Abstract

The purpose of our project is to select the best possible models to accurately detect digits from zero to nine and thereby achieve high accuracy rates using the MNIST dataset. We implemented four different CNN models which were LeNet-5, LeNet_Improved, AlexNet and VGG-16. Initially, we had trouble training the VGG-16 model as it required too much memory and was not learning properly as well as the accuracy for AlexNet was fluctuating too much. We set the batch size to 4 and the learning rate to 1×10^{-6} for VGG-16 to work. For AlexNet, we set the learning rate to 1×10^{-4} to stabilise the accuracy graph. Furthermore, we discovered Adam was a much better optimiser than SGD which increased accuracies for all our models by 1 - 2% and reduced runtimes by ~2 minutes. In conclusion, after completing the runtimes of our models, the test accuracies are LeNet-5 with 96.39%, LeNet_improved with 97.59%, AlexNet with 98.68% and finally, VGG-16 with 98.96%.

Project Demo video: https://webdropoff.auckland.ac.nz/cgi-bin/pickup/e034e4e7d9b72cb634fdb83792256c7a/627249

Table of Contents

Introduction

Database

Link to database: http://yann.lecun.com/exdb/mnist/

We will be using a database called MNIST (Modified National Institute of Standards and Technology database) since we are creating a system that recognises handwritten digits. The total size of the database is 10 MB and contains a training set of 60,000 and a test set of 10,000. Each image consists of a 28x28 pixel square coloured in greyscale where it features a picture of the digits from zero to nine.

Initial Findings

After researching numerous papers online, it was astounding to discover how software can precisely detect differing images of a digit from zero to nine. For example, the image below is from a model with a 99.7% accuracy showing it can classify even the worst drawn-out number where most humans cannot make sense of.

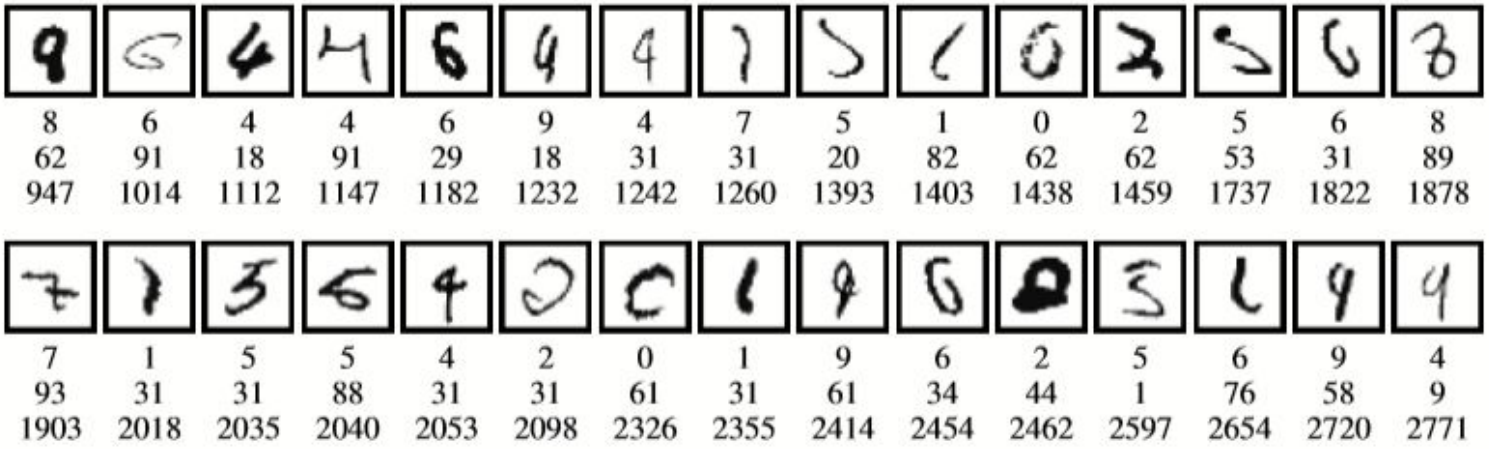


figure 1. Machine classifying digits (Adam Byerly, Tatiana Kalganova 2020)

Algorithm Reasoning

LeNet-5 (1998)

Since LeNet-5 (1998) is one of the simplest and earliest models for CNN, we can study the changes and improvements that have increased the prediction accuracy over time.

