Kernel Support Vector Machines and Convolutional Neural Networks

Shihao Jiang*†, Richard Hartley*†, Basura Fernando*†
*Australian National University
†Australian Centre for Robotic Vision, Australia

Abstract—**Convolutional Neural Networks (CNN) have achieved great success in various computer vision tasks due to their strong ability in feature extraction. The trend of development of CNN architectures is to increase their depth so as to increase their feature extraction ability. Kernel Support Vector Machines (SVM), on the other hand, are known to give optimal separating surfaces by their ability to automatically select support vectors and perform classification in higher dimensional spaces. We investigate the idea of combining the two such that best of both worlds can be achieved and a more compact model can perform as well as deeper CNNs. In the past, attempts have been made to use CNNs to extract features from images and then classify with a kernel SVM, but this process was performed in two separate steps. In this paper, we propose one single model where a CNN and a kernel SVM are integrated together and can be trained end-to-end. In particular, we propose a fully-differentiable Radial Basis Function (RBF) layer, where it can be seamless adapted to a CNN environment and forms a better classifier compared to the normal linear classifier. Due to end-to-end training, our approach allows the initial layers of the CNN to extract features more adapted to the kernel SVM classifier. Our experiments demonstrate that the hybrid CNN-kSVM model gives superior results to a plain CNN model, and also performs better than the method where feature extraction and classification are performed in separate stages, by a CNN and a kernel SVM respectively.**

I. Introduction

The trend of the development of deep learning, is making networks deeper by adding more layers to the architecture. The argument is that deeper neural networks have larger capacity and can extract more abstract features from the input data. Thus, they often give better performance than shallower networks. This is evidenced by the top-5 error on ImageNet validation set of ResNet models. ResNet models with layers 50, 101 and 152 give the top-5 error rates (%) 5.25, 4.60 and 4.49 respectively [1].

However, the trend of going deep in deep networks is not without its drawbacks. One obvious problem is that as the architecture becomes deeper, the demand for computational resources also increases accordingly, which includes both memory and computation time. ResNet-50, ResNet-101 and ResNet-152 have 25.5 million, 44.5 million and 60.3 million parameters respectively. Also, longer training time slows down the progress for research and development.

The reason for adding layers in a deep network, is to enhance the feature extraction ability of a network, such that the extracted features can be linearly separable and thus correctly classified in classification problems. Since a CNN can be viewed as an integration of feature extractors and classifiers, one would ask whether improving the classifier in a network could achieve the same performance as adding more layers to the network, or improving performance without adding a significant number of parameters.

To this end, we revisit a more traditional approach, Support Vector Machines (SVMs) [2], in particular kernel SVMs, and adapt them into a CNN environment. The motivation behind this work is that although CNNs have proven effective in feature extraction due to their multi-layer structure, they are not always optimal for classification. The linear classifier of a CNN can potentially be replaced by a classifier able to separate more complex surfaces, thus improve classification accuracy. This leads us to investigate kernel SVMs, which were the standard method of solving classification problems before CNN became popular. Kernel SVMs compute optimal separating surfaces with a maximal margin criterion and are able to classify data not linearly separable. We are particularly interested in the interaction of kernel SVMs and CNNs and see whether the hybrid architecture can improve performance.

In this paper, we propose a fusion CNN-kSVM model, with CNN as the feature extractor and Gaussian-kernel SVM as the classifier, replacing the linear classifier of an original CNN. Previous works [3], [4] also investigated the idea of using a CNN to extract features and classify with an SVM, and claimed to achieve better results than a CNN softmax classifier. However, two separate models were built and optimized separately. In our case, we combine a CNN and a kernel SVM into a single model. In particular, our model consists of a fully-differentiable Gaussian Radial Basis Function (RBF) layer, which combined with a fully-connected layer, forms a kernel SVM classifier. The kernel SVM classifier can be optimized together with the CNN using gradient descent methods. To our knowledge, this is the first time an SVM is embedded in a CNN architecture and the two are optimized together.

