Modified MNIST Classification Task

Team yimapa - COMP551-001

Yi Qiao Wang
yi.wang14@mail.mcgill.ca
260682080

Mathew Wright
mathew.wright@mail.mcgill.ca
260681105

Patrick Beland
patrick.beland2@mail.mcgill.ca
260688796

I. Introduction

Image classification is a widely studied and highly applicable field in machine learning. A popular database used in computer vision to train and evaluate image processing systems is the MNIST, a dataset of handwritten digits.

For this project, we were tasked with a modification to the classic goal of classifying the digits of the MNIST dataset. This variation modifies the original MNIST dataset and requires the construction of a machine learning model capable of identifying the handwritten digit encompassing the largest surface area of an image containing other handwritten digits. Each grayscale image is set over some background pattern while each digit contained in the image is rotated and randomly scaled. Provided as part of this modified dataset were 50,000 training examples on which to train our models. These models would then be entered into a Kaggle competition where their performance would be evaluated on 10,000 test cases.

To begin, we present the motivation and methodology employed in tackling this classification task. In particular, the algorithms and machine learning models applied to the problem are described. In addition to the algorithms, we provide details and considerations in the implementation of said models such as feature design and hyper-parameter selection. Subsequently, the results and performance of the three considered models are given and investigated in detail. In conclusion, an analysis of the strengths and weaknesses of the taken approach as well as avenues for future work are discussed.

II. Methodology

The methodology used throughout the project as well as its justification is presented below.

A. Classification Algorithms

To better understand the subsequent sections, a brief overview of the the algorithms pertaining to the machine learning models utilized in the classification task is provided.

As baseline classifiers, we choose to use the linear support vector machine as well as the decision tree. To compare with these baselines, two neural network models were employed. The first of which was a fully connected feed-forward neural network. In addition to the fully connected neural network, a convolutional neural network was considered.

In the implementation of these aforementioned models, several popular software libraries were used. Throughout all three models and in data manipulation, NumPy was used for general purpose linear algebra operations. In addition to NumPy, specialized libraries were used in the baseline and convolutional neural networks. In these last two models, we took advantage of scikit-learn and PyTorch.

Mathematical notation used in this section is as follows. The input feature matrix is denoted as $X \in \mathbb{R}^{n \times m}$, where an individual example is denoted as $x_i \in \mathbb{R}^m$. Furthermore, we denote the target classes for i examples as $y \in \mathbb{R}^n$ and the weight vector as $\mathbf{w} \in \mathbb{R}^m$.

1) Linear Support Vector Machine:

The support vector machine (SVM) is a discriminative binary classifier that models data as points in $\mathbb{R}^m$ and attempts to find the largest separating hyperplanes between the two classes with respect to some loss function. Once this largest separating hyperplane H, also known as the maximum margin hyperplane, is chosen from the training data, unclassified data is then projected into the same space and then assigned a class based on the decision boundary dictated by H. In the multi-class classification setting involving C classes, scikit-learn uses the *one-against-one* approach in which $\frac{C(C-1)}{2}$ SVM models are learned, one for each pair of distinct classes. For any given SVM the maximum margin hyperplane can be fully specified by its normal vector as well as two support vectors. Indeed, the orientation of H can be viewed as a set of weights $\mathbf{w}$ and its position by some bias $b \in \mathbb{R}^m$, both which can be learned from the data. Given a training data set X and targets $y \in \{-1, 1\}^n$, computing $H = \mathbf{w} + b$ is the equivalent to the optimization task

$$\begin{aligned} \min_{\mathbf{w},b} \quad & L(\mathbf{w}, b) + C \sum_{i=1}^{n} \epsilon_i \\ \text{subject to} \quad & y_i \mathbf{w}^T x_i + b \geq 1 - \epsilon_i, \epsilon_i \geq 0 \\ & \text{for } 1 \leq i \leq n. \end{aligned}$$

where the function $L(\mathbf{w}, b) = \frac{\mathbf{w}^T \mathbf{w}}{2}$ is the loss function, C is a penalty term for misclassification and ϵ_i is the classification error. Furthermore, additional constraints on $\sum_{i=1}^{n} \epsilon_i$ can be imposed to limit the number of misclassified training examples. The penalty term C is viewed as a hyper-parameter and can be optimized for by tuning via cross-validation, for example.

