COMP 551 – Applied Machine Learning
Mini-Project 3

Omar Abdel Baky
Tal Elbaz
Devin Kreuzer

November 13, 2019

Abstract

Machine learning algorithms are at the forefront of data analysis. A well-known problem in Machine Learning algorithm development is the MNIST database problem, where the task is to develop a method to predict the value of hand-written digits in image data. The following paper will elaborate on methods to solve a modified version of the MNIST problem, in which we must predict the largest of three hand-written digits in images with non-empty backgrounds. In order to do so, we developed our own Convolutional Neural Networks (CNNs) and touched upon well-known algorithms that excel for the standard MNIST problem. We modified these networks by adding extra layers and training it with different parameters, batch sizes, activation functions among others to develop the best model possible. Our best model achieved a 96.50 % accuracy score on the Kaggle competition dataset.

1 Introduction

Machine learning has become the new popular means for analyzing various forms of information. In this project, we explore a specific type of information; images. The MNIST problem is a ubiquitous image processing problem in which participants must develop an algorithm capable of recognizing images of hand-written digits. The following paper describes the process of utilizing well-known machine algorithms, notably convolutional neural networks, to recognize the largest of three hand-written numerical digits arranged in a single image.

1.1 Preliminaries

Neural Network: A model which is a network of many base models whose outputs are relayed as inputs to other models to continuously improve outputs at every layer. This algorithm enables the computer to recognize underlying patterns in the data that are not obvious to the human eye. A popular technique in neural networks is to relay all outputs of one layer to all inputs of the next layer, which is defined as a fully connected layer. By finding the optimal weights to attribute to the base models, it is capable of optimizing the error function of the predictions.

Convolutional Neural Network (CNN): A specific class of neural networks consisting of deep networks (many layers), widely used in image classification. What differentiates them from other neural networks is their attempts to avoid over-fitting of the data, which occurs when using fully connected layers. CNNs attempt to reduce the number of connections and parameters across the network without decreasing any efficiency. In the example for image classification, pixels are grouped through a mapping in sizes of 3x3 or 5x5 depending on the method used.

VGGnet: A CNN designed to compete in image processing/classification competitions that scored same of the best results. The model is available online and is widely recognized as one of the best algorithms for image recognition. It is composed of a series of layers best represented in Figure 1.

[**?**].

Inception V3: The runner up CNN in the 2015

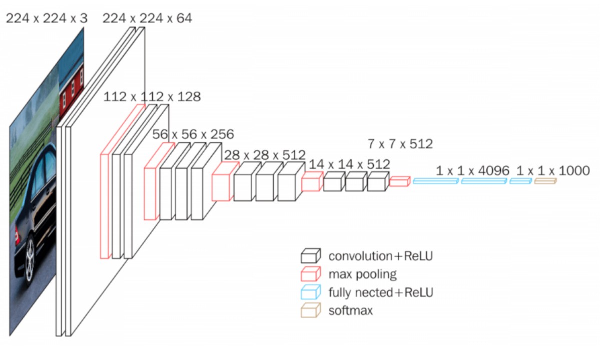

Figure 1: Depicition of VGG16

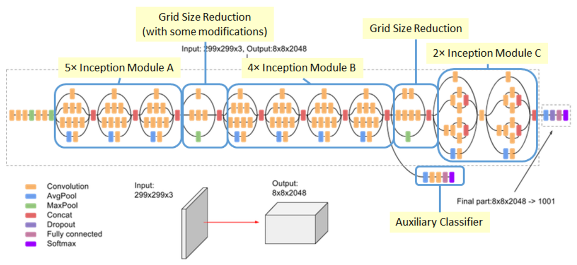

Figure 2: Depiction of IV3

ILSVRC competition with an error rate of 3.58%. The model was designed to compete with VGG16 and contains 42 layers across its network that can be visualized in Figure 2.

1.2 Task

The goal of this project is to create a model capable of predicting the largest of three hand-written numbers in an image which also contains non-empty backgrounds.

2 Related Work

There has been lots of work done in attempts to maximize the accuracy of classifying the MNIST images. One of the most popular techniques was presented by Lecun et al. [?], which consisted of training a multi-layer neural network using the back-propagation algorithm (computes the gradient of every node and propagates the error backwards from the output node). After trying many different algorithms, the one that obtained the best error consisted of running the same algorithm connected to itself three times. This technique is called boosting; combining multiple classifiers to create one optimal classifier. The base algorithm consisted of 6 layers, with 5 convolutional ones (only one layer was fully connected). Using this method, they trained the first base model with the original images as inputs. The second model is then trained on patterns recognized by the first one, meaning patterns where the model was correct in addition to patterns where it got it wrong. For example, say got the number 8 right almost every time, or if it confused 1 and 7 many times. These would be the kinds of patterns it would take into account. Lastly, the third model trains on the points where the first two models disagree. This is done to optimize when the first model is correct versus when the second one is. All together, it provided their most effective classifier.

