COMP 551 - Mini-Project 3

Jessica Helena Chan
William Bouchard
Frederic Beaupre

December 14, 2020

Abstract

In this project, we develop deep learning models to classify image data from a modified version of the MNIST dataset. Each gray-scale (64x64) pixel image is made up of 1-5 handwritten digits. We developed different Convolutional Neural Networks (CNNs) and implemented several methods to improve the training and validation accuracy. Two of those strategies allowed to surpass 99% test accuracy, one inspired by the VGGNet16 model and another which made use of OpenCV to separate and classify digits individually. After running multiple experiments, the model that yielded the best accuracy on the test dataset was the Medium Deep model (architecture detailed in Appendix) with Adam optimizer, trained on a pre-processed version of the dataset in which images were cropped and augmented.

1 Introduction

The goal of this project was to develop deep learning models to classify image data. The dataset used was a modification of the well-known MNIST Digits dataset. The modified dataset consists of images composed of 1-5 digits from the original MNIST dataset. We created a CNN that outputs a list of digits seen in the image. The CNN architecture we opted for was inspired from Simonyan and Zisserman's VGG-16 net [3]. We built different model versions of this architecture at varying depths and found that the Medium Deep model performed the best. For each model we experimented with the inclusion of maxpooling layers, tuned for hyper-parameters and varied the pre-processing of the input data to find the model which performed best. We refer to those models as "Original Model", "Medium-Deep Model", "Deep Model", and "Original-Pooled Model", whose architectures can be found in the Appendix. It was found that the model which performed best was the Medium Deep model with Adam optimization, trained on cropped and sheared input data.

2 Dataset

The dataset to be processed is the modified MNIST Digits dataset consisting of 56,000 gray-scale (64x64) pixel images each composed of 1-5 digits from the original MNIST dataset for training and 14,000 images of the same kind of images for testing. Each of the images were labelled with 5 consecutive numbers from 0 to 10, with 0-9 denoting the presence of that digit and 10 denoting the absence of a number. We loaded the dataset using the h5py package and converted it into a numpy array for use in this project. Labels were one hot encoded with Keras utils.to_categorical with shape (5x11) to preserve the order of numbers. We then used some heuristics to reduce the size of the data points and experimented with augmentation of the data. Specifically, we cropped the surrounding uninformative sections of the image data above and below the numbers at rows 20 and 40. To augment the data, we used keras ImageDataGenerator API to configure two data generators, one with shearing range 0.3 to account for italicized numbers [2], and another with rotation range 10, zoom range 0.1, and vertical and horizontal shift range 0.1 (referred to as "Aug-2" in Table 1). We found that cropping the image data yielded improvements to both run-time and accuracy, while both data generators resulted in a slight increase in accuracy.

Figure 1: Example of cropping and keras generated image shearing with range 0.3.

In addition, we also altered the dataset so as to make it possible to train our network on individual digits using OpenCV [1]. We then merged the altered dataset with the original MNIST Digits dataset, which consists of 70,000 gray-scale (28x28) pixel images of individual digits, which resulted in a total of 240,000 individual digit images in the training set. These images were labelled by classes 0-9 denoting the number depicted. This was used in the image segmentation approach (denoted OpenCV in Table 1) and discussed in the following section.

3 Strategies

We developed 2 different strategies in parallel for training and testing in order to compare test accuracies.

In our first approach, we drew inspiration from the VGGNet16 deep learning model to create a shallower version that would work with our data [3]. We started with our Original Model, which consists of triplets of convolutional layers with filter sizes (3x3), (3x3), (5x5). At the first triplet, we used 32 filters and increased this number by a factor of 2 as the model got deeper. Batch normalization was performed after each convolutional layer and a dropout layer of 0.4 was added after each triplet. We further deepened this model to compare results and found that the Medium Deep model yielded the best results, while the deepest model yielded the worst. We did not pursue further evaluation of the Deep model due to an excessively high runtime with low accuracy compared to the Original and Medium Deep models. Maxpooling2D with pool size (2x2) was added after each triplet in the Original Model but we found that this addition was detrimental to accuracy and did not pursue further evaluation.