We have conducted experiments on both binary classification problems and multi-class classification problems. For binary classification, we use a dataset containing dog and cat images, with 20000 images for training and 4000 images for testing. We demonstrate that the hybrid CNN-kSVM model yields significant performance improvement compared with a stand-alone CNN, without significantly increasing the number of parameters. We also compare with using a libSVM [5] to classify CNN features, the same as [4], and have discovered that our approach yields better results. For multi-class classification, we use the CIFAR-10 dataset [6]. We have tested our kernel SVM classifier based on a small CNN model and a deep CNN model, and have found improvements in classification results in both architectures.

II. Related Works

A. Support Vector Machines

Support Vector Machines developed by Cortes and Vapnik [7] are one of the most robust classifiers due to their ability to find optimal separating planes with maximum margin. Kernel SVMs can also implicitly map their inputs to very high dimensional feature spaces and perform linear classification in feature space, thus performing non-linear classification in input spaces.

In the case of a CNN, the final fully-connected layer used for classification is no different to a linear SVM, except optimized with gradient-based approaches. Therefore, various approaches have attempted the idea of combining CNNs and SVMs.

B. Combination of CNNs and SVMs

Various papers have explored the idea of combining CNNs and SVMs to take advantage of the two different models.

Huang *et al.* [3] first implemented the idea of combining a CNN with an SVM in 2006, when CNN was not the major model used to solve image classification problems. In [3], a convolutional net was first trained in a supervised way as a feature extractor. The features were extracted from the final layer of the network and were used as input to train a Gaussian SVM. In our approach, we use the extracted features to initialize Gaussian centres and train the hybrid model end-to-end with gradient descent. The paper also mentioned the drawback for SVMs, that is they are not suitable for problems with large training set. Our on-line method of training does not suffer from this problem.

Tang [8] proposed the idea of replacing the softmax layer with a linear SVM and training the CNN with a margin-based loss instead of cross-entropy loss. The loss function the author used was an L2-SVM instead of the standard hinge loss. They demonstrated superior performance on MNIST, CIFAR-10 datasets and a Kaggle competition compared to using a softmax top layer. This work changed the loss function rather than the network structure, whereas in our approach we modified the linear classifier into a Gaussian-kernel classifier.

Niu *et al.* [4] implemented the idea of combining CNN and kernel SVM to solve handwritten digits recognition problem. The idea was largely similar to [3]. Firstly an original CNN was trained until the training process converged. Then they replaced the output layer with an SVM with RBF kernel and trained the SVM using CNN feature vectors. They used libSVM software to build their SVM model. After the SVM classifier is well trained, it performs the recognition task and makes decisions based on features extracted by the CNN. Elleuch *et al.* applied CNN and SVM approaches to solve the more challenging Arabic Handwritten Recognition. The network structure was similar to [4] except that dropout was added to give further accuracy improvement.

Shi *et al.* [9] incorporated Radial Basis Function (RBF) in a new setting. [9] solved action anticipation problem by using an RNN to extract features and then classifying with a multilayer perceptron with RBF kernels. Even though this paper solved a different problem, it demonstrated the advantage of RBF kernels in representing complex feature mappings.

In our implementation, we build an end-to-end model such that a CNN and SVM are combined together and optimized simultaneously. CNN feature vectors are used to initialize Gaussian-kernel SVM. Then the model is trained end-to-end such that the initial CNN layers are optimized for the kernel SVM classification layer.

III. Method

We first highlight the essential similarity between a linear SVM and the final fully-connected layer of a CNN. Then we introduce the construction of a kernel SVM and how it can be incorporated into a CNN. We then explain our motivation of combining a kernel SVM with a CNN and present our network architecture.

A. Linear SVM

We formulate the idea of SVMs in a way that is related to neural networks.

A linear SVM takes a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ as input, and outputs a value $y \in \mathbb{R}$, defined by

$$y = \langle \mathbf{w}, \mathbf{x} \rangle + b, \tag{1}$$

where a vector $\mathbf{w} \in \mathbb{R}^n$ and offset $b \in \mathbb{R}$ are learnt during a training process. This provides a classifier for input $\mathbf{x}$; the vector $\mathbf{x}$ is assigned to class 1 or -1 depending on whether $y > 0$ or $y < 0$.