2) Fully Connected Feed-Forward Neural Network:

A neural network, or artificial neural network (ANN), is a machine learning model that takes its inspiration and namesake

from the biological neural networks found in animal brains. This class of algorithms are highly capable and can perform classification tasks without a priori domain specific knowledge programmed into the model. The fundamental unit of an ANN are nodes which themselves are organized into layers. Each node in all but the first (input) layer and the last (output) layer accepts inputs and produces and output. These intermediate layers are known as *hidden* layers. Information in the form of data passes through the network by traversing through the hidden layers, starting at the input layer and culminating at the output layer.

In the fully connected feed-forward setting, every node in layer i is connected to every node in layer $i+1$ through some weight. That is, if there are n_i nodes in layer i, each node ξ_k in layer $j = i + 1$ has some weight vector $\mathbf{w}_k^{ij}$ of dimension $(n\times 1)$ associated with its n inputs. Each node has an activation function a that is a function of its inputs multiplied by its weight vector $\mathbf{w}^T X$. The output of $a(\mathbf{w}^T X)$ is then fed forward to each node in the subsequent hidden layer. The ANN can then classify some input with some output by interpreting the vector produced by the output layer's activation functions. In our implementation, the sigmoid function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

was chosen as the activation function for each neuron.

To train such a model is an optimization problem of some loss function $L(x_i)$ with respect to the weights in the ANN via gradient descent. This is done via an algorithm known as backpropagation. Backpropagation trains the weights of an ANN in a two-pass approach. The forward pass involves classifying some labeled example x_i and comparing the predicted class p_i with the true class y_i. Classification error is then propagated backwards through the network in a biased manner by adjusting weights that contributed more to the misclassification more aggressively compared to weights that had little effect on the misclassification. Mathematically, these correction terms can be expressed as

Layer	δ
Output Node	$(y - \xi)\sigma'(\xi)$
Hidden Node at Layer l	$(w^{l+1})^T \delta^{l+1} \sigma'(\xi)$

where ξ is the output of some hidden unit and $\sigma'(x) = \big[\sigma(x)(1 - \sigma(x))\big]$.

Our implementation of the fully connected network utilized the mean squared error loss function L defined for prediction p_i of input x_i in the training set as

$$L(p_i) = \big(p_i - y_i\big)^2$$

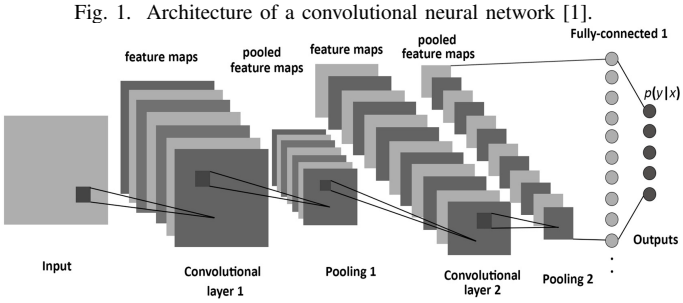
where y_i is the true class.

Once the weights of a network are learned, a new example x_i is then classified by passing x_i into the first layer, which then propagates its signal forward through the entire network. The output of the final layer is what determines which class x_i is predicted to belong to.

The ANN model is very flexible as it contains a large number of parameters. Therefore, it can learn extremely complex functions. Indeed, by the universal approximation theorem, an ANN with a single hidden layer containing finitely many nodes can approximate any continuous function, under some mild regularity conditions. However, in practice, it is often not feasible to use only one layer. Therefore, many ANN models utilize architectures involving many hidden layers, which can be viewed as a hyper-parameter of the model. In addition to the number of hidden layers, one can also vary the learning rate of the weights, the number of neurons per layer, the activation function, as well as the loss function. Indeed, it is this flexibility that gives rise to the following special case of the ANN, the convolutional neural network.

3) Convolutional Neural Network:
A convolutional neural network (CNN) is a special case of the regular ANN that is often applied to great success in image recognition tasks. Like the ANN, it's underlying principles are the same. The CNN model comprises of layers of neurons, each with learned weights as the estimated parameters of the CNN. Furthermore, a CNN is trained on example data in the same way as a general ANN, by gradient descent over the space of weights through backpropagation. What differentiates the CNN is its architecture, which exploits an assumed input format, images in this case, to achieve improved performance over its fully connected counterpart.

Fig. 1. Architecture of a convolutional neural network [1].