In order to reduce the noise of having multiple digits in a single image, we used OpenCV to do image segmentation and separated the multiple digits into individual (28x28) pixel images. Considering that OpenCV rarely detected multiple boundaries for only one digit, an insignificant percentage of digits were wrongly extracted from the original image. This happened for around 300 digits out of 150,000 extracted digits. Then, we trained the model on this new dataset of individual digits and tested on the test dataset, which we had segmented using the same strategy. This gave a respectable accuracy of 98.99% on the test data. To make this model more robust, we added more training data from the MNIST dataset. Since we had constructed a dataset similar to the MNIST dataset using this modified version, we appended the original MNIST dataset's 70,000 images, which brought our number of training images to around 240,000. On this dataset, 10% was used for validation and 90% for training.

4 Models and Hyper-parameter Tuning

In trying to find the model that would achieve the best test accuracy, we built upon our Original model by adding MaxPooling2D layers, Dropout layers and convolutional layers with increasing numbers of filters. We started with a CNN whose full architecture summary can be found in the appendix (Figure 2). For these models, we drew inspiration from VGGNet16 to create a shallower version that would suit our task [3]. All models were implemented with Keras Sequential module [2].

The models were built using triplets of 2D convolutional layers with filter sizes (3x3), (3x3), (5x5). The final convolution of each triplet utilized a stride of size 2. Batch normaliza-

Model	Original Data	Sheared	Cropped	Aug-2	Sheared & Cropped	Adaptive-lr	OpenCV
`Original`	97.0	98.7	97.4	98.7	99.6	98.1	99.6
`Original-pooled`	95.8	96.3	98.2	96.2	97.4	96.5	98.2
`Medium-Deep`	98.9	99.8	99.7	99.8	99.8	99.8	99.1
`Deep`	82.8	-	-	-	-	-	-

Table 1: Validation accuracies for different models and experiments. Note that Adaptive-lr used the original dataset as input.

tion layers were inserted between convolutions and Dropout layers followed each triplet of convolutions. The triplets were followed by one 2D convolutional layer of filter size (4x4) followed by batch normalization and dropout. The number of filters used in each triplet started at 32 and increased by a factor of 2 for each triplet. All convolutional layers used ReLU activation. Finally, the resulting output was flattened and a dense layer with output size 55 was applied with softmax. This output was reshaped to (5x11) to match the shape of the one-hot encoded label matrix, and categorical cross entropy loss was used for learning. A visualization of the model architecture is available in the appendix (Figure 7).

The Original model consists of 2 triplets of convolutional layers with 32 filters in the first triplet and 64 filters in the second. The Medium Deep model added a third triplet with 128 filters. The Deep model added a fourth and fifth triplet with 256 and 512 filters respectively.

In hyper-parameter tuning our best model, we first addressed the dropout layer's parameter, which represents the probability of dropping a datapoint. We evaluate the model hyper-parameters with 5-fold cross validation, incrementing the dropout parameter by 0.1 starting at 0.1 such that we cover the following dropout probability values: $[0.1, 0.2, 0.3, 0.4, 0.5]$.

We also experimented with adding max pooling layers, cropping the images, augmenting the data, utilizing an adaptive learning rate and segmentation with OpenCV. We also searched for the best hyper-parameters for the Adam optimizer (learning rate α, β_1, β_2) using 5-fold cross-validation, which resulted in the graphs shown in Appendix A (Figures 4 and 5), giving us optimal values of ($\alpha = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.99$). Investigating the learning rate parameter further, we used a scheduler to allow the learning rate to vary over the training process. Based on observations from Figure 6 in the appendix, we see that the accuracy drops significantly as the learning rate moves away from $0 + \epsilon$, where ϵ is infinitesimally small. Hence we choose a scheduler $LR = 0.001 \times 0.95 \times LR$, which updates the learning rate at the beginning of every epoch, making it smaller and smaller from an initial, relatively large value of 0.1. This allows for large weight updates in the beginning of the learning process and smaller, more fine-tuned updates towards the end of the learning process.