As such, a linear SVM is identical to a final "fully-connected" layer in a CNN, which performs classification. A CNN can thus be viewed as two parts: one part performs feature extraction, which implements a non-linear function $\Phi : \mathbb{R}^m \to \mathbb{R}^n$ (where m is the dimension of the input of the CNN); the second part is the final fully-connected layer, which is essentially a linear SVM and is used for classification.

Commonly, a stand-alone SVM is trained using convex optimization methods in order to find a maximum margin hyperplane, whereas in the case of a CNN, the linear SVM is trained together with the non-linear function Φ, often with gradient-based methods.

Figure 1 illustrates how a CNN can be viewed as a non-linear function Φ and a linear SVM.

B. Kernel SVM

A kernel SVM can be described in a similar fashion. An imagined non-linear function $\Phi : \mathbb{R}^n \to \mathcal{H}$ is applied on the input to transform it to a higher dimensional space $\mathcal{H}$, possibly inifinite-dimensional. $\mathcal{H}$ is a Hilbert space, where inner products can be taken. The introduction of the mapping Φ is to highlight the essential similarity between kernel SVMs and linear SVMs. Given an input $\mathbf{x} \in \mathbb{R}^n$, we may carry out two-label classification by forming the expression

$$y = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle_{\mathcal{H}} + b, \tag{2}$$

where $\mathbf{w} \in \mathcal{H}$ and $b \in \mathbb{R}$.

In a kernel SVM, one need only define a suitable kernel K, instead of explicitly performing the mapping Φ and taking inner products in Hilbert space. This is known as the "kernel

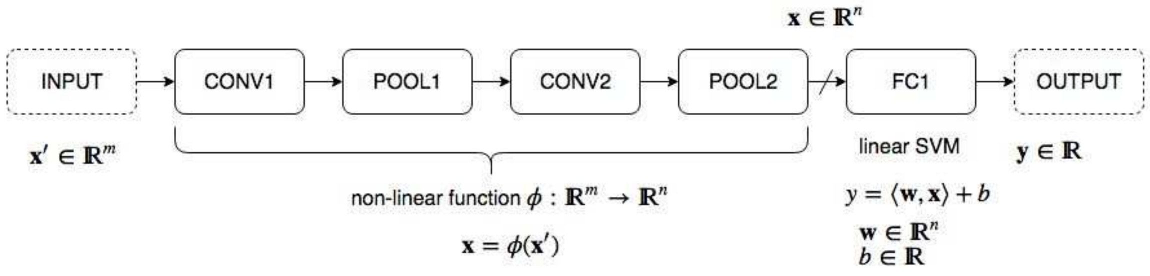

$\mathbf{x} \in \mathbb{R}^n$
INPUT
CONV1
POOL1
CONV2
POOL2
FC1
OUTPUT
$\mathbf{x}' \in \mathbb{R}^m$
linear SVM
$\mathbf{y} \in \mathbb{R}$
non-linear function $\phi : \mathbb{R}^m \to \mathbb{R}^n$
$\mathbf{x} = \phi(\mathbf{x}')$
$y = \langle \mathbf{w}, \mathbf{x} \rangle + b$
$\mathbf{w} \in \mathbb{R}^n$
$b \in \mathbb{R}$

Fig. 1: An example CNN which can be viewed as a non-linear function Φ and a linear SVM classifier. During training, the output is fed to a loss layer where it is compared with ground truth.

trick". The kernel function is related to the non-linear mapping Φ through the formula

$$K(\mathbf{x}_1, \mathbf{x}_2) = \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}}. \tag{3}$$

One commonly used kernel is the Gaussian Radial Basis Function (RBF) kernel, defined by

$$K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\lambda \|x_1 - x_2\|^2). \tag{4}$$

We make the assumption that $\mathbf{w}$ lies in the span of $\Phi(\mu_i)$, for some values $\mu_i \in \mathbb{R}^n$. Thus, one may write

$$\mathbf{w} = \sum_{i=1}^{k} a_i \Phi(\mu_i),$$

where $a_i \in \mathbb{R}$ are the scalars. The μ_i are referred to as the support vectors. With this assumption, we may write

$$\begin{aligned} y &= \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle_{\mathcal{H}} + b \\ &= \left\langle \sum_{i=1}^{k} a_i \Phi(\mu_i), \Phi(\mathbf{x}) \right\rangle_{\mathcal{H}} + b \\ &= \sum_{i=1}^{k} a_i \langle \Phi(\mu_i), \Phi(\mathbf{x}) \rangle_{\mathcal{H}} + b \\ &= \sum_{i=1}^{k} a_i K(\mu_i, \mathbf{x}) + b. \end{aligned}$$

