Yang, 2009] domain adaptation falls under general transfer learning as transductive transfer learning as described in Section 6.5. Others define it as a separate area [Goodfellow et al., 2016]. Either way domain adaptation is not a focus of this review as it is rare in image classification that the source and target domain have the exact same class labels. This situation is more common in natural language processing where you may find for example, a sentiment analysis model transferred between two slightly different product review domains [Csurka, 2017; Wang and Deng, 2018; Zhang et al., 2017b]. The few datasets that are used in domain adaptation for image classification are carefully curated to have identical object classes in different domains [Gong et al., 2012; Saenko et al., 2010; Tommasi and Tuytelaars, 2014].

Despite there being few examples of domain adaptation problems in object classification, there are techniques that have been developed for these tasks that could be more broadly useful to deep transfer learning for object classification. Wang and Deng state that "When the labeled samples from the target domain are available in supervised DA, soft label and metric learning are always effective" [Wang and Deng, 2018]. Metric learning has been shown to be effective in K-shot learning as described in Section 6.7.3. It has recently been shown to improve performance with slightly larger but still small fine-grained target datasets with up to 42 examples per class [Ridnik et al., 2020]. It is also common in domain adaptation to use a multi-stage adaptation similar to that shown to be effective for facial expression recognition in [Ng et al., 2015].

6.7.2 Concept drift and multitask learning

The concept drift problem is related to domain adaptation in that it deals with adapting models to distributions that change over time. Multitask learning is another related problem domain where the focus is on learning one model that can perform well at multiple tasks in multiple domains. While both can be seen as types of transfer learning, the distinction between pure transfer learning focused tasks and concept drift or multitask learning is an important one. The focus in the former is to learn just the target task as well as possible and performance is only judged on this. Whereas in both concept drift and multitask learning the focus is to learn more than one task well:

- in concept drift learning the new distribution needs to be modelled well without forgetting all learning about the old distribution, and
- in multitask learning many tasks need to be learned well. This needs to be achieved without new tasks overwriting previously learned tasks.

The issues described above are examples of catastrophic forgetting. Catastrophic forgetting is defined as the scenario where learning a new set of patterns suddenly and completely erases a network's knowledge of what it has already learned [French, 1999; McCloskey and Cohen, 1989; Ratcliff, 1990]. The major difference between general transfer learning and concept drift or multitask learning is that usually catastrophic forgetting is not a consideration in the former. This is because the focus is entirely on the model's performance on the new task.

6.7.3 K-shot or few shot learning

The K-shot (also referred to as few shot) learning problem domain is specifically focused on learning from very few or even zero labelled examples. The focus of new work in this domain is generally on improving metrics for defining and learning the best embedding spaces and comparisons to new examples, plus tricks of data augmentation. Transfer learning is implicit in most of the techniques used, as weights are pretrained on a related task [Wang et al., 2020b; Kaya and Bilge, 2019].

Methods for refining deep transfer learning methods for k-shot image classification tasks can be divided into:

- data augmentation and other methods for improving the source dataset,
- metrics that define the "goodness" of the embedding space and how that information should be used to classify small target classes, and
- methods that look at the best way to update weights based on the target dataset.

We focus on methods that fall under the last heading, as they are most relevant to this review. However, as stated previously, we note that metric methods have recently been shown to improve transfer learning performance for some small datasets that fall outside of the standard K-shot learning definition [Ridnik et al., 2020]. More research is recommended to extend these results.

Weight update methods Weight update methods look at the best way to update weights from the source or other related tasks, including the decision not to update

any weights. Many models across a broad spectrum of K-shot learning techniques do not perform any updates to weights using the target dataset. There is some recent evidence that shows that weight updates are superior in a variety of cases [Scott et al., 2018], but again given the small target dataset sizes the right fine-tuning settings must be used. Many models that do not update weights to the target task do simulate the few-shot learning scenario in training on related datasets. They train on a subset of the available classes, then optimize the model by maximising performance on the unseen classes. This results in a model that generalises well to unseen examples [Vinyals et al., 2016; Snell et al., 2017].

In Generalizing from a Few Examples: A Survey on Few-shot Learning [Wang et al., 2020b], they create a useful taxonomy of methods to use prior knowledge to solve the few-shot or k-shot learning task. These are:

- data augmentation,
- informing the model space, and
- informing the parameter space.

They explicitly define transfer learning as falling under the informing the parameter space umbrella. Under this general umbrella transfer learning and related topics can be split into several different strategies for dealing with very small target datasets:

1. Refining pretrained parameters: these methods explicitly look at ways for improving deep transfer learning methods for very small target datasets. They are grouped into ways of explicitly regularizing to avoid overfitting, including:
    - early stopping,
    - selectively updating only certain weights,
    - updating certain groups of weights together, and
    - weight regularization.
2. Refining meta-learned parameters: these methods are drawn from multitask learning. Parameters W_0 are learned from several related tasks, then again fine-tuned on the target task. The models differ in the way the weights are updated from the multitask to the task specific model. Examples include sharing lower layer weights between the multitask and task specific models and regularizing by penalising differences in task specific weights on different tasks to encourage all task specific models to have similar weights. Regularizing methods that use task specific information or model the uncertainty of using meta-learned parameters can also be included in this heading.

3. Learning the optimizer: using optimizers that were designed to achieve high level performance with large numbers of training examples may not perform well in settings with limited training examples. These methods work on learning optimizers that can perform better than hand designed optimizers in a specific problem domain when fine-tuning a pretrained model with a small number of training examples.

While there is a lot of cross-over of deep transfer learning for image classification from the refining pretrained parameters category of K-shot learning, the latter do not focus on the subtleties of image classification as an application. As they focus only on very small target datasets, they also do not consider how the change in size of the target dataset and the relationship between the source and target dataset affect the optimal transfer learning methods and hyperparameters.

6.7.4 Unsupervised or self-supervised learning

In unsupervised or self-supervised learning (self-supervised) a model of a dataset distribution is learned by using an aspect of the data itself as a training signal. The most general example is autoencoders which consist of an encoder and decoder with the target being the same as the input. The middle encoding is regularized in some way, in the hopes of producing a meaningful semantic encoding. Other examples of task specific self-supervised learning include predicting the next frame of a video [Srivastava et al., 2015], predicting the next word in a sentence [Brown et al., 2020], image inpainting or upsampling [Pathak et al., 2016; Ledig et al., 2017], and many more. For a detailed treatment of self-supervised methods for learning image representations see [Jing and Tian, 2020]. Generative Adversarial Networks (GANs) are also an example of self-supervised learning [Radford et al., 2015; Goodfellow et al., 2014]. In learning to classify real vs fake training examples the discriminator in a GAN learns useful features of the data distribution and the weights can then be used to initialise a classification model.

Self-supervised learning relates closely to transfer learning as it is often used for pretraining when there is limited labelled data for a source task, but a large amount of related unlabelled data. Self-supervised learning has been shown to be beneficial to performance when the ratio of unlabelled to labelled training examples is high, but may harm performance when the ratio is low [Paine et al., 2014]. This relates back to the question of more versus better related source training data that comes up regularly in deep transfer learning and is covered in Sections 6.4.6 and 6.4.6.1.

6.8 Discussion and suggestions for future directions

6.8.1 Summary of current knowledge

Early work in deep transfer learning for image classification showed that transfer learning is effective compared to training from randomly initialised weights, particularly when it involves small target datasets and the source and target datasets are similar [Agrawal et al., 2014; Yosinski et al., 2014; Azizpour et al., 2015].

Our review has highlighted the limits of our knowledge in each area and suggested future research directions to expand the current body of knowledge:

1. Fine tuning tends to work better than freezing weights in most cases [Agrawal et al., 2014; Yosinski et al., 2014; Azizpour et al., 2015]. Freezing lower layers may work better in limited cases where the target domain is small and the tasks are extremely similar [Plested and Gedeon, 2019]. More work is needed to see whether this applies with modern deep CNNs.

2. Recent regularization techniques such as L2-SP, DELTA, BSS and stochastic normalization tend to improve performance when source and target tasks are very similar. However, it has been shown that L2-SP and DELTA particularly can result in minimal improvement or even worse performance when the source and target datasets are less related [Li et al., 2020a; Wan et al., 2019; Plested et al., 2021; Chen et al., 2019]. Most work so far has focused on how these techniques compare when the source and target dataset are very closely related. More work is needed to show how each of these methods perform when they are applied to less similar datasets. Recent work has shown that in some cases using L2-SP regularization for lower layers and L2 regularization for higher layers can improve performance over using either one for all layers [Plested et al., 2021]. While the evidence for this is so far limited it does align with observations from [Yosinski et al., 2014] that lower layers are more transferable than higher layers. Thus the former may need less flexibility to move from their pretrained values.

3. Both the learning rate and momentum should be lower during fine-tuning for more similar source and target tasks and higher for less closely related tasks [Li et al., 2020a; Plested et al., 2021]. The learning rate should also be decayed more quickly the more similar the source and target tasks are, so as not to change the pretrained parameters as much [Kolesnikov et al., 2019]. Similarly the learning rate should be decayed more quickly with smaller target datasets where the empirical risk estimate is likely to be less reliable and overfitting

more of a problem [Plested and Gedeon, 2019; Kolesnikov et al., 2019]. However, when the target data set is small it must be taken into account that the number of weight updates per epoch will be low and the number of updates should be reduced, not necessarily the number of epochs. When the source and target datasets are less similar it may be optimal to fine-tune higher layers at a higher learning rate than lower layers [Plested et al., 2021]. More work is needed to show how the learning rate, momentum and number of updates before decaying the learning rate change when the source and target tasks are very different.

4. Transferring more layers is better than less layers when source and target datasets are large and very similar and learning rates are high for all layers [Yosinski et al., 2014]. This does not necessarily hold true when target datasets are smaller and fine tuning hyperparameters are more optimal [Plested and Gedeon, 2019; Plested et al., 2021]. This relates back to Point 3 that the learning rate and momentum should be lower for more similar tasks. As the empirical risk minimizer is unreliable for small datasets it is more prone to overfitting. Pretraining weights limits how far they can move based on this unreliable risk minimizer [Neyshabur et al., 2020; Liu et al., 2019a], so it acts as a regularizer to prevent overfitting. Another way to limit overfitting is to use a smaller learning rate and momentum, and also decay the learning rate quickly to prevent weights becoming too large based on unreliable statistics. These two items combined together mean it is likely that a larger fine-tuning learning rate is optimal when more layers are pretrained and a lower learning rate when less layers are pretrained [Plested and Gedeon, 2019; Plested et al., 2021]. For this reason it is important to tune both the number of layers pretrained and optimal learning rate together. Recent work has also shown that using fixed pretrained features from lower layers without fine-tuning can result in better performance than features from higher layers when source and target datasets are less similar and the target dataset is small [Mormont et al., 2018].

5. More source and target training data is better in general. Improvements in performance from pretraining on datasets that are up to 3,000 times larger than ImageNet 1K have shown that the largest modern models are overfitting on ImageNet 1K. The regularization associated with pretraining with much larger datasets helps prevent this [Ngiam et al., 2018; Mahajan et al., 2018; Kolesnikov et al., 2019]. However, closely related data is better than more data when data is abundant [Ngiam et al., 2018; Mahajan et al., 2018]. Self-supervised learning

on a more related unlabelled source dataset has been shown to improve performance over supervised learning on a less related labelled dataset in some cases [Heker and Greenspan, 2020; Zoph et al., 2020]. More work is needed to show under what circumstances the former is better than the latter. A multi-step fine-tuning process including a more closely related interim task has been shown to improve performance over a single step transfer from ImageNet 1K in a number of specialised tasks with target datasets that are very different from ImageNet 1K [Ng et al., 2015; Gonthier et al., 2020; Azizi et al., 2021]. More work is needed to see if this technique could benefit other tasks where limited closely related data is available.

6. Measures of transferability can predict the performance of a pretrained model on a particular target task [Bao et al., 2019; Tran et al., 2019; Nguyen et al., 2020]. The performance of a simple classification model with frozen weights can give insight into transfer learning best practice for the target task [Plested et al., 2021]. More work is needed to determine if transferability measures in general can help determine optimal transfer learning hyperparameters across a wide range of source and target datasets.

6.8.2 Recommendations for best practice

Our recommendations for best practice in deep transfer learning for image classification broken down by target dataset size and similarity to the source dataset are summarized below.

Larger, similar target datasets Most techniques will work well in this case, so less time needs to be spent finding the best technique for the given task. A lower learning rate and momentum are better for fine-tuning when the source and target datasets are similar. However, this may not be necessary when the target dataset is very large, for example ImageNet 1K, and overfitting a poor empirical risk minimizer is less of a problem. All weights should be fine-tuned not frozen. L2-SP regularization or other more recent techniques like DELTA, BSS, stochastic normalization etc. are likely to work reasonably well. However, less regularization may be needed in general with a very large target dataset.

Larger, less similar target datasets In this case pretraining weights may not improve performance and negative transfer is more likely, so care needs to be taken in

using pretraining. When fine-tuning in this scenario a higher learning rate, momentum and training for longer before decaying the learning rate should be used. This is to attempt to move weights out of the flat basin of the loss landscape that is created when using pretrained weights and further away from their pretrained values [Neyshabur et al., 2020; Liu et al., 2019a]. Recent regularization techniques like L2-SP, DELTA, BSS, stochastic normalization, etc. should not be used in this case as weights should be allowed to move freely away from their pretrained values based on the larger target dataset. Pretraining fewer layers of weights could also be attempted in order to take advantage of the more general lower layers, while allowing the more task specific higher layers to train from random initialization [Yosinski et al., 2014; Plested et al., 2021].

Smaller, more similar datasets This scenario is where transfer learning really excels, although more care needs to be taken to use optimal hyperparameters when the dataset is small [Plested and Gedeon, 2019]. The optimal learning rate and momentum will likely be low as the weights will not need to be moved far from their pretrained values and if they are, overfitting the unreliable empirical risk minimizer is more likely with a small dataset. Recent regularization techniques like L2-SP, DELTA, BSS, stochastic normalization, etc. are likely to improve performance significantly in this case.

Smaller, less similar datasets In this case transfer learning can be very useful if done well, but can also lead to poor results. In this case it is difficult to strike and optimal balance between:

1. allowing the weights to move far enough from their pretrained values that the model is not using inappropriate features for the classification task, and
2. not allowing the weights to overfit the unreliable empirical risk minimizer.

Given this if there is any way to use a more closely related dataset it is likely to improve results. This could be:

1. doing supervised pretraining on a large unlabelled more closely related dataset instead of a large less similar dataset, and
2. using a moderately sized closely related dataset either instead of a large source dataset for pretraining or as an interim fine-tuning step when transferring weights from a less related source task to the final target task.

If it is not possible to use a more related dataset then a high initial learning rate and momentum should be used, but the learning rate should be decayed quickly. Recent regularization techniques like L2-SP, DELTA, BSS, stochastic normalization, etc. should not be used on all layers, but in some cases may be beneficial on lower layers.

Chapter 7

An Analysis of the Interaction Between Transfer Learning Protocols in Deep Neural Networks

7.1 Preamble

In this Section I extend work on the transferability of features in deep neural networks to explore the interaction between training hyperparameters, optimal number of layers to transfer and the size of a target dataset. I show that using the commonly adopted transfer learning protocols result in increased overfitting and significantly decreased accuracy compared to optimal protocols, particularly for very small target datasets. I demonstrate that there is a relationship between fine-tuning hyperparameters used and the optimal number of layers to transfer. My research shows that if this relationship is not taken into account, the optimal number of layers to transfer to the target dataset will likely be estimated incorrectly. Best practice transfer learning protocols cannot be predicted from existing research that has analysed transfer learning under very specific conditions that are not universally applicable. Extrapolating transfer learning training settings from previous findings can in fact be counterintuitive, particularly in the case of smaller datasets. I present optimal transfer learning protocols for various target dataset sizes from very small to large when source and target datasets and tasks are similar. My results show that using these settings results in a large increase in accuracy when compared to commonly used transfer learning protocols. These results are most significant with very small target datasets. I show an increase in accuracy of 47.8% on the smallest dataset which comprised of only 10 training examples per class. These findings are important as they are likely to improve outcomes from past, current and future research in transfer learning. I expect that researchers will want to re-examine their experiments to incorporate our find-

ings and to check the robustness of their existing results. This work is my original ideas and implementation, with support from my supervisor to discuss and iterate ideas.

7.2 Introduction

Transfer learning in neural networks generally involves pretraining weights on a large source dataset then applying these weights to a problem on a related target dataset. In this paradigm either:

- all but the last layer of weights are frozen, essentially using these weights as a feature detector for another classification algorithm [Sharif Razavian et al., 2014; Azizpour et al., 2016; Donahue et al., 2014], or
- fine-tuning is performed where all or some of the weights are retrained to fit the target dataset and the original pretrained weights act as a regularizer to prevent overfitting [Agrawal et al., 2014; Huh et al., 2016; Girshick et al., 2014; Kornblith et al., 2019; Mormont et al., 2018; Tan and Le, 2019].

The former is generally used when the target dataset is small and the latter when it is larger.

Deep convolutional neural networks (CNNs) have revolutionized computer vision [Krizhevsky et al., 2012]. Since then a large body of research has shown that performing transfer learning by pretraining a CNN on a large dataset like ILSVRC2012 [Deng et al., 2009], and fine-tuning the pretrained weights on a related target dataset, can improve results on a wide range of target tasks [Girshick et al., 2014; He et al., 2017; Huh et al., 2016; Sun et al., 2017; Mahajan et al., 2018; Kornblith et al., 2019; Mormont et al., 2018; Tan and Le, 2019].

The most systematic and thorough analysis of transfer learning on CNNs to date is the work of Yosinski et al. [Yosinski et al., 2014]. They showed that transferring weights pretrained on a related dataset and then fine-tuning them, results in networks that generalize better than those trained directly on a large target dataset. They also showed that when fine tuning is done with standard hyperparameters, routinely used for training from scratch (initial learning rate of 0.01, decay 0.1 every 30 epochs), accuracy on the target dataset increases as the number of layers transferred also increases.

The results of Yosinski et al. have been regularly adopted as an accepted best practice paradigm for transfer learning with CNNs, notwithstanding differences

from Yosinski et al.'s target dataset size. There are many examples in the literature that have adopted this practice of all but the final classification layer being transferred with training settings identical or similar to those outlined above[He et al., 2018; Scott et al., 2018; Wu et al., 2018; Kornblith et al., 2019; Mormont et al., 2018; Tan and Le, 2019]. These settings have been used to:

- show when transfer learning is effective [He et al., 2018],
- compare transfer learning with other methods for small target datasets [Scott et al., 2018], and
- question whether transfer learning is useful at all [He et al., 2018; Scott et al., 2018].

