# Digit Classification by Maximum Area

Section 551-001
Team Name: EIS

Elias Stengel-Eskin
260609642
elias.stengel-eskin@mail.mcgill.ca

Bing'er Jiang
260668025
binger.jiang@mail.mcgill.ca

Sheldon Benard
260618386
sheldon.benard@mail.mcgill.ca

## I. Introduction

Object recognition in image data is an important application of computer vision, as it strives to automate tasks typically performed by the human visual system. The quintessential demonstration of object recognition is the classification of handwritten digits from visual data. In our study, we explore a variation of this problem. Our task is to classify grayscale images, containing two or three handwritten digits, by the numerical digit whose bounding rectangle occupies the largest area (see Figure 1). In the study, we use a Linear Support Vector Machine (SVM) baseline, Forward-Feeding Neural Network (MLP), and Convolutional Neural Network (CNN) for the classification task, comparing the accuracies of the three models. After preprocessing the image data, the hyperparameters were tuned using a validation set, and predictions were generated using the tuned models. The CNN outperformed the other models, yielding a test set prediction accuracy of 93.7%, and, after utilizing ensemble optimization techniques, a test set accuracy of 96.3%.

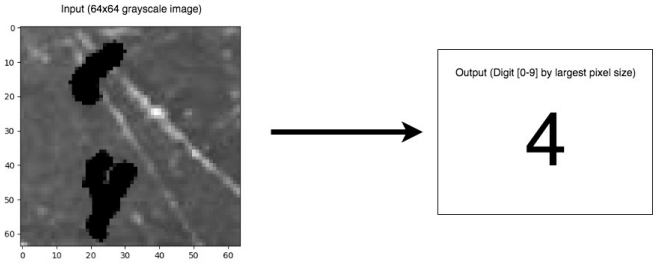


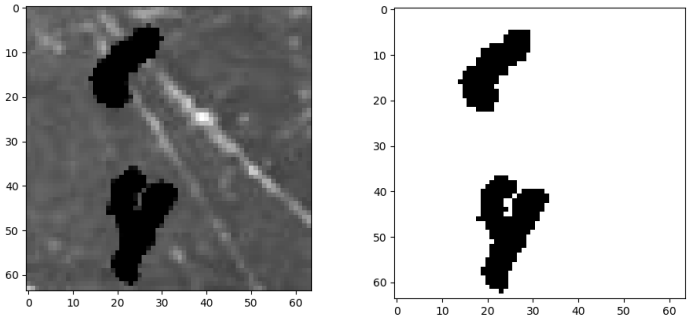

Fig. 1. Left: A raw input 64×64 greyscale image containing two hand-written digits; Right: Class label as output (the bottom digit 4 occupying larger space in the pitcture).

## II. Feature Design

### A. Threshold

The dataset used for the study is a variant of the popular MNIST dataset [1]. The dataset was obtained by generating random grayscale images composed of two or three digits that are 40%, 60%, 80%, 100%, or 120% of their original size.

Each image in the dataset is 64x64 pixels, meaning that the input contains 4096 features (each feature corresponding to 1 pixel of the image). An example of such an image is depicted in figure 2(a), where the digits (whose pixels have an intensity of 255) are the objects of interest. All pixels with an intensity less than 255 are not part of the digit, and thus can be ignored. Thus, the image data was preprocessed with a pixel intensity threshold: all pixels less than 255 were set to 0 (c.f. figure 2(b)). Thus, the noisy features (the non-digit pixels) are set to 0, but the features of the digit remain, greatly simplifying the problem.



(a) (b)

Fig 2.(a): the original image containing two hand-written digits; (b): preprocessed image after removing non-255 values

### B. Feature scaling

Tt has been shown that changes in the distribution between layers of a deep neural network can slow down gradient-descent-based optimizers [2]. To avoid changes in the distribution between the input layer and the first convolutional layer, MinMax scaling was used. For each pixel, the feature was scaled using formula (1). This rescales the features to be in the range [0, 1].

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad (1)$$

Moreover, feature scaling is very important for classifiers that involve the calculation of Euclidean distance between two points, like Linear Support Vector Machine. Feature scaling avoids any occurence of features with greater numeric values dominating those with smaller numeric values.