One main issue with fully connected neural networks is the exploding number of parameters. For example, to handle an image of size $w \times h \times d$, a fully connected network would require some neurons to contain $w \times h \times d$ weights. However, a CNN avoids this problem by sharing weights across the d dimension, as well as imposing constraints on the number of connections in the network from one layer to the next.

To achieve this, a CNN makes use of convolution layers and pooling layers. A convolution layer C comprises of multiple feature maps, one for each of the d input dimensions to C. Each of these feature maps, or filters, share the same set of weights. These filters need not encompass the entire image and thus can specialize to "activate" only when given particular inputs. The size of a filter as well as the number of filters per convolution layer are hyper-parameters of the model.

Furthermore, our CNN model also utilizes pooling layers. Pooling layers seek to reduce the size of the image via some variance reducing, albeit bias inducing, procedure. It is

this pooling mechanism that reduces the number of weights connecting one layer to the next. A pooling layer's kernel size as well as the method it uses to reduce the spacial size of the input are both hyper-parameters of a particular CNN model.

Finally, the last architectural layer that we employed in our CNN is the dropout layer. This layer probabilistically sets individual weights of the previous layer to 0 with some probability p. The purpose of this layer is to control over-fitting to the training set by purposely removing weights and thus simplifying the model.

Another important hyper-parameter in the CNN model is the stride size. Since the filters in both the convolution and pooling layers need not be applied to the entirety of the image but are instead mapped to subsections of the image, the stride parameter specifies the details of this mapping. The stride of a particular layer, be it convolution or pooling, determines how many pixels a filter moves between applications of its filtering operation in the generation of the output of said layer. A layer with a larger stride will produce an output that is smaller and more compressed compared to the same layer using a lower stride.

Prediction in the CNN setting is exactly the same as that in the classical ANN model.

In our implementation of the CNN, the activation of each neuron was set as the rectifier function

$$f(x) = x^+ = \max(0, x).$$

In addition, we chose to optimize the weights with respect to the categorical cross entropy loss function

$$L(x_i) = -\sum_{c=1}^{C} \log(p_{i,c}) y_{i,c}$$

where $p_{i,j}$ is the probability of x_i belonging to class c_j and $y_{i,c}$ is a binary indicator equal to 1 if x_i belongs to c_j and 0 otherwise. By setting the activation function of the output layer to be the soft-max function, the output of the CNN on some input x_i is exactly the vector $[p_{i,1}, p_{i,2}, \ldots, p_{i,C}]$.

B. Feature Design

1) Data Preprocessing:

In attempt to improve performance of the classifiers, a data preprocessing pipeline was used. This preprocessing pipeline takes advantage of some patterns and regularities observed in the data. One of the principle assumptions made is that the elements of interest in an image (the digits) were all white, while the background and noise were various shades of gray and black. Another observation is that human written digits are largely uniform in shape; they are usually written in such a way that they fit into some possibly skewed rectangle. By exploiting these assumptions, powerful transformations to the images are possible.

To facilitate the preprocessing, the computer vision library OpenCV was used. The steps involved in this pipeline are as follows.

1) Thresholding is applied to the image.

Fig. 2. Example image from the training dataset which demonstrates the observed patterns, before preprocessing.

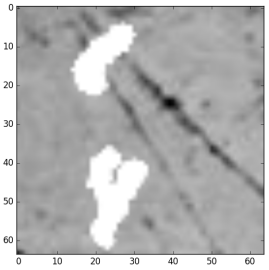
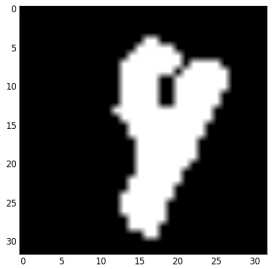


Fig. 3. The same image from 2 after the preprocessing pipeline.



2) The image's contours are then located.
3) For each contour, the minimum enclosing rectangle is determined.
4) For each contour, the minimum enclosing square is inferred from its minimum enclosing rectangle.
5) The contour with the maximum minimum enclosing square C_{max} is identified and a binary mask is applied using its minimum enclosing rectangle.
6) The image is cropped and rescaled to be of dimension $28 \times 28 \times 1$.