Other works ostensibly appear to back up the conclusions adopted by the community from Yosinski et al.'s results. However, the results were only demonstrated to apply under very specific experiment conditions. For example Azizpour et al. [Azizpour et al., 2016] used transferred layers of weights as a feature detector with a linear Support Vector Machine (SVM). They showed that in similar tasks all except the final layer of a CNN should be transferred and for even the least related image tasks all but the final two or three layers should be transferred. Their protocol differs from current standard transfer learning practices in that they performed no fine tuning on the transferred weights, and they did not replace and reinitialize the layers not transferred. Replacing more layers with an SVM results in fewer parameters. This likely biases the experiments towards reporting higher accuracy with more layers transferred, particularly with larger target datasets.

Recent work drives this transfer learning paradigm further by pretraining on datasets that are 6× [He et al., 2017], 300× [Sun et al., 2017], and even 3000× [Mahajan et al., 2018] larger than ILSVRC2012. While this body of work demonstrates significant improvements on image classification transfer learning tasks with large target datasets the opposite has often been shown for less related tasks or smaller target datasets [He et al., 2018; Scott et al., 2018]. This led us to question whether transfer learning protocols developed in [Yosinski et al., 2014] and widely adopted by the community are actually best practice for target datasets that are smaller than those used for the experiments performed by Yosinski et al. To answer the above question this paper explores the relationship between:

- the optimal number of pretrained layers to transfer to a target task,
- fine-tuning hyperparameters including initial learning rate for each, and layer and learning rate decay settings

- the size of the target dataset.

We show that the interaction between these settings results in the optimal number of layers to transfer being overestimated or underestimated, depending on the size of the target dataset, if transferred weights are trained for too many epochs at too high a learning rate on the target dataset. This relationship is accentuated for smaller datasets and can result in transfer learning showing no improvement or even a negative effect when performed with the protocols adopted from [Yosinski et al., 2014] as was shown in [Scott et al., 2018].

In this paper we make several contributions. We show:

1. There is an interaction between optimal number of layers to transfer and fine-tuning hyperparameters.

2. Freezing layers of pretrained weights after transferring them rarely produces the best results.

3. Using optimal transfer learning protocols we developed, based on the outcomes of our experiments, produces large changes in results compared to the commonly adopted existing protocols. This particularly applies when working with smaller target datasets. This is likely to impact results that have compared transfer learning, using commonly adopted protocols, with other methods.

4. Optimal transfer learning protocols cannot be predicted from existing research.

We also provide optimal transfer learning protocols for use with a range of target dataset sizes from very small to large where the source and target datasets are similar.

7.3 Experiments

Our goal is to extend existing work on transfer learning in particular the seminal work of [Yosinski et al., 2014] and to identify limitations of its application. We aim to develop transfer learning and fine-tuning protocols based on the size of the target dataset. Given our intention is to compare transfer learning paradigms only we have used a standard Pytorch [Paszke et al., 2017] implementation of the model AlexNet [Krizhevsky et al., 2012] with no modern architectural improvements. This model is widely used in existing works, allowing our results to be more universally comparable and applicable.

We followed the same experiment protocols as [Yosinski et al., 2014]. We split the 1000 object classes in ILSVRC2012 randomly into 500 classes for pretraining with

the remaining 500 classes acting as the target dataset. We repeated this process four times for each experiment to create four separate splits of 500 classes for pretraining and 500 classes for the target dataset. We refer to the target dataset that includes all training examples available for each of the 500 random classes as full. The full target datasets have around 645,000 training examples and a minimum of 1,000 per class. We then created different sized target datasets by drawing the first x number of training examples per class to create datasets of size 500x. We selected the first x per class to ensure the same training examples were used across different target dataset sizes and class splits. The other target dataset training set sizes have 500, 250, 100, 50, 25 and 10 training examples per class. We use all the standard ILSVRC2012 validation examples from each random set of 500 target dataset classes for testing. This means there are 50 test examples per class.

7.4 Results and Discussion

We performed six sets of experiments. The first experiments varied the number of layers transferred for each of a standard set of target dataset sizes and is discussed in Section 7.4.1. Experiments where we varied the number of layers trained at an initial high learning rate are presented in Section 7.4.2. Section 7.4.3 revisits the experiments varying the number of layers transferred incorporating the best settings from the experiments in Section 7.4.2. Results from experiments varying the number of epochs before the initial high learning rate is decayed are discussed in Section 7.4.4. Experiments to determine the optimal number of layers to freeze are presented in Section 7.4.5. Finally the results using the optimal transfer learning protocols derived from all previous experiments are compared to results using commonly adopted protocols from [Yosinski et al., 2014] as well as training from scratch with no transfer learning in Section 7.4.6.

7.4.1 Vary number of layers transferred for different size target datasets

Figure 7.1 shows the results of all our transfer learning experiments varying the number of layers transferred for our standard set of target dataset sizes while keeping the training hyperparameters constant. The training hyperparameters used were as per [Yosinski et al., 2014] with an initial learning rate of 0.01 for all layers reducing by 0.1 every 30 epochs. These results show that for the full target dataset size the accuracy increases slightly with the number of layers transferred. This is in keeping with the outcomes of [Yosinski et al., 2014]. However, as soon as the size of the dataset

Figure 7.1: Number of layers transferred vs dataset size. Each bar represents the average accuracy over the validation set for a model fine-tuned on one size of target training dataset with a particular number of layers of pretrained weights transferred.

Table 7.1: Overfitting on small target datasets

		training examples per class			
layers	**loss**	**10**	**25**	**50**	**average training and validation loss difference**
4	training	1.55	1.22	1.35	2.91
	validation	5.07	4.27	3.51	
5	training	0.71	0.82	1.06	3.63
	validation	5.38	4.43	3.67	
6	training	0.49	0.60	0.83	3.90
	validation	5.34	4.48	3.81	
7	training	0.39	0.48	0.71	4.72
	validation	6.26	5.18	4.29	

is halved this result no longer holds. It is better to transfer only five layers rather than all seven as per standard practice. This result is most pronounced for the smallest dataset where accuracy increases from 20.9% when transferring all seven layers to 23.4% when only transferring five layers. This is an increase of 12.2% compared to the original accuracy. These results show that for smaller datasets:

- transferring the correct number of layers has an increasing impact on accuracy, and
- transfer learning in general increases in importance as training from scratch is no longer viable.

Another surprising result is that for smaller target datasets overfitting is much more pronounced when more layers are transferred to the target task. This is shown

in Table 7.1. The aim of pretraining is to act as a regularizer and reduce overfitting. Contrary to this, on a small dataset transferring more layers of pretrained weights exacerbates the overfitting problem compared to transferring less layers. This finding is more intuitive than it sounds. With smaller target datasets the model does not see enough training examples in the fine-tuning stages to make significant changes to the large pretrained weights. This leaves it with features that are not well suited to the target task, so when aggressive training hyperparameters are used the model overfits to the small training set.

7.4.2 Vary number of layers trained at a high learning rate initially

Starting from the final classification layer, this experiment varied the number of layers trained at an initial high learning rate of 0.01 for 30 epochs before decaying it by 0.1. The remaining layers were trained with the same regime but using an initial learning rate of 0.001. The number of layers transferred was set at 6 as it produced either the top or comparable to the top accuracy across all target dataset sizes in our other experiments. Figure 7.2 shows the results of this experiment across the full set of target dataset sizes. It demonstrates that accuracy increases as the number of layers trained at a high learning rate decreases, for all target dataset sizes. This indicates that the fine-tuning hyperparameters used in [Yosinski et al., 2014] and in much of the literature since, are too aggressive for the cases examined in our work where source and target datasets and tasks are similar. Another point to note about the results shown in Figure 7.2 is that there is a noticeable drop in accuracy when the final two layers are trained with a high learning rate compared to either one or three layers. This is interesting, as it coincides with the number of layers reinitialized. This again highlights overfitting due to aggressively training with features that are not well suited to the target task.

7.4.3 Compare number of layers to transfer with low and high initial decay epochs

We replicated experiment 3.1 examining the number of layers to transfer, this time only the final layer was trained at a high learning rate initially. We performed the experiment with a high setting of 30 epochs before the initial high learning rate was decayed. We then repeated this with a low setting of five epochs The chart on the left in Figure 7.3 shows that with a high setting for initial decay epochs the optimal number of layers to transfer is five, for all but the two smallest target datasets. However, when a low setting for the initial decay epochs is used the optimal number

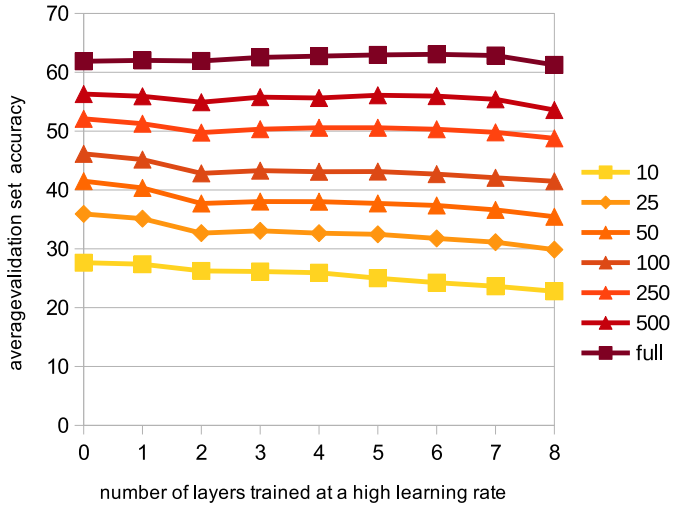


Figure 7.2: Number of layers trained at a high learning rate. The degradation in performance, across all target dataset sizes, as more layers are trained with a high initial learning rate is shown

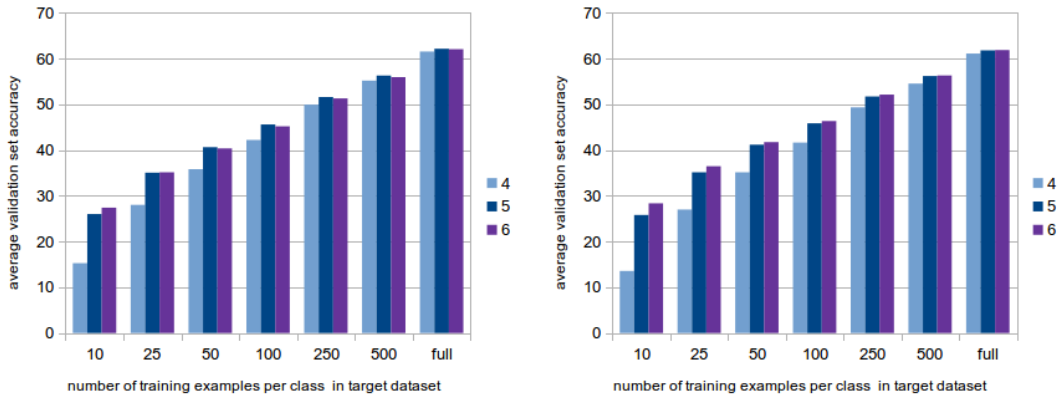


Figure 7.3: Number of layers to transfer with low and high initial decay epochs. Comparison of the change in the average accuracies across number of layers transferred with a low (5) left and high (30) right initial number of epochs before decay. All experiments are performed with only the final classification layer trained at a high learning rate initially.

of layers to transfer is six, for all target dataset sizes except the largest. The lower initial decay setting also results in higher accuracies for all but the largest dataset size. The biggest increase in accuracy is 3.5% on the smallest dataset. Whereas the

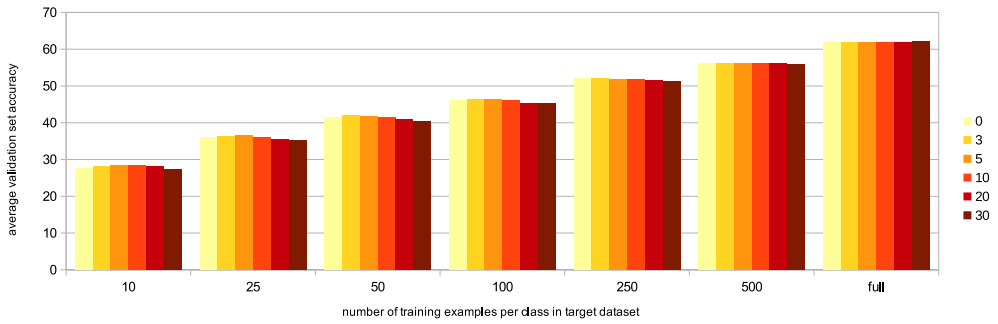


Figure 7.4: Initial decay epochs. Changes in the average validation set accuracy are plotted, across all target dataset sizes, as the number of epochs before decay are varied for the high learning rate on the final classification layer.

increase on the second largest dataset is only 0.07% and the accuracy on the largest dataset is lower when a small number of epochs before decay are used.

The findings from this experiment are significant as they show the interaction between number of layers to transfer and fine-tuning hyperparameters when pinpointing the optimal transfer learning protocols. This result can also be observed by comparing Figure 7.1 to Figure 7.3. When all layers are trained with an initial high learning rate of 0.01 for 30 epochs, transferring four layers rather than five or six produces only marginally lower results. However, when just the final classification layer is trained with a high learning rate initially, transferring only four layers produces significantly lower results. Both these observations show that if initial learning rates and decay epochs are set too high the optimal number of layers to transfer will be overestimated or underestimated depending on the size of the target dataset.

7.4.4 Vary initial decay epochs

Our hypothesis was that training for 30 epochs initially at a high learning rate of 0.01 is too aggressive for fine tuning. To test this, we varied the initial decay epochs between 0 and 30 while fixing the number of layers transferred at six. The results of these experiments are shown in Figure 7.4. Confirming this hypothesis our results show that training for 30 epochs initially at a high learning rate of 0.01 is too aggressive when fine-tuning on a target dataset and task that is very similar to the pretraining dataset. This continues the pattern noted in the previous experiments that an aggressive training scheme causes overfitting. This experiment shows that the pattern is evident even when only training the final classification layer heavily. A surprising result in these experiments is that as the size of the target dataset de-

creases the number of epochs to train the classification layer at a high learning rate increases. An unexpected curve can be seen in the optimal number of epochs to train at a high learning rate before decay for each target dataset size. These epochs are as follows:

- for the smallest dataset 10 epochs,
- for medium sized datasets with 50, 100 and 250 training examples per class 0 to 3 epochs, and
- for the largest dataset 30 epochs is optimal.

This may seem counterintuitive, however, the very small number of training examples per epoch for the smallest datasets needs to be considered. When this is taken into account these results once again show the effects of overfitting when fine-tuning transferred weights on smaller target datasets. Our smallest target dataset of 10 training examples per class has 5, 10 and 25 fewer batches per epoch than our target datasets with 50, 100 and 250 training examples respectively. Because of this the weights are changed far fewer times during each training epoch. This means that training for 10 epochs on our smallest dataset results in the same number of weight updates as training for only 1 epoch on a dataset with 100 training examples. This further extends our findings that the fine-tuning hyperparameters commonly adopted from [Yosinski et al., 2014] are too aggressive to produce the optimal results, particularly for smaller datasets. The weights for smaller datasets should be updated less times at a high learning rate to achieve the best accuracy.

The optimal number of epochs, to train the final classification layer at a high learning rate, increases significantly when fine-tuning with the full set of target dataset training examples. This is in keeping with the discussion above. With a large dataset there are enough training examples to ensure that:

- fine tuning the transferred weights at lower learning rates changes the features used for classification enough, and
- performing classification with features that are not optimal for the task does not result in as much overfitting as for smaller datasets.

For this reason, it is not possible to use optimal settings for transfer learning with very large target datasets to perform transfer learning with smaller target datasets or predict the latter from the former.

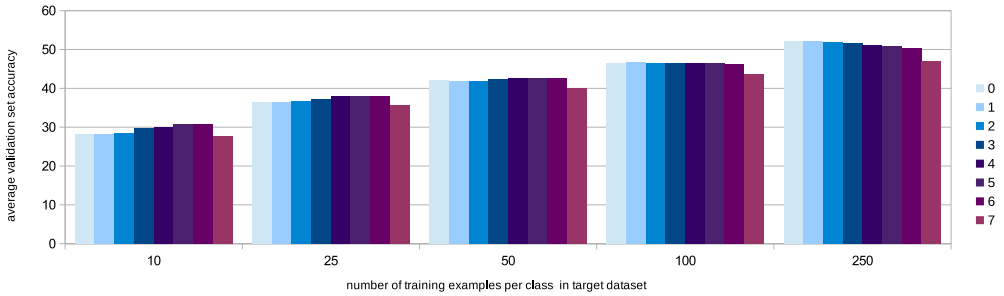


Figure 7.5: Number of frozen pretrained layers. The effect on performance of the number of pretrained layers of weights frozen during fine-tuning are shown across various target dataset sizes.

7.4.5 Vary number of frozen layers

Current practice for transfer learning with CNNs generally involves locking or freezing the weights of some of the lower layers of a pretrained CNN while fine-tuning on the target task to prevent overfitting. We tested the assumption that this practice improves performance in transfer learning on CNNs, particularly for small datasets, by varying the number of layers frozen during fine-tuning while keeping the number of layers transferred and hyperparameters constant. The results of the experiments in Figure 7.5 show that freezing a high number of layers results in a significant increase in accuracy for very small target datasets of 10 and 25 training examples per class. However, once the target dataset increases to 50 training examples per class, freezing layers has very little effect and freezing more than one layer quickly becomes detrimental as the size of the target dataset continues to grow. When considering these results for freezing layers it should be kept in mind that the source and target datasets for these experiments are as similar as possible, both being drawn from ILSVRC2012, and the tasks, image classification, are identical. We expect that freezing layers will be less useful for very small target datasets and more detrimental for anything larger, for less related source and target datasets and/or tasks. We conclude that freezing layers has very limited scope to improve results on transfer learning.

7.4.6 Optimal results

Figure 7.6 shows the results of using optimal number of layers to transfer and hyperparameters for fine-tuning when performing transfer learning. These results show that while using the optimal transfer learning protocols improves results for all dataset sizes it becomes particularly important as the size of the dataset decreases.

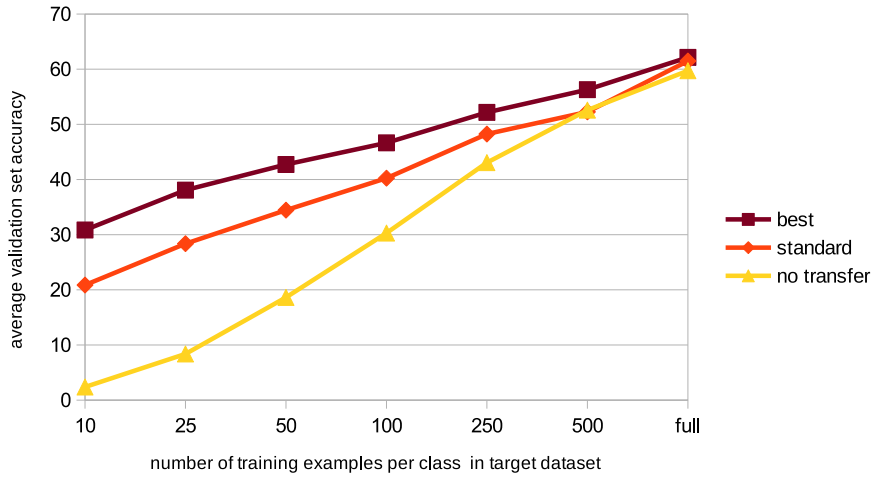


Figure 7.6: Optimal transfer learning protocol vs commonly used vs no transfer learning. The final results of using the optimal transfer learning protocols we developed compared to those commonly used in previous research and no transfer learning

For smaller datasets:

- transferring the optimal number of layers and using the best hyperparameters has an increasing impact on accuracy, and
- the importance of transfer learning increases because training from scratch is no longer viable.