### C. Elastic distortion

During the execution of the optimal model (determined by hyper-parameter tuning, see section 4D), the model displayed an overfitting to the training data. Thus, we expanded the dataset by adding elastic distortions. Elastic-distorted data mimics the deformations of uncontrolled oscillations of hand muscles [14], thus allowing us to expand the dataset. Furthermore, supervised systems benefit from larger high-quality datasets, and elastic distortions have been shown to improve neural network results [5].

Using elastic distortions, we expanded our original dataset from 50,000 samples to 100,000 samples. The elastic distortion step is shown schematically in Figure 3(a). This expanded

dataset was used to train the CNN model after hyper-parameter tuning to avoid overfitting.

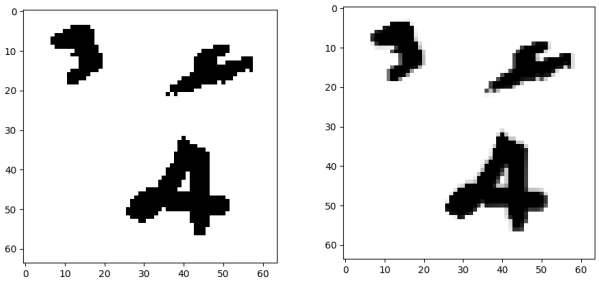


Fig 3.(a): the same input image in figure 1 after removing non-255 values; (b): preprocessed image after elastic distortion

## III. Algorithms

We compared the performance of three learning algorithms for the largest digit recognition task: a linear support vector machine, a feed-forward neural network, and a convolutional neural network. The models are described below.

### *A. Baseline: Linear SVM*

Linear SVM, a supervised learning model, is used as a linear classifier to establish baseline model performance. For a given classification task, Linear SVM attempts to introduce hyperplanes that separate the classes so that the distance between the hyperplanes is maximal. The samples that creat te maximum margin between the different classes (i.e. the sample closest to the hyperplane) are called support vectors. The model attempts to optimize the tradeoff between increasing the margin size and ensuring that data points of the training set are on the right side of the margin.

### *B. Baseline feed-forward neural network*

We implemented a Feed-Forward Neural Network as the benchmark for more complicated neural networks (e.g. CNNs). Feed-Forward Neural Networks transform a vector input using a series of non-linear transformations; a linear combination of features and weights is passed through a non-linear activation function for each neuron in a layer.. Each neuron of a layer is connected to all the neurons of the previous layer and are independent from other neurons in the same layer. For a classification task, the output layer is a vector of class scores.

For our neural network architecture, the network has one input layer and varying numbers of hidden layers and neurons per layer using tanh as the activation function (more details in section 4d). The hyperparameters (layer number, layer size, and learn rate) were tuned via cross-validation. The output of the hidden layer was then pushed through softmax to predict class labels (ten digits from 0 to 9). The network was trained using stochastic gradient descent with cross-entropy loss.

### *C. Convolutional neural network*

Many studies have shown that CNNs are profoundly powerful classifiers for image recognition tasks [5] [6]. As compared to Feed-Forward Neural Networks, CNNs make the explicit assumption that the input dataset consists of images. Utilizing this assumption, CNNs are able to extract and combine local features, which are fed into a set of alternating convolutional and pooling layers. CNNs have an advantage over traditional neural network architectures because they have fewer connections and fewer parameters; since the neurons of a layer are learning only local connections the total number of connections is reduced; the smaller number of parameters is a results of parameter sharing. These factors make CNNs less computationally expensive to train.

The CNN architecture used in the study is shown in Figure 4; this visualizes the best-performing model after hyper-parameter tuning. The CNN has 10 hidden layers (4 convolutional layers, with rectified linear unit activation (ReLU), each followed by 4 max-pooling layers, and finally 2 fully-connected dense layers). All convolutional layers use a local receptive field of size 5×5 and filter number of 48, each followed by a 2×2 max-pooling layer for feature reduction. The two fully-connected layers contain 1024 and 512 ReLU neurons respectively, and the output layer contains 10 neurons, corresponding to 10 digits.

## IV. Methodology

The following subsections outline all the decisions we made for training and testing.

### *A. Training-validation split*

Initially, the data was split into 95% training and 5% validation (80% training, 20% validation for the MLP), but, according to [7], "the commonly used strategy of allocating 2/3rd of cases for training was close to optimal for reasonable sized datasets (n $\geqslant$ 100)." Thus, ultimately, we moved to using a dataset split of 70% training and 30% validation (70,000/30,000 split on expanded dataset).