LeNet_Improved

We want to study the effects of having the activation function as ReLU and the pooling as max-pooling; this will have on the testing accuracy compared to the original LeNet, which uses Tanh and average pooling.

AlexNet (2012)

We want to study the effects of having more parameters (60k vs 60M) and more layers (five vs eight layers) that will have on the prediction accuracy compared to the previous models.

VGG-16 (2014)

The main difference between VGG-16 (2014) and the other previous models is that this model uses a smaller filter size and is a deeper network overall. We will study how the changes in a much higher parameter (138M), more layers and smaller filter size will change the accuracy of our test.

Reasons for choosing the four models

We decided to not use Inception-v1 due to the complexity of creating the algorithm from scratch and how long it takes to train the model (e.g. VGG-16 took 6 hours). We also decided not to use RNN as CNN models are more suited for our MNIST dataset, thus achieving less accuracy as a result.

Background and literature review

Background research about Deep Neural Network

Deep neural networks (DNN) is a technology that tries to replicate the thinking process of the human brain by adding the inputs through multiple layers to obtain an output. The learning in DNN can be supervised, semi-supervised or unsupervised where many real-world applications use this technology.

Sequence of CNN

Convolutional neural network (CNN) is a type of neural network used for classifying a particular characteristic of an image. The algorithm consists of 5 main layers in the system; including Convolution, activation function, Flattening, Pooling and fully connected layers. The convolution layer takes in the characteristics from the input data by creating smaller pixels through the filtering of the original image. The activation function increases the non-linearity of the picture, where the pooling layer then decreases the size of the image so it can focus on the desired features. Finally, the flattening step lines up the pooled feature map into vertical columns into the input layers of the fully connected layer where it becomes our model for learning and testing.

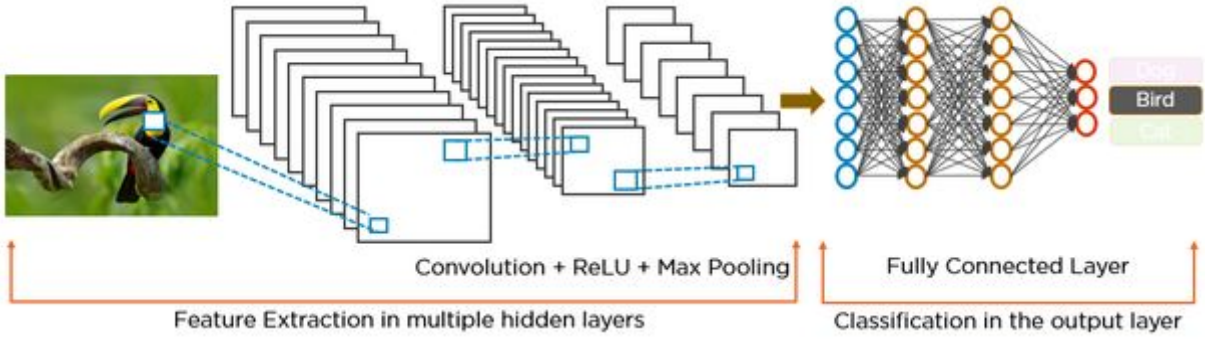


Figure 2. Fully Sequence of CNN (SuperDataScience Team, 2018)

State of art result

link: https://arxiv.org/pdf/2001.09136v3.pdf

The state of art research paper named, "*A Branching and Merging Convolutional Network with Homogeneous Filter Capsules*" published on 24 Jan 2020 is a model currently ranked number one using the MNIST dataset. Their convolutional neural network consists of additional layers to detect specific features with different receptive fields. They transformed each last filter into homogenous vector capsules from each branch. They then compared three different methods of merging with the branches, with equal weights, learned weights and finally with varying weight initialization methods. The combined techniques of randomly initiated augmentation techniques with domain-specific produce a state of art accuracy of 99.84% with an error rate of 0.16%. They used Adam optimizer for all training and experienced no overfitting.

Methodology

We used existing models proven to be the best for MNIST dataset that has achieved 95%+ accuracy. We researched numerous coding templates online; including PyTorch's official tutorial website and a basic model from 'debugger cafe.' Our code is a heavily modified version of the templates found online to suit our purposes for our research.

Link to templates:

1. *https://debuggercafe.com/deep-learning-with-pytorch-image-classification-using-neural-networks/*
2. *https://pytorch.org/docs/stable/nn.html*

Where is the learning ability and why?

