COMP 551 Project 3 Report: Group 25

Elie Climan 260686400, Dominik Stolz 260926700

November 14, 2019

Abstract

The image processing analysis challenge presented in this report sought to train a Machine Learning model which for each image sample predicts the largest out of several handwritten digits on that image. Ultimately, the goal was to identify the Convolutional Neural Networks which yields the best results, which we deem as a high prediction accuracy given a dataset of MNIST numbers. This involved light preprocessing, training various types of neural networks and optimizing using different activation and regularization functions. Ultimately, we found that a 4 layer VGG16 net style Convolutional Neural Network using the Keras API yielded the best resulted with the highest accuracy recorded on Kaggle being %93.6.

Introduction

Handwritten digit analysis is one of the most important underlying techniques in Machine Learning. Its application can be easily expanded to include any image processing such as document scanning or extracting key information from images such as a HazMat logo on the back of a truck. The most common training data used in this field is the MNIST dataset.

The purpose of this work was to implement a Convelutional Neural Network (CNN) in Python using libraries such as PyTorch and Keras in order to apply machine learning techniques on the famous MNIST Dataset, which has been modified for the purposes of this exercise. Ultimately, the goal was to design a model capable of predicting the largest digit value given an image composed of three digits from the original MNIST dataset as well as a noisy background. Many of the most successful MNIST classification algorithms have been championed by CNNs, such as the work done by *dan,ueli,luca,juergen* in 2010 which also evaluated a CNN on the, more challenging, NIST dataset acheiving a $\%0.27 \pm 0.02$ accuracy.

With limited prepossessing and using CNNs of different sizes and complexity, a confidence rate of " " was achieved. In order to achieve said results, we explored the effect of various hyper-parameters such as batch size, the numbers of epochs, network depth, activation functions and regularization values in order to yield the best results. After 40 epochs, we found that using Keras and designing several nets to mimic a similar style as the VGG16, as per Simonyan Zisserman [4], with a ReLU activation function and Adam optimizer, yielded the best results.

Related Work

- VGG16 Net. Proposed by **Simonyan and Zisserman [2015]** [4], essentially broke down the idea of a 16 net Neural Network with max pooling coupled with a ReLu activation function. This is what drove the creation of the Keras model that doubles the nets in size after each Dropout period.

- Neural network-based symbol recognition using a few labeled samples by **Fu, Kara [2011]** [1] represented the link between Keras and the VGG16 net. This paper provided many graphi-

cal representations about how a VGG16 can be refashioned.

- Neural network-based symbol recognition using a few labeled samples by **Fu, Kara [2011]** [1] represented the link between Keras and the VGG16 net. This paper provided many graphical representations about how a VGG16 can be refashioned.

- Building off *e Tato Roger Nkambou* [5], we were able to further understand the role of the Adam optimizer in dynamically optimizing the learning rate

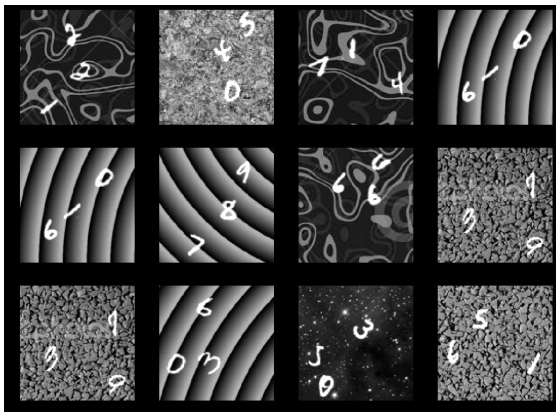

Figure 1: Original Images

Datasets

The Dataset is a modified version of the MNIST dataset, containing 50 000 training images and additional 10 000 test images. Each image contains a three handwritten digits 0-9 in RGB value 255 – completely white. The background was relatively dark in colour and has various curvatures and shapes, otherwise called the noise. Prepossessing for CNNs is fairly straightforward. A binary value was assigned to all RGB colours under a 255 value threshold in order to separate the background from the number itself. Once the noise was completely dropped as per Figure 2, we used a contour package to isolate the digits from the background as seen in Figure 3. However, in our experiments, we learned that isolating each of the three numbers constituted an extra effort not worth the gain in performance. Since Neural Networks are not predicated on any hyper-plane or linear classification, so simply making the relevant data more pronounced relative to the noise was the only prepossessing that was done. As seen in Figure 3, The distribution of the labels is heavily skewed towards the larger range of the spectrum. This can be explained by the fact that the largest number of each image is chosen to be its representative, therefore, with each lower number, it is to be expected that it is trumped more often by a larger one.