In the case of the RBF kernel, this becomes

$$y(\mathbf{x}) = \sum_{i=1}^{k} a_i \exp(-\lambda \|\mathbf{x} - \mu_i\|^2) + b. \tag{5}$$

Compared with Equation (1), which is classifying input with a hyperplane, we apply multiple Gaussian functions to approximate the separating function.

In normal practice with SVMs, the Gaussian centres, μ_i are chosen from among the training samples $\{\mathbf{x}_j\}$, but we propose relaxing this restriction so that the μ_i are trainable parameters.

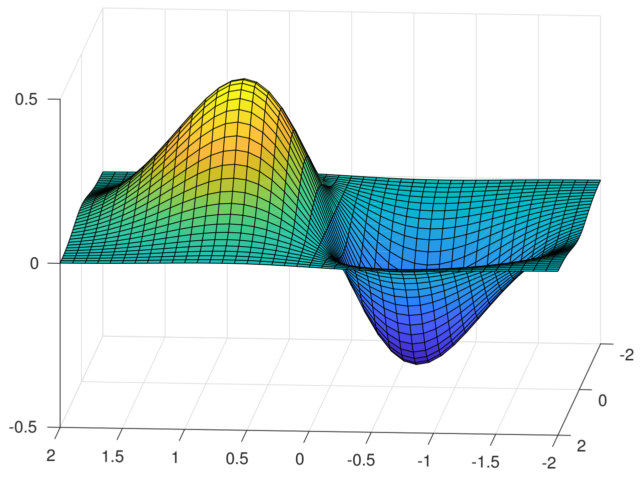
It is easy to see that a kernel SVM is a superior classifier compared with a linear SVM. Linear SVMs can only classify data by a hyperplane, while kernel SVMs can classify data based on more complex surfaces. In the case of a Gaussian RBF kernel, by adding a sufficient number of Gaussians, one can clearly describe the data separation between different classes and thus improve classification accuracy. Additionally, a linear SVM can be regarded as a special case of an RBF-kernel SVM using two Gaussians. For example, $y = \exp(-\|\mathbf{x} - \mathbf{x}_1\|^2) - \exp(-\|\mathbf{x} - \mathbf{x}_2\|^2)$ will be positive for points closer to $\mathbf{x}_1$ and negative for points closer to $\mathbf{x}_2$. This creates a hyperplane placed mid-way between $\mathbf{x}_1$ and $\mathbf{x}_2$. This is demonstrated in the surface plot and contour plot in Figure 2. Thus, a kernel SVM with two support vectors should not perform worse than a linear SVM. By adding more support vectors, we are able to generate a more complex separating surface and thus hope to achieve better performance. This is demonstrated in Figure 3.

The approach of placing Gaussians in the Euclidean space is similar to the idea of Parzen windows [10], in which distributions are approximated by summing normal Gaussian distributions placed at locations within the support region of the distribution. In the method suggested here by RBF-kernel SVMs, approximations to the support regions of the two classes are formed by summing Gaussians placed within those regions, and classifying test points according to the values of the approximation functions to the two different support regions. Thus, RBF-kernel SVMs may be understood in this simple way, without the machinery of Hilbert spaces, or the positive-definiteness of the kernel. In fact, the approach applies equally well to any of the "kernel" used to approximate a function.