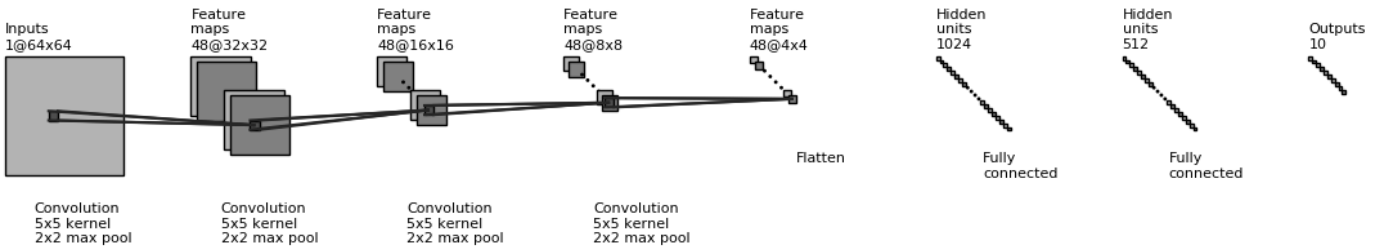



Fig 4. The schematic of CNN architecture

### B. Regularization

Dropout was introduced to prevent overfitting in neural networks,; this strategy has been successfully used to improve neural network performance in past supervised learning tasks [8]. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents neurons, in the network, from co-adapting (or, being useful as a feature detector only in the context of other feature detector) too much.

In our study, dropout regularization was implemented by adding 0.5 dropout after each of the two dense layers. The dropout regularization imposes 0.5 probability of any given element being dropped during training. According to [9], a dropout rate of 0.5 resulted in the most regularization for a single linear unit, and so, we adopted the 0.5 dropout rate for our study.

### C. Optimization

This section gives a brief description of the optimization techniques used in training:

*1). Batch Normalization.* As mentioned above, changes in the distribution between layers of a deep neural network can slow down gradient-descent-based optimizers. Batch normalization eliminates these changes in the distribution, by explicitly forcing the activations throughout the network to take on a unit Gaussian distribution, and this accelerates the training process [2]. In order to optimize the speed of the learning algorithm, we implemented batch normalization in our convolutional neural network.

*2). Elastic Distortion.* Expanding the original dataset via elastic distortion has been shown to improve neural network results [2]. Further, as shown in [10], expanding the original dataset via data transformations limits the extent to which the model overfits. Thus, we expanded our original dataset from 50,000 to 100,00 using elastic distortion (alpha = 15, sigma = 15) in the second stage of hyper-parameter tuning where overfitting occured.

*3). Adam Optimizer.* Adam has been shown to efficiently and effectively solve problems related to deep learning [11] . Adam is robust and requires little memory. Given the memory limitations of the CPU-based computers used to train the models described in this paper, the Adam optimizer was the optimal choice for our study.

*4). Ensemble Learning.* Majority voting has been shown to improve performance of an ensemble of the same network trained multiple times [12]. Further, Checkpoint ensembles, ensemble of same network training session but at different lengths of training (checkpoints), have been shown to improve prediction performance as well [13].

To reduce the variation in our predictions, a hybrid of these two ensemble methods was used for the CNN model. In total, 5 convolution neural networks were trained, each with 5 checkpoints saved. Thus, a total of 25 different model states were used together, in a majority voting fashion, to generate the test predictions.

### D. Hyper-parameter tuning

*1). Linear SVM.* For the Linear SVM, we used an L2 penalty and the Squared-Hinge loss function. The hyper-parameter tuned was the cost (C):

Table 1. Hyper-parameters of Linear SVM

| Hyper-parameters | Range of Values |
|---|---|
| Penalty | L2 |
| Loss Function | Squared Hinge Loss |
| Cost (C) | [0.01, 1.0] |

*2) Feed-forward Neural Network.* For feed-forward neural network, the hyper-parameters are listed below.

Table 2. Hyper-parameters of fead-forward neural network

| Hyper-parameters | Range of Values |
|---|---|
| Number of hidden layers | {1, 2, 3} |
| Learning rate | {0.001, 0.01, 0.1]} |
| Neurons per layer | {128, 256, 512} |
| Activation | tanh |
| Training epochs | 10 |
| Batch size | 128 |

3) Convolutional Neural Netwok.