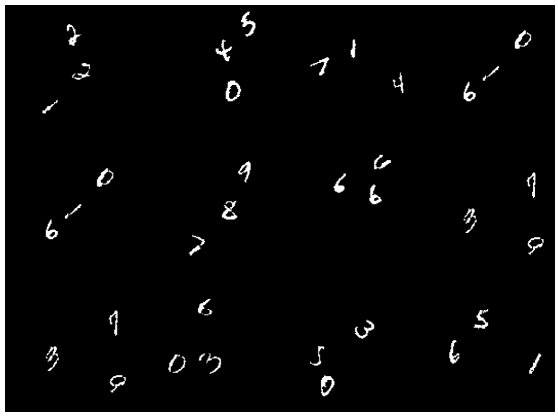

Figure 2: Post Preprocessing Step 1

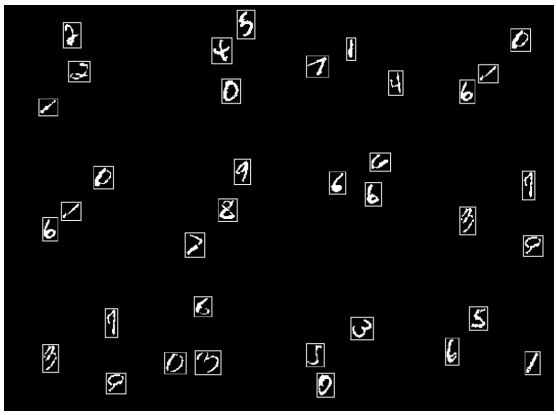

Figure 3: Post Preprocessing Step 2

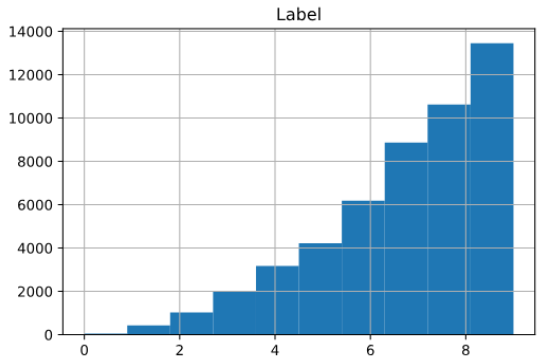

Figure 4: Distribution of image labels.

Proposed Approach

Before presenting our proposed approach we will discuss alternative approaches we investigated, which however were found to be inferior to our main approach. Our first approach involved using pretrained image classification networks such as VGG16 or ResNet-18 as feature extractors. In literature this approach is also known as Transfer Learning [3]. This allows much faster training, as only the weights of the last, fully-connected layers have to be learned while the kernels of the convolutional layers stay unchanged.

Another approach which we ended up discarding, was inspired by the Spatial Transformer Network architecture [2]. In this architecture, the input image is transformed with a learned tranformation matrix prior to being fed into the convlutional layers. The transformation matrix is learned using another smaller convolutional net called the localization network and a fully connected layer to reduce its output to a 3x2 matrix. The benefit of Spatial Transformer Networks is improved invariance to rotation, scale and translation.

A core part of our proposed approach is a validation pipeline which uses 15% of the labeled training data as validation set. After each epoch, the model was evaluated using this validation set to recognize overfitting early.

Concerning the model architecture, we found the VGG16 architecture using Keras to be the most straightforward and intuitive to implement. The model was designed using 2D Convolutional Neural Nets (CNN), Max pooling and dropout layers for regularization. Each layer of the neural net employed a filter which doubled, as per the VGG16 net packaged with a ReLu activation function and followed by a Max Pooling function which downscales the output by factors of 2 (vertically and horizontally) at each layer of the net. Lastly, a Dropout rate of 0.5 was added at the end of each layer, except the first layer. This acts as a form of regularization which arbitrarily ignores half of the neurons of that previous layer in order to improve the bias of the model.