For our smallest sized dataset using the best transfer learning settings compared to those often adopted from [Yosinski et al., 2014] results in a 47.88% increase in accuracy. The settings used to produce the best accuracy for each target dataset size are shown in Table 7.2.

7.5 Discussion

To our knowledge we have performed the most thorough analysis of transfer learning using CNNs to date. Based on this research we identified optimal transfer learning protocols for target dataset sizes from very small to large, when source and target datasets and tasks are similar. Our results show:

1. Using the commonly adopted transfer learning practice of transferring all but the final classification layer and training for 30 epochs at a learning rate of 0.01

Table 7.2: Optimal transfer learning settings

	Training examples per class						
	10	**25**	**50**	**100**	**250**	**500**	**full**
layers to transfer	6	6	6	6	6	6	5
initial decay epochs	10	5	3	3	3	0	30
layers to freeze	6	5	5	1	1	0	0
average validation set accuracy	30.8	38.1	42.7	46.6	52.2	56.3	62.1

causes a significant increase in overfitting compared to using optimal transfer learning protocols, particularly as the size of the target dataset decreases.

2. Best practice transfer learning and fine-tuning protocols for smaller target datasets cannot be derived using the results of the commonly adopted protocols used in performing transfer learning with a large target dataset.

3. There is an interaction between the optimal number of layers to transfer and fine-tuning hyperparameters. If this is not taken into account, the optimal number of layers to transfer will be overestimated for small and medium sized target datasets and underestimated for large target datasets.

4. Freezing layers of pretrained weights after transferring them rarely results in the best results. It is only worth considering for very small target datasets.

5. Using our identified best practice transfer learning and fine-tuning protocols produces large changes in results compared to the existing protocols. This result again is particularly pronounced when applied to smaller datasets.

These new findings are likely to impact results of past and current research, which used commonly adopted protocols.

7.5.1 Final recommendations

The scope of the following recommendations is for source and target datasets that are very similar such as the subsets of ImageNet 1K used in these experiments. Our recommendations are:

1. only consider freeze layers when the target dataset is small (less than 50,000 training examples),

2. try reinitialising more than one layer in all cases, but reduce the learning rate as you increase the number of layers reinitialised,
3. decay the learning rate more quickly for smaller target datasets, but keep in mind that there are less batches per epoch so you may need to increase the number of epochs before decay while reducing the number of weight updates,
4. lower layers should be trained at a lower learning rate than higher layers for small to medium target datasets (less than 250,000 training examples).

7.6 Conclusion

We have shown that using our identified optimal protocols when performing transfer learning results in a significant increase in accuracy when compared to the commonly adopted protocols. This is particularly evident for small target dataset sizes where we observed a 47.88% increase in accuracy compared to using the more standard protocols. Best practice transfer learning protocols for small target datasets cannot be predicted from previous research that has analysed transfer learning under very specific conditions that are not universally applicable. We have presented optimal transfer learning protocols for various target dataset sizes, from very small to large, when source and target datasets and tasks are similar. We intend to extend our work to analyse more diverse datasets and tasks, along with looking at ways to predict optimal transfer learning protocols. This paper and our ongoing research are significant as it is likely to improve outcomes from past, current and future research in transfer learning.

This work shows that standard transfer learning practices aren't optimal when the target dataset size is reduced. The following chapters introduce a model for learning optimal non-binary transfer learning hyperparameters for a variety of target dataset sizes and test out best practice non-binary transfer learning hyperparameters on target datasets that transfer learning does not usually perform as well on.

Chapter 8

Learning to transfer learn: beyond binary transfer learning hyperparameters

8.1 Preamble

In this work I designed a neural architecture search style controller model to predict the best fine-tuning hyperparameters for a particular target dataset with various numbers of training examples. The hyperparameters predicted went beyond standard fine-tuning practice to include different learning and decay rates for different layers. I tested the model using two different datasets, ImageNet and Stanford Cars, and two different CNN child model architectures, AlexNet and Inception v4. The final model performed much better than standard practice transfer learning for an AlexNet child model and an ImageNet target task. The model designed to predict fine-tuning hyperparameters for an Inception version 4 model applied to the Stanford Cars dataset produced results comparable to but not above standard practice. I conclude that the model design shows good potential to be used for the purpose of predicting optimal non-binary transfer learning hyperparameters. Finally I give recommendations for improving performance when training it on more complex modern models such as Inception version 4. This work is my original ideas and implementation, with support from my supervisor to discuss and iterate ideas.

8.2 Introduction

Transfer learning has been shown to be improve performance in a wide variety of computer vision tasks, particularly when the source and target tasks are closely related and the target task is small [Ngiam et al., 2018; Mahajan et al., 2018; Cui et al.,

2018; Ge and Yu, 2017; Sabatelli et al., 2018; Ng et al., 2015; Li et al., 2018, 2019b; Wan et al., 2019]. In fact it become standard practice to pretrain on Imagenet 1K for many different tasks where the number of training examples in the target dataset are orders of magnitude fewer than Imagenet 1K [Li et al., 2018, 2019b; Wan et al., 2019; Masi et al., 2018; Li and Deng, 2020; Ng et al., 2015; Mazurowski et al., 2019; Mormont et al., 2018; Girshick et al., 2014].

The generally accepted transfer learning strategy is to transfer all layers apart from the final classification layer, then use a search strategy to find the best single initial learning rate and decay rate as well as other hyperparameters. This commonly used strategy originates from various works showing that performance on the target task increases as the number of layers transferred increases [Yosinski et al., 2014; Chu et al., 2016; Agrawal et al., 2014; Azizpour et al., 2015]. However it has recently been shown that this work does not take into account the relationship between the learning rate and the implicit regularisation provided by transferring more pretrained layers [Plested et al., 2021; Plested and Gedeon, 2019; Neyshabur et al., 2020]. When the number of layers transferred is greater a higher learning rate is more likely to be optimal as the implicit regularisation is higher and when the number of layers transferred is fewer a lower learning rate must be used in order to avoid overfitting.

This work attempts to discard all assumptions about how to transfer learn and allows a neural architecture search style reinforcement learning controller model [Zoph and Le, 2016] to decide on all the parameters for transfer learning, including number of layers to transfer and different learning rates for different layers. Our contributions are:

1. the first model that learns transfer learning hyperparameters,
2. results comparable to or above expert human performance in many areas, and
3. many insights about best practice for transfer learning and suggestions for change to current paradigms.

8.3 Related Work

Fine-tuning hyperparameter searches several studies include extensive searches over learning rate, weight decay and other fine-tuning hyperparameters [Kornblith et al., 2019; Mahajan et al., 2018; Li et al., 2020a, 2018]. Further studies have shown the combination of a lower learning rate and fewer layers transferred may be optimal for some datasets [Plested and Gedeon, 2019; Plested et al., 2021].

Adaptive fine-tuning hyperparameters Guo et al. [2019] make two copies of their ResNet models pretrained on ImageNet 1K. One model is used as a fixed feature selector with the pretrained layers frozen and the other model is fine-tuned. They reinitialize the final classification layer in both. A policy net trained with reinforcement learning is then used to create a mask to combine layers from each model together in a unique way for each target example. They show that SpotTune improves performance compared to fine-tuning with an equivalent size single model (double the size of the two individual SpotTune models) and achieves close to or better than state of the art in most cases. MultiTune [Wang et al., 2020a] simplifies SpotTune by removing the policy network and concatenating the features from each model prior to the final classification layer. It also improves on SpotTune by using two different non-binary fine-tuning [Plested et al., 2021] hyperparameter settings rather than one fine-tuned and one frozen model. The results show that MultiTune improves or equals accuracy compared to SpotTune in most cases tested with significantly less training time. AMF:Adaptable Weight Fusion [Shen et al., 2022] further improves on MultiTune by adding a policy network that learns the optimal combination of the final features of each model, rather than simply concatenating them.

Adaptive source dataset weights in [Zhu et al., 2020] the joint loss for the target dataset and weighted source dataset is optimised. Adaptive weights for the loss generated by each training example in the source dataset are learned via reinforcement learning.

8.4 Overall methodology

We use a neural architecture style controller, child architecture for each experiment similar to [Zoph and Le, 2016]. The design of the controller is specific to each experiment and is described in the relevant section.

8.5 Experiment 1 learning the best transfer learning hyperparameters - highly related datasets

This experiment expanded on previous work [Yosinski et al., 2014; Plested and Gedeon, 2019] splitting the 1,000 Imagenet 1K [Deng et al., 2009] classes into 500 for the source dataset and the remaining 500 for the target dataset. An AlexNet model [Krizhevsky et al., 2012] was pretrained on one set then transferred and fine-tuned on the second

set. The aim was to train one controller model for all target dataset sizes. The target dataset sizes were arranged in bins, with smaller sizes closer together:

$$[5, 10, 15, 25, 50, 75, 125, 250, 500, 1,000] * 1,000$$

A particular target dataset size input is defined as a normal distribution with mean being the size of the dataset given as a decimal bin number and standard deviation of 0.5. For example with a target dataset size of 20,000:

$$bin = 2 + (1 - (25,000 - 20,000)/(15,000 - 25,000)) = 2.5$$

$$dist = N(bin, 0.5)$$

then for each bin i:

$$input\,[i] = dist.cdf\,(i + 0.5) - dist.cdf(i - 0.5) \tag{8.1}$$

Encoding the dataset size input as a binned normal distribution allowed less training of more computationally expensive large target datasets. This is because the pattern of how optimal settings change as the target dataset increases in size can be learned through the differences between small to medium dataset sizes. Dataset sizes were selected in a log normal distribution centred around 25,000 ensuring far fewer target datasets with greater than 75,0000 training examples are trained.

The target dataset size input was initially encoded as a single float, being the ratio of the current target dataset size to the full dataset size (being 1,000 training examples per class). So with this encoding a target dataset size of 10 training examples per class would be encoded as 10/1,000 = 0.01. Experiments showed that this encoding did not achieve the aim of allowing one controller model to learn the fine-tuning hyperparameters for all target dataset sizes with limited examples of larger target dataset sizes. This is likely because the relationship between the target dataset size and the optimal fine-tuning hyperparameters is complex and not simply monotonically increasing or decreasing.

8.5.1 Parameters to be learned

For this experiment the transfer learning parameters to be learned were the number of layers to be reinitialized from 1-7, plus the initial learning rate, the initial number of epochs before decaying the learning rate by 0.1 and the subsequent number of epochs before decaying the learning rate by 0.1. All hyperparameters are shown in Table 8.1.

Table 8.1: AlexNet hyperparameters for each layer

Initial Learning Rate	First Decay Epochs	Ongoing Decay Epochs
0, 0.001, 0.002, 0.005, 0.01	3, 5, 10, 20	5, 10, 20, 30

8.5.2 Controller

A hierarchical Long Short Term Memory recurrent neural network [Hochreiter and Schmidhuber, 1997] controller was trained with reinforcement learning to predict the best transfer learning hyperparameters for a given target dataset size. The controller model is shown in figure 8.1 and the design is based [Zoph and Le, 2016]. The first LSTM layer predicts the number of layers to reinitialise then an embedded representation for each layer. The second LSTM layer then decodes the representation for each layer to the final prediction from each parameter category defined above.

For each set of predicted hyperparameters a child AlexNet model [Krizhevsky et al., 2012] was trained with those settings and the reward for the controller model was the final accuracy of the child model. We used the policy gradient method REINFORCE [Williams, 1992] to train the controller model.

8.5.3 Child network

An AlexNet model [Krizhevsky et al., 2012] for each set of hyperparameters is initialised with weights pretrained on the source dataset 500 classes. A number of layers are reinitialised to random weights as per the specifications for that set of hyperparameters and the model is trained with the learning rate, the number of epochs until the learning rate is initially decayed by 0.1 and the number of epochs before each subsequent decay for each layer, as defined by the set of hyperparameters. Training is stopped and the accuracy returned when the loss has not gone down for 4 epochs or a minimum loss is not reached after a certain number of epochs for a particular dataset size.

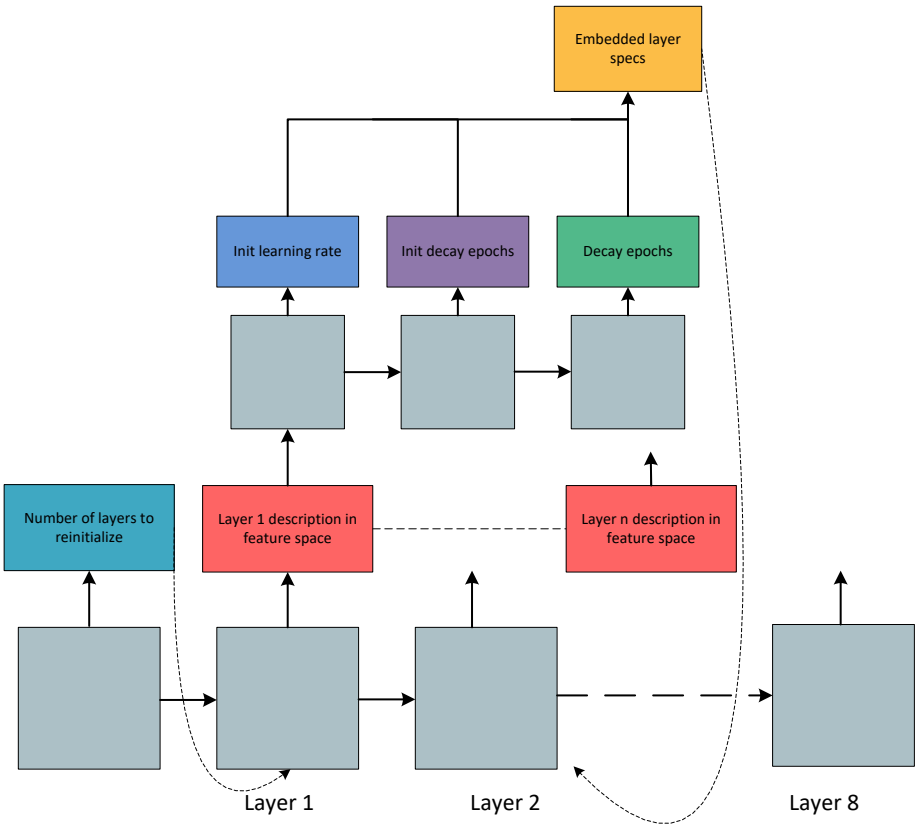


Figure 8.1: LSTM controller

8.5.4 Results

The results from the controller compared to the same model trained from random initialisation, standard transfer learning practice following [Yosinski et al., 2014], and best transfer learning practice following [Plested and Gedeon, 2019] are shown in figure 8.2. This shows that the best performance by the controller beats standard practice and random initialisation in all cases and best known practice in some cases.

Table 8.2 shows the hyperparameters for the top performing models. There are several interesting points about the top performing hyperparameters for each size:

1. All the top performing hyperparameters reinitialise two layers rather than the general practice of only reinitialising the final classification layer.

2. There are very few frozen layers lower (0 learning rates with no decay epochs)

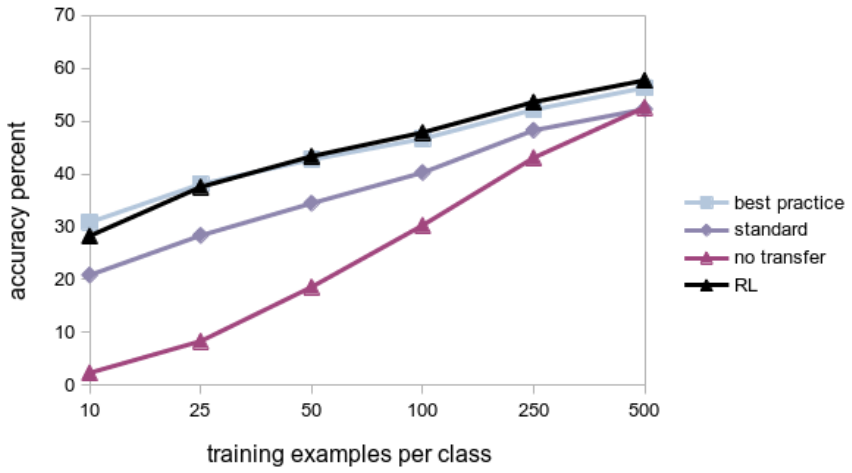


Figure 8.2: Controller vs no transfer, standard practice and best practice

and all but one of them are in the two smallest target dataset sizes. This coincides with previous work showing that freezing transferred layers is only optimal for very small target datasets [Plested and Gedeon, 2019].

3. Two of the classification layers are frozen, and apart from the smallest target dataset the other classification layers either start with a very low learning rate, decay the initial learning rate very quickly, or both. We found throughout training that there were many instances of a frozen classification layer performing as well as or better than a classification layer with a high initial learning rate. This goes against existing thinking that the classification layer should always be trained at a high learning rate as it is starting from randomly initialised weights. However this is more intuitive than it seems at first glance as when the target dataset is much smaller than the source dataset the transferred weights will likely be too big to change much from their pretrained values with the smaller number of updates per epoch when fine-tuning with the target dataset. This means that when the classification layer is initialised with much smaller random weights and trained for a large number of epochs with a high learning rate it is very likely to overfit the unsuitable features transferred from the source dataset.

Table 8.2: Top performing model hyperparameters

Training examples per class	Number of new layers	Classification layer	Other layers 7-1 [lr, first decay epochs, subsequent decay epochs]
10	2	[0.01, 10, 20]	[0.005, 3, 5],[0.002, 20, 10],[0.01, 5, 10], [0.005, 10, 30],[0.002, 5, 10],[0],[0.001, 10, 20]
25	2	[0]	[0.001, 20, 10],[0.01, 20, 10],[0.001, 5, 10], [0.005, 5, 10],[0.002, 5, 5],[0],[0]
50	2	[0.005, 5, 10]	[0.005, 10, 10],[0.005, 3, 10],[0.005, 5, 20],[0.01, 10, 10], [0.002, 5, 10],[0.002, 10, 10],[0.001, 20, 10]
100	2	[0.01, 3, 10]	[0.01, 10, 5],[0.01, 20, 20],[0.002, 10, 5],[0.005, 5, 10], [0.002, 20, 30].[0.01, 3, 20],[0.002, 3, 20]
250	2	[0.002, 3, 20]	[0.005, 5, 30],[0.01, 10, 10].[0.01, 20, 30],[0.01, 10, 30], [0.01, 10, 30],[0.002, 20, 5],[0]
500	2	[0]	[0.01, 5, 20],[0.01, 10, 10],[0.002, 20, 30],[0.002, 10, 30], [0.002, 20, 20],[0.01, 5, 10],[0.002, 5, 10]

8.5.5 Discussion

We have shown that a reinforcement learning controller model that learns to predict the optimal transfer learning hyperparameters for a given size of target dataset can significantly outperform using the standard practice settings. Some of the results specifically contradict generally accepted assumptions about best practice of transfer learning. These include reinitialising only the classification layer, freezing layers and training the classification layer at a high learning rate for a large number of epochs. When these assumptions are relaxed transfer learning performance improves significantly, especially for target datasets with fewer training examples.