Another very successful algorithm achieving an even better error in the MNIST problem than the one presented above (Ciresan et al. [?]) uses a very similar technique. They use six convolutional mapping layers to subset the pixels, of increasing sizes. After these six layers, each node of the last layer is fully connected to a layer of 150 neurons, and then outputted to 10 nodes. They tried different numbers of convolutional layers, and achieved increasing accuracy with every added layer. However, the more layers, the more computationally expensive and was optimized at 6 layers. This model achieved the record accuracy for the MNIST dataset when it was published.

Finally, a model was created by Chen et al. [?] that was so good it was capable of beating human accuracy on the MNIST dataset. Their model, following the same idea as the ones above, has 8 layers in its network. In addition, a voting scheme was also used, but this time with 5 base models.

From these projects, we may deduce that this type of framework has proved to be the most effective, and while the MNIST problem is not quite identical to ours, it has many similarities.

3 Dataset

The dataset consists of Matrices representing grey-scale images with random backgrounds and random digits from the MNIST handwritten digit dataset in random orientations. Each entry of the matrix represented the brightness value of a pixel, taking on values from 0-255. The dataset contains 50000 total images.

3.1 Pre-processing

In order to properly run the algorithm, the data must be organized and pre-processed. To pre-process the data we divided the pixel values by 255 to scale down the values to 0-1 from 0-255 and we set the pixel values for all pixels with brightness value less than 50/255 to 0 to whiten the digits. In the following figures, we can see the difference between the raw input data and the input data following our pre-processing. Furthermore, we

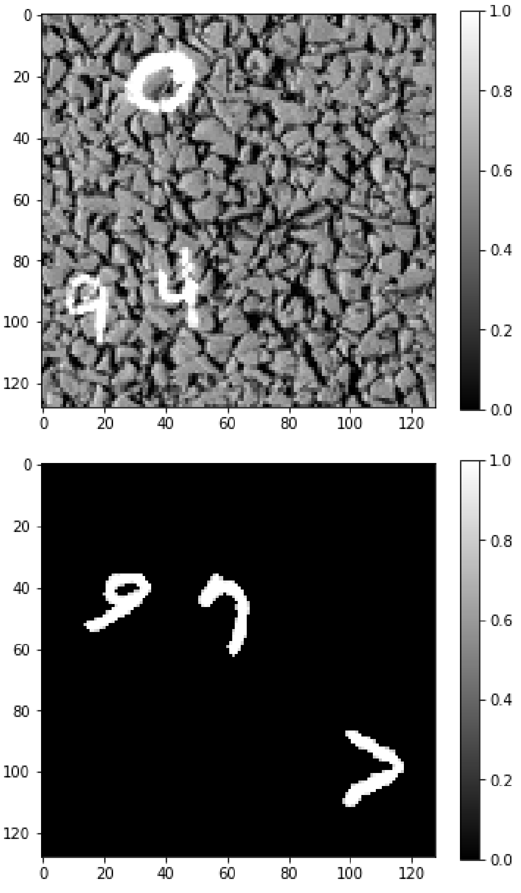

Figure 3: Image data before and after pre-processing

will be using Tensorflow's ImageDataGenerator, which generates batches of images and augments them in accordance to some pre-built functions. Notably, we will be using the built-in rotation range feature, which will account for the fact that digits can be rotated within the image.

4 Proposed approach

Our first proposed approach is to develop a simple CNN with Keras with Tensorflow backend using varying numbers of convolutional layers with max pooling, dropout(1%), and a final fully-connected 10-node layer with a softmax activation.

Our second and third approaches will be to stack layers on top of the pre-loaded VGG16 and IV3 networks to cater them to our modified version of MNIST. The layers we will include for the IV3 include a 2D Average Pooling as well as 1024, 512, 256 and 10-node fully connected layers with Relu activation (we learnt in class that these tended to be optimal). 2D average pooling consists of taking the average value of the input matrix, which in the this case is the subset of pixels. This reduces the size of the data as well as controlling for overfitting. We decided to add the dense layers because IV3 outputs a 2048x8x8 tensor, and we wish to gradually diminish the dimension of the output down to 10 (the dimension of our output after categorical encoding). From the IV3 output layer, we will experiment with Flattening and Average pooling. For the VGGnet, since the model ends with a fully-connected 1000 values, we will gradually reduce the dimension down to 10 as well. To properly compare all methods, we will be maintaining the adams optimizer throughout.

Furthermore, we will be experimenting with different parameters for the ImageDataGenerator function including batch sizes, pre-processing functions as well as rotation ranges. Since it often takes multiple epochs to train the model, we will be saving and reloading the weights between training periods.

