MiniProject 3: Modified MNIST

Abstract **-**The MNIST (Mixed National Institute of Standards and Technology) database consists of handwritten digits and is commonly used for training various image processing systems. The modified MNIST dataset contains multiple digits of varying sizes per sample 64x64 pixel greyscale image. In this project, we are tasked with designing and validating a supervised classification model to perform on the latter dataset. The object is to correctly classify the digit that occupies the largest square bounding box. Several multi-layer convolutional neural network (CNN) models are trained and their outputs are ensembled. This is carried out using the Keras Python Deep Learning library. The data is preprocessed using a heuristic approach that locates and scales the digit with the largest bounds. Performance using preprocessed and unprocessed data is compared with processed data outperforming original data. The best performing model reports an accuracy score of 92.5% on the final test set.

I. INTRODUCTION

Given a set of modified MNIST data, students are tasked with correctly classifying the digit occupying the largest bounding box. The nature of the dataset itself presented some challenges.

The MNIST database contains 60,000 training images and 10,000 testing images where half of the training set and half of the test set were taken from the NIST's testing dataset. A cursory search on Google reveals the traditional MNIST digits to be either binary (i.e. black and white) or very close to it. Furthermore, sample images seem to contain only one digit each. The modified MNIST dataset not only contains between 2 and 3 digits per image, but also provides a textured background. These characteristics complicated the training in that a variability in background lighting, digit size, and touching digits was introduced. Therefore, preprocessing was judged necessary.

The final preprocessing algorithm which isolated and scaled the largest digit allowed individual CNN models to achieve between 88 and 91% on validation. These models used between 3 and 4 convolutional layers. However, their accuracy was still not enough as other teams were posting accuracies of 95% and more on Kaggle.

The solution to this was ensembling. A simple stacking scheme was set up where 3 models' predictions were voted on. This increased the final accuracy to 92.5%. Though not as high as the upper percentile of the Kaggle competition, it was deemed a respectable final effort.

Modelling the classifiers was done on the Keras Python Deep Learning library which enabled rapid prototyping compared to some of the other frameworks.

II. RELATED WORK

In deep learning, a convolutional neural network (CNN) is a class of deep neural networks which is mostly applied to analyze visual imagery. A CNN consists of an input layer and an output layer, as well as multiple hidden layers. The hidden layers typically consist of convolutional layers, RELU layers, pooling layers, fully connected layers and normalization layers. CNNs use relatively little pre-processing compared to other image classification algorithms. Also, CNN's are generally easier to train for vision problems [1]. One such example of using a hierarchical system of convolutional neural networks achieved an error rate on the MNIST database of about 0.23% [2].

Ensemble learning is the method of combining multiple models trained over the same dataset or a random set of datasets to improve the model performance [3]. Ensembling methods are widely used in deep learning [4] to improve the overall model accuracy. In fact, ensembling multiple classifiers generally outperforms individual classifiers [5]. Furthermore, model averaging (i.e. different models vote on the final output) generally works because different models usually make different mistakes [6].

Segmentation is a popular image processing task which separates image components into groups that share characteristics [7]. The modified MNIST dataset contains digits drawn on textured backgrounds. Therefore, segmentation is judged relevant insomuch as to separate the digits from their background. On the other hand, Hochuli *et al.* argue that without a segmentation module, strings of digits that are isolated or touching can be correctly identified [8]. However, their dataset differs from ours in that it consists of binary images -there is already a clear distinction between background (black) and regions of interest (white digits). That being said, [8] is capable of overcoming the problem of touching digits by using 4 task specific classifiers, the first of which estimates the amount of such digits.

III. DATASET AND SETUP

The dataset consists of modified MNIST digits. There are 2-3 digits per sample image. The student is tasked with classifying the single digit that occupies the largest square bounding box. The data was pre-processed using a heuristic segmentation approach that identified the largest bounding box. Firstly, several filters were applied and the image was converted to binary. Once the largest binary digit was identified, it was extracted from the original (i.e. unprocessed) image and scaled to size (i.e. 64x64 pixels.) This was then fed into the network as the training data. The following figure illustrates an example of the steps. Specific details of the preprocessing will be addressed in the following section.

