COMP 551 Project 3: Modified MNIST

Daniel Chernis	260707258
Archit Gupta	260720757
Chenthuran Sivanandan	260749843

Abstract

The goal of this project was to correctly identify the maximum number in an image, given three numbers in the image (with numbers ranging from 0 to 9). We classified each image as belonging to one of ten classes. Using Keras, we implemented a Convolutional Neural Network (CNN) with 6 Convolution2D and Max Pooling layers. We tested different models by varying the number of epochs, batch sizes and the number of layers. We evaluated the performance of each model in terms of accuracy on our validation set. Using 10 epochs, 6 layers and a batch size of 200 we reached an accuracy of 93.22% on our validation set. To speed up training we used the GPU's available on Google Colaboratory. To reduce overfitting in the fully connected layers we used dropout.

Introduction

As suggested in [5], to build the Convolutional Neural Network (CNN), a combination of 2D convolution and Max Pooling layers was applied with a ReLU activation function in between. Given the complexity in the images, the number of such layers was increased to 6 to capture low-level details even further, but at the cost of more computational power. ReLU was applied since it is generally considered one of the most reliable activation functions [2].

Since "adaptive" optimization algorithms modify the learning rate depending on how the training is progressing, we used Adam, one of the most popular optimization algorithms in deep learning. Adam scales the learning rate for each parameter based on the statistical history of gradient updates for that parameter. The intuition is to increase the update strength for parameters that have had smaller updates in the past.

The dataset consisted of 50,000 grey scale images each of which was 128 x 128 pixels. These pixels served as the features for our model. Each image had a label assigned to it to represent the largest number that should be found in the image. This value represents the expected output.

Related Work

In [1], the authors used a deep CNN to classify the 1.2 million high resolution images in the ImageNet contest into 1000 different classes. On the test data, they achieved top-1 and top-5 error rates of 37.5% and 17.0% which was a higher accuracy than the state of the art model of 2011. The neural network (NN) had 60 million parameters, and 650,000 neurons. Additionally there were 5 convolution layers, some of which were followed by a pooling layer, as well as 3 fully connected layers.

In [7], the authors investigated the effect of the convolutional network depth on it's accuracy in the large scale image recognition setting. They used an architecture with 3x3 convolutional filters. They found that increasing the network depth improves results significantly. The authors final model was able to secure the first and second place spot in the localisation and classification tracks of the 2014 ImageNet Challenge.

The original MNIST dataset was already classified using Tensorflow Keras [3],[6], with an accuracy of over 99.2%. Using that setup creates a straightforward and efficient way to classify that data using CNNs. Furthermore, [4] showed that Echo State Networks (Recurrent Neural Networks) were not able to outperform CNNs to classify the digits of the MNIST dataset. CNNs are better suited for image classification. This pushes the idea that CNNs will provide the optimal solution.

Dataset and Setup

During the fitting of our CNN, we divided the data into a 90-10 training-validation split. After deserializing (byte stream to an object structure, Figure 1) the training and test images using Python's pickle module, the images were binarized (converted to black and white, Figure 2). A binary image is a digital image that has only two possible values for each pixel. Typically, the two colors used for a binary image are black and white. The color used for the object in the image is the foreground color while the rest of the image is the background color. The pixels of the black numbers in the original dataset were set to white while the remaining pixels were set to black.

In order to determine the intensity at which a pixel is considered black or white, a threshold of 220px (on a scale from 0 to 255) was found by trial and error i.e. if intensity of a particular pixel exceeds the threshold value it symbolizes light region or binary 1 else it represents dark region or binary 0.