We suggest future work should include further relaxing the assumption that the learning rate should always be decreased monotonically throughout training on the target dataset. It is possible that starting the classification layer learning rate low or at zero, then increasing it once the lower layer weights are fine-tuned on the target dataset may allow for better performance due to less overfitting.

8.6 Experiment 2 learning the best transfer learning hyperparameters - less related datasets

The aim of this experiment was to see if the algorithm designed in experiment one could be extended to a very deep modern CNN and a dataset that was different to Imagenet [Krizhevsky et al., 2012]. Inception v4 [Szegedy et al., 2017] was chosen as the model because it has been shown to have state of the art performance for transferring from Imagenet 1K to a broad range of target tasks [Kornblith et al., 2019]. Stanford cars [Krause et al., 2013] was chosen as it tends to be a more difficult target task to transfer to from Imagenet with performance often being similar to training from random initialisation [Kornblith et al., 2019]. Every possible size of target dataset was experimented with from a minimum of one training example per class to a maximum of around 41 training examples per class (not all classes have the same number of examples).

The controller model and parameters predicted were adjusted to account for the order of magnitude more layers in Inception v4 compared to AlexNet. The target dataset sizes were arranged in bins, with smaller sizes closer together as per experiment one, but the sizes were expressed in number of training examples per class with 41 being the maximum and the bins were arranged as follows:

$$[1, 5, 10, 20, 41]$$

The final input vector was calculated as per equation 8.1.

Similar to experiment one, fewer larger datasets were trained with the idea again being that the bins would allow the larger datasets to be partially learned from the smaller ones. The datasets for each epoch were picked with a two uniform distributions.

8.6.1 Parameters to be learned

The hyperparameters are shown in Table 8.3 Instead of predicting all layers individually we divided the 21 blocks of layers into the classification layer, higher layers and lower layers. Splitting the remaining layers after the classification layer into higher and lower is based on knowledge that features from lower layers are likely to be more general and from higher layers more task specific [Yosinski et al., 2014]. Because of this the hypothesis is that the model will perform better when some number of lower layers are trained at a lower learning rate and/or decayed more quickly than

the higher layers. However the parameter choice allows for all layers to be trained at the same learning rate by either selecting the number of layers at a high learning rate to the maximum, in order to select the higher learning rates for all, or selecting the same learning rate for all selections for lower learning rates.

The decay rate was also allowed to vary for this experiment and three of each were selected for decay epochs and decay rate, being the first, second and subsequent number of epochs before decay and amount to decay by.

The total number of hyperparameter combinations in the optimisation space was 612,360,000 for each target dataset size.

Table 8.3: Inception v4 Fine-Tuning Hyperparameters

Number of new layers	Number of high lr layers	classification layer lr	high lr	low lr	decay epochs (3 selected)	decay rate (3 selected)
1 - 5	1 - 21	0, 0.001, 0.005, 0.01, 0.05	0, 0.001, 0.005, 0.01, 0.05	0, 0.001, 0.002, 0.005, 0.01	5, 10, 15, 20, 30, 40	0.1, 0.2, 0.3, 0.5, 0.7

8.6.2 Controller

The controller is single layer LSTM. The first five hyperparameter selections correspond to the first five outputs of the LSTM through time. Each hyperparameter selection then has a unique decoder that produces the logits and the final distribution that is sampled to get each value. The selected values are then encoded and become the input for the next LSTM time step. The final two hyperparameters being the three lots of decay epochs and decay rates are selected in the same way, as the 6th to 11th timesteps of the LSTM, with three decay epochs selected then 3 decay rates selected.

Again we used the policy gradient method REINFORCE [Williams, 1992] to train the controller model. The accuracy returned by the child network trained with the specified settings is used as the reward as per [Zoph and Le, 2016]. After running short ablation studies the learning rate was set low at 0.0002 to ensure hyperparameter selections don't converge too early and more of the large and complex search space was explored.

8.6.3 Child network

An Inception v4 [Szegedy et al., 2017] for each set of hyperparameters is initialised with weights pretrained on Imagenet [Deng et al., 2009] taken from [Wightman, 2019]. The number of layers specified by the setting "Number of new layers" are reinitialised with small random weights. The weights for the classification layer, high layers and low layers are set as per the specified hyperparameters and training proceeds with the first, second and subsequent learning rate decays scheduled as per the specified hyperparameters. Training is stopped and the accuracy returned when the loss has not gone down for over 10 epochs or a minimum loss for a particular dataset size is not reached after a certain number of epochs.

8.6.4 Results

Figure 8.3 shows the accuracy result on the validation set for the best hyperparameters prediction for each number of examples per class in the training set. The smooth curve of the results shows that the model is able to predict effective transfer learning hyperparameters for every different dataset size. Despite having extremely limited training examples for datasets with number of training examples great than 20. This shows that the model has learned reasonably well.

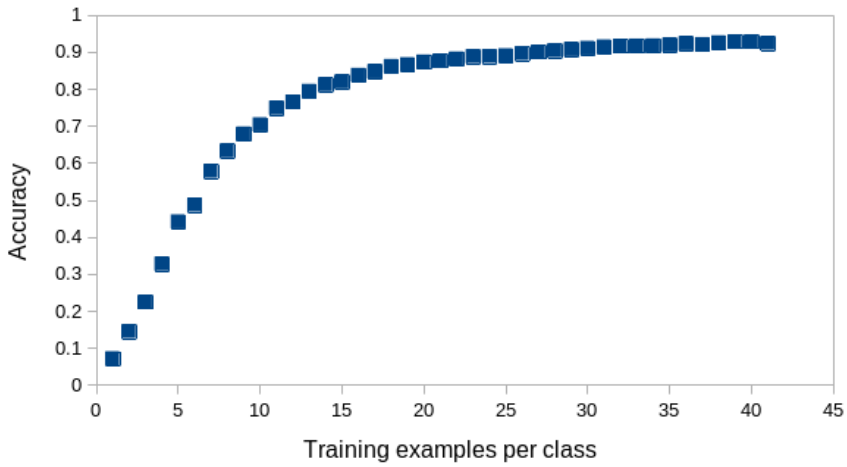


Figure 8.3: Inception v4 Stanford cars

8.6.4.1 Statistics on best hyperparameters

Table 8.4 shows the statistics of the best hyperparameters for each size. Also no 0 learning rate for the classification layer for sizes above 33 training examples per class.

Table 8.4: Statistics of best hyperparameters

Training examples per class	% of 1 layer reinitialised only	% of 0 learning rate for classification layer	% with classification layer learning rate lower than middle layers	% of frozen lower layers
1-10	50%	40%	90%	20%
11-20	20%	40%	100%	0
21-41	33%	19%	76%	0
all	34%	29%	85%	5%

Table 8.5: Predicted hyperparameters

Training examples per class	New layers	High lr layers	Classification layer lr	High lr	Low lr	Decay epochs	Decay rates
1-2	1	7	0.01	0.05	0.005	[10, 10, 10]	[0.1, 0.1, 0.1]
3-5	1	15	0.01	0.05	0.005	[10, 10, 10]	[0.1, 0.1, 0.1]
6-7	4	15	0.01	0.05	0.01	[10, 10, 10]	[0.1, 0.1, 0.1]
8-12	4	7	0.01	0.05	0.01	[10, 10, 10]	[0.1, 0.1, 0.1]
13-18	4	7	0.01	0.05	0.005	[10, 10, 10]	[0.1, 0.1, 0.1]
19-33	1	7	0.01	0.05	0.005	[10, 10, 10]	[0.1, 0.1, 0.1]
34-41	1	7	0.01	0.05	0.01	[10, 10, 10]	[0.1, 0.1, 0.1]

It is not possible to find best practice transfer learning settings among the 612,360,000 hyperparameter combinations for each target dataset size. So for the final performance comparisons a range of target dataset sizes were chosen, being [2, 4, 6, 8, 14, 27, 41]. The following experiments were performed:

1. The best predicted hyperparameters from the model were fully trained.
2. The final converged predicted hyperparameters from the model were fully trained.
3. The best traditional transfer learning hyperparameters were chosen. This meant the number of layers reinitialised was set to 1, only one learning rate was chosen for all layers and only one each of decay rate and number of epochs until decay were chosen.

The final results for these experiments are shown in Table 8.6. The final predictions from the model are comparable to standard best practice, however the final test results for the hyperparameters chosen from the best results on the validation set are not. This is likely because using non-standard fine-tuning hyperparameters such as reinitialising more layers than just the final classification layer produces a lot more variation in the performance of the model based on where the new weights are reinitialised. For this reason best hyperparameters are selected based on outliers in performance.

It seems that final predictions could be improved with a better training strategy. Particularly since it is known that there are fine-tuning hyperparameters that improve on standard performance for the Stanford Cars dataset [Plested et al., 2021]. We suggest using suggestions from [Andrychowicz et al., 2020] to improve the performance of the controller model, particularly their recommendation to "initialize the last policy layer with 100× smaller weights". This helps to insure the initial predictions have zero mean and small standard deviation so there is no bias in initial predictions and thus early learning.

Another element that likely effected the convergence of the model was the early stopping policy used in the training the child models. Both criteria favour models that converge early but are not necessarily optimal overall. An effect of this is shown in that all decay epochs are low in the final predicted models. Final hyperparameter predictions are also extremely similar between target dataset sizes even though it is known that the size of the target dataset has a significant impact on best fine-tuning practice [Plested and Gedeon, 2019]. This is an indication that the model has converged too early without learning the full complexity of the optimisation space.

Table 8.6: Final Results Inception v4 Stanford Cars

Training examples per class	Best prediction	Final prediction	Standard
2	18.65%	17.32%	19.05%
4	34.25%	41.14%	45.70%
8	67.60%	73.28%	75.20%
14	83.82%	86.38%	86.17%
27	88.81%	91.64%	92.23%
41	91.57%	93.96%	94.22%

8.7 Conclusion

We designed a neural architecture search style controller model to predict the best fine-tuning hyperparameters with various numbers of training examples for a particular target dataset. The hyperparameters predicted went beyond standard fine-tuning practice to include different learning and decay rates for different layers. We tested the model using two different datasets, ImageNet and Stanford Cars, and two different CNN child model architectures, AlexNet and Inception v4. We also used a broad range of different target dataset sizes for each target dataset. For the Imagenet target task with the AlexNet CNN child model the results were significantly above those achieved through standard fine-tuning practice. For the Stanford Cars target task with the Inception v4 CNN child model the final predicted hyperparameters were comparable to but did not improve on standard fine-tuning hyperparameters. We believe this second result was because:

1. the model was allowed to converge too early in this extremely complex optimisation space and we make recommendations as to how to avoid this in future, and

2. the early stopping mechanisms used to allow for faster training of the child models favoured hyperparameters which converged quickly but were not necessarily optimal.

Our model has shown its ability to learn hyperparameters that improve on standard practice for the simpler optimisation space of the smaller AlexNet model. As it is known that there are fine-tuning hyperparameters that improve on standard performance for the Stanford Cars dataset we expect that our recommendations to improve training will allow our model to learn these superior hyperparameters for the more complex Inception v4 optimisation space too. We also recommend future experiments using our model to predict non-standard fine-tuning hyperparameters for target datasets where it is known that using more optimal settings can produce a more significant improvement in results [Plested et al., 2021], such as FGVC Aircraft [Maji et al., 2013] and Describable Textures [Cimpoi et al., 2014].

This work and the previous chapter include a number of discoveries about best practice for non-binary transfer learning. In the following work I applied best practice to a modern deep CNN and a variety of commonly used small target image classification datasets to show that using best practice improves results for target datasets that are more difficult to transfer to for various reasons.

Chapter 9

Rethinking binary hyperparameters for deep transfer learning for image classification

9.1 Preamble

The current standard for a variety of computer vision tasks using smaller numbers of labelled training examples is to fine-tune from weights pretrained on a large image classification dataset such as ImageNet. The application of transfer learning and transfer learning methods tends to be rigidly binary. A model is either pretrained or not pretrained. Pretraining a model either increases performance or decreases it, the latter being defined as negative transfer. Application of L2-SP regularisation that decays the weights towards their pretrained values is either applied or all weights are decayed towards 0. In this chapter I re-examine these assumptions. My recommendations are based on extensive empirical evaluation that demonstrate the application of a non-binary approach to achieve optimal results. (1) Achieving best performance on each individual dataset requires careful adjustment of various transfer learning hyperparameters not usually considered, including number of layers to transfer, different learning rates for different layers and different combinations of L2SP and L2 regularization. (2) Best practice can be achieved using a number of measures of how well the pretrained weights fit the target dataset to guide optimal hyperparameters. I include methods for non-binary transfer learning including combining L2SP and L2 regularization and performing non-traditional fine-tuning hyperparameter searches. Finally I suggest heuristics for determining the optimal transfer learning hyperparameters. The benefits of using a non-binary approach are supported by final results that come close to or exceed state of the art performance on a variety of tasks that have traditionally been more difficult for transfer learning. This work is my original

ideas and implementation, with support from my supervisor to discuss and iterate ideas. A few final experiments and cleaning up of code was performed by one of my Masters students Xuyang Shen.

9.2 Introduction

Convolutional neural networks (CNNs) have achieved many successes in image classification in recent years [Krizhevsky et al., 2012; Girshick et al., 2014; Li and Deng, 2020; Masi et al., 2018; Mazurowski et al., 2019]. It has been consistently demonstrated that CNNs work best when there is abundant labelled data available for the task and very deep models can be trained [Ngiam et al., 2018; Mahajan et al., 2018; Kolesnikov et al., 2019]. However, there are many real world scenarios where the large amounts of training data required to obtain the best performance cannot be met or are prohibitively expensive. Transfer learning has been shown to improve performance in a wide variety of computer vision tasks, particularly when the source and target tasks are closely related and the target task is small [Ngiam et al., 2018; Mahajan et al., 2018; Cui et al., 2018; Ge and Yu, 2017; Sabatelli et al., 2018; Ng et al., 2015; Li et al., 2018, 2019b; Wan et al., 2019]. It has become standard practice to pretrain on ImageNet 1K for many different tasks where the available labeled datasets are orders of magnitude smaller than ImageNet 1K [Li et al., 2018, 2019b; Wan et al., 2019; Masi et al., 2018; Li and Deng, 2020; Ng et al., 2015; Mazurowski et al., 2019; Mormont et al., 2018; Girshick et al., 2014]. Several papers published in recent years have questioned this established paradigm. They have shown that when the target dataset is very different from ImageNet 1K and a reasonable amount of data is available, training from scratch can match or even out perform pretrained and fine-tuned models [He et al., 2018; Shen et al., 2017b,a; Zhu et al., 2019b; Mahajan et al., 2018].

The standard transfer learning strategy is to transfer all layers apart from the final classification layer, and either use a single initial learning rate and other hyperparameters for fine-tuning all layers, or freeze some layers. Given that lower layers in a deep neural network are known to be more general and higher layers more specialised [Yosinski et al., 2014], we argue that the binary way of approaching transfer learning and fine-tuning is counter intuitive. There is no intuitive reason to think that all layers should be treated the same when fine-tuning, or that pretrained weights from all layers will be applicable to the new task. If transferring all layers results in negative transfer, could transferring some number of lower more general layers improve performance? If using an L2SP weight decay on all transferred layers for regularisation decreases performance over decaying towards 0, might applying

the L2SP regularisation to some number of lower layers that are more applicable to the target dataset result in improved performance?

We performed extensive experiments across four different datasets to:

- re-examine the assumptions that transfer learning hyperparameters should be binary, and
- find the optimal settings for number of layers to transfer, initial learning rates for different layers, and number of layers to apply L2SP regularisation vs decaying towards 0.

We developed methods for non-binary transfer learning including combining L2SP and L2 regularization and performing non-traditional fine-tuning hyperparameter searches. We show that the optimal settings result in significantly better performance than binary settings for all datasets except the most closely related. Finally we suggest heuristics for determining the optimal transfer learning hyperparameters.

9.3 Related work

The standard transfer learning strategy is to transfer all layers apart from the final classification layer, then use a search strategy to find the best single initial learning rate and other hyperparameters. Several studies include extensive hyperparameter searches over learning rate and weight decay [Kornblith et al., 2019; Mahajan et al., 2018], momentum [Li et al., 2020a], and L2SP [Li et al., 2018]. This commonly used strategy originates from various works showing that performance on the target task increases as the number of layers transferred increases [Yosinski et al., 2014; Chu et al., 2016; Agrawal et al., 2014; Azizpour et al., 2015]. All these works were completed prior to advances in residual networks [He et al., 2016], and searches for optimal combinations of learning rates and number of layers to transfer were not performed. Additionally, in two of the works layers that were not transferred were discarded completely rather than reinitialising and training them from scratch. This resulted in smaller models with less layers transferred and a strong bias towards better results when more layers were transferred [Agrawal et al., 2014; Azizpour et al., 2015]. Further studies have shown the combination of a lower learning rate and fewer layers transferred may be optimal [Plested and Gedeon, 2019]. However, again modern residual networks were not used and only very similar source and target datasets were selected.

It has been shown that the similarity between source and target datasets has a strong impact on performance for transfer learning:

1. More closely related datasets can be better than more source data for pretraining [Ngiam et al., 2018; Mahajan et al., 2018; Cui et al., 2018; Ge and Yu, 2017; Sabatelli et al., 2018].
2. A multi-step pretraining process where the interim dataset is smaller and more closely related to the target dataset can outperform a single step pretraining process when originating from a very different, large source dataset [Ng et al., 2015].
3. Self-supervised pretraining on a closely related source dataset can be better than supervised training on a less closely related dataset [Zoph et al., 2020].
4. L2SP regularization, where the weights are decayed towards their pretrained values rather than 0 during fine-tuning, improves performance when the source and target dataset are closely related, but hinders it when they are less related [Li et al., 2018, 2019b; Wan et al., 2019]
5. Momentum should be lower for more closely related source and target datasets [Li et al., 2020a].

These five factors demonstrate the importance of the relationship between the source and target dataset in transfer learning.

9.4 Methodology

9.4.1 Datasets

We performed evaluations on four different small target datasets, each being less than 10,000 training images. We chose one that is very similar to ImageNet 1K that transfer learning and sub methods have traditionally performed best on. This is used as a baseline for comparison to show that the default methods that perform badly on the other datasets chosen do perform well on this one. For each of the other target datasets it has been shown that traditional transfer learning strategies:

- do not perform well on them, and/or
- they are very different to the source dataset used for pretraining.

We used the standard available train, validation and test splits for the three datasets for which they were available. For Caltech256-30 we used the first 30 items from each class for the train split, the next 20 for the validation split and the remainder for the test split.