5 Results

For our base CNNs, we used a 5:1 training validation split to train and monitor performance. With two convolutional layers, the performance was 25% accuracy. Four convolutional layer CNNs had significantly better performance, with our first prototype network reaching 82% success rate after 400 epochs and our improved CNN reaching 84% after 200 epochs. Interestingly, the slower learning CNNs overfit while our faster CNNs

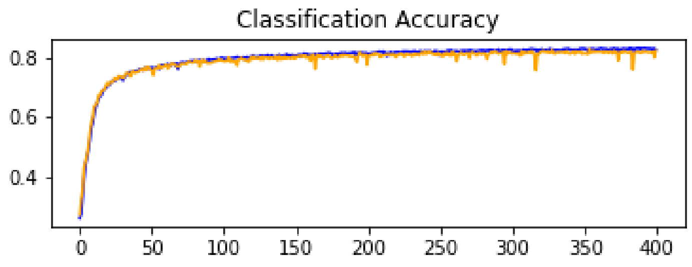

Figure 4: Original CNN training. Training accuracy (blue) and Validation accuracy (gold)

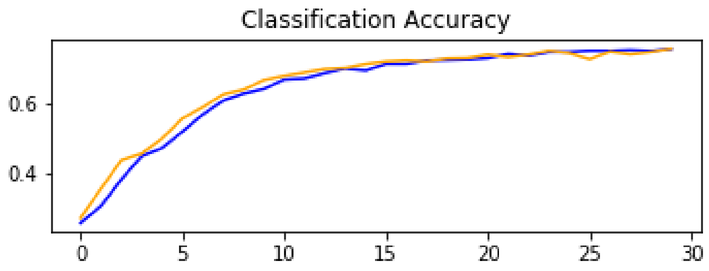

Figure 5: Improved CNN training. Training accuracy (blue) and Validation accuracy (gold)

showed no signs of significant overfit with the validation accuracy staying very close to training accuracy only getting a little more unstable even when learning was plateauing (see figure 4 and figure below.). Our improved model scored 84% on the leaderboard at 200 epochs.

As for the pre-trained models, we used an 80-20 validation split and observed much better performance, though it took many epochs and many hours of training to arrive at our best model. Since we struggled to keep our VGGnet to work, we maintained our focus on the IV3 model.

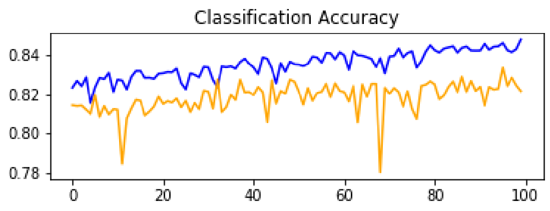

Figure 6: Improved CNN training Epoch 100-200. Training accuracy (blue) and Validation accuracy (gold)

Since the model took long to run and we wanted to experiment with different amounts of layer-freezing and activation functions, we continuously saved and re-loaded the weights between different training blocks. The following figure depicts the one of the last training blocks in which our model improved tremendously. During this phase, we unfroze the layers of the model and trained the entire network. Our final accuracy on the Kaggle test set reached 96.50%.

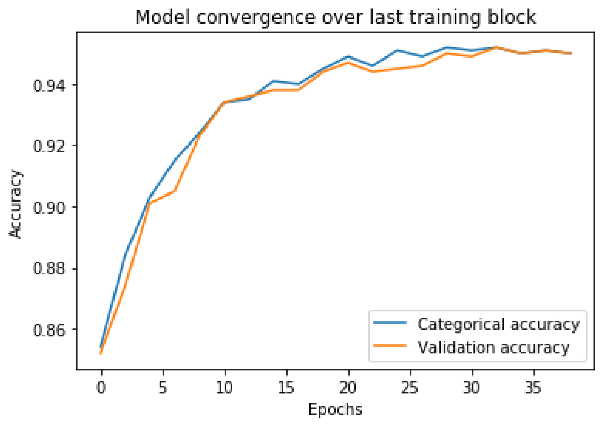

Figure 7: IV3 model convergence

	IV3	Our CNN
Duration of one epoch (s)	173	57
Validation accuracy (%)	96.73	84.64
Kaggle score (%)	96.50	84.00

Table 1: IV3 vs our CNN

6 Discussion and Conclusion

In conclusion, convolutional neural networks are indeed very good models for image processing problems. Of our three proposed approaches, using IV3 as a base model and adding layers to it worked best. This is similar to the boosting method used by Lecun et al. [?], as the base model's outputs are evaluated and then re-evaluated when saving the weights at the end of every iteration and reloading them at the beginning of the next, allowing the model to find underlying patterns in the data for more accurate predicting. We would like to gain better insight into the VGG network as we had difficulty implementing additional layers to the model in such a way that it boosted its performance.

7 Statement of Contributions

Omar: Training and improvement of Homemade CNNs Devin: Testing and training the IV3 and VVG networks Tal:Testing and training the IV3 and VVG networks