5 Results and Discussion

The medium-deep model performed better than the original and deep models in all the experiments conducted in terms of validation accuracy. It achieved its best accuracy on 60% of the test data when trained on cropped and sheared images with 5-fold training. 5-fold training utilized the same structure as 5-fold cross validation, but conserved the model parameters over each fold. We believe that this model's architecture was sufficient to learn optimal weight parameters accurately without overfitting due to its moderate depth and number of parameters. We quickly ruled out the even deeper model as a best model candidate, as it seemed to succumb to overfitting and not generalize well to new data, yielding drastically reduced validation accuracy on the original and cropped data. We did not pursue further evaluation of the Deep model due to an excessively high run-time and the reduction in accuracy.

In our experiments, we found that cropping and augmenting the data increased validation accuracy across all viable model architectures. Combining cropping and image shearing measures yielded the best results, improving accuracy by 1-3 percent in the Original and Medium Deep models. The addition of 2D MaxPooling in the Original model resulted in a reduction in validation accuracy and we decided not to pursue further evaluation on this model. Since the image data consisted of thin lines, we concluded that MaxPooling would cause the the data to be coarsened in a manner detrimental to our model. Finally, we found that our adaptive learning rate implementation improved accuracy by 1-2% across all models.

6 Conclusion

In this project, we implemented deep learning models for multi-label classification on the modified version of the MNIST dataset. To do so, we drew inspiration from the VGGNet16 model [3] and implemented various experiments including maxpooling, normalization techniques and performing image segmentation with OpenCV. With our best model, we reached a test accuracy of 99.416% with the Medium Deep model using Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.99$, and learning rate = 0.001 trained on cropped and sheared images with 5-fold training.

In terms of improvements, we believe that given more computing power, we could have performed more extensive hyper-parameter tuning, which probably would have yielded some improvement on the accuracy. In addition to that, we probably would have had more time to run other experiments. Indeed, some of those experiments took more than half a day to run, which restricted our ability to thoroughly test alternatives due to time and computational power constraints. For further investigation, we would have tried to apply further preprocessing on the OpenCV segmented images in order to rotate the numbers, emphasize the contours, etc.

7 Statement of Contributions

The work was distributed evenly.

References

[1] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[2] Francois Chollet et al. *Keras*. 2015. URL: `https://github.com/fchollet/keras`.

[3] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: `1409.1556 [cs.CV]`.

8 Appendix

Model	Learning Rate	β_1	β_2	Dropout Rate	Test Accuracy
Medium Deep	0.01	0.90	0.99	0.4	99.416

Table 2: Best Model (Medium Deep) Hyper-Parameters and Test Accuracy

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	320
batch_normalization (BatchNo	(None, 62, 62, 32)	128
conv2d_1 (Conv2D)	(None, 60, 60, 32)	9248
batch_normalization_1 (Batch	(None, 60, 60, 32)	128
conv2d_2 (Conv2D)	(None, 30, 30, 32)	25632
batch_normalization_2 (Batch	(None, 30, 30, 32)	128
dropout (Dropout)	(None, 30, 30, 32)	0
conv2d_3 (Conv2D)	(None, 28, 28, 64)	18496
batch_normalization_3 (Batch	(None, 28, 28, 64)	256
conv2d_4 (Conv2D)	(None, 26, 26, 64)	36928
batch_normalization_4 (Batch	(None, 26, 26, 64)	256
conv2d_5 (Conv2D)	(None, 13, 13, 64)	102464
batch_normalization_5 (Batch	(None, 13, 13, 64)	256
dropout_1 (Dropout)	(None, 13, 13, 64)	0
conv2d_6 (Conv2D)	(None, 10, 10, 128)	131200
batch_normalization_6 (Batch	(None, 10, 10, 128)	512
flatten (Flatten)	(None, 12800)	0
dropout_2 (Dropout)	(None, 12800)	0
dense (Dense)	(None, 55)	704055
reshape (Reshape)	(None, 5, 11)	0