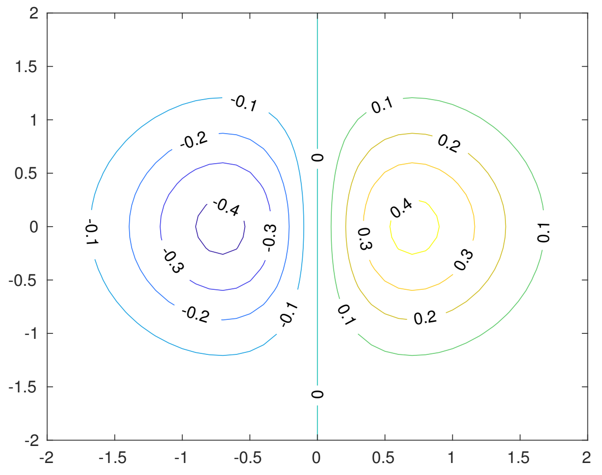
C. Network Architecture for Binary Classification

We explain our CNN-kSVM model architecture for binary classification here. We employ a CNN architecture with four convolution layers and one fully-connected layer. The first two convolution layers have kernel size 5×5 and zero-padding 2. The third convolution layer has kernel size 3×3 and zero-padding 1 and the fourth convolution layer has kernel size 7×7. All convolution layers have stride 1. Each of the convolution layers is followed by a rectified linear unit (ReLU) activation function. Three max pooling layers are added after convolutions to subsample feature maps. The first two max pooling layers have kernel size 4×4 and stride 4 and the third max pooling layer has kernel size 2×2 and stride 2. The architecture is demonstrated in Figure 4.

The kernel SVM classifier implements a Gaussian RBF

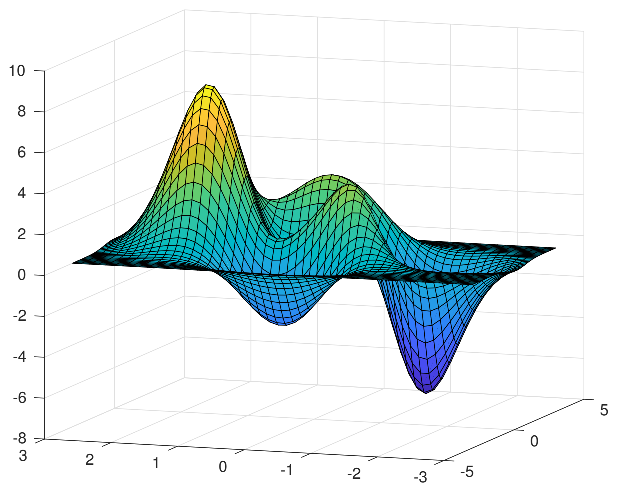


(a) Surface plot of an SVM classifier composed by two Gaussians with weight -1 and 1. Equivalent to a binary linear classifier.

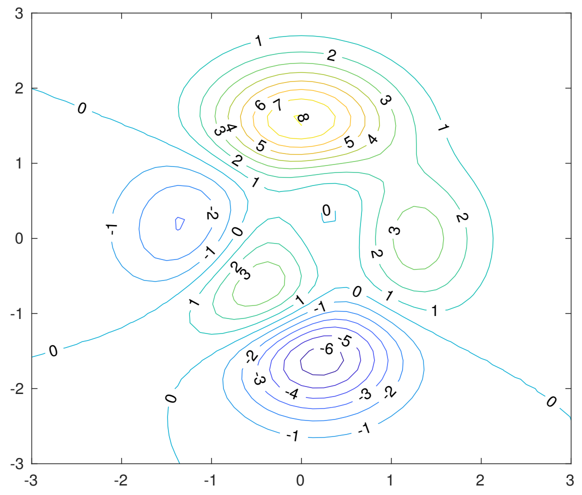


(b) Contour plot of an SVM classifier composed by two Gaussians with weight -1 and 1. Level 0 indicates the separating plane, in this case, a line in $\mathbb{R}^2$.

Fig. 2: Kernel SVM classifier formed by 2 Gaussian functions with weights -1 and 1



(a) Surface plot of an SVM classifier composed by five Gaussians.



(b) Contour plot of an SVM classifier composed by five Gaussians. Level 0 indicates the separating plane. One can see that this classifier is able to separate data that is not linearly-separable.