In summary, the model obtains its learning ability by using gradient descent to detect the local minimum of a differential equation. In other words, it allows the model to alter its parameters to minimise the output deviation so it can learn and predict more accurately. The loss function optimises the network by calculating the difference between output and expected output. This difference enables the movement of the gradient descent (i.e. up or down the curve), so it can further reduce errors in the prediction. The gradient descent then continuously updates the model through step size (i.e. learning rate) until it finds the minimum.

LeNet-5:

What is LeNet-5?

LeNet-5 consists of 3 fully-connected layers and two convolutional resulting in the digit ‘5’ in its name. It also has 60k parameters, uses average-pooling instead of max-pooling like other later models created and uses Tanh as the activation function.

Features of LeNet-5 that may impact the learning ability:

Average pooling takes the average value of the pixels, so it extracts features smoothly to detect desired characteristics of an image. This process helps the model to learn more about the non-important features of an image compared to max-pooling, where it retains only the most critical characteristic and removes the rest.

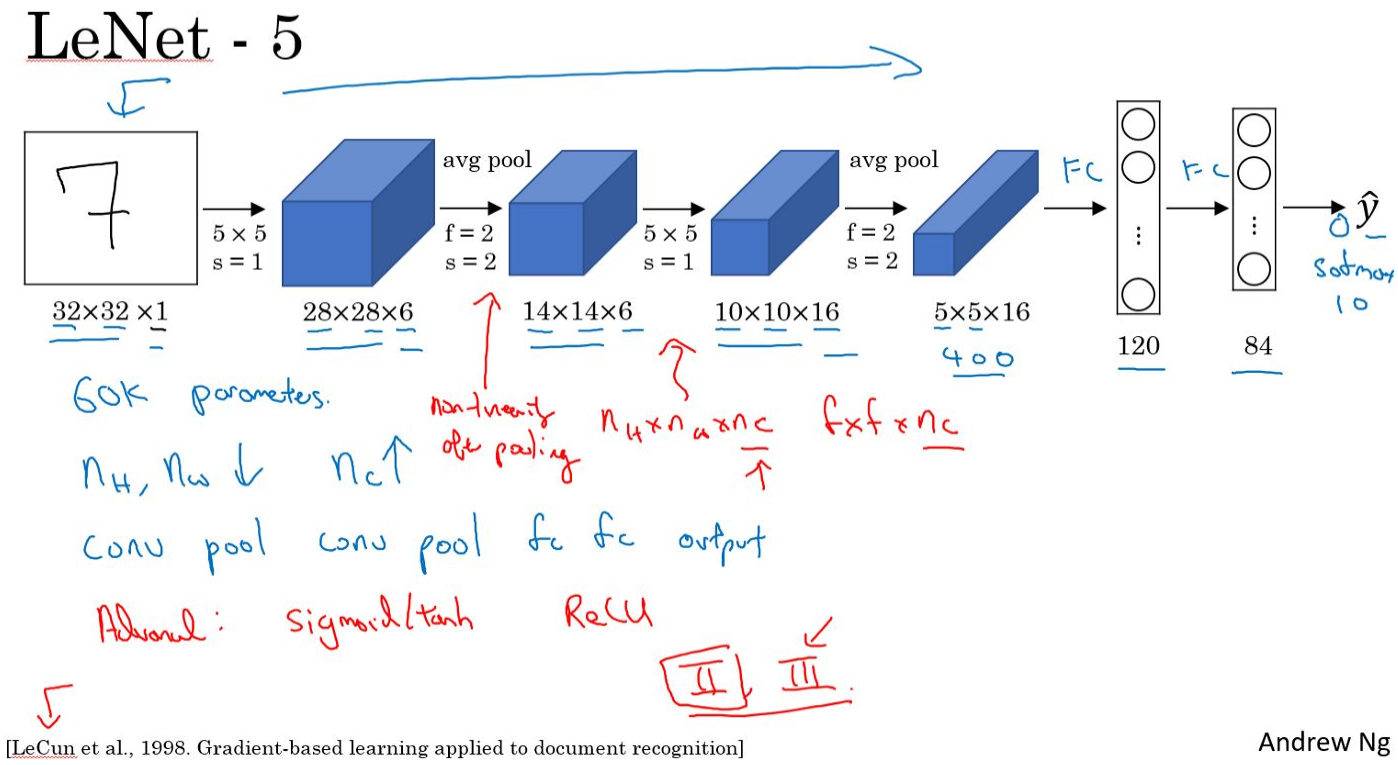


Figure 3. LeNet-5 illustration (Raimi Karim, 2019)