9.4.1.1 Source dataset

We used ImageNet 1K as the source dataset as it is the most commonly used source dataset and therefore the most suitable to demonstrate our approach.

9.4.1.2 Target datasets

These are the final datasets that we transferred the models to and measured performance on. We used a standard 299×299 image size for all datasets.

Most similar to ImageNet 1K We chose Caltech 256-30 (Caltech) [Griffin et al., 2007] as the most similar to ImageNet. It contains 256 different general subordinate and superordinate classes. As our focus is on small target datasets, we chose the smallest commonly used split with 30 examples per class.

Fine-grained Fine-grained object classification datasets contain subordinate classes from one particular superordinate class. We chose two where standard transfer learning has performed badly [Guo et al., 2019; Kornblith et al., 2019].

- Stanford Cars (Cars): Contains 196 different makes and models of cars with 8,144 training examples and 8,041 test examples [Krause et al., 2013].
- FGVC Aircraft (Aircraft): Contains 100 different makes and models of aircraft with 6,667 training examples and 3,333 test examples [Maji et al., 2013].

Very different to ImageNet 1K We evaluated performance on another dataset that is intuitively very different to ImageNet 1K as it consists of images depicting adjectives instead of nouns.

- Describable Textures (DTD) [Cimpoi et al., 2014]: consists of 3,760 training examples of texture images jointly annotated with 47 attributes. The texture images are collected "in the wild" by searching for texture adjectives on the web.

9.4.2 Model

Inception v4 [Szegedy et al., 2017] was selected for evaluation as it has been shown to have state of the art performance for transferring from ImageNet 1K to a broad range of target tasks [Kornblith et al., 2019]. Our code is adapted from [Wightman, 2019] and we used the publicly available pretrained Inception v4 model. Our code is

available at https://anonymous.4open.science/r/Non-binary-deep-transfer-learning-for-image-classification-872D. We did not experiment with pretraining settings, for example removing regularization settings as suggested by [Kornblith et al., 2019], as it was beyond the scope of this work and the capacity of our compute resources.

9.4.3 Evaluation metric

We used top-1 accuracy for all results for easiest comparison with existing results on the chosen datasets. Graphs showing ablation studies with different hyperparameters show results on one run on the validation set. Final reported results are averages and standard deviations over four runs on the test set. We used the standard training and test sets for all datasets and the standard validation set where available. For the Cars dataset where no standard validation set was available we selected the first 10 examples from each class in the training set to use as a validation set. For all experiments we used single crop and for our final comparison to state of the art for each dataset we used an ensemble of all four runs and 10-crop.

9.4.4 Compute resources

The experiments were run on compute nodes containing four Nvidia V100 GPUs and two 24-core Intel Xeon Scalable 'Cascade Lake' processors.

Each individual experiment in the ablation studies ran on one GPU and 12 cores from one processor. Running time varied between approximately six hours for DTD to one day for Caltech.

Each final experiment was run on a full compute node with four GPUs and two 24-core processors. Running time varied between approximately one day for DTD and Aircraft to two days for Caltech.

9.5 Results

9.5.1 Assessing the suitability of the fixed features

We first examined performance using the pretrained Inception v4 model as a fixed feature extractor for comparison and insight into the suitability of the pretrained features. We trained a simple fully connected neural network classification layer for each dataset and compared it to training the full model from random initialization. The comparisons for all datasets are shown in Table 9.1.

The class normalized Fisher score on target datasets using the pretrained but not fine-tuned weights, was used as a measure of how well the pretrained weights separate the classes in the target dataset [Fukenaga, 1990]. A larger normalized Fisher score shows more separation between classes in the feature space.

$$F(\mathbf{W}) = Tr\left\{s_w^{-1} s_B\right\} / N_c \tag{9.1}$$

where s_w^{-1} is the inverse of the within class covariance, s_B is the between class covariance and N_c is the number of classes for the target dataset being measured. See [Fukenaga, 1990] for further details.

We also calculated domain similarity between the fixed features of the source and target domain using the earth mover distance (EMD) [Rubner et al., 2000; Peleg et al., 1989] and applying the procedure defined in [Cui et al., 2018].

The domain similarity calculated using the EMD has been shown to correlate well with the improvement in performance on the target task from pretraining with a particular source dataset [Cui et al., 2018].

Table 9.1: Domain measures

	Trained from random initialization	Fixed features classification	EMD domain similarity	Normalised Fisher score	State of the art
Caltech	67.2	83.4	0.568	1.35	86.6 [Li et al., 2019b]
Cars	92.7	64.2	0.536	1.18	96.0 (@448) [Ridnik et al., 2020]
Aircraft	88.8	59.9	0.557	0.83	93.9 (@448) [Zhuang et al., 2020b]
DTD	66.8	74.6	0.540	3.47	78.9 [Chen et al., 2020]

9.5.2 Default settings

For the first experiment we transferred all but the final classification layer and trained all layers at the same learning rate as per standard transfer learning practice. We performed a search using the validation set to find the optimal single fine-tuning learning rate for each dataset. The results in Figure 9.1 show the accuracy for each

learning rate for each dataset.

The optimal single learning rate for Caltech, the most similar dataset to ImageNet, is an order of magnitude lower than the optimal learning rate for the fine-grained classification tasks Stanford Cars and Aircraft. This shows that the optimal weights for Caltech are much closer to the pretrained values. The surprising result was that the optimal learning rate for DTD was very similar to Caltech and also an order of magnitude lower than Stanford Cars and Aircraft. Final results on the test set for each dataset are shown in Table 9.2.

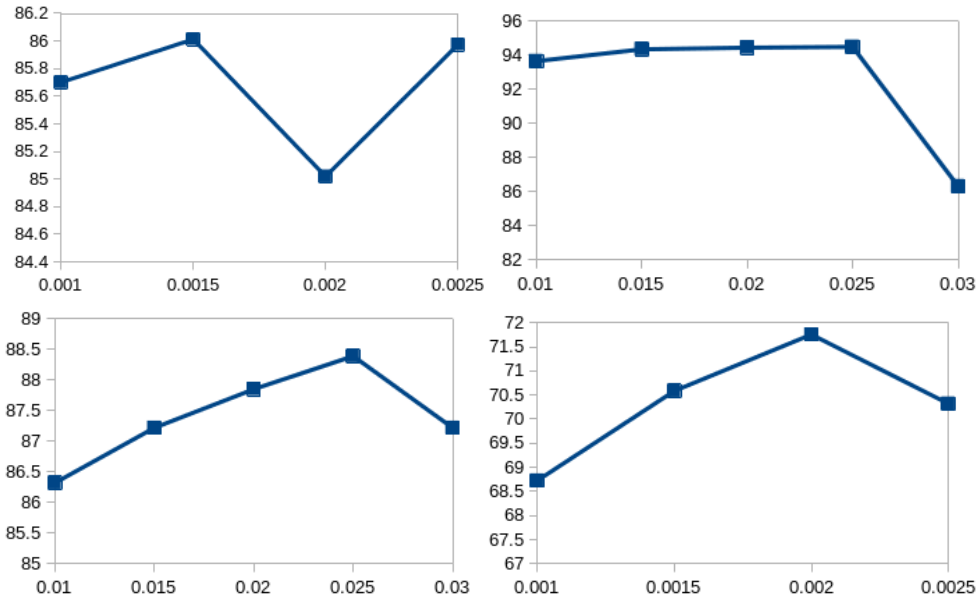


Figure 9.1: Optimal learning rates for each dataset. Left to right Caltech, Cars, Aircraft, DTD

Table 9.2: Final results default settings

	Caltech	Stanford Cars	Aircraft	DTD
Learning rate	0.0025	0.025	0.025	0.002
Accuracy	83.69±0.0784	94.59±0.110	93.78±0.137	77.30 ±0.726

9.5.3 Optimal learning rate decreases as more layers are reinitialized and transferring all possible layers often reduces performance on the target task

To the best of our knowledge the combination of learning rate and number of layers reinitialized when performing transfer learning has not been examined in modern residual models. To examine the relationship between learning rate and number of layers reinitialized, we searched for the optimal learning rate for each of 1-3 blocks of layers reinitialized across each of the four datasets. One layer is the default with the final classification layer only being reinitialized. Two and three involve reinitializing an additional one or two Inception C blocks of layers respectively as well as the final layer. For consistency we refer to both the final classification layer and the Inception C blocks as layers. Figure 9.2 shows the performance of fine-tuning with various combinations of learning rate and layers reinitialized. The optimal learning rate when reinitializing more than one layer is always lower than when reinitializing only the final classification layer. Also the optimal number of layers to reinitialize is more than one for all datasets except Cars. Table 9.3 shows the final results for the optimal learning rate and accuracy for each number of layers reinitialized.

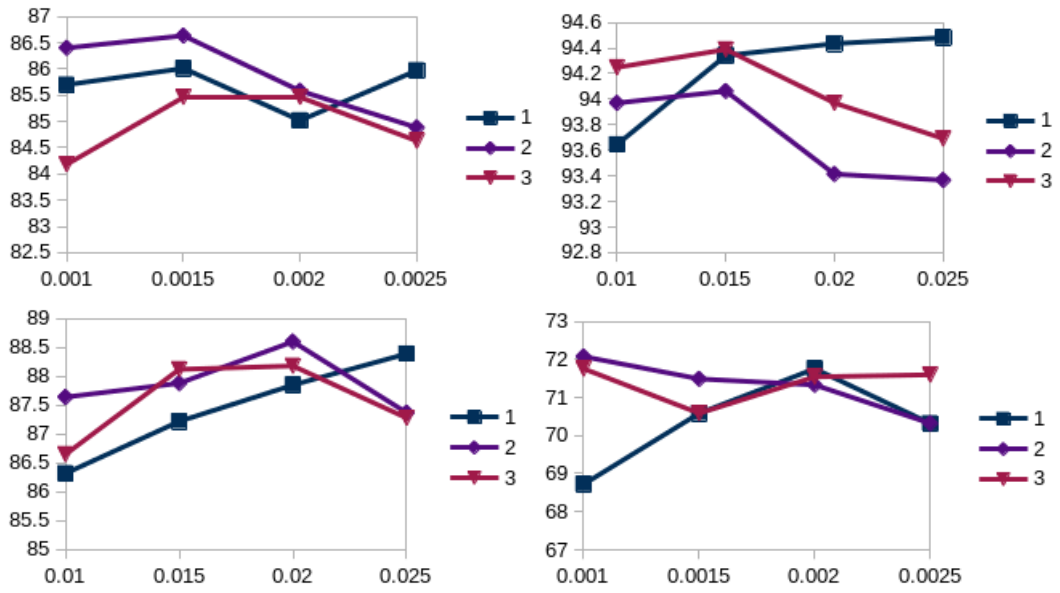


Figure 9.2: Number of layers reinitialized versus learning rate. Left to right, top to bottom Caltech, Cars, Aircraft, DTD

Table 9.3: Final optimal learning rate for each number of layers reinitialized

	Caltech	Cars	Aircraft	DTD
1 layer lr	0.0025	0.025	0.025	0.002
Accuracy	83.69±0.078	94.59±0.110	93.78±0.137	77.30 ±0.726
2 layers lr	0.015	0.015	0.02	0.001
Accuracy	84.31±0.114	94.59± 0.205	93.56±0.942	78.92 ±0.191
3 layers lr	0.015	0.015	0.02	0.0015
Accuracy	83.57±0.104	94.46±0.348	92.96±1.588	78.90±0.243

Reinitializing more layers has a more significant effect when the optimal learning rate for reinitializing just one layer is higher

Caltech and DTD have an order of magnitude lower optimal learning rates than Cars and Aircraft, but reinitializing more layers for the former results in a significant increase in accuracy whereas there is none for the latter. This result initially seems counter intuitive as Cars and Aircraft are less similar to ImageNet than Caltech and DTD. However, a lower learning rate tempers the ability of the upper layers of the model to specialise to the new domain and even very closely related source and target domains have different final classes and thus optimal feature spaces. The combination of a lower learning rate and reinitializing more layers likely allows the models to keep the closely related lower and middle layers and create new specialist upper layers.

9.5.4 Lower learning rates for lower layers works better when optimal learning rate is higher

Conventionally learning rates for different layers are set so that either all layers are fine-tuned with the same learning rate or some are frozen. The thinking is that as lower layers are more general and higher layers more task specific [Yosinski et al., 2014] the lower layers do not need to be fine-tuned. Recent work has shown that fine-tuning tends to work better in most cases [Plested and Gedeon, 2019]. However, setting different learning rates for different layers is not generally considered. We examined the effects of applying lower learning rates to lower layers that are likely to generalise better to the target task. Figure 9.3 shows that for the Stanford Cars and Aircraft where the optimal initial learning rate is higher, setting lower learning rates for lower layers significantly improves performance whereas for Caltech and DTD it does not.

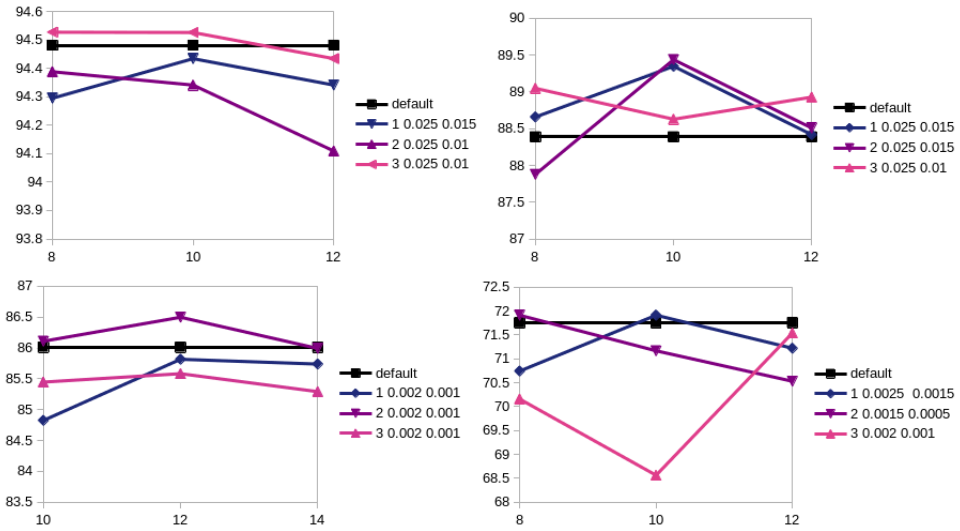


Figure 9.3: Lower learning rates for lower layers. Left to right Caltech, Cars, Aircraft, DTD. Legend shows number of layers reinitialized, high learning rate, low learning rate. X axis is number layers trained starting with the low learning rate

9.5.5 L2SP

The L2SP regularizer decays pretrained weights towards their pretrained values rather than towards zero during fine-tuning. In the standard transfer learning paradigm with only the final classification reinitialized, when the L2SP regularizer is applied the final classification layer only is decayed towards 0.

The values α and β are tunable hyperparameters to control the amount of regularization applied to the pretrained and randomly initialized layers respectively.

The original experiments showing the effectiveness of the L2SP regularizer [Li et al., 2018] were done on target datasets that are extremely similar to the source datasets used. They showed that a high level of regularization decaying towards the pretrained weights is beneficial on these datasets. It has since been shown that the L2SP regularizer can result in minimal improvement or even worse performance when the source and target datasets are less related [Li et al., 2020a; Wan et al., 2019].

Our results shown in Figure 9.4 align with the original paper [Li et al., 2018] for the dataset Caltech showing that standard L2SP regularization does result in an improvement in performance over L2 regularization. Our results also align with [Li et al., 2020a; Wan et al., 2019] in showing that for the datasets we have chosen to be different from ImageNet, and known to be more difficult to transfer to, L2SP regularization performs worse than L2 regularization. In general the lower the setting

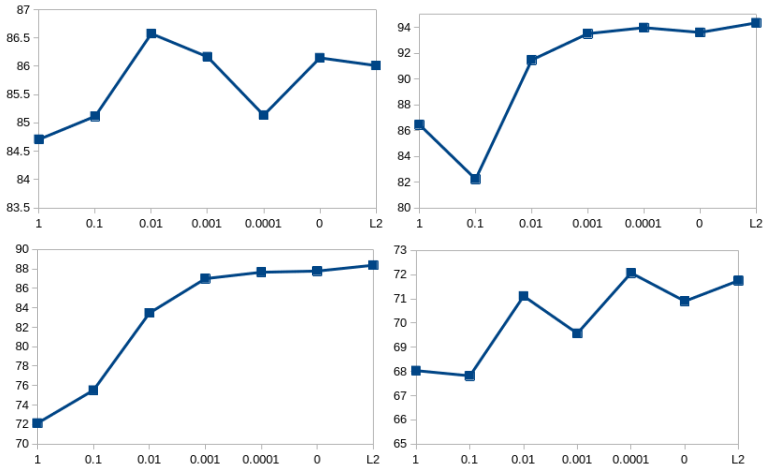


Figure 9.4: L2SP with default settings. Left to right, top to bottom, Caltech, Cars, Aircraft, DTD

for alpha (the L2SP regularization hyperparameter) the better the performance.

We relaxed the binary assumption that L2SP regularization must be applied to all pretrained layers. We used L2SP regularization for lower layers that we expected to be more similar to the source dataset and L2 regularization for upper layers to allow them to specialise to the target dataset. We searched for the optimal combinations of L2SP and L2 weight regularization along with the α and β hyperparameters for each dataset. We show the best settings for number of layers and amount of L2SP regularization in Table 9.5 and graph the optimal settings compared to the best default settings in Figure 9.5.

We make the following observations:

1. A combination of L2SP and L2 regularization is optimal for most settings of the L2SP regularization hyperparameter (α) for all datasets except Caltech.

2. When more layers are trained with L2 rather than L2SP regularization the optimal L2 regularization hyperparameter (β) is lower as the squared sum of the weights in these layers will be larger for the same model.

Further experiments were performed to search for the combination of optimal L2SP vs L2 regularization settings with optimal number of layers transferred and different learning rates for different layers. These results are also shown in Table 9.5.

Table 9.4: Optimal learning rates for each number of layers reinitialized with different learning rates for lower layers. Learning rates are in the format (high layers learning rate, low layers learning rate, number of low layers)

	Caltech	Cars	Aircraft	DTD
Default accuracy	83.69±0.0784	94.59±0.110	93.78±0.137	77.30±0.726
1 layer lrs	0.002 0.001 12	0.025 0.01 8	0.025 0.015 10	0.0025 0.0015 10
Accuracy	83.95±0.196	94.86±0.0460	94.32 ±0.144	77.753±0.456
2 layers lrs	0.002 0.001 12	0.025 0.01 8	0.025 0.015 10	0.0015 0.001 14
Accuracy	84.15±0.553	94.78 ±0.132	94.17 ±0.311	78.59±0.127
3 layers lrs	0.002 0.001 12	0.025 0.01 10	0.01 0.025 0.01 8	0.0015 0.001 12
Accuracy	83.46±0.143	94.83 ±0.0832.	94.50 ±0.192	78.84 ±0.464

9.5.6 Final optimal settings and how to predict them

The final results and optimal settings are shown in Table 9.6 and Figures 9.6 to 9.9. Table 9.7 shows a clear distinction between target datasets that are more similar to the source dataset and for which the pretrained weights are better able to separate the classes in feature space and those that are less similar with pretrained weights that fit the target task poorly. The best measure for determining whether the pretrained weights are well suited is a comparison of the performance achieved with fixed pretrained features and that achieved through training from random initialization. As this method is computationally intensive a reasonable alternative may be the normalized Fisher Ratio, but as the differences are not as pronounced in all cases it should be further investigated on more datasets to see how reliable it is as a heuristic. The EMD measure of domain similarity is a poor predictor of the suitability of the pretrained weights.