The principal motivation behind this preprocessing approach is to improve classification performance by removing as much redundant information and features in the input image as possible. By doing so, we reduce biases caused by the the noise and aid the network in focusing its efforts on only learning the features that influence classification performance. Indeed, in step *(1)* a hard threshold is applied to the image. This hard thresholding alters the pixel value of all pixels p with a new value p^{new} by the following rule

$$p^{new} \leftarrow \begin{cases} 0, & \text{if } p < 250. \\ 255, & \text{otherwise.} \end{cases}$$

This rule makes use of the fact that the pixels values of the handwritten images are white. A small slack in the threshold is given to better handle the transition area between a digit and the background. Next, steps *(2)* - *(6)* involve identifying and then isolating the digit d that encompasses the largest surface area of the image. Since d is the only digit that influences the class of the image, a binary mask is applied to hide all pixels not a part of d. Finally, step *(7)* regularizes and centers d so

that every training image is uniformly shaped and oriented. By doing so, we reduce the image's spatiality and thus the number of parameters of the network, which improves the generalization capability of the CNN by preventing it from over fitting to the training set.

C. Model Optimization Strategy: CNN

The performance of a classifier is highly dependent on how its hyper-parameters are configured. Due to its flexibility as well as its suitability to the problem domain, established in both research and practical applications, our team choose to focus our optimizing efforts on the CNN model.

1) Hyper-parameter Selection: To pick the best combination of hyper-parameters, we conducted an initial, high-level search of the parameter space by using functionality from the scikit-learn library. Cross-validation was used to evaluate the performance of different hyper-parameters to avoid particularities of any fixed training/validation split. Once the best performing set of hyper-parameters was identified, manual search on a more granular scale was conducted in hopes of improving upon the accuracy.

2) Choice of Optimization Algorithm: In our implementation of the CNN, we chose to use the Adam optimization algorithm instead of the more common stochastic gradient descent (SDG) method. This choice of optimization strategy can be seen as an extension of the SDG algorithm and is particularly suitable for computer vision problems. This is due to the fact that it maintains individual learning rates for each parameter (AdaGrad) that depend on the most recent k magnitudes of the weight gradient (RMSProp).

3) Cross-validation: In order to gauge the performance of a particular hyper-parameter configuration, it is required to first train all hyper-parameter configurations on the same training set, and then evaluate performance on some holdout set known as the validation set. To avoid particularities of any given training set, the common practice of cross-validation was used. This technique creates k separate datasets from the original dataset, each with a random but equally sized train/validation split. Each hyper-parameter configuration is then tested on each of the k datasets, with its overall accuracy assigned as the average of its accuracies on the k individual data sets.

In our experiment, a 5-fold cross-validation split of 10% was chosen. That is, the data was split up into five sets, with 10% of the $50,000$ training examples used as a validation set in each of the five splits.

III. Results

A. Local Results

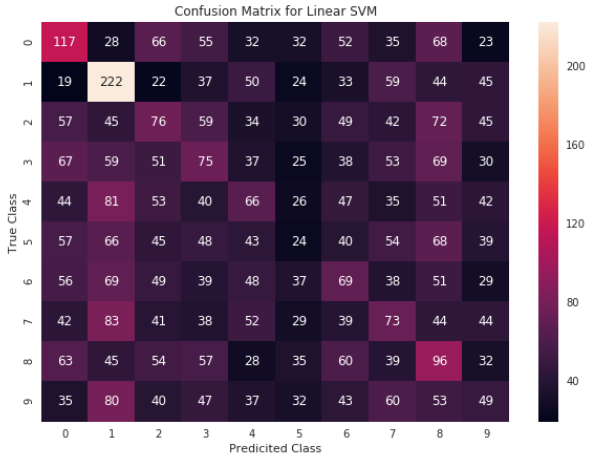
Testing of the classifiers based on cross validation of the training data was done to evaluate each of the three models that were constructed. The best performing hyper-parameter configuration of each model, the configuration's precision, accuracy, recall and $F1$-metric, as well as relevant plots are provided below.

TABLE I
PERFORMANCE METRICS OF THE LINEAR SVM

Accuracy	Precision	Recall	$F1$-metric
0.173	0.162	0.170	0.164

1) Linear Support Vector Machine: Through a preliminary hyper-parameter search over the penalty term C in the linear SVM model, it was found that a penalty term of $C = 3.3$ was optimal. The performance of the optimal SVM baseline model obtained from cross-validation is given in table I. We also provide the associated confusion matrix of the optimized SVM in figure 4.