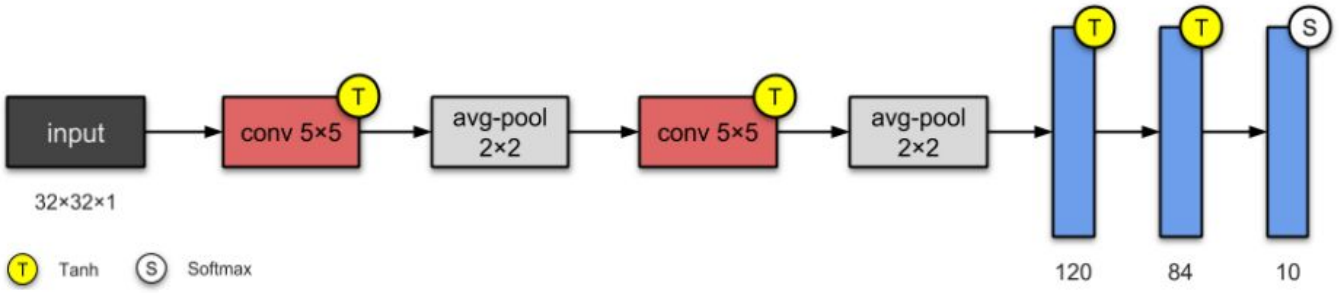


Figure 4. LeNet-5 (1998) (Slide)

- *LeNet-5 uses convolution stacking, pooling layer until it reaches to the end of the sequence with one or more fully-connected layers.*
- *The input data is 32 x 32 x 1*

LeNet_Improved:

What is LeNet_Improved?

LeNet_Improved is a custom made model that uses ReLU as it's activation function and max-pooling compared to LeNet's Tanh and average-pooling.

Features of LeNet_Improved that may impact the learning ability:
Max-pooling is more suited for our dataset as images in MNIST are greyscale. This feature means it extracts the most prominent features (e.g. edges or brightest part of the image) and in turn, increases its learning ability furthermore.

AlexNet:

What is AlexNet?
AlexNet (2012) is very similar to LeNet-5, but the main distinction is it has more layers. It has three fully-connected layers and five convolutional resulting in a total of eight layers. It has 60M parameters compared to 60k of LeNet-5 (1998) and uses max-pooling instead of average-pooling. AlexNet is the first model to implement ReLU as its activation function.

Features of AlexNet that may impact the learning ability:
AlexNet uses ReLU where the gradient is non-saturated, meaning the convergence of the gradient descent is much higher than Tanh. Additionally, it uses Dropout to prevent overfitting from happening. Overfitting causes the model to learn from its noise and thereby reduce it's learning capability.