Targets datasets for which the pretrained weights are well suited need:

- a much lower learning rate for fine-tuning,
- more than one layer to be reinitialized from random weights, and
- some or all layers trained with L2SP regularization.

Target datasets where the pretrained weights are not well suited need:

Table 9.5: Best default and optimal L2SP settings and results

	Caltech	Cars	Aircraft	DTD
L2	83.694	94.59	93.78	77.30
Default L2SP 1 new layer α, β	0.01 0.01	0.0001 0.01	0.0001 0.01	0.0001 0.01
result	84.52	94.42	93.29	78.32
Optimal L2SP 1 new layer α, β	0.01 0.01 L2SP layers all	0.01 0.001 L2SP layers 10	0.0001 0.001 L2SP layers 10	0.001 0.001 L2SP layers 10
result	84.52	94.57	93.86	78.32
Optimal L2SP overall α, β	0.01 0.01 L2SP layers all	0.01 0.001 L2SP layers 10	0.0001 0.001 L2SP layers 10	0.001 0.001 1.0 L2SP layers 14
result	84.52	94.65	94.22	78.90

- a much higher learning rate for fine-tuning with a lower learning rate for lower layers,
- only the final classification layer reinitialized, and
- L2 rather than L2SP regularization.

Using the above best practice, non-binary transfer learning procedures we achieved state of the art or close to, on three out of the four datasets. We used publicly available pretrained weights and no additional methods for either pretraining or fine-tuning.

9.6 Discussion

Traditional binary assumptions about transfer learning hyperparameters should be discarded in favour of a tailored approach to each individual dataset. These assumptions include transferring all possible layers or none, training all layers at the same learning rate or freezing some, and using L2SP regularization or L2 regularization for all layers. Our work demonstrates that optimal transfer learning non-binary hyperparameters are dataset dependent and strongly influenced by how well the pretrained weights fit the target task. For a particular dataset, optimal non-binary transfer learning hyperparameters can be determined based on the difference between model performance when fixed features are used and when the full model is trained from random initialization as shown in Table 9.7. We recommend using the settings shown on the left in this Table for target datasets where the difference

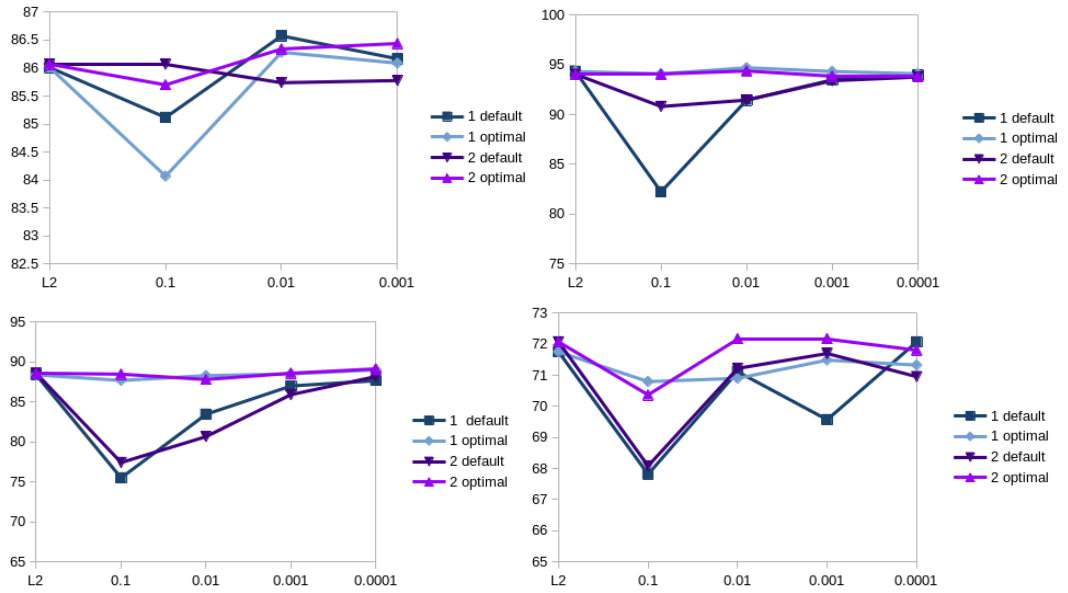


Figure 9.5: Best default and optimal L2SP settings. Left to right Caltech, Cars, Aircraft, DTD

Table 9.6: Final optimal settings versus best default settings. P-values are comparing the optimal result to the best of the default or default L2SP results, with the null hypothesis being that there is no difference. The results for Cars and Aircraft are significant at $p < .05$

	Caltech	Cars	Aircraft	DTD
Default settings	0.0025	0.025	0.025	0.002
Default result	83.69	94.59	93.78	77.30
Default L2SP	84.52	94.42	93.29	78.32
Optimal settings	1 new layer lr 0.0025 L2SP 0.01 0.01	1 new layer high lr 0.025 low lr 0.01 low layers 8 no L2SP	3 new layers FC lr 0.01 high lr 0.025 low lr 0.01 low layers 8 no L2SP	new layers 2 lr 0.002 L2SP 0.001 0.001 L2SP layers 14
Optimal result	84.52	94.86	94.50	78.90
P-value	0	0.0039	0.00088	0.052
Ensemble	85.94	95.35	95.11	79.79
State of the art	86.6 [Li et al., 2019b]	96.0 (@448) [Ridnik et al., 2020]	93.9 (@448) [Zhuang et al., 2020b]	78.9 [Chen et al., 2020]

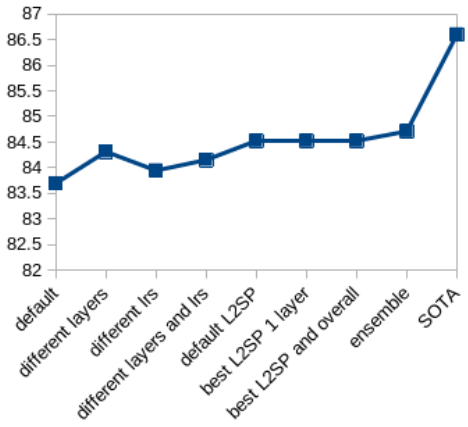


Figure 9.6: Optimal settings versus best default settings Caltech

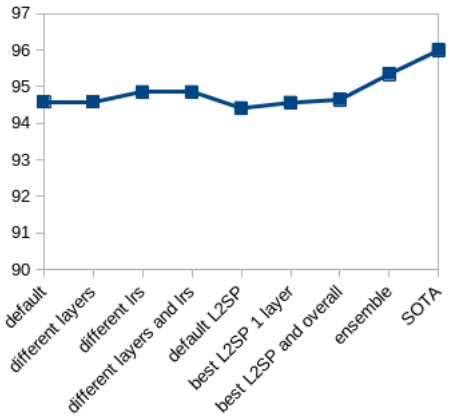


Figure 9.7: Optimal settings versus best default settings Cars

is negative and settings shown on the right for positive differences. Target datasets for which the pretrained weights are well suited and target datasets for which they are not, result in large differences in this value. These differences should still be pronounced even if suboptimal learning hyperparameters are used for this initial test due to limited resources for hyperparameter search. This heuristic for determining optimal hyperparameters should be useful in most transfer learning for image classification cases. The normalized Fisher Ratio may be useful in some cases, however, care should be taken because the differences are not as pronounced. The EMD domain similarity measure should not be used to determine transfer learning hyperparameters.

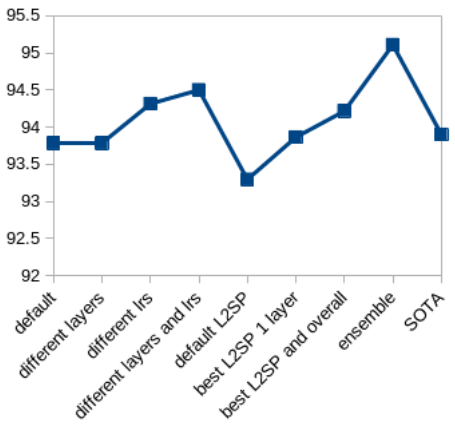


Figure 9.8: Optimal settings versus best default settings Aircraft

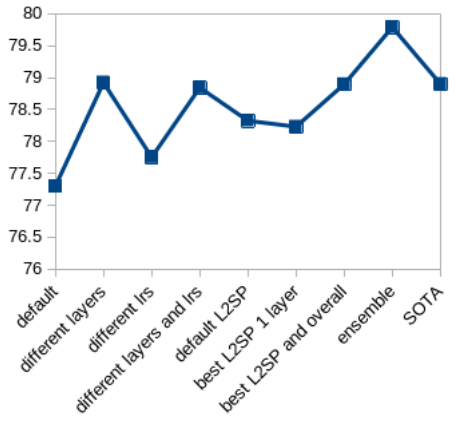


Figure 9.9: Optimal settings versus best default settings DTD

9.7 Broader Impact

This research is focused on improving transfer learning for small target image classification datasets. The positive impacts are easy to observe in improved performance of models utilising transfer learning in a wide range of real world applications. For example, faster and more accurate medical diagnostics where there is limited data available.

However, improvements to image classification models can have positive or negative impacts. More accurate models could reduce errors but result in overconfidence and more reliance on results in areas where the full limitations are not well understood and taken into account. Improved image classification models could also be

Table 9.7: Predicting optimal settings based on pretrained features

	Caltech	DTD	Cars	Aircraft
Fisher ratio	1.35	3.47	1.18	0.83
EMD similarity	0.568	0.540	0.536	0.557
Random initialisation minus fixed features	-16.2	-7.8	28.5	28.9
Optimal learning rate	Low	Low	High	High
L2SP	Yes	Yes	No	No
More layers reinitialized	Yes	Yes	No	No
Low layers at low learning rate	No	No	Yes	Yes

used in areas with potential negative impacts, for example:

- use of intrusive facial recognition systems
- discrimination based on subtle items such as religious symbols

9.8 Limitations and future work

The main aim of this work is to highlight the need for a non-binary approach to achieving optimal performance in transfer learning applied to image classification on small target datasets. The relationship between the source and the target dataset and how well the pretrained weights fit the target task has been shown to be important to transfer learning performance numerous times [Ngiam et al., 2018; Mahajan et al., 2018; Cui et al., 2018; Ge and Yu, 2017; Sabatelli et al., 2018; Li et al., 2018, 2019b; Wan et al., 2019; Ng et al., 2015; Zoph et al., 2020; Li et al., 2020a]. However, the guidance as to how to set transfer learning hyperparameters based on the heuristics presented in this chapter is based only on the experiments on the four datasets outlined in this chapter. Care should be taken if applying this guidance to new datasets, particularly outside of image classification.

Chapter 10

Conclusion

This thesis defines deep transfer learning and the problem it attempts to solve in relation to image classification. It provides a new taxonomy of the applications of transfer learning for image classification. This taxonomy makes it easier to see overarching patterns where transfer learning has been effective and where it has failed to fulfill its potential. The thesis also provides the first comprehensive review of deep transfer learning as it relates to image classification overall and suggestions for future research directions in the field. While there have been recent general surveys of deep transfer learning, and ones that relate to particular specialised target image classification tasks, there have been none that review the area as a whole. It is important for future progress in the field that all current knowledge on deep transfer learning in image classification is collated and the overarching patterns are analysed and discussed.

My research outlined in this thesis introduces an alternative to the traditional approach of selecting deep transfer learning and fine-tuning hyperparameters. I define binary thinking in deep transfer learning as primarily focusing on the two extreme ends of the scales when selecting deep transfer learning and fine-tuning hyperparameters:

- Taking an all or nothing approach to transfer learning is binary thinking. Either transferring all possible layers or reinitializing all layers. The non-binary approach would consider reinitializing more than just the final prediction layer.

- Selecting one best learning rate for all layers or freezing some of the lower layers that are considered more generalizable is also binary thinking. The non-binary approach would allow flexibility for some layers, particularly lower layers, to be trained at a lower learning rate.

- Selecting one type of regularization parameter is a third kind of binary thinking. Either L2SP regularization that decays weights towards their pretrained

values (or another recent transfer learning specific regularization approach), or L2 regularization that decays weights towards 0. The non-binary approach would allow for some lower layers to be regularized with transfer learning specific regularization, and higher layers to be decayed towards 0.

Experiment results using best practice non-binary hyperparameters are either comparable to or exceed state of the art results for a number of and variety of small target datasets.

The work in this thesis shows methods for predicting the best transfer learning hyperparameters without being limited by binary constraints. Heuristics are introduced that show good potential to predict the optimal non-binary hyperparameters on a range of target datasets, however, more work should be done in testing these heuristics on a wider range of target datasets of a variety of sizes.

A model that can be trained to predict optimal hyperparameters was designed and it was shown to significantly improve on standard transfer learning hyperparameters when training an AlexNet model and transferring between two subsets of ImageNet of various sizes. A similar model was trained to predict optimal hyperparameters for an Inception version 4 model with the target dataset being the Stanford Cars dataset with all possible numbers of training examples per class. In this case the reinforcement learning model converged too early resulting in a collapse of hyperparameter predictions and sub optimal performance. Most of the reasons for early convergence of the reinforcement learning model were due to methods used to speed up training to make best use of limited compute resources. Recommendations are made on how to avoid this in future and based on the experiments so far, the models designed have shown good potential to improve on standard binary hyperparameters.

10.1 Future Work

I recommend several directions for future work:

1. Expand all results to a broader range of image understanding target datasets.

2. Apply non-binary hyperparameters for deep transfer learning more broadly. I have started the work of applying non-binary transfer learning to better retrain when encountering an out-of distribution failure in reinforcement learning. Initial results are promising.

3. Combining more than one set of best practice transfer learning hyperparameters for an application in mixture model datasets. I currently have a student working on this who has a proof of concept model showing strong results.

4. Application of best practice to physiological signals and other human centered computing applications. Physiological signals are unique to each subject but show similarities. Applications in this space can be thought of as a transfer learning problem with each individual's signals being a small target dataset. Given the methods described in this thesis are particularly designed to improve results for small target datasets I anticipate that they would have the potential to improve results in this area.

5. I have started some initial work to identify a point in between fully reinitializing a layer or block of a deep neural network and keeping the pretrained weights when they are far from a good minimum for the new task. Keeping the distribution of the pretrained weights in higher layers but normalizing them to be of the same size as reinitialized weights has showed some success for small datasets.

Bibliography

Agrawal, P.; Girshick, R.; and Malik, J., 2014. Analyzing the performance of multilayer neural networks for object recognition. In *European conference on computer vision*, 329–344. Springer. (cited on pages 67, 70, 103, 110, 124, and 139)

Andrychowicz, M.; Raichuk, A.; Stańczyk, P.; Orsini, M.; Girgin, S.; Marinier, R.; Hussenot, L.; Geist, M.; Pietquin, O.; Michalski, M.; et al., 2020. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, (2020). (cited on page 135)

Arras, L.; Arjona-Medina, J.; Widrich, M.; Montavon, G.; Gillhofer, M.; Müller, K.-R.; Hochreiter, S.; and Samek, W., 2019. Explaining and interpreting lstms. In *Explainable ai: Interpreting, explaining and visualizing deep learning*, 211–238. Springer. (cited on page 46)

Azizi, S.; Mustafa, B.; Ryan, F.; Beaver, Z.; Freyberg, J.; Deaton, J.; Loh, A.; Karthikesalingam, A.; Kornblith, S.; Chen, T.; et al., 2021. Big self-supervised models advance medical image classification. *arXiv preprint arXiv:2101.05224*, (2021). (cited on pages 86, 90, 97, and 105)

Azizpour, H.; Razavian, A. S.; Sullivan, J.; Maki, A.; and Carlsson, S., 2015. Factors of transferability for a generic convnet representation. *IEEE transactions on pattern analysis and machine intelligence*, 38, 9 (2015), 1790–1802. (cited on pages 67, 70, 103, 124, and 139)

Azizpour, H.; Razavian, A. S.; Sullivan, J.; Maki, A.; and Carlsson, S., 2016. Factors of transferability for a generic convnet representation. *IEEE transactions on pattern analysis and machine intelligence*, 38, 9 (2016), 1790–1802. (cited on pages 110 and 111)

Baker, B.; Gupta, O.; Raskar, R.; and Naik, N., 2017. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, (2017). (cited on page 53)

Bao, Y.; Li, Y.; Huang, S.-L.; Zhang, L.; Zheng, L.; Zamir, A.; and Guibas, L., 2019. An information-theoretic approach to transferability in task transfer learning.