Table 3. Hyper-parameters of CNN

| Hyper-parameters | Range of Values |
|---|---|
| Number of filters | {48, 64, 96} |
| Kernel size | {3x3,5x5,7x7} |
| Number of convolutional layers | {2, 4, 6} |
| Learning rate | {0.001, 0.0001, 0.0009 with beta=0.8} |
| Activation | ReLU |
| Dropout | 0.5 |
| Max-pooling size | 2 |
| Stride | 2 |
| Batch size | 64 |

## V. RESULTS

This section shows classification accuracy achieved by different classifiers and compares the performance between different model configurations.

### A. Linear SVM

Table 4 below shows the hyper-parameter setting that yields the highest accuracy, determined by grid search. We

used 70% of data for training and 30% of data for validation on the original dataset, with L2 regularization and squared hinge loss.

Table 4. Validation and test results for linear SVM

| Hyper-parameter | Validation accuracy | Test accuracy |
|---|---|---|
| C = 0.01 | 13.52% | 14.87% |

### B. *Feed-forward Neural Network*

Table 5 shows the 5-fold cross-validation accuracy (using an 80-20 split) for a variety of hyperparameter settings. The best hyperparameter settings are reported for each layer. Each network was trained for 10 epochs with a batch size of 128 using cross-entropy loss.

Table 5. Results for feed-forward neural network with different hyper-parameters from 5-fold cross validation

| | 1-layer | 2-layer | 3-layer |
|---|---|---|---|
| Best learning rate | 0.01 | 0.001 | 0.001 |
| Best layer size | 512 | 128 | 128 |
| Accuracy | 23.52 | 11.41 | 12.67 |

### C. *Convolutional Neural Network (CNN)*

Results of hyper-parameter tuning of CNN are shown in Table 6 and 7. There are two stages of hyper-parameter tuning. During the first stage, the purpose was to select the best architecture of CNN by varying three parameters: number of layers, number of filters, and the size of the filters (Table 6). The configuration that produced best validation results was selected for the second stage of tuning, where learning rate was varied (Table 7).

As shown in Table 6, the best performance was achieved with the combination of 48 filters of size 5×5 with 4 convolutional layers. Interestingly, the results show that a larger number of filters did not necessarily ensure better performance, nor does a deeper configuration. Moreover, a comparison with the baseline linear SVM and feed-forward neural network shows that a relatively simple 2-layer CNN, after training for 10 epochs of 300 steps, can reach an accuracy of 86.80%, which beats the SVM as well as the best configurations of the MLP.

It it important to note that, referring to Table 6, several configurations seemed to get stuck at an accuracy of 10.82%, which corresponds to a loss of around 2.30. We suspect there may be a local minimum in the error surface, and these configurations were unable to escape the well of this local minimum.

Tabel 6. First stage of hyper-parameter tuning of CNN with stride = 2, learning rate =0.001, trained for 10 epoches with 300 steps per epoch on the original dataset. The split was 5% validation, 95% training

| Filters | | Number of layers | | |
|---|---|---|---|---|
| Number | Size | 2-layer | 4-layer | 6-layer |
| 96 | 3×3 | 10.82 | 10.82 | 10.82 |
| 96 | 5×5 | 10.82 | 92.58 | 90.98 |
| 96 | 7×7 | 86.80 | 93.82 | 93.56 |
| 64 | 3×3 | 10.82 | 92.40 | 10.82 |
| 64 | 5×5 | 10.82 | 10.82 | 93.42 |
| 64 | 7×7 | 10.82 | 93.48 | 92.06 |
| 48 | 3×3 | 10.82 | 10.82 | 10.82 |
| 48 | 5×5 | 10.82 | **93.96** | 10.82 |
| 48 | 7×7 | 10.82 | 93.64 | 91.26 |

The results for different learning rates are shown in Table 7. The best result was achieved with learning rate of 0.0009 with a $\beta 1 = 0.8$ ($\beta 1$ is the exponential decay rate for the gradient's running average).