Fig. 4. Confusion matrix of the linear SVM



From the confusion matrix, we notice that the linear SVM managed to be significantly more accurate at classifying one class in particular, the class of images with label *1*. Since the linear SVM will only find a separating hyperplane if the problem space is linearly separable, we hypothesize that the $C - 1$ spaces involving images labeled as *class 1* was much less homogeneous than those spaces that did not. Indeed, the other entries in the matrix seem to all show similar performance. This hypothesis is further supported by the observed performance metrics, which are slightly better than a random guess.

2) Fully Connected Feed-forward Neural Network: In our fully connected feed forward ANN, a preliminary hyper-parameter search was performed on the number and dimensions of the hidden layers, the number of iterations and the learning rate. It was determined through a preliminary hyper-parameter search that the following architecture was optimal.

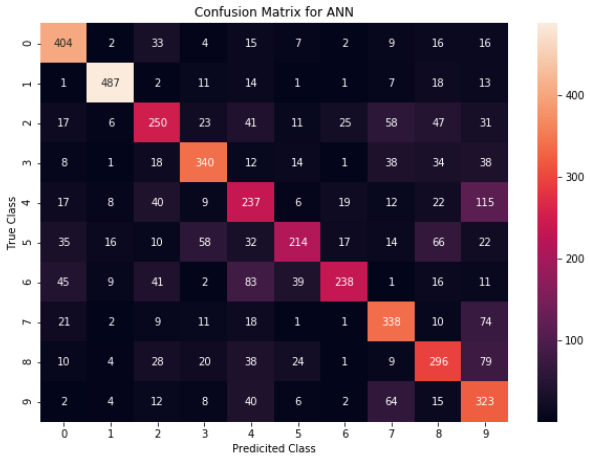
1) Input Layer: nodes = 784
2) Hidden Layer : nodes = 450
3) Hidden Layer : nodes = 250
4) Hidden Layer : nodes = 50
5) Output Layer : nodes = 10

TABLE II
PERFORMANCE METRICS OF THE FULLY-CONNECTED NEURAL NETWORK.

Accuracy	Precision	Recall	$F1$-metric
0.625	0.625	0.625	0.625

In addition, the hyper-parameter search resulted in a learning rate of $\alpha = 0.001$ trained over 250 iterations. The performance of the optimal ANN is given in table II. We also provide the associated confusion matrix in figure 5.

Fig. 5. Confusion matrix of the fully-connected neural network.



We observe that the ANN model can classify with much greater success compared to the SVM model. This is to be expected since the network learns nonlinear projections than enable it to separate the classes, something that the linear SVM model can not accomplish.

3) Convolutional Neural Network: Following an extensive hyper-parameter search, it was determined that a CNN with the following architecture was optimal.

1) Input, Convolution: nodes = 60, filters = 5, stride = 1
2) Pooling: kernel size = 2, stride = 2
3) Convolution: nodes = 30, filters = 3, stride = 1
4) Pooling: kernel size = 2, stride = 2
5) Fully Connected: nodes = 512
6) Dropout: dropout probability = 0.25
7) Fully Connected: nodes = 256
8) Dropout: dropout probability = 0.25
9) Output, Fully Connected: nodes = 10

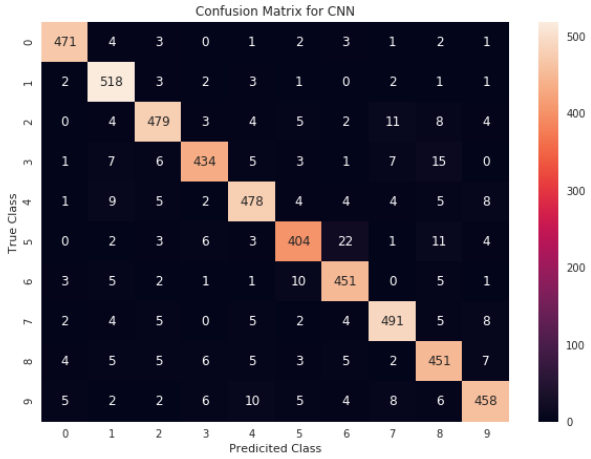
In addition to the optimal architecture, all non-pooling layers but the output layer used a ReLU activation function with the output layer utilizing the softmax activation layer. The pooling layers utilized the max pooling strategy. The same search resulted in a optimal learning rate of $\alpha = 0.001$ trained on 10 epochs with a batch size of 100.

The performance of the optimal CNN classifier is given in table III. We also provide the associated confusion matrix of the optimized SVM in figure 6.