In *2019 IEEE International Conference on Image Processing (ICIP)*, 2309–2313. IEEE. (cited on pages 79 and 105)

Bengio, S., 2015. Sharing representations for long tail computer vision problems. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, 1–1. (cited on page 1)

Bengio, Y.; Lamblin, P.; Popovici, D.; and Larochelle, H., 2007. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, 153–160. (cited on page 17)

Berg, T.; Liu, J.; Woo Lee, S.; Alexander, M. L.; Jacobs, D. W.; and Belhumeur, P. N., 2014. Birdsnap: Large-scale fine-grained visual categorization of birds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2011–2018. (cited on page 64)

Bishop, C. M., 2006. *Pattern recognition and machine learning*. springer. (cited on page 13)

Bossard, L.; Guillaumin, M.; and Van Gool, L., 2014. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*. (cited on page 64)

Bottou, L. and Bousquet, O., 2008. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, 161–168. (cited on page 58)

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al., 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, (2020). (cited on page 102)

Cai, H.; Chen, T.; Zhang, W.; Yu, Y.; and Wang, J., 2018. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32. (cited on page 53)

Cai, L.; Zhang, Z.; Zhu, Y.; Zhang, L.; Li, M.; and Xue, X., 2022. Bigdetection: A large-scale benchmark for improved object detector pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4777–4787. (cited on page 83)

Chen, L.-C.; Zhu, Y.; Papandreou, G.; Schroff, F.; and Adam, H., 2018. Encoder-decoder with atrous separable convolution for semantic image segmentation. In

Proceedings of the European conference on computer vision (ECCV), 801–818. (cited on page 92)

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G., 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, 1597–1607. PMLR. (cited on pages 143 and 151)

Chen, X.; Wang, S.; Fu, B.; Long, M.; and Wang, J., 2019. Catastrophic forgetting meets negative transfer: Batch spectral shrinkage for safe transfer learning. (2019). (cited on pages 72, 73, 77, and 103)

Chu, B.; Madhavan, V.; Beijbom, O.; Hoffman, J.; and Darrell, T., 2016. Best practices for fine-tuning visual classifiers to new domains. In *European conference on computer vision*, 435–442. Springer. (cited on pages 70, 124, and 139)

Cimpoi, M.; Maji, S.; Kokkinos, I.; Mohamed, S.; and Vedaldi, A., 2014. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3606–3613. (cited on pages 65, 136, and 141)

Csurka, G., 2017. Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*, (2017). (cited on page 99)

Cubuk, E. D.; Zoph, B.; Mane, D.; Vasudevan, V.; and Le, Q. V., 2019. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 113–123. (cited on page 52)

Cui, Y.; Song, Y.; Sun, C.; Howard, A.; and Belongie, S., 2018. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4109–4118. (cited on pages 76, 95, 123, 138, 140, 143, and 154)

Cui, Y.; Zhou, F.; Lin, Y.; and Belongie, S., 2016. Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1153–1162. (cited on page 76)

Dabbura, I., 2017. Gradient descent algorithm and its variants. https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3. (cited on page 23)

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L., 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*. (cited on pages 33, 62, 67, 81, 84, 88, 91, 110, 125, and 133)

Deng, J.; Guo, J.; Xue, N.; and Zafeiriou, S., 2019. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4690–4699. (cited on page 88)

Deshpande, M., 2017. Recurrent neural networks for language modeling. (cited on page 44)

Do, D. T.; Lee, J.; and Nguyen-Xuan, H., 2019. Fast evaluation of crack growth path using time series forecasting. *Engineering Fracture Mechanics*, 218 (2019), 106567. (cited on page 42)

Donahue, J.; Jia, Y.; Vinyals, O.; Hoffman, J.; Zhang, N.; Tzeng, E.; and Darrell, T., 2014. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, 647–655. (cited on page 110)

Dowling, J., 2017. Large scale machine learning and deep learning. https://www.kth.se/social/files/5a04101256be5be762e60d54/ID2223-03-gradient-descent-spark-ml.pdf. (cited on page 11)

Duchi, J.; Hazan, E.; and Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12, 7 (2011). (cited on page 24)

Evans, N.; Plested, J.; and Gedeon, T., 2020. Exploring the correlation between random convolutional architectures and the trained equivalent. In *2020 International Joint Conference on Neural Networks (IJCNN)*, 1–6. IEEE.

Everingham, M.; Van Gool, L.; Williams, C. K. I.; Winn, J.; and Zisserman, A. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html. (cited on page 64)

Fang, J.; Sun, Y.; Peng, K.; Zhang, Q.; Li, Y.; Liu, W.; and Wang, X., 2020. Fast neural network adaptation via parameter remapping and architecture search. *arXiv preprint arXiv:2001.02525*, (2020). (cited on page 53)

Fei-Fei, L.; Fergus, R.; and Perona, P., 2004. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, 178–178. IEEE. (cited on page 64)

Feng, C., 2021. Loss function overview. https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/basic-overview/loss_function_overview. (cited on page 19)

French, R. M., 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3, 4 (1999), 128–135. (cited on page 100)

Fukenaga, K., 1990. Introduction to statistical pattern recognition. *2ed, Academic Press, San Diego*, (1990). (cited on page 143)

Garcia-Garcia, A.; Orts-Escolano, S.; Oprea, S.; Villena-Martinez, V.; Martinez-Gonzalez, P.; and Garcia-Rodriguez, J., 2018. A survey on deep learning techniques for image and video semantic segmentation. *Applied Soft Computing*, 70 (2018), 41–65. (cited on page 92)

Ge, W. and Yu, Y., 2017. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1086–1095. (cited on pages 95, 96, 124, 138, 140, and 154)

Ghosh, S.; Das, N.; Das, I.; and Maulik, U., 2019. Understanding deep learning techniques for image segmentation. *ACM Computing Surveys (CSUR)*, 52, 4 (2019), 1–35. (cited on page 92)

Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 580–587. (cited on pages 1, 33, 91, 110, 124, and 138)

Glorot, X. and Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256. JMLR Workshop and Conference Proceedings. (cited on page 26)

Gong, B.; Shi, Y.; Sha, F.; and Grauman, K., 2012. Geodesic flow kernel for unsupervised domain adaptation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2066–2073. IEEE. (cited on page 99)

Gonthier, N.; Gousseau, Y.; and Ladjal, S., 2020. An analysis of the transfer learning of convolutional neural networks for artistic images. *arXiv preprint arXiv:2011.02727*, (2020). (cited on pages 76, 95, and 105)

Goodfellow, I.; Bengio, Y.; Courville, A.; and Bengio, Y., 2016. *Deep learning*, vol. 1. MIT press Cambridge. (cited on pages 18, 25, 26, and 99)

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y., 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27 (2014). (cited on page 102)

Graves, A., 2012. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, 5–13. Springer. (cited on page 45)

Griffin, G.; Holub, A.; and Perona, P., 2007. Caltech-256 object category dataset. (2007). (cited on pages 64 and 141)

Guo, Y.; Shi, H.; Kumar, A.; Grauman, K.; Rosing, T.; and Feris, R., 2019. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4805–4814. (cited on pages 75, 125, and 141)

He, K.; Girshick, R.; and Dollár, P., 2018. Rethinking imagenet pre-training. *arXiv preprint arXiv:1811.08883*, (2018). (cited on pages 67, 68, 82, 86, 91, 111, and 138)

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R., 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, 2961–2969. (cited on pages 92, 110, and 111)

He, K.; Zhang, X.; Ren, S.; and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 1026–1034. (cited on page 26)

He, K.; Zhang, X.; Ren, S.; and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778. (cited on pages 34, 35, 40, 70, 82, 93, and 139)

Heidenreich, H., 2018. What are the types of machine learning? https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f. (cited on page 7)

Heker, M. and Greenspan, H., 2020. Joint liver lesion segmentation and classification via transfer learning. *arXiv preprint arXiv:2004.12352*, (2020). (cited on pages 67, 83, 90, and 105)

Hendrycks, D. and Dietterich, T., 2019. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, (2019). (cited on page 94)

Hendrycks, D.; Zhao, K.; Basart, S.; Steinhardt, J.; and Song, D., 2021. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 15262–15271. (cited on page 94)

Hinton, G.; Srivastava, N.; and Swersky, K., 2012a. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14, 8 (2012), 2. (cited on page 24)

Hinton, G.; Vinyals, O.; and Dean, J., 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, (2015). (cited on page 62)

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. R., 2012b. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, (2012). (cited on pages 35 and 36)

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, 9, 8 (1997), 1735–1780. (cited on pages 44 and 127)

Hornik, K.; Stinchcombe, M.; White, H.; et al., 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2, 5 (1989), 359–366. (cited on page 15)

Hu, B.; Lu, Z.; Li, H.; and Chen, Q., 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, 2042–2050. (cited on page 29)

Huh, M.; Agrawal, P.; and Efros, A. A., 2016. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, (2016). (cited on pages 67, 70, 92, and 110)

Ioffe, S., 2017. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *arXiv preprint arXiv:1702.03275*, (2017). (cited on page 75)

Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, (2015). (cited on pages 36, 75, and 77)

Jeon, Y.; Choi, Y.; Park, J.; Yi, S.; Cho, D.; and Kim, J., 2020. Sample-based regularization: A transfer learning strategy toward better generalization. *arXiv preprint arXiv:2007.05181*, (2020). (cited on page 74)

Jing, L. and Tian, Y., 2020. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, (2020). (cited on pages 97 and 102)

Kalchbrenner, N.; Grefenstette, E.; and Blunsom, P., 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, (2014). (cited on page 29)

Karpathy, A., 2015. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/. (cited on page 43)

Kaur, A., 2018. Neural network hyperparameter tuning in a nutshell. https://medium.com/@arshdeepkaur/neural-network-hyperparameter-tuning-in-a-nutshell-8c2b9a14a2c0. (cited on page 21)

Kaya, M. and Bilge, H. Ş., 2019. Deep metric learning: A survey. *Symmetry*, 11, 9 (2019), 1066. (cited on page 100)

Khosla, A.; Jayadevaprakash, N.; Yao, B.; and Li, F.-F., 2011. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, vol. 2. (cited on page 65)

Kingma, D. P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, (2014). (cited on page 25)

Kolesnikov, A.; Beyer, L.; Zhai, X.; Puigcerver, J.; Yung, J.; Gelly, S.; and Houlsby, N., 2019. Big transfer (bit): General visual representation learning. *arXiv preprint arXiv:1912.11370*, (2019). (cited on pages 1, 77, 78, 92, 103, 104, and 138)

Kornblith, S.; Shlens, J.; and Le, Q. V., 2019. Do better imagenet models transfer better? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2661–2671. (cited on pages 39, 67, 68, 71, 76, 77, 82, 84, 85, 86, 87, 110, 111, 124, 131, 139, 141, and 142)

Kou, Z.; You, K.; Long, M.; and Wang, J., 2020. Stochastic normalization. *Advances in Neural Information Processing Systems*, 33 (2020). (cited on pages 73, 75, and 77)

Kraus, O. Z.; Grys, B. T.; Ba, J.; Chong, Y.; Frey, B. J.; Boone, C.; and Andrews, B. J., 2017. Automated analysis of high-content microscopy data with deep learning. *Molecular systems biology*, 13, 4 (2017), 924. (cited on pages 83 and 90)

Krause, J.; Stark, M.; Deng, J.; and Fei-Fei, L., 2013. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia. (cited on pages 62, 65, 131, and 141)

Krishna, S. T. and Kalluri, H. K., 2019. Deep learning and transfer learning approaches for image classification. *International Journal of Recent Technology and Engineering (IJRTE)*, 7, 5S4 (2019), 427–432. (cited on page 69)

Krizhevsky, A.; Hinton, G.; et al., 2009. Learning multiple layers of features from tiny images. (2009). (cited on page 64)

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105. (cited on pages 1, 18, 33, 34, 81, 84, 110, 112, 125, 127, 131, and 138)

Kuznetsova, A.; Rom, H.; Alldrin, N.; Uijlings, J.; Krasin, I.; Pont-Tuset, J.; Kamali, S.; Popov, S.; Malloci, M.; Kolesnikov, A.; et al., 2020. The open images dataset v4. *International Journal of Computer Vision*, 128, 7 (2020), 1956–1981. (cited on page 83)

Lake, B. M.; Salakhutdinov, R.; and Tenenbaum, J. B., 2015. Human-level concept learning through probabilistic program induction. *Science*, 350, 6266 (2015), 1332–1338. (cited on page 66)

LeCun, Y.; Bengio, Y.; and Hinton, G., 2015. Deep learning. *nature*, 521, 7553 (2015), 436–444. (cited on page 17)

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. D., 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1, 4 (1989), 541–551. (cited on pages 29, 33, and 34)

Ledig, C.; Theis, L.; Huszár, F.; Caballero, J.; Cunningham, A.; Acosta, A.; Aitken, A.; Tejani, A.; Totz, J.; Wang, Z.; et al., 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4681–4690. (cited on pages 97 and 102)

Lee, H.; Grosse, R.; Ranganath, R.; and Ng, A. Y., 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, 609–616. (cited on page 31)

Li, C.; Yuan, X.; Lin, C.; Guo, M.; Wu, W.; Yan, J.; and Ouyang, W., 2019a. Am-lfs: Automl for loss function search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 8410–8419. (cited on page 52)

Li, H.; Chaudhari, P.; Yang, H.; Lam, M.; Ravichandran, A.; Bhotika, R.; and Soatto, S., 2020a. Rethinking the hyperparameters for fine-tuning. *arXiv preprint arXiv:2002.11770*, (2020). (cited on pages 67, 68, 72, 76, 77, 103, 124, 139, 140, 147, and 154)

Li, L. and Talwalkar, A., 2018. What's the deal with neural architecture search. https://www.determined.ai/blog/neural-architecture-search. (cited on page 54)

Li, S. and Deng, W., 2020. Deep facial expression recognition: A survey. *IEEE Transactions on Affective Computing*, (2020). (cited on pages 1, 33, 88, 124, and 138)

Li, X.; Grandvalet, Y.; and Davoine, F., 2018. Explicit inductive bias for transfer learning with convolutional networks. *arXiv preprint arXiv:1802.01483*, (2018). (cited on pages 72, 73, 76, 77, 94, 124, 138, 139, 140, 147, and 154)

Li, X.; Grandvalet, Y.; and Davoine, F., 2020b. A baseline regularization scheme for transfer learning with convolutional neural networks. *Pattern Recognition*, 98 (2020), 107049. (cited on page 72)

Li, X.; Xiong, H.; Wang, H.; Rao, Y.; Liu, L.; and Huan, J., 2019b. Delta: Deep learning transfer using feature map with attention for convolutional networks. *arXiv preprint arXiv:1901.09229*, (2019). (cited on pages 72, 76, 77, 124, 138, 140, 143, 151, and 154)

Lin, M.; Chen, Q.; and Yan, S., 2013. Network in network. *arXiv preprint arXiv:1312.4400*, (2013). (cited on page 32)

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L., 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*, 740–755. Springer. (cited on pages 83, 91, and 93)

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K., 2018a. Progressive neural architecture search. In

Proceedings of the European conference on computer vision (ECCV), 19–34. (cited on page 53)

Liu, H.; Long, M.; Wang, J.; and Jordan, M. I., 2019a. Towards understanding the transferability of deep representations. *arXiv preprint arXiv:1909.12031*, (2019). (cited on pages 60, 79, 83, 104, and 106)

Liu, H.; Simonyan, K.; and Yang, Y., 2018b. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, (2018). (cited on page 53)

Liu, W.; Wen, Y.; Yu, Z.; Li, M.; Raj, B.; and Song, L., 2017. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 212–220. (cited on page 88)

Liu, W.; Wen, Y.; Yu, Z.; and Yang, M., 2016. Large-margin softmax loss for convolutional neural networks. In *ICML*, vol. 2, 7. (cited on page 88)

Liu, Y.; Yao, Y.; Wang, Z.; Plested, J.; and Gedeon, T., 2019b. Generalized alignment for multimodal physiological signal learning. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–10. IEEE.

Maas, A. L.; Hannun, A. Y.; Ng, A. Y.; et al., 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, vol. 30, 3. Citeseer. (cited on page 18)

Mahajan, D.; Girshick, R.; Ramanathan, V.; He, K.; Paluri, M.; Li, Y.; Bharambe, A.; and van der Maaten, L., 2018. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 181–196. (cited on pages 1, 63, 67, 76, 83, 91, 92, 93, 95, 104, 110, 111, 123, 124, 138, 139, 140, and 154)

Maji, S.; Kannala, J.; Rahtu, E.; Blaschko, M.; and Vedaldi, A., 2013. Fine-grained visual classification of aircraft. Technical report, Toyota Technological Institute at Chicago. (cited on pages 65, 136, and 141)

Mao, H.; Alizadeh, M.; Menache, I.; and Kandula, S., 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, 50–56. (cited on page 48)

Maria Carlucci, F.; Porzi, L.; Caputo, B.; Ricci, E.; and Rota Bulo, S., 2017. Autodial: Automatic domain alignment layers. In *Proceedings of the IEEE International Conference on Computer Vision*, 5067–5075. (cited on page 74)

Masi, I.; Wu, Y.; Hassner, T.; and Natarajan, P., 2018. Deep face recognition: A survey. In *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*, 471–478. IEEE. (cited on pages 1, 33, 87, 124, and 138)

Mazurowski, M. A.; Buda, M.; Saha, A.; and Bashir, M. R., 2019. Deep learning in radiology: An overview of the concepts and a survey of the state of the art with focus on mri. *Journal of magnetic resonance imaging*, 49, 4 (2019), 939–954. (cited on pages 1, 34, 83, 89, 124, and 138)

McCloskey, M. and Cohen, N. J., 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, vol. 24, 109–165. Elsevier. (cited on page 100)

Miller, G. A., 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38, 11 (1995), 39–41. (cited on pages 63 and 64)

Minaee, S.; Boykov, Y.; Porikli, F.; Plaza, A.; Kehtarnavaz, N.; and Terzopoulos, D., 2020. Image segmentation using deep learning: A survey. *arXiv preprint arXiv:2001.05566*, (2020). (cited on page 92)

Minsky, M. and Papert, A. S., 1969. Perceptrons. (cited on page 15)

Mitchell, T. M. et al., 1997. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 37 (1997), 870–877. (cited on page 6)

Mormont, R.; Geurts, P.; and Marée, R., 2018. Comparison of deep transfer learning strategies for digital pathology. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2262–2271. (cited on pages 82, 83, 89, 104, 110, 111, 124, and 138)

Moudiki, T., 2020. Linear model, xgboost and randomforest cross-validation using crossval::crossval$_ml$.. (cited on page 9)

Munder, S. and Gavrila, D. M., 2006. An experimental study on pedestrian classification. *IEEE transactions on pattern analysis and machine intelligence*, 28, 11 (2006), 1863–1868. (cited on page 66)

Musgrave, K.; Belongie, S.; and Lim, S.-N., 2020. A metric learning reality check. In *European Conference on Computer Vision*, 681–699. Springer. (cited on page 88)

Mustafa, B.; Loh, A.; Freyberg, J.; MacWilliams, P.; Wilson, M.; McKinney, S. M.; Sieniek, M.; Winkens, J.; Liu, Y.; Bui, P.; et al., 2021. Supervised transfer learning at scale for medical imaging. *arXiv preprint arXiv:2101.05913*, (2021). (cited on page 90)

Nayef, B. H.; Abdullah, S. N. H. S.; Sulaiman, R.; and Alyasseri, Z. A. A., 2022. Optimized leaky relu for handwritten arabic character recognition using convolution neural networks. *Multimedia Tools and Applications*, 81, 2 (2022), 2065–2094. (cited on page 18)

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y., 2011. Reading digits in natural images with unsupervised feature learning. (2011). (cited on page 66)

Neyshabur, B.; Sedghi, H.; and Zhang, C., 2020. What is being transferred in transfer learning? *arXiv preprint arXiv:2008.11687*, (2020). (cited on pages 60, 61, 76, 78, 83, 104, 106, and 124)

Ng, A., 2022. Neural networks and deep learning. https://www.coursera.org/learn/neural-networks-deep-learning. (cited on page 23)

Ng, H.-W.; Nguyen, V. D.; Vonikakis, V.; and Winkler, S., 2015. Deep learning for emotion recognition on small datasets using transfer learning. In *Proceedings of the 2015 ACM on international conference on multimodal interaction*, 443–449. (cited on pages 76, 88, 91, 93, 99, 105, 124, 138, 140, and 154)

Ngiam, J.; Peng, D.; Vasudevan, V.; Kornblith, S.; Le, Q. V.; and Pang, R., 2018. Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056*, (2018). (cited on pages 1, 76, 92, 93, 95, 96, 104, 123, 138, 140, and 154)

Ngoma, Y. M. et al., 2017. *Analysis of Control Attainment in Endogenous Electroencephalogram Based Brain Computer Interfaces*. Ph.D. thesis, Tshwane University of Technology. (cited on page 8)

Nguyen, C.; Hassner, T.; Seeger, M.; and Archambeau, C., 2020. Leep: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*, 7294–7305. PMLR. (cited on pages 80 and 105)

Nilsback, M.-E. and Zisserman, A., 2008. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, 722–729. IEEE. (cited on page 65)

Oord, A. v. d.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; and Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, (2016). (cited on page 29)

Paine, T. L.; Khorrami, P.; Han, W.; and Huang, T. S., 2014. An analysis of unsupervised pre-training in light of recent advances. *arXiv preprint arXiv:1412.6597*, (2014). (cited on pages 97 and 102)

Palaz, D.; Collobert, R.; and Doss, M. M., 2013. End-to-end phoneme sequence recognition using convolutional neural networks. *arXiv preprint arXiv:1312.2137*, (2013). (cited on page 29)

Pan, S. J. and Yang, Q., 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22, 10 (2009), 1345–1359. (cited on pages 58, 61, 96, and 99)

Parkhi, O. M.; Vedaldi, A.; Zisserman, A.; and Jawahar, C., 2012. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, 3498–3505. IEEE. (cited on page 65)

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A., 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*. (cited on page 112)

Pathak, D.; Krahenbuhl, P.; Donahue, J.; Darrell, T.; and Efros, A. A., 2016. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2536–2544. (cited on pages 97 and 102)

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12 (2011), 2825–2830. (cited on page 10)

Peleg, S.; Werman, M.; and Rom, H., 1989. A unified approach to the change of resolution: Space and gray-level. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11, 7 (1989), 739–742. (cited on pages 95 and 143)

Pham, H.; Guan, M.; Zoph, B.; Le, Q.; and Dean, J., 2018. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, 4095–4104. PMLR. (cited on page 53)

Pineda, F., 1987. Generalization of back propagation to recurrent and higher order neural networks. In *Neural information processing systems*, 602–611. (cited on page 43)

Plested, J. and Gedeon, T., 2019. An analysis of the interaction between transfer learning protocols in deep neural networks. In *International Conference on Neural Information Processing*, 312–323. Springer. (cited on pages 3, 62, 67, 70, 81, 82, 83, 84, 103, 104, 106, 124, 125, 128, 129, 135, 139, and 146)

Plested, J.; Gedeon, T. D.; Zhu, X.; Dhall, A.; and Geocke, R. R., 2017. Detection of universal cross-cultural depression indicators from the physiological signals of observers. In *2017 Seventh International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*, 185–192. IEEE.