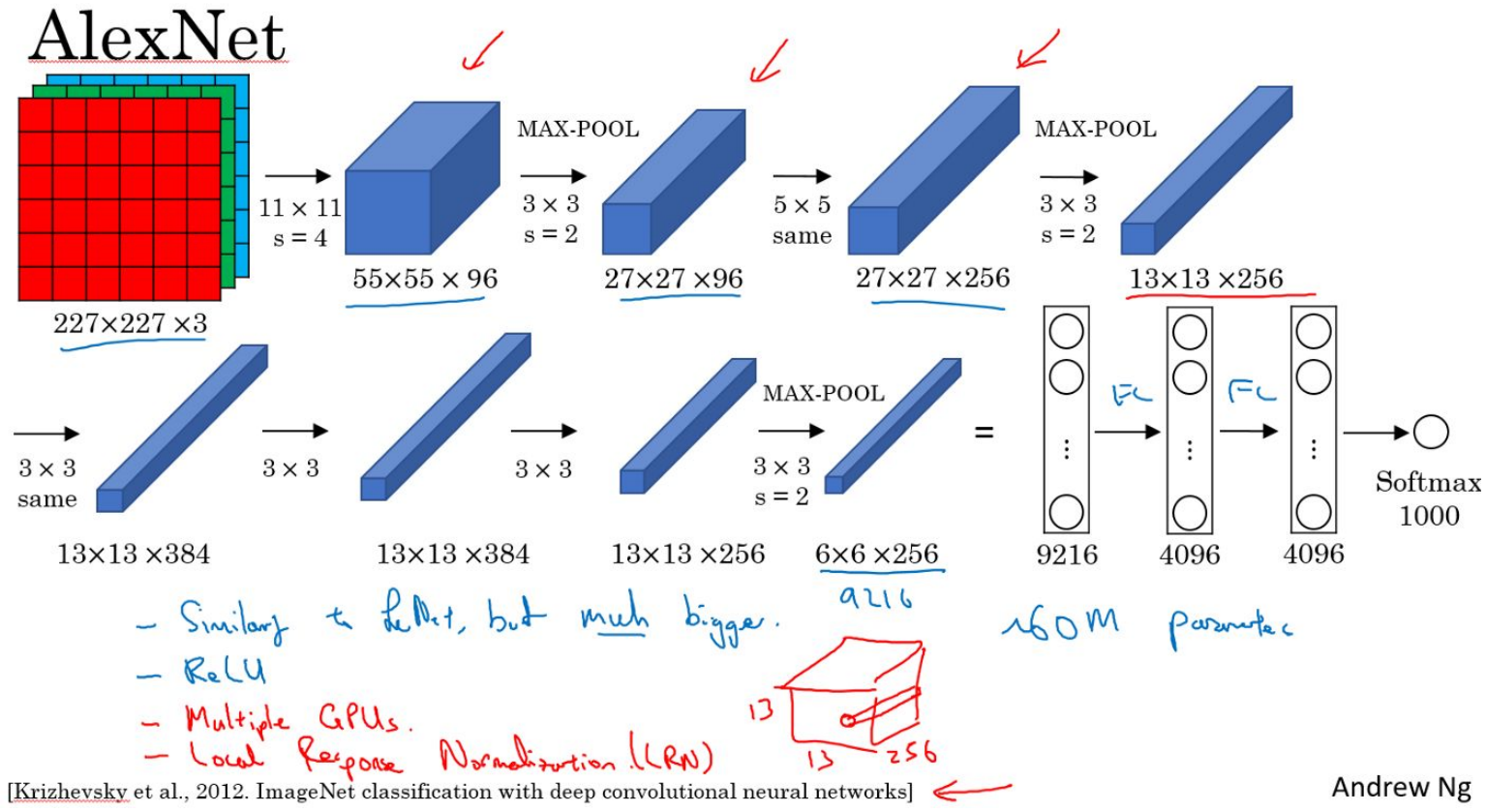


Figure 5. AlexNet illustration (Raimi Karim, 2019)

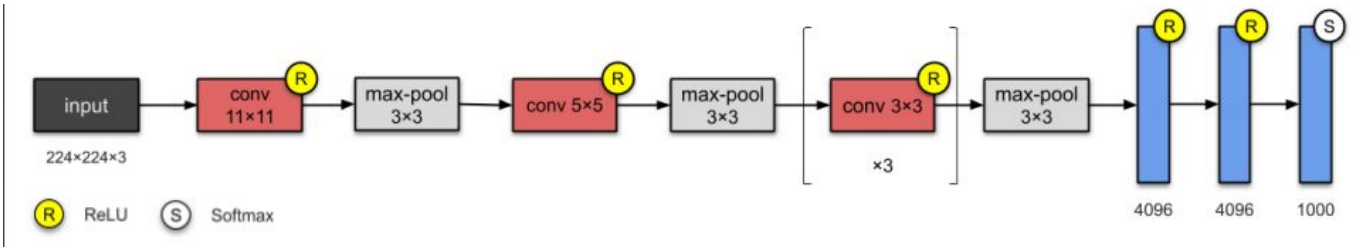


Figure 6. AlexNet (2012) (Slide)

- *AlexNet (2012) is the first model to use ReLU (activation function) in it's model as the training time is much faster)*
- *The input data is 227 x 227 x 3 (original document had an error of 224x224 dimension)*

VGG-16:

What is VGG-16?

VGG-16 (2014) is also very similar to AlexNet (2012) and LeNet-5 (1998). It consists of three fully-connected layers and thirteen convolutional resulting in sixteen total layers. It has 138M parameters and uses max-pooling.

Features of VGG-16 that may impact the learning ability:

VGG-16 has more layers and parameters compared to our other models. One problem that arises from having a deeper network is gradient vanishing. Due to VGG-16 using a gradient-based method of learning, the model will stop learning if the gradient relative to its output and parameters is minimal. In theory, increasing the number of layers increases the accuracy depending on the size and complexity of the dataset. Since our model is very complicated for our small dataset, the model may overfit and reduce it's learning ability.