Fig. 3: Kernel SVM classifier formed by 5 Gaussian functions

function. The classifier consists of an RBF layer and a fully-connected layer; which together can be represented by

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{N} a_i \exp\left(-\frac{\|\mathbf{x}-\mu_i\|^2}{\sigma_i^2}\right) + b. \tag{6}$$

In Equation 6, the vector $\mathbf{x} \in \mathbb{R}^{500}$ is the input for the classifier and $\hat{y} \in \mathbb{R}$ is the predicted output scalar. The parameters are μ_i, σ_i, a_i and b. The $\mu_i \in \mathbb{R}^{500}$ represent Gaussian means and $\sigma_i \in \mathbb{R}$ represent the standard deviations for Gaussian functions; both are in the RBF layer and learned through training. There are N Gaussians in the RBF layer and the 500-dimensional input $\mathbf{x}$ is duplicated N times for N Gaussians. The parameters a_i and b represent weights and bias of the fully-connected layer and perform a linear combination on the outputs of the Gaussians; they are also learned through training. The complete model is illustrated in Figure 5.

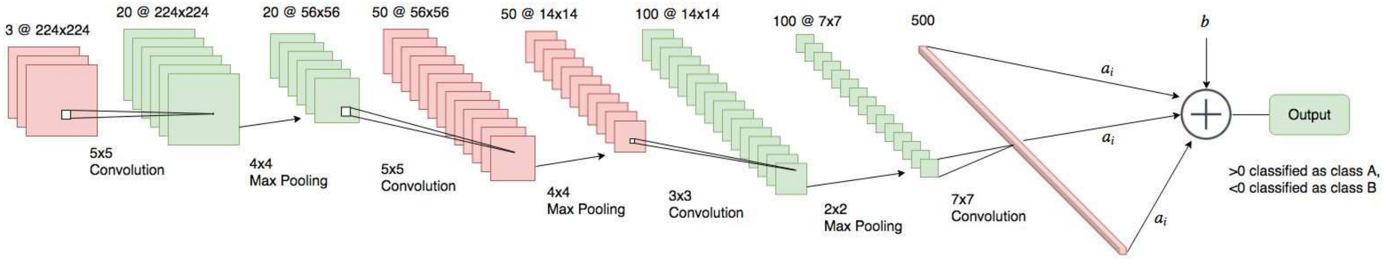


Fig. 4: Architecture of the original CNN model in binary classification

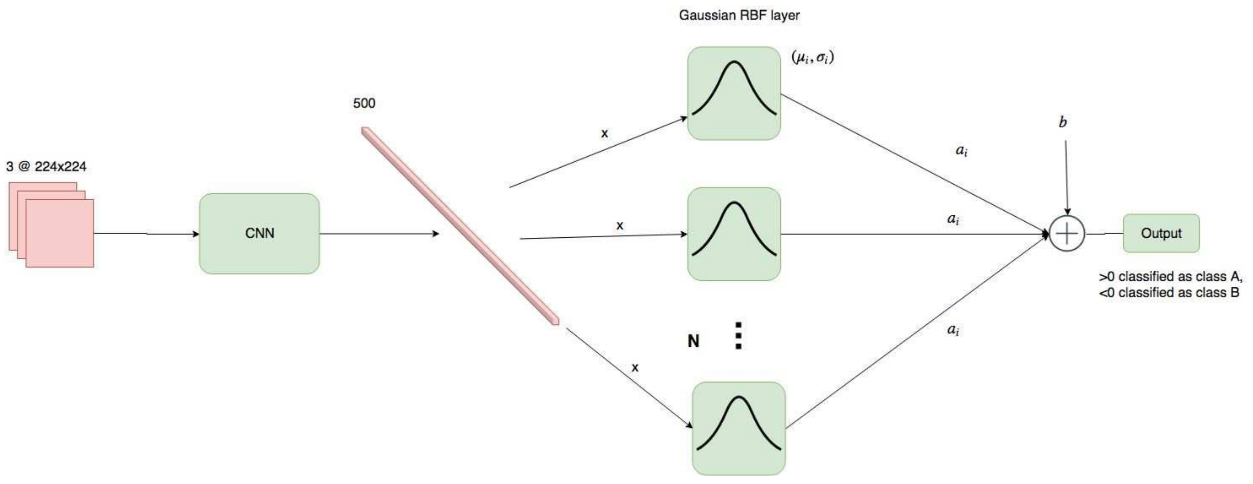


Fig. 5: Architecture of the proposed model combining CNN and RBF-kernel SVM. In our experiments, N = 2 or N = 20.