Plested, J.; Shen, X.; and Gedeon, T., 2021. Rethinking binary hyperparameters for deep transfer learning. In *International Conference on Neural Information Processing*, 463–475. Springer. (cited on pages 67, 68, 72, 76, 77, 78, 82, 85, 103, 104, 105, 106, 124, 125, 135, and 136)

Polyak, B. T., 1964. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4, 5 (1964), 1–17. (cited on page 22)

Puigcerver, J.; Riquelme, C.; Mustafa, B.; Renggli, C.; Pinto, A. S.; Gelly, S.; Keysers, D.; and Houlsby, N., 2020. Scalable transfer learning with expert models. *arXiv preprint arXiv:2009.13239*, (2020). (cited on pages 76, 93, 95, and 96)

Quattoni, A. and Torralba, A., 2009. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 413–420. IEEE. (cited on page 65)

Radford, A.; Metz, L.; and Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, (2015). (cited on pages 97 and 102)

Raghu, M.; Zhang, C.; Kleinberg, J.; and Bengio, S., 2019. Transfusion: Understanding transfer learning for medical imaging. *arXiv preprint arXiv:1902.07208*, (2019). (cited on pages 67, 83, 86, and 90)

Ramachandran, P.; Zoph, B.; and Le, Q. V., 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, (2017). (cited on page 52)

Ranjan, R.; Castillo, C. D.; and Chellappa, R., 2017. L2-constrained softmax loss for discriminative face verification. *arXiv preprint arXiv:1703.09507*, (2017). (cited on page 88)

Ratcliff, R., 1990. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97, 2 (1990), 285. (cited on page 100)

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V., 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 4780–4789. (cited on page 52)

Rebuffi, S.-A.; Bilen, H.; and Vedaldi, A., 2017. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, 506–516. (cited on page 66)

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788. (cited on page 91)

Ren, P.; Xiao, Y.; Chang, X.; Huang, P.-Y.; Li, Z.; Chen, X.; and Wang, X., 2021. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54, 4 (2021), 1–34. (cited on page 52)

Ren, S.; He, K.; Girshick, R.; and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28 (2015), 91–99. (cited on page 91)

Ribani, R. and Marengoni, M., 2019. A survey of transfer learning for convolutional neural networks. In *2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, 47–57. IEEE. (cited on page 69)

Ridnik, T.; Lawen, H.; Noy, A.; and Friedman, I., 2020. Tresnet: High performance gpu-dedicated architecture. *arXiv preprint arXiv:2003.13630*, (2020). (cited on pages 99, 100, 143, and 151)

Rosenblatt, F., 1962. Principles of neurodynamics: Perceptions and the theory of brain mechanisms. (1962). (cited on pages 13 and 14)

Rosenstein, M. T.; Marx, Z.; Kaelbling, L. P.; and Dietterich, T. G., 2005. To transfer or not to transfer. In *NIPS 2005 workshop on transfer learning*, vol. 898, 1–4. (cited on page 60)

Rubner, Y.; Tomasi, C.; and Guibas, L. J., 2000. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40, 2 (2000), 99–121. (cited on pages 95 and 143)

Rumelhart, D. E.; Durbin, R.; Golden, R.; and Chauvin, Y., 1995. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, (1995), 1–34. (cited on page 18)

Sabatelli, M.; Kestemont, M.; Daelemans, W.; and Geurts, P., 2018. Deep transfer learning for art classification problems. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 0–0. (cited on pages 95, 124, 138, 140, and 154)

Saenko, K.; Kulis, B.; Fritz, M.; and Darrell, T., 2010. Adapting visual category models to new domains. In *European conference on computer vision*, 213–226. Springer.

(cited on page 99)

Schroff, F.; Kalenichenko, D.; and Philbin, J., 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 815–823. (cited on page 88)

Scott, T.; Ridgeway, K.; and Mozer, M. C., 2018. Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning. In *Advances in Neural Information Processing Systems*, 76–85. (cited on pages 82, 101, 111, and 112)

Shao, L.; Zhu, F.; and Li, X., 2014. Transfer learning for visual categorization: A survey. *IEEE transactions on neural networks and learning systems*, 26, 5 (2014), 1019–1034. (cited on page 69)

Shao, S.; Li, Z.; Zhang, T.; Peng, C.; Yu, G.; Zhang, X.; Li, J.; and Sun, J., 2019. Objects365: A large-scale, high-quality dataset for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, 8430–8439. (cited on page 83)

Sharif Razavian, A.; Azizpour, H.; Sullivan, J.; and Carlsson, S., 2014. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 806–813. (cited on pages 67, 70, and 110)

Shaw, S. A comparative study of activation functions. https://wandb.ai/shweta/Activation%20Functions/reports/A-Comparative-Study-of-Activation-Functions--VmlldzoxMDQwOTQ. (cited on page 17)

Shen, X.; Plested, J.; Caldwell, S.; and Gedeon, T., 2021a. Exploring biases and prejudice of facial synthesis via semantic latent space. In *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–8. IEEE.

Shen, X.; Plested, J.; Caldwell, S.; Zhong, Y.; and Gedeon, T., 2022. Amf: Adaptable weighting fusion with multiple fine-tuning for image classification. *arXiv preprint arXiv:2207.12944*, (2022). (cited on page 125)

Shen, X.; Plested, J.; and Gedeon, T., 2021b. Feature selection on thermal-stress dataset. *arXiv preprint arXiv:2109.03755*, (2021).

Shen, X.; Plested, J.; Yao, Y.; and Gedeon, T., 2020. Pairwise-gan: Pose-based view synthesis through pair-wise training. In *International Conference on Neural Information Processing*, 507–515. Springer.

Shen, Z.; Liu, Z.; Li, J.; Jiang, Y.-G.; Chen, Y.; and Xue, X., 2017a. Dsod: Learning

deeply supervised object detectors from scratch. In *Proceedings of the IEEE international conference on computer vision*, 1919–1927. (cited on pages 91 and 138)

Shen, Z.; Shi, H.; Feris, R.; Cao, L.; Yan, S.; Liu, D.; Wang, X.; Xue, X.; and Huang, T. S., 2017b. Learning object detectors from scratch with gated recurrent feature pyramids. *arXiv preprint arXiv:1712.00886*, 1 (2017). (cited on pages 91 and 138)

Shin, R.; Packer, C.; and Song, D., 2018. Differentiable neural network architecture search. (2018). (cited on page 53)

Shu, J.; Xu, Z.; and Meng, D., 2018. Small sample learning in big data era. *arXiv preprint arXiv:1808.04572*, (2018). (cited on page 69)

Singh, A.; Sarkhel, R.; Das, N.; Kundu, M.; and Nasipuri, M., 2020. A skip-connected multi-column network for isolated handwritten bangla character and digit recognition. *Sensing and Imaging*, 21, 1 (2020), 1–25. (cited on page 30)

Snell, J.; Swersky, K.; and Zemel, R. S., 2017. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, (2017). (cited on page 101)

Soomro, K.; Zamir, A. R.; and Shah, M., 2012. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, (2012). (cited on page 66)

Srivastava, N.; Mansimov, E.; and Salakhudinov, R., 2015. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, 843–852. (cited on pages 97 and 102)

Stallkamp, J.; Schlipsing, M.; Salmen, J.; and Igel, C., 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32 (2012), 323–332. (cited on page 66)

Strubell, E.; Ganesh, A.; and McCallum, A., 2019. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, (2019). (cited on page 1)

Stuart, R. and Peter, N., 2016. Artificial intelligence-a modern approach 3rd ed. (cited on page 38)

Sun, C.; Shrivastava, A.; Singh, S.; and Gupta, A., 2017. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, 843–852. (cited on pages 63, 91, 92, 93, 110, and 111)

Sutton, R. S. and Barto, A. G., 2018. *Reinforcement learning: An introduction*. MIT press. (cited on pages 45, 46, and 47)

Szegedy, C.; Ioffe, S.; Vanhoucke, V.; and Alemi, A. A., 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*. (cited on pages 35, 41, 131, 133, and 141)

Tajbakhsh, N.; Shin, J. Y.; Gurudu, S. R.; Hurst, R. T.; Kendall, C. B.; Gotway, M. B.; and Liang, J., 2016. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35, 5 (2016), 1299–1312. (cited on page 89)

Tan, C.; Sun, F.; Kong, T.; Zhang, W.; Yang, C.; and Liu, C., 2018. A survey on deep transfer learning. In *International conference on artificial neural networks*, 270–279. Springer. (cited on page 69)

Tan, M. and Le, Q. V., 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, (2019). (cited on pages 82, 94, 110, and 111)

Thanaki, J., 2017. *Python natural language processing*. Packt Publishing Ltd. (cited on page 16)

Tommasi, T. and Tuytelaars, T., 2014. A testbed for cross-dataset analysis. In *European Conference on Computer Vision*, 18–31. Springer. (cited on page 99)

Tran, A. T.; Nguyen, C. V.; and Hassner, T., 2019. Transferability and hardness of supervised classification tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 1395–1405. (cited on pages 79 and 105)

Van Horn, G.; Mac Aodha, O.; Song, Y.; Cui, Y.; Sun, C.; Shepard, A.; Adam, H.; Perona, P.; and Belongie, S., 2018. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8769–8778. (cited on pages 62 and 64)

Vapnik, V., 1992. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, 831–838. (cited on page 56)

Vinyals, O.; Blundell, C.; Lillicrap, T.; Wierstra, D.; et al., 2016. Matching networks for one shot learning. *Advances in neural information processing systems*, 29 (2016), 3630–3638. (cited on page 101)

Wah, C.; Branson, S.; Welinder, P.; Perona, P.; and Belongie, S., 2011. The caltech-ucsd birds-200-2011 dataset. (2011). (cited on pages 65 and 93)

Wan, R.; Xiong, H.; Li, X.; Zhu, Z.; and Huan, J., 2019. Towards making deep transfer learning never hurt. In *2019 IEEE International Conference on Data Mining*

(ICDM), 578–587. IEEE. (cited on pages 68, 72, 73, 76, 77, 103, 124, 138, 140, 147, and 154)

Wang, H.; Wang, Y.; Zhou, Z.; Ji, X.; Gong, D.; Zhou, J.; Li, Z.; and Liu, W., 2018. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 5265–5274. (cited on page 88)

Wang, M. and Deng, W., 2018. Deep visual domain adaptation: A survey. *Neurocomputing*, 312 (2018), 135–153. (cited on page 99)

Wang, X.; Jin, Y.; Long, M.; Wang, J.; and Jordan, M., 2019a. Transferable normalization: Towards improving transferability of deep neural networks. (2019). (cited on pages 61, 67, 68, and 74)

Wang, Y.; Plested, J.; and Gedeon, T., 2020a. Multitune: Adaptive integration of multiple fine-tuning models for image classification. In *International Conference on Neural Information Processing*, 488–496. Springer. (cited on page 125)

Wang, Y.; Yao, Q.; Kwok, J. T.; and Ni, L. M., 2020b. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53, 3 (2020), 1–34. (cited on pages 55, 58, 100, and 101)

Wang, Z.; Dai, Z.; Póczos, B.; and Carbonell, J., 2019b. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11293–11302. (cited on pages 61, 67, and 68)

Weiss, K.; Khoshgoftaar, T. M.; and Wang, D., 2016. A survey of transfer learning. *Journal of Big data*, 3, 1 (2016), 9. (cited on page 68)

Wen, Y.; Zhang, K.; Li, Z.; and Qiao, Y., 2016. A discriminative feature learning approach for deep face recognition. In *European conference on computer vision*, 499–515. Springer. (cited on page 88)

Wightman, R., 2019. Pytorch image models. https://github.com/rwightman/pytorch-image-models. doi:10.5281/zenodo.4414861. (cited on pages 133 and 141)

Williams, R. J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8, 3-4 (1992), 229–256. (cited on pages 48, 50, 51, 127, and 132)

Williams, R. J. and Peng, J., 1991. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3, 3 (1991), 241–268. (cited on pages 48, 50, and 51)

Wu, Y.; Hassner, T.; Kim, K.; Medioni, G.; and Natarajan, P., 2018. Facial landmark detection with tweaked convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 40, 12 (2018), 3067–3074. (cited on pages 82 and 111)

Wu, Y. and He, K., 2018. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, 3–19. (cited on pages 37 and 77)

Xiao, J.; Hays, J.; Ehinger, K. A.; Oliva, A.; and Torralba, A., 2010. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE computer society conference on computer vision and pattern recognition*, 3485–3492. IEEE. (cited on page 65)

Xie, Q.; Luong, M.-T.; Hovy, E.; and Le, Q. V., 2020. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10687–10698. (cited on pages 92 and 94)

Yalniz, I. Z.; Jégou, H.; Chen, K.; Paluri, M.; and Mahajan, D., 2019. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, (2019). (cited on page 94)

Yani, M. et al., 2019. Application of transfer learning using convolutional neural network method for early detection of terry's nail. In *Journal of Physics: Conference Series*, vol. 1201, 012052. IOP Publishing. (cited on page 33)

Yao, Y.; Plested, J.; and Gedeon, T., 2018a. Deep feature learning and visualization for eeg recording using autoencoders. In *International Conference on Neural Information Processing*, 554–566. Springer.

Yao, Y.; Plested, J.; and Gedeon, T., 2018b. A feature filter for eeg using cyclegan structure. In *International Conference on Neural Information Processing*, 567–576. Springer.

Yao, Y.; Plested, J.; and Gedeon, T., 2020. Information-preserving feature filter for short-term eeg signals. *Neurocomputing*, 408 (2020), 91–99.

Yao, Y.; Plested, J.; Gedeon, T.; Liu, Y.; and Wang, Z., 2019. Improved techniques for building eeg feature filters. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–6. IEEE.

Yepez, J. and Ko, S.-B., 2020. Stride 2 1-d, 2-d, and 3-d winograd for convolutional neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28, 4 (2020), 853–863. (cited on page 32)

Yoon, J.; Arik, S.; and Pfister, T., 2020. Data valuation using reinforcement learning. In *International Conference on Machine Learning*, 10842–10851. PMLR. (cited on page 96)

Yosinski, J.; Clune, J.; Bengio, Y.; and Lipson, H., 2014. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, 3320–3328. (cited on pages 62, 67, 70, 80, 81, 82, 84, 90, 98, 103, 104, 106, 110, 111, 112, 113, 115, 118, 120, 124, 125, 128, 131, 138, 139, and 146)

You, K.; Kou, Z.; Long, M.; and Wang, J., 2020. Co-tuning for transfer learning. *Advances in Neural Information Processing Systems*, 33 (2020). (cited on page 75)

Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D., 2017a. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, (2017). (cited on pages 38 and 77)

Zhang, J.; Li, W.; and Ogunbona, P., 2017b. Transfer learning for cross-dataset recognition: a survey. *arXiv preprint arXiv:1705.04396*, (2017). (cited on page 99)

Zheng, Y.; Pal, D. K.; and Savvides, M., 2018. Ring loss: Convex feature normalization for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 5089–5097. (cited on page 88)

Zhou, B.; Lapedriza, A.; Khosla, A.; Oliva, A.; and Torralba, A., 2017. Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40, 6 (2017), 1452–1464. (cited on pages 63 and 93)

Zhu, L.; Arik, S. O.; Yang, Y.; and Pfister, T., 2019a. Learning to transfer learn. (2019). (cited on page 96)

Zhu, L.; Arik, S. Ö.; Yang, Y.; and Pfister, T., 2020. Learning to transfer learn: Reinforcement learning-based selection for adaptive transfer learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII 16*, 342–358. Springer. (cited on page 125)

Zhu, R.; Zhang, S.; Wang, X.; Wen, L.; Shi, H.; Bo, L.; and Mei, T., 2019b. Scratchdet: Training single-shot object detectors from scratch. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2268–2277. (cited on pages 91 and 138)

Zhuang, F.; Qi, Z.; Duan, K.; Xi, D.; Zhu, Y.; Zhu, H.; Xiong, H.; and He, Q., 2020a. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, (2020). (cited on page 68)

Zhuang, P.; Wang, Y.; and Qiao, Y., 2020b. Learning attentive pairwise interaction for fine-grained classification. In *Proceedings of the AAAI Conference on Artificial*

Intelligence, vol. 34, 13130–13137. (cited on pages 143 and 151)

Zoph, B.; Ghiasi, G.; Lin, T.-Y.; Cui, Y.; Liu, H.; Cubuk, E. D.; and Le, Q. V., 2020. Rethinking pre-training and self-training. *arXiv preprint arXiv:2006.06882*, (2020). (cited on pages 68, 76, 86, 92, 97, 105, 140, and 154)

Zoph, B. and Le, Q. V., 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, (2016). (cited on pages 52, 53, 124, 125, 127, and 132)

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V., 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8697–8710. (cited on pages 52 and 53)