Type	Output	Kernel	Activation
Conv2D	32	3	relu
Conv2D	32	3	relu
MaxPool2D		2	
Conv2D	64	3	relu
Conv2D	64	3	relu
MaxPool2D		2	
Dropout			
Conv2D	128	3	relu
Conv2D	128	3	relu
MaxPool2D		2	
Dropout			
Conv2D	256	3	relu
Conv2D	256	3	relu
MaxPool2D		2	
Dropout			
Dense	512		relu
Dropout			
Dense	128		relu
Dense	10		softmax

Table 1: Model architecture

Once the model was trained, we entered an epoch of 40. Due to the rate of convergence, we found that 40 provided enough passes through the data to guarantee a reasonably good global maximum in terms of accuracy and global minimum in terms of loss. Furthermore, given the size of the dataset, we went with a batch size of 200 which we found yielded the best results at a reasonable speed without sacrificing accuracy.

Results

Ultimately, there was a stark difference upon running the model against the test and validation data. We found that, while the pre-trained model was deeper, boasting a 16 layer neural net, its "black box" nature made it hard to optimize and ameliorate. As shown in Table 2 below, the Keras implemented 3 layer VGG16 style implementation consistently yielded better results with the test and training data. Based on our iterations, we found that being able to change the parameters of each layer and custom fit an activation function allowed us to fine tune to a final model much more precisely and with more understanding than with the pre-trained model. Some of these notable parameters include the number of epochs which, as per Table 1 shows rapid growth through the 6th first epochs but began to converge beyond that point.

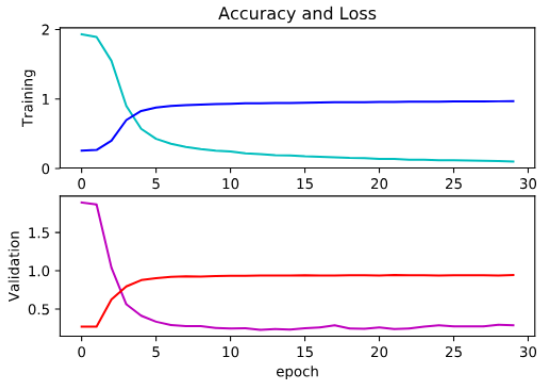

Figure 5: Accuracy (blue/red) and loss (cyan/magenta) during training.

	Validation Accuracy [%]
Epoch 1	27.05
Epoch 2	27.05
Epoch 3	62.65
Epoch 4	77.19
Epoch 5	86.76
Epoch 6	92.47

Table 2: Val loss by epoch, data began to converge beyond this point

	Keras	VGG16
Train Accuracy [%]	90.47	59.87
Test Accuracy [%]	96.6	59.53

Table 3: Comparison of computation time and accuracy between PyTorch Pretrained VGG16 and a VGG16 based approach using Keras

Discussion and Conclusion

For CNNs, it is imperative to understand your dataset. It is also imperative to identify the hyper-parameters available to you when designing your neural net. It can be as simple as the number of epochs and the batch size or as complex as the choice in optimization function. Furthermore, preprocessing is a small step which yielded very limitted improvements but proved paramount to the functionality of the neural net. We found that improper thresholding resulted in numerous adverse effects such as rapid convergence on the training set with high overfitting. Once this was implemented, however, these is bountiful research about the benefits and downsides of various activation functions, optimizers and their organizational structure relative to the neural net itself. To address this complexity of order and how to optimize the hyper-parameters at our disposal, we found ourselve, largely reading the Keras documentation and making qualitative observations about our data in order to arrive at the most optimal hyper-parameters. Such as the categorical crossentropy and the adam optimizer which used momentum to greatly speed up the processing time of the training. We also found out how over-fitting can be addressed using dropout rates in between layers in order to add more randomization to the learning procedure before entering the fully connected layer.

Ultimately, we provided meaningful results on the data-set, the %93.6 accuracy is representative of the hyper-paramaters which were carefully chosen and a Convolutional Neural Network that was well crafted. To address this challenge of learning rate selection, we implemented learning rate decay so that we can use large learning rates in the early stages of training (helping rapid convergence towards the neighborhood of the loss function minimum), and lower learn-

ing rates in the later stages (helping prevent instability around the minimum). While this results in a better speed/stability trade-off for training, the end results is practically the same as long as the global minimum is reached.

Statements of Contribution

- Elie Climan worked on the Keras model and the report
- Dominik Stolz worked on the data preprocessing, PyTorch models and training on the Google Compute Engine

References

[1] Luoting Fu and Levent Burak Kara. "Neural network-based symbol recognition using a few labeled samples". In: *Computers & Graphics* 35.5 (2011), pp. 955–966.

[2] Max Jaderberg et al. *Spatial Transformer Networks*. 2015. arXiv: `1506.02025 [cs.CV]`.

[3] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[4] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[5] Ange Tato and Roger Nkambou. "Improving adam optimizer". In: (2018).