IV. Experiments

A. Binary Classification

We first evaluate our CNN-kSVM model on a binary classification problem. We use a binary dataset containing dog and cat images. The dataset contains $24,000$ images, with $12,000$ cats and $12,000$ dogs. Sample images are shown in Figure 6. We separate the dataset such that $20,000$ images are used for training and $4,000$ images are used for testing, with equal number of dog and cat images in both. Images were resized to 224×224 when training the network. We label cats as 0 and dogs as 1.

We first train a CNN from scratch with architecture shown in Figure 4, which achieves test set performance 91.625%. The CNN is trained with data augmentation including random crop and random horizontal flip. The trained CNN is our baseline model. We then investigate how a kernel SVM classifier can improve the performance. We pass training images through the trained CNN and collect $20,000$ 500-dimensional feature vectors.

1) LibSVM: As a baseline, following [4]: we pass the collected 500-dimensional feature vector into a RBF kernel SVM classifier. LibSVM [5] is used to build the classifier. We use grid search to determine C and γ, which are two parameters for RBF kernel and the best classification result obtained is 92.05%. We also try initializing the RBF layer with support vectors generated by LibSVM and then training the hybrid model but do not observe significant improvement.

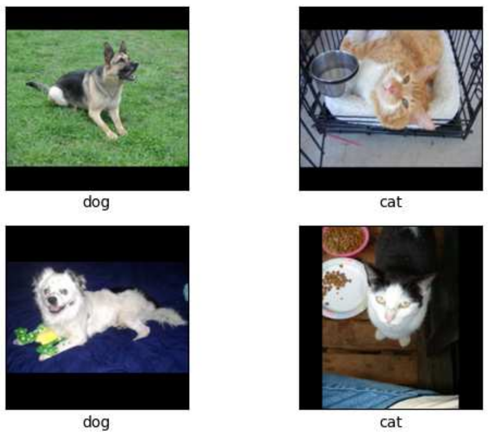


Fig. 6: Sample images of the dog and cat dataset, partly taken from Kaggle dataset.

2) N = 2: We first initialize a kernel SVM classifier with 2 Gaussians. These two Gaussians are initialized such that they form a separating hyperplane equivalent to the linear classifier (fully connected layer in CNN). The means of the two Gaussians are selected in the following manner. We first compute the mean feature vector for each class. Then we find two Gaussian means that are closest to the two mean feature vectors and are also symmetrical with respect to the hyperplane represented by the linear classifier. This initialization ensures that the CNN-kSVM model gives the exactly same classification accuracy as the original CNN. We then train this model and have obtained 92.85% classification accuracy.

3) N = 20: We then add multiple Gaussians for each class. The Gaussian centres are also initialized such that they are symmetrical with respect to the separating hyperplane and also have a good coverage of the data cloud. We train this model and achieve a classification accuracy of 93.2%.

TABLE I: Test Accuracy for Different Models

Model	Test Accuracy
Baseline CNN	91.625%
LibSVM [4]	92.05%
CNN with RBF layer, N=2	92.85%
CNN with RBF layer, N=20	93.2%

Test results are shown in Table I. We can see that the hybrid CNN-kSVM model is able to increase classification accuracy from 91.625% to 93.2% with N = 20. This is 18.8% decrease in error rate from 8.375% to 6.8%, which is a significant improvement when only increasing 0.38% number of parameters. The number of parameters for each model is shown in Table II, and we can see that there is not much difference in them.

TABLE II: Model Parameters for Different Models

Model	Parameters
Baseline CNN	2522671
CNN with RBF layer, N=2	2523175
CNN with RBF layer, N=20	2532211

Also, given the size of the test set 4000, the result 93.2% is better than 91.625% at an significance level of 99.987%. The result given by [4], which is an improvement from 91.625% to 92.05%, has a significance level of 99.66%. This probability shows that this improvement is not due to chances.

B. Multi-class Classification

We also apply our model on a multi-class classification problem. The dataset which we evaluate our model on is CIFAR-10 [6]. The CIFAR-10 dataset contains 60,000 32×32 colour images in 10 classes, with 6,000 images in each class. The training set contains 50,000 images and the test set contains 10,000 images.