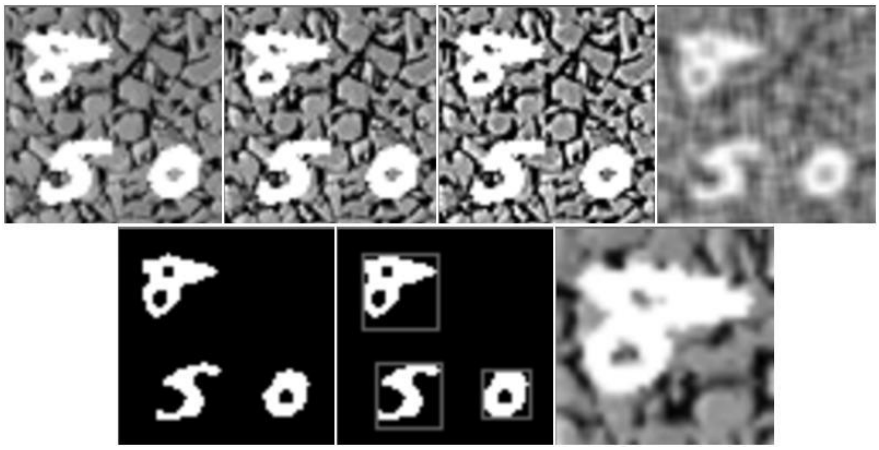


Figure 1: Image Processing: Initial Image (Top Left), Final Image (Bottom Right)

IV. PROPOSED APPROACH

A.PREPROCESSING

Generally, three preprocessing algorithms were tested and compared. They were developed heuristically, through a trial-and-error approach.

The first produced binary images where only the largest digit remained (i.e. everything else was set to black.) Whilst training, the team achieved 86% accuracy on validation and a testing accuracy of 87.5% on Kaggle.

The second algorithm aimed to reduce training errors from faulty preprocessing. A basic sharpening kernel was added as an algorithm step and all images were darkened by division to eliminate bright backgrounds. The final images produced were still binary, however, the improvement was marginal.

The third preprocessing attempted used a finer sharpening technique (i.e. unsharpening mask) and further darkened images beyond a certain threshold. Lastly, once the largest bounding box was located, it was scaled to the full 64x64 pixels (see Figure 1) and returned to its original greyscale state. It was assumed that the convolutional neural net would take care of its own processing beyond this point. This allowed our best performing individual model to achieve around 91% on validation data. However, there remained processing errors as shown in Figure 2.

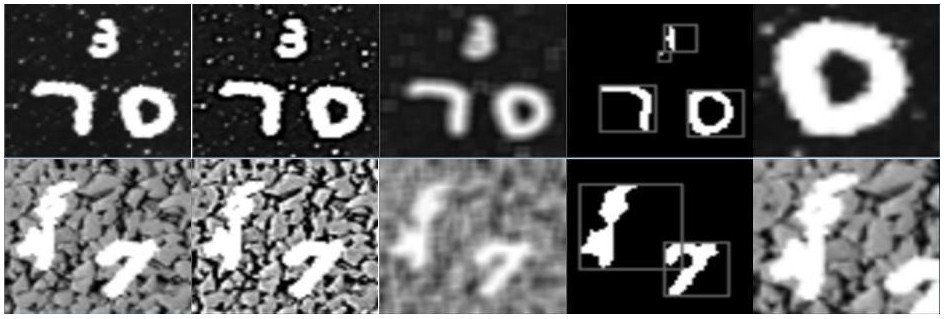


Figure 2: Processing Errors (Top Row : Label is 7, but we isolate 0; Bottom row: Touching Digits Error)

B.CLASSIFICATION MODELS

After the data is processed, it is fed into the convolution neural network (CNN) as the input. In this project, Keras [9], the deep learning library is used to construct our models. We first build a single multilayer neural network, as the advantage of multiple layers allows the model to learn features at various levels of abstraction. For our first model, we built a 4-layer convolutional neural network followed by a flattening layer. The output layer is a dense layer consisting of 10 nodes (i.e. 10 possible classes per images) with softmax activation function. It alternates between convolutional and pooling layers ; with dropout regularization used to prevent overfitting. We experiment with various filter size and stride size, and we found that the filter size of (3,3) and stride of (1,1) provides the best accuracy for the single model. Also, we find that any more layers added after the 4^{th} layers do not significantly improve the model. The model architecture is shown in the Figure 3. The second model we built consists of 3 consecutive convolution layers, and uses GlobalAveragePooling2D to compute the spatial average of the 10 output classes. The difference between Global Average Pooling and Max Pooling is that Max Pooling extracts the max values, while average pooling takes all into account and uses the average, as shown in Figure 3 [10]. The third CNN we built uses 2 consecutive convolution layers, followed by Max Pooling , and 64 as the first hidden neurons in the first layer. See *Appendix* for all of our model's structure summary. To further improve the accuracy of our model, we used the ensemble method to stack these 3 CNNs together.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 62, 62, 32)	320
max_pooling2d_1 (MaxPooling2	(None, 31, 31, 32)	0
conv2d_2 (Conv2D)	(None, 29, 29, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 12, 12, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 128)	0
conv2d_4 (Conv2D)	(None, 4, 4, 256)	295168
max_pooling2d_4 (MaxPooling2	(None, 2, 2, 256)	0
dropout_1 (Dropout)	(None, 2, 2, 256)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_1 (Dense)	(None, 128)	131200
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