TABLE III
PERFORMANCE METRICS OF THE CNN

Accuracy	Precision	Recall	$F1$-metric
0.927	0.927	0.927	0.927

Fig. 6. Confusion matrix of the CNN



The CNN demonstrated impressive performance at the classification task. It was able to generalize well and classify more than 90% of the validation set correctly. While, the CNN does manage to classify a large majority of the validation images correctly, it still fails on a portion of them. We suspect that this classification error comes from bias introduced by our preprocessing method. This method may erroneously choose the incorrect digit when selecting the contour with the largest surface area, thus feeding the network with incorrectly labeled training data. This is discussed in more detail in the analysis of the disadvantages of our approach.

4) Model Comparison: By comparing the three evaluated models, we can see that the CNN performs much better than both the SVM and the ANN. This result was expected due to the amount of effort put into its optimization as well as the fact that it is especially suited to the task of image classification as its architecture exploits the structure of its input.

Another noteworthy observation is the power that nonlinearity brings. The performance of the two neural network models greatly surpasses that of the linear SVM. Indeed, without much effort in terms of optimization other than a simple search through the hyper parameter space, we were able to achieve more than three times the classification accuracy with the ANN compared to the linear SVM.

IV. DISCUSSION

A. Analysis of Approach

The methods taken by our team in approaching this challenge contained both advantages and disadvantages. These

merits and faults are henceforth detailed and improvements proposed.

1) Advantages: One major advantage in our approach are the preprocessing methods applied to the dataset. By exploiting *a priori* knowledge of the problem domain and regularities in the dataset, we were able to reduce the difficulty of the task. The preprocessing pipeline removed much of the noise present in the images. In doing so, we were able to simplify the inputs thereby increasing classification accuracy. Furthermore, this also enabled us to reduce the complexity of our models by reducing the number of parameters resulting in improved generalization.

Another beneficial facet of our approach were the decisions made regarding convergence rate improvement. These choices allowed our team to iterate faster by reducing the amount of time taken to train and test different models and hyper-parameter configurations.

2) Disadvantages: While the preprocessing pipeline was considered a major advantage to our approach, there is a negative aspect to it as well. Indeed, there is a critical stage in the preprocessing where we allow for the introduction of bias to the network. When the largest contour is selected for input image x_i there is a chance that the wrong digit is chosen when compared to the target class y_i. This erroneous event was observed during manual testing with certain input examples. Nonetheless, while this does introduce bias with respect to method in which the largest area contour is selected, our team deemed this trade-off as ultimately being beneficial to classification performance. The hope was that significantly more images would be properly preprocessed, compared to those that would fail in the preprocessing step.

B. Future work

There are several aspects of the project that present suitable and interesting prospects for future investigation.

The first of which is exploring the potential and pushing the boundaries of other models to see what they are capable of. Due to our approach in choosing to focus on the CNN as well as the nature of the competition, we did not have the resources to commit to optimizing and exploring other models; those that we implemented as well as novel models that we did not.

One particular area of interest would be to test the performance of the SVM with a nonlinear kernel. As was mentioned before, it was hypothesized that the problem space is highly nonlinear, causing the linear SVM to demonstrate poor accuracy. This hypothesis could be tested by comparing a nonlinear SVM with the linear SVM.

Finally, the prospect of evaluating different CNN architectures is another area worth exploring. Indeed, further investigation into the effect of different combinations of convolution-pooling layers and different filtering and pooling strategies would likely produce an even more capable classifier. Due to the flexibility of the CNN model, we were unable to search a larger hyper-parameter space, and thus, could not optimize our model to as great of an extent as is possible.

V. Statement of Contributions

Yi Qiao was responsible for the implementation of the fully connected feed-forward neural network and provided contributions to the construction of the CNN. Patrick was responsible for the implementation of the linear SVM and supported Mathew in the tuning of the CNN. Mathew was largely responsible for the implementation and optimization of the CNN. Both Mathew and Patrick contributed to the data preprocessing efforts while Yi Qiao produced the written report.

We hereby state that all the work presented in this report is that of the authors.

References

[1] S. Albelwi and S. Mahmood, *A Framework for Designing the Architectures of Deep Convolutional Neural Networks*, in Entropy vol. 19 no. 6, pp. 242, 2017. [Online]. Available: http://www.mdpi.com/entropy/entropy-19-00242/article_deploy/html/images/entropy-19-00242-g001.png[Accessed 21 Mar. 2018].