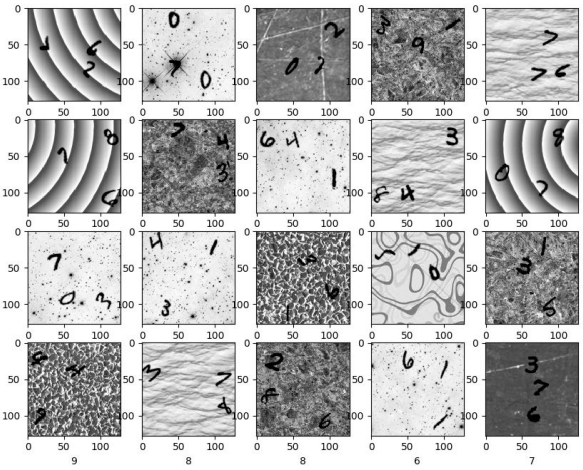

Figure 1 ~ Original Dataset

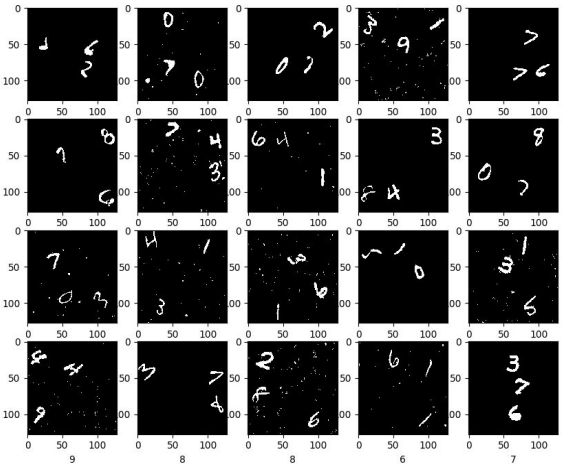

Figure 2 ~ Binarized Dataset

Proposed Approach

We decided to use Tensorflow's Keras library to build the CNN. After the data preprocessing, a model for the CNN was set up as a model entity. The model type used was Sequential since it the easiest way to build a model in Keras. It allows you to build a model layer by layer. The structure of our CNN that builds up to our model is the following:

1. 2 layers of 2D Convolution with a filter of 32 and a kernel size of 5x5
2. ReLU Activation Function
3. Max Pooling with weight matrix of size 2x2
4. 2D Convolution with a filter of 64 and a kernel size of 3x3
5. ReLU Activation Function
6. Max Pooling with weight matrix of size 2x2
7. 2D Convolution with a filter of 128 and a kernel size of 3x3
8. ReLU Activation Function

9. Max Pooling with weight matrix of size 2x2
10. 2D Convolution with a filter of 256 and a kernel size of 3x3
11. ReLU Activation Function
12. Max Pooling with weight matrix of size 2x2
13. Dropout with rate 0.25
14. Flatten
15. Dense with dimensionality 128 for the number of rows
16. Dropout with drop rate of 0.5
17. Dense with dimensionality of 10 (classification of 0 to 9) and Softmax activation function.

The first layer is two 2D Convolutions as suggested in [3]. These layers will deal with our input images, which are seen as 2-dimensional matrices. The filter number is the number of nodes in each layer. It is adjusted to be higher as we go along since we want to get more and more detail out of the dataset. The Kernel size is the size of the filter matrix for our convolution. It starts as size 5 since we want the model to start faster even at the cost of losing some information at the beginning. Then, we pair max pool with weight matrix of 2x2 and a 2D Convolution as it is considered a good match up [5], and pooling discards some noise, and performs dimensionality reduction to avoid overfitting and training too long.

The activation function used in between all layers is the ReLU as it has been proven to work well in NN. Our first layer also takes in an input shape (128,128,1), the dimensionality of each input image with the 1 signifying that the images are grayscale.

After 4 ‘convolution-pooling’ layers, there is a ‘Dropout’ which consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting. Then there is a ‘Flatten’ layer which serves as a connection between the convolution and dense layers. Flatten serves as a connection between the convolution and dense layers. The ‘Dense’ layer with default linear activation function was used for our output layer. It is a standard layer type that is used in many cases for NN. We will have 10 nodes in the output layer, one for each possible outcome (0–9).

The final activation is ‘Softmax’ as it makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability to find the largest number in each image.

After setting it up, we need to compile our model. Compiling the model takes 3 parameters: optimizer, loss and metrics. The optimizer controls the learning rate. We picked the adaptive optimization ‘Adam’ as our optimizer since it is a standard in deep learning and a good optimizer to use for many cases. It improves the learning rate through training which determines how fast the optimal weights for the model are calculated. A smaller learning rate may lead to more accurate weights (up to a certain point), but the learning time will be longer. We used ‘categorical_crossentropy’ for our loss function. This is the most common choice for classification. A lower score indicates that the model is performing better. We used the ‘accuracy’ metric to see the accuracy score on the validation set when we train the model.

Finally, peak accuracy was calculated by running the model many times for 1 - 10 epochs.

Results

Our Kaggle leaderboard test set accuracy is 92.4 %.

The following are the results of our experiments on the validation set based on the number of epochs. Using Google GPU's on Google Colab, the runtime for 1 epoch is on average 1.5 minutes. Therefore, even though we get a higher validation set accuracy when we increase the number of epochs, this comes at a direct cost of greater runtime. The graph for the runtimes vs the number of epochs is shown in figure 4.