Total params: 520,330
Trainable params: 520,330
Non-trainable params: 0

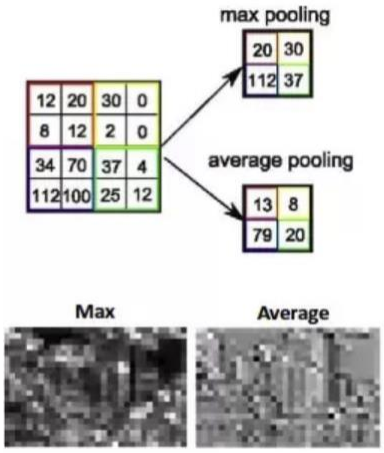


Figure 3: Model Architecture and Pooling

C.ENSEMBLING SCHEME

We used stacking as a means of ensembling. First, we trained each of the three models separately on the training dataset and evaluated their accuracy on the validation set. The batch size of all the models was set to 64, with all of them training on 10 epochs. Next, the three models were put in an ensemble, where the average outputs of each of the models were taken and combined as a single output, as shown in Figure 4 [11]. Stacking different models together allowed the ensemble to compensate for the weaknesses of individual models, rather than relying on a single model. The results of individual models and the ensembled model will be presented in the result section below.

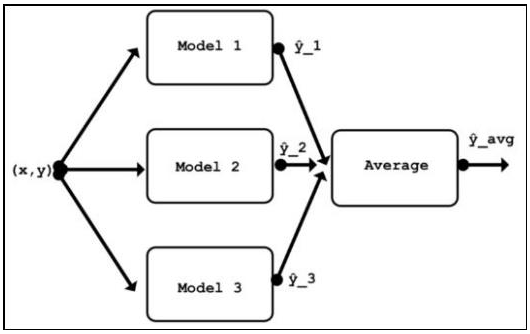


Figure 4: Ensemble Model Architecture

V. RESULTS

Using the set up described above, the three models and the ensembled model accuracy are presented in Table 1. The ensembled model did show an improvement on the accuracy of our prediction, with an accuracy of 0.9201 compared to all of the single models. As for the best performing single model, Model 1 seemed to performed the best with an accuracy of 0.9165. Model 2 performed the worst, with an accuracy of 0.8972. This could be due to the use of Global Average Pooling layer rather than using Max Pooling layer. Max pooling extracts the most important features such as edges, while Global Average Pooling takes all features into count and results in an average value, it couldn't extract the most important features. This could lead to bad accuracy as not all features are of equal importance, and that not all inputs from the convolution layer into the next are needed. We tried to ensembled only Model 1 and Model 3 together, it has an accuracy of 0.9200, so adding Model 2 into the ensembled model only improved the accuracy by 0.0001. When all the data was used for training, the ensembled model reached an accuracy of 0.925 on Kaggle.

Models	Validation Accuracy
Model 1	0.9165
Model 2	0.8972
Model 3	0.9114
Ensemble: (Model 1 + Model 2 + Model 3)	0.9205

Table 1: Results of Model Performance on Processed Data

Models	Validation Accuracy
Model 1 (unprocessed)	0.9105
Model 2 (unprocessed)	Does not Converge
Model 3 (processed)	0.9114
Ensemble: (Model 1 + Model 3)	0.879

Table 2: Results of Model Performance on Processed and Unprocessed Data

We also tested out the model using the unprocessed data for learning, as shown in Table 2. Whereas one model using unprocessed learning data does not converge (after 20 epochs), the other reaches an accuracy of 0.9105. However, when ensembled in a 2 model system, there is an actual decrease in total accuracy (0.879.)