Total params: 1,030,007
Trainable params: 1,029,175
Non-trainable params: 832

Figure 2: Original Network Architecture

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	320
batch_normalization (BatchNo	(None, 64, 64, 32)	128
conv2d_1 (Conv2D)	(None, 64, 64, 32)	9248
batch_normalization_1 (Batch	(None, 64, 64, 32)	128
conv2d_2 (Conv2D)	(None, 32, 32, 32)	25632
batch_normalization_2 (Batch	(None, 32, 32, 32)	128
dropout (Dropout)	(None, 32, 32, 32)	0
conv2d_3 (Conv2D)	(None, 32, 32, 64)	18496
batch_normalization_3 (Batch	(None, 32, 32, 64)	256
conv2d_4 (Conv2D)	(None, 32, 32, 64)	36928
batch_normalization_4 (Batch	(None, 32, 32, 64)	256
conv2d_5 (Conv2D)	(None, 16, 16, 64)	102464
batch_normalization_5 (Batch	(None, 16, 16, 64)	256
dropout_1 (Dropout)	(None, 16, 16, 64)	0
conv2d_6 (Conv2D)	(None, 16, 16, 128)	73856
batch_normalization_6 (Batch	(None, 16, 16, 128)	512
conv2d_7 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_7 (Batch	(None, 16, 16, 128)	512
conv2d_8 (Conv2D)	(None, 8, 8, 128)	409728
batch_normalization_8 (Batch	(None, 8, 8, 128)	512
dropout_2 (Dropout)	(None, 8, 8, 128)	0
conv2d_9 (Conv2D)	(None, 8, 8, 32)	65568
batch_normalization_9 (Batch	(None, 8, 8, 32)	128
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 55)	112695
reshape (Reshape)	(None, 5, 11)	0

Total params: 1,005,335
Trainable params: 1,003,927
Non-trainable params: 1,408

Figure 3: Medium Deep Model Architecture

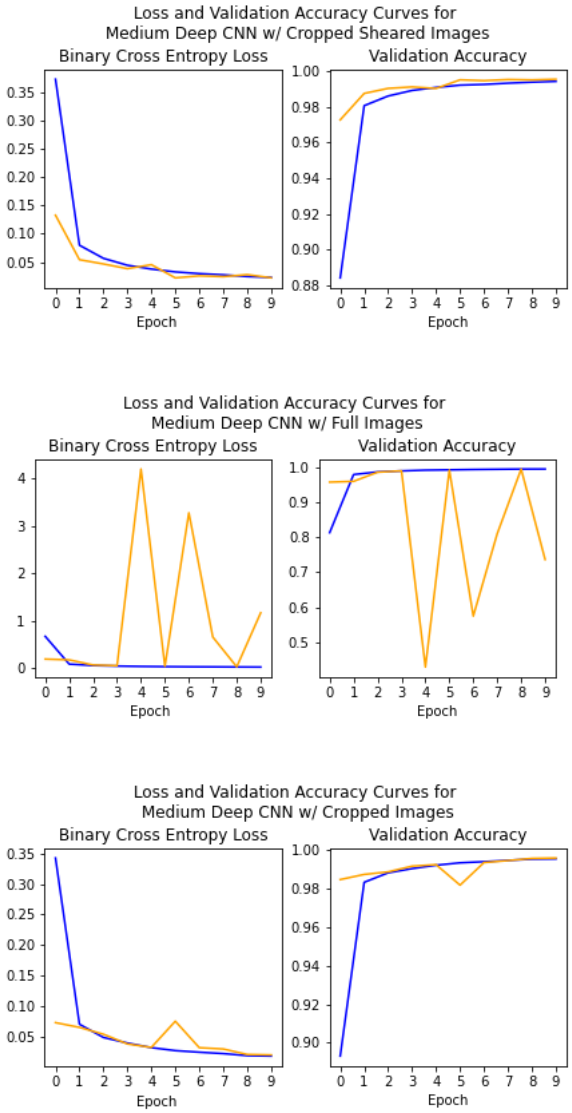

Figure 4: Training/validation curves. Where orange represents validation and blue represents training.