Table 7. Second stage of hyper-parameter tuning of CNN with original data varying over different learning rates and weight decays, trained for 10 epoches with 500 steps per epoch, using 30% validation, 70% training

| Learning rate | | Validation |
|---|---|---|
| $\alpha$ | $\beta 1$ | |
| 0.001 | 0.9 | 92.23 |
| 0.0001 | 0.9 | 86.28 |
| 0.0009 | 0.8 | 93.28 |

With this best configuration for the CNN, we trained the model for 20 epochs, 500 steps per epoch. On the original dataset, with 30% validation - 70% training split, we obtained the following results:

Table 8. Training/ validation/ test accuracy for best model.

| | Training | Validation | Test |
|---|---|---|---|
| Accuracy | 98.44 | 93.68 | 93.10 |

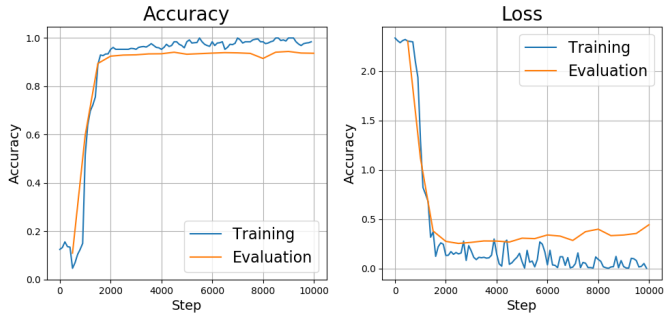



Fig 5. Accuracy (left) and loss(right) for best-performance model configurations trained for 10,000 steps: model overfitting

Figure 5 shows that the model was overfitting to the training data, and the training and validation loss values diverged significantly. Since other studies have shown that increasing the dataset through transformations reduce model overfitting [10], we expanded the original dataset from 50,000 to 100,000 using elastic distortions. The results for the optimal CNN model on the expanded dataset are shown in Table 9 and Figure 6, and the amount of overfitting was reduced by the introduction of elastic distortion.

Table 9. Training/ validation/ test accuracy for best model after elastic distortion.

| | Training | Validation | Test |
|---|---|---|---|
| Accuracy | 98.07 | 96.64 | 93.70 |

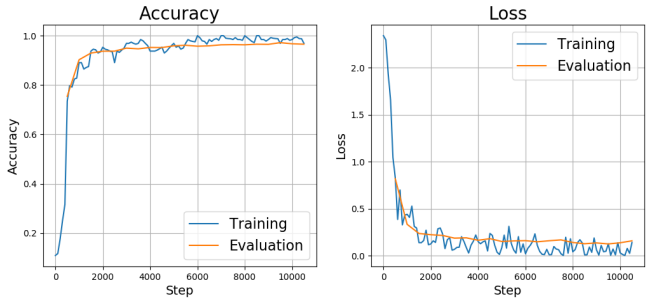



Fig 6. Accuracy (left) and loss(right) for best-performance model configurations trained for 10,000 steps after elastic distortion: no overfitting

Finally, the best test accuracy was achieved by an ensemble of 5 optimally-configured models, each with 5 checkpoints. The test accuracy was 96.33%.

## VI. Discussion

### A. Baseline Models

By tuning the baseline models' hyperparameters, the performance of these models was enhanced. The results from the Linear SVM shows that a linear decision boundary is not suitable for modeling this problem -- even with the best hyperparameters, the model hardly outperformed the baseline. The MLP model, on the other hand, is able to model non-linear decision boundaries. While MLP model outperformed the Linear SVM, it had a maximum accuracy of 23.52%. This is in part due to the sub-optimality of the features. Feed-forward neural networks require a vector input, whereas the image has two-dimensional features; thus the image was flattened into a one-dimensional vector before being fed to the MLP, stripping away many of these features and dependencies. Typically, deeper networks lead to an increase in accuracy. However, in this case, exploding and vanishing gradient problems led to a breakdown in accuracy for 2- and 3-layer networks, explaining their performance (which was comparable to the baseline).

On the other hand, CNNs solve the problem of maintaining the structural nature of the image input. In this report, we used CNNs for the task of largest pixel digit classification. This section briefly summarizes the pros and cons of our approach, and possible future works.

### B. Pros

One advantag of our approach was thorough hyper-parameter tuning. For each of the classifiers, we considered several configurations and compared and contrasted their validation accuracies. In addition, we were able to spot overfitting in our optimal model, and were able to effectively reduce the amount of overfitting using elastic distortions. Lastly, we explored several optimization techniques (MinMax scaling and batch normalization, ensemble learning, dropout) which optimized our CNNs efficiency and, ultimately, accuracy, as ensemble learning boosted our test accuracy to 96.33%.