The right graphs of Figure 5 show that the model has difficulty training in the beginning, as it does not gain in accuracy in the first 4 epochs. There is improvement after the 4th epoch: the validation accuracy reaches around 0.9105, using unprocessed data to train.

This is opposed to the processed data where the model experiences a sharp increase in accuracy from the first epoch and onwards.

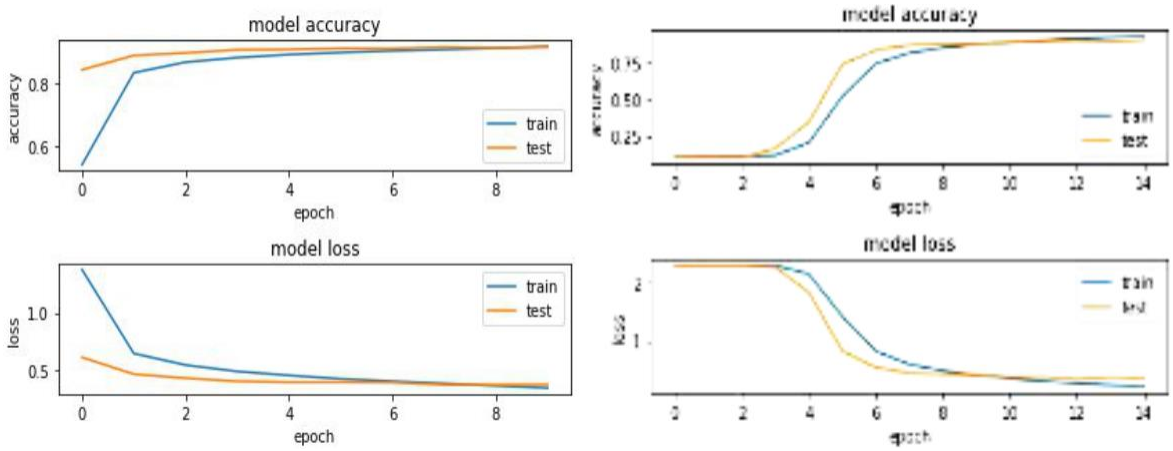


Figure 5: Model 1 Training on Processed (Left) and Unprocessed (Right) Data

VI. DISCUSSION AND CONCLUSION

In this project, we found that using a single multi-layered network allow us to achieve an accuracy of 0.9165, and that using the ensemble method of stacking 3 models together improved the accuracy by 0.0036, achieving an accuracy of 0.925 on the Kaggle leaderboard. We used the method of stacking to ensemble the models together, however, we find that the accuracy did not improve as much as we had assumed it would.

Furthermore, though the processing was very useful in achieving a high learning rate and accuracy for a single model, it seems it was in fact a limiting factor to obtaining higher accuracies. Later work could ensemble models using processed data with those using unprocessed data.

For future work, we would like to try a different ensembling method, such as with Adaboost ensemble, or trying stacking more CNN's together. Also, other techniques such as transfer learning and data augmentation can be attempted. For transfer learning, we would fine-tune the last layer of a pre-trained network (eg. VGGNet) to improve accuracy. For data augmentation, data size can be increased by using augmentation, such as applying rotation,flipping,introducing variations,etc.to the images.

VII. STATEMENT OF CONTRIBUTIONS

A group project for COMP 551 – Applied Machine Learning. All members contributed to the final report, specifically, to their respective tasks and, generally, to the report as a whole.

References

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*,pp. 1097-1105. 2012.

[2] Cireşan, Dan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification." *arXiv preprint arXiv:1202.2745*(2012).

[3] Rokach, Lior. "Ensemble-based classifiers." *Artificial Intelligence Review 33*,no. 1-2 (2010): 1-39.

[4] Dietterich, Thomas G. "Ensemble methods in machine learning." In *International workshop on multiple classifier systems*,pp. 1-15. Springer, Berlin, Heidelberg, 2000.

[5] Opitz, David, and Richard Maclin. "Popular ensemble methods: An empirical study." *Journal of artificial intelligence research*11 (1999): 169-198.

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*,2016.

[7] Yuheng, S., & Hao, Y. "Image Segmentation Algorithms Overview." *arXiv preprint arXiv:1707.02051.* (2017).

[8] A. G. Hochuli, L. S. Oliveira, A. Britto Jr, R. Sabourin, "Handwritten digit segmentation: Is it still necessary?", *Pattern Recognition*78 (2018) 1-11.