We have experimented on two CNN architectures: one is a small model, as shown in Table III; the other is a deep CNN model, which is DenseNet [11]. In our small CNN model in Table III, each 5×5 convolution has stride 1 and zero-padding 2. The pooling layers have stride 2 and zero-padding (0,1,0,1).

TABLE III: Small CNN Model

Conv 5×5, 32, Max Pooling 3×3 stride=2, ReLU
Conv 5×5, 32, ReLU, Avg Pooling 3×3 stride=2
Conv 5×5, 64, ReLU, Avg Pooling 3×3 stride=2
Conv 4×4, 64, ReLU
Fully Connected, 10

Similar to the architecture shown in Figure 5, we also extract features before the final fully-connected layer of a CNN, then pass them into our kernel SVM classifier, which contains an RBF layer with N Gaussian functions and a fully-connected layer.

1) Small CNN: For the small CNN model, we first train the CNN model and the test set accuracy is 80.65%. Then following the baseline [4], we extract 64-dimensional feature vectors before fully-connected layer and classify with RBF-kernel SVM using libSVM software. Surprising, this does not give better results than original CNN.

In our CNN-kSVM model, we first set $N = 10$ by having one Gaussian function per class. The mean of each Gaussian is initialized by computing the mean of the 64-dimensional feature vectors for that particular class. After training, this gives the same results as the original CNN. Then we add more Gaussian functions for each class, to have a better representation of the feature space. We perform PCA on the 64-dimensional feature vectors for each class to obtain the first and second principal components. We then place the Gaussian means along the axis of the first and second principal components. When we have 2 Gaussian functions for each class, i.e. $N = 20$, the test accuracy is 80.95%. When we have 5 Gaussian functions for each class, i.e. $N = 50$, the test accuracy is 81.65%, which gives us 1% improvements from the baseline CNN. The complete results are shown in Table IV.

TABLE IV: Test Accuracy for CIFAR-10, Small CNN

Model	Test Accuracy
Baseline CNN	80.65%
LibSVM [4]	80.62%
CNN with RBF layer, N=10	80.65%
CNN with RBF layer, N=20	80.95%
CNN with RBF layer, N=50	81.65%

2) DenseNet: We also test our RBF layer on a deeper CNN, to investigate whether our kernel SVM model will be beneficial to a network that already gives high classification accuracy on test set. Here, we use the basic DenseNet model with 40 layers. We train DenseNet-40 on CIFAR-10 and the test set accuracy was 94.58%. We then insert our RBF layer to form the DenseNet with RBF layer architecture, where we place 200 Gaussian functions in total, $N = 200$. This gives us a test set accuracy of 94.8%, which is a 4% decrease in error rate, from 5.42% to 5.2%. The result is shown in Table V.

TABLE V: Test Accuracy for CIFAR-10, DenseNet

Model	Test Accuracy
DenseNet	94.58%
DenseNet with RBF, N=200	94.8%

V. Conclusion

In this paper, a new hybrid CNN-kSVM model has been proposed. In particular, we leverage the feature extraction capability of CNNs and the optimal classification performance of kernel SVMs and demonstrate that combining two models can give superior performance to a single CNN. Moreover, rather performing feature extraction and classification in two different steps like in previous works, we are able to combine CNN and kernel SVM into a single model and perform end-to-end training. Various experiment results have demonstrated that this model is able to improve the performance of an original CNN, in both binary classification and mutil-class classification problems.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[3] F. J. Huang and Y. LeCun, "Large-scale learning with svm and convolutional for generic object categorization," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1. IEEE, 2006, pp. 284–291.

[4] X.-X. Niu and C. Y. Suen, "A novel hybrid cnn–svm classifier for recognizing handwritten digits," *Pattern Recognition*, vol. 45, no. 4, pp. 1318–1325, 2012.

[5] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[6] A. Krizhevsky, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[7] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[8] Y. Tang, "Deep learning using linear support vector machines," *arXiv preprint arXiv:1306.0239*, 2013.

[9] Y. Shi, B. Fernando, and R. Hartley, "Action anticipation with rbf kernelized feature mapping rnn," in *ECCV*, 2018.

[10] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, pp. 832–837, 1956.

[11] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, vol. 1, no. 2, 2017, p. 3.