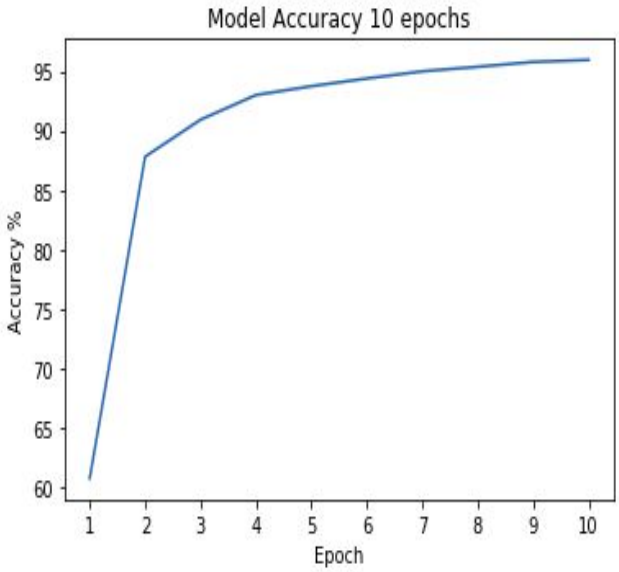

Figure 3 - Epochs v/s Validation Accuracy

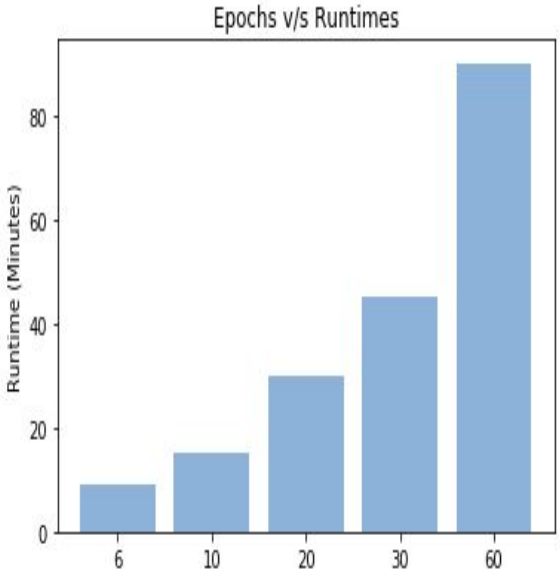

Figure 4 - Epochs v/s Runtimes

We also looked into the effect that the batch size had on the accuracy of the validation set. In general, we noticed that a higher batch size led to a decrease in the accuracy of the validation set. This makes sense when we consider the fact that the size of the batch is essentially the frequency of updates, so with a smaller batch size we will be updating the weights more frequently than with a larger batch size. We ran our experiments with 10 epochs.

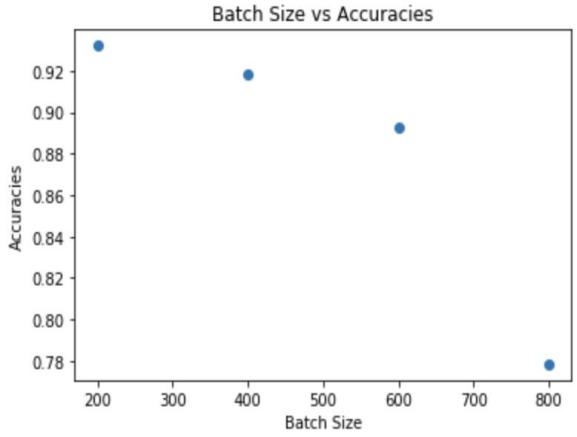

Figure 5 - Batch Size v/s Validation Accuracy

Discussion and Conclusions

As the complexity of our model increases, our accuracy increased. We tested different epoch sizes until we reached a maxima. We then tried adding a Convolution with 512 filters but the accuracy decreased to 90%. This implied that after a certain complexity and epoch count of our CNN, we start losing rather

useful information that would otherwise help classification. If we train for too long, the model's learning rate slows down and starts causing overfitting which in return decreases classification accuracy. Experimentally, the model needs to balance the complexity of the model and the training time to maximize accuracy.

In the future we could diversify the CNN by adding different types of layers like VGGNets mentioned previously. They also seemed to give higher accuracies for our colleagues doing the same experiments.

Another potential improvement is to add Momentum. At each iteration of the gradient descent, we are computing an update based on the derivative of the current (mini) batch of training examples. This would allow to overcome some lower local minima and find higher ones for accuracy.

Statement of Contribution

All members contributed equally

References

[1] Alex Krishevsky, Ilya Sutskever, and Geoffrey E. HintonG. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. ImageNet Classification with Deep Convolutional Neural Networks 2012.
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[2]
Anish Singh Walia,
https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

[3] Eijaz Allibhai,
https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5

[4] Nils Schaetti, Michel Salomon, and Raphaël Couturier. Echo state networks-based reservoir computing for mnist handwritten digits recognition. In Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC), 2016 IEEE Intl Conference on, pages 484–491. IEEE, 2016. https://hal.archives-ouvertes.fr/hal-02131170/document

[5] Sumit Saha,
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[6] Tyler Elliot Bettilyon,
https://medium.com/tebs-lab/how-to-classify-mnist-digits-with-different-neural-network-architectures-39c75a0f03e3

[7] Karen Simonyan & Andrew Zisserman. Very Deep Convolutional Neural Networks For Large Scale Recognition.2015. https://arxiv.org/pdf/1409.1556v6.pdf