[9] Keras: The Python Deep Learning Library https://keras.io/

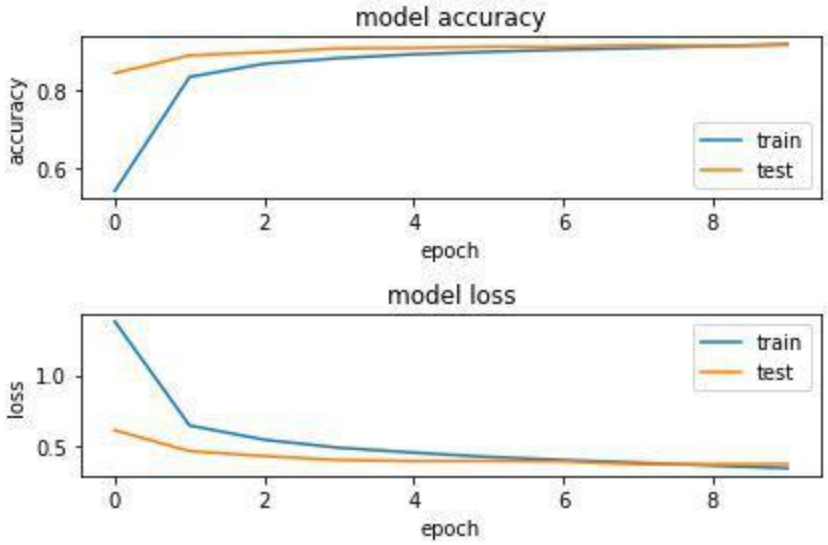
[10] Lawnboy,M. "Ensembling ConvNets using Keras", Retrieved March 15,2019, from https://towardsdatascience.com/ensembling-convnets-using-keras-237d429157eb

[11] Saha, S. "A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way", Retrieved March 16,2019,from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a 53

APPENDIX

Model 1 : Sample Result

```
Test loss: 0.3695907964035869
Test accuracy: 0.9165
```



Model 2

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 64, 64, 1)	0
conv2d_14 (Conv2D)	(None, 62, 62, 32)	320
conv2d_15 (Conv2D)	(None, 60, 60, 32)	9248
conv2d_16 (Conv2D)	(None, 58, 58, 32)	9248
max_pooling2d_8 (MaxPooling2	(None, 29, 29, 32)	0
dropout_6 (Dropout)	(None, 29, 29, 32)	0
conv2d_17 (Conv2D)	(None, 27, 27, 64)	18496
conv2d_18 (Conv2D)	(None, 25, 25, 64)	36928
conv2d_19 (Conv2D)	(None, 23, 23, 64)	36928
max_pooling2d_9 (MaxPooling2	(None, 11, 11, 64)	0
dropout_7 (Dropout)	(None, 11, 11, 64)	0
conv2d_20 (Conv2D)	(None, 9, 9, 128)	73856
conv2d_21 (Conv2D)	(None, 7, 7, 32)	36896
conv2d_22 (Conv2D)	(None, 5, 5, 10)	2890
global_average_pooling2d_1 (	(None, 10)	0
activation_1 (Activation)	(None, 10)	0

```
Total params: 224,810
Trainable params: 224,810
Non-trainable params: 0
```

Model 3

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 64, 64, 1)	0
conv2d_23 (Conv2D)	(None, 64, 64, 64)	640
conv2d_24 (Conv2D)	(None, 62, 62, 64)	36928
max_pooling2d_10 (MaxPooling	(None, 31, 31, 64)	0
conv2d_25 (Conv2D)	(None, 31, 31, 128)	73856
conv2d_26 (Conv2D)	(None, 29, 29, 128)	147584
max_pooling2d_11 (MaxPooling	(None, 14, 14, 128)	0
conv2d_27 (Conv2D)	(None, 14, 14, 256)	295168
conv2d_28 (Conv2D)	(None, 14, 14, 256)	590080
max_pooling2d_12 (MaxPooling	(None, 7, 7, 256)	0
dropout_8 (Dropout)	(None, 7, 7, 256)	0
flatten_3 (Flatten)	(None, 12544)	0
dense_5 (Dense)	(None, 64)	802880
dense_6 (Dense)	(None, 10)	650

Total params: 1,947,786
Trainable params: 1,947,786
Non-trainable params: 0