### C. Cons

Although we did a thorough hyper-parameter tuning, there are several configurations that we were unable to consider due to limited time and computational power. Moreover, the number of training epochs had to be kept small, due to time constraints, and so model configuration training was not done to our desired number of epochs. Lastly, we did not explore separating the problem statement. Our task was to find the largest digit (by bounding-box area) in a 64x64 image, and our approach was to train a CNN to learn to both find the largest digit and then recognize the largest digit. Thus, errors can occur via the model not properly identifying a digit or the model not properly identifying the largest digit. An alternate approach would be to programmatically determine the largest digit, since this task can be reduced to image segmentation, and then, using the largest digit, have a CNN (or another classifier) identify the digit.

### D. Future work

Future improvements can be made by adopting different methodology and increasing computational power.

As alluded to in the previous section, the current approach does not make explicit distinction between largest digit identification and digit recognition, which may lead to suboptimal results. Image segmentation could be used to determine the largest digit, and a classifier could be trained to recognize the digits 0-9. An example of the image segmentation procedure is shown in Figure 7 below.

Splitting the task this way also has an advantage of making use of more training data. Specifically, for the second task (i.e. single digit recognition), one would be able to incorporate all the original MNIST dataset for training.

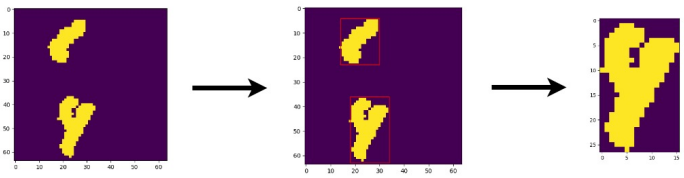


Fig 7. The schematic of explicitly modeling two tasks, the first one being the identification of largest digit (middle) from the original image (left), and the second one being digit recognition purely based on the single digit (right)

Also, GPUs could be used to train more complicated networks with larger epoches and different hyper-parameter

combinations. Moreover, different methods of preprocessing and regularization could be implemented and different optimizers used (other than Adam) to see if any improvement can be made.

## VII. Statement of Contributons

All of us participated in defining the problem and developing the methodology. Individual contributions are listed below:

Elias Stengel-Eskin: coded feed-forward neural network, ran models for hyper-parameter tuning for both feed-forward neural networks and CNNs, revised the report.

Bing'er Jiang: drafted and revised the report, coded and ran linear models for baseline.

Sheldon Benard: coded CNN, ran models for hyper-parameter tuning, data visualization, revised the report.

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

## References


[1] LeCun, Y., Cortes, C., & Burges, C. J. (2010). MNIST handwritten digit database. AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist, 2.

[2] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456).

[3] Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003, August). Best practices for convolutional neural networks applied to visual document analysis. In ICDAR (Vol. 3, pp. 958-962).

[4] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[5] Labusch, K., Barth, E., & Martinetz, T. (2008). Simple method for high-performance digit recognition based on sparse coding. IEEE transactions on neural networks, 19(11), 1985-1989.

[6] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[7] Dobbin, K. K., & Simon, R. M. (2011). Optimally splitting cases for training and testing high dimensional classifiers. BMC medical genomics, 4(1), 31.

[8] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

[9] Baldi, P., & Sadowski, P. J. (2013). Understanding dropout. In Advances in neural information processing systems (pp. 2814-2822).

[10] Lu, Y. (2016). Food Image Recognition by Using Convolutional Neural Networks (CNNs). arXiv preprint arXiv:1612.00983.

[11] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[12] Ju, C., Bibaut, A., & van der Laan, M. (2018). The relative performance of ensemble methods with deep convolutional neural networks for image classification. Journal of Applied Statistics, 1-19.

[13] Chen, H., Lundberg, S., & Lee, S. I. (2017). Checkpoint Ensembles: Ensemble Methods from a Single Training Process. arXiv preprint arXiv:1710.03282.

[14] Krishnan, P., & Jawahar, C. V. (2018). HWNet v2: An Efficient Word Image Representation for Handwritten Documents. arXiv preprint arXiv:1802.06194.