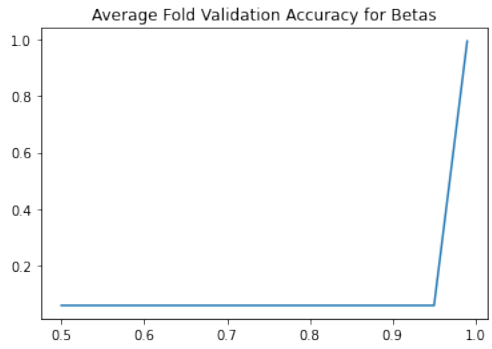

Figure 5: Testing for β_1 and β_2 in Adam optimizer with search space: $[0.5, 0.75, 0.8, 0.9, 0.93, 0.95, 0.99]$. Where y-axis represent validation accuracy and x-axis represents β values.

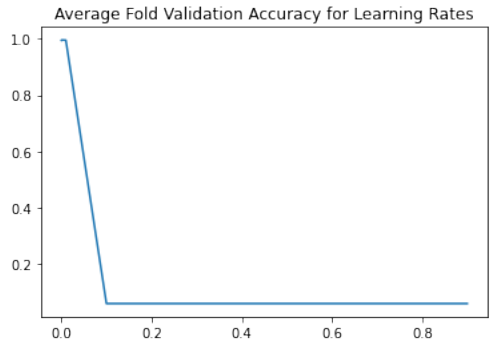

Figure 6: Testing for learning rates in Adam optimizer with search space: $[0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.8, 0.9]$.

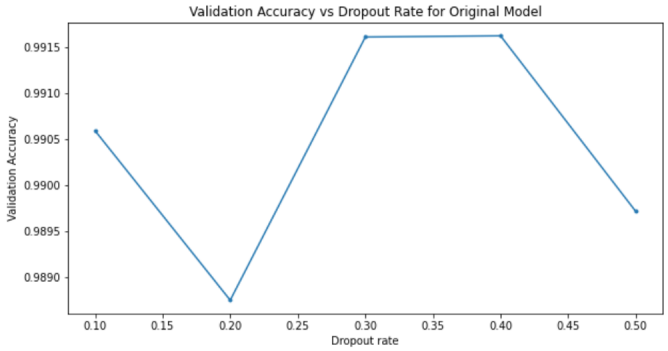

Figure 7: Testing for dropout rate in Original Model with search space: $[0.1, 0.2, 0.3, 0.4, 0.5]$. Where y-axis represent validation accuracy and x-axis represents learning rate.

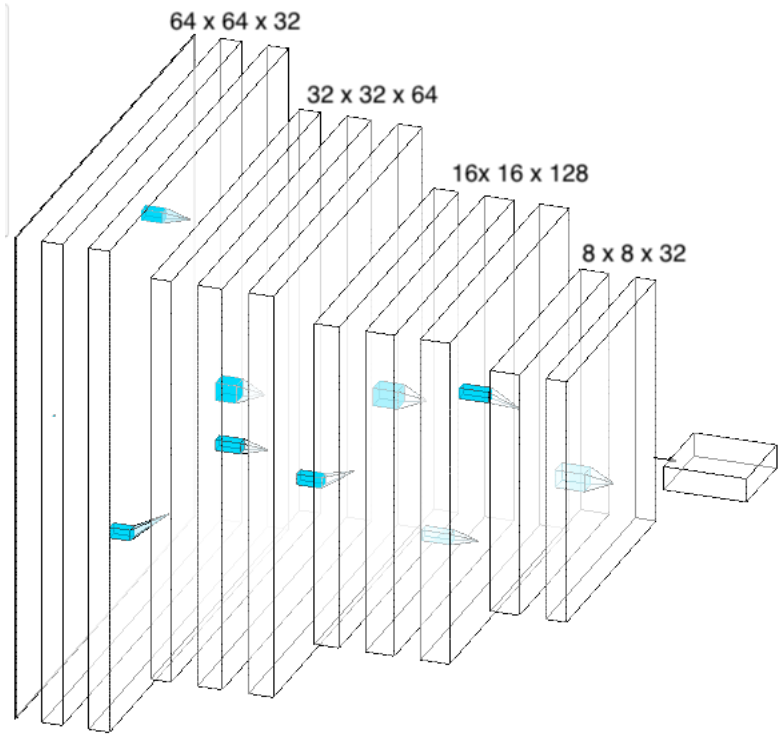

Figure 8: Visualization of convolutions in best (Medium Deep) model.