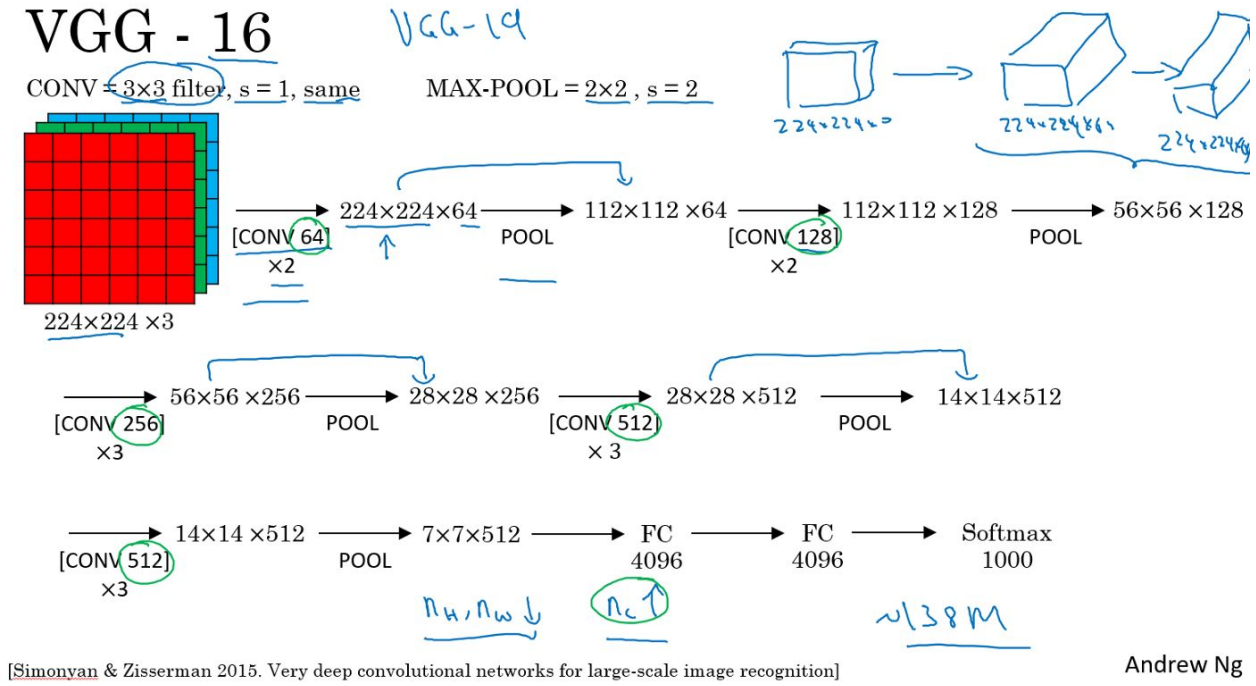


Figure 7. VGG16 illustration Raimi Karim, 2019)

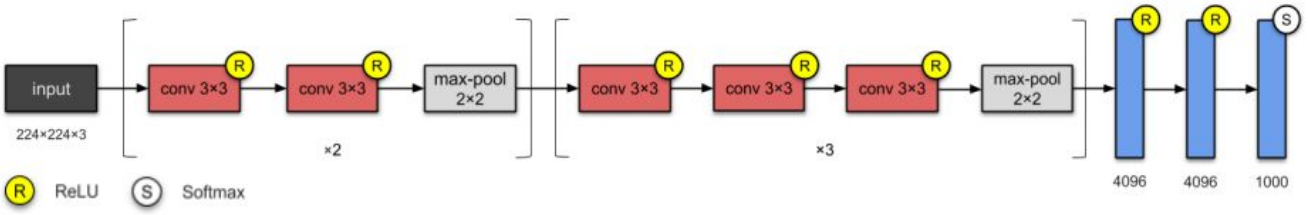


Figure 8. VGG-16 (2014) (Slide)

- *The input data is 224 x 224 x 3*
- *Smaller filter size (2x2,3x3)*

Evaluation

Confusion Matrix Features:

Precision rate: The system's prediction accuracy and classification of the correct number for a given image. I.e. if all nine predictions are identified correctly as a nine or not.

$$\frac{True\ Positives}{True\ Positives + False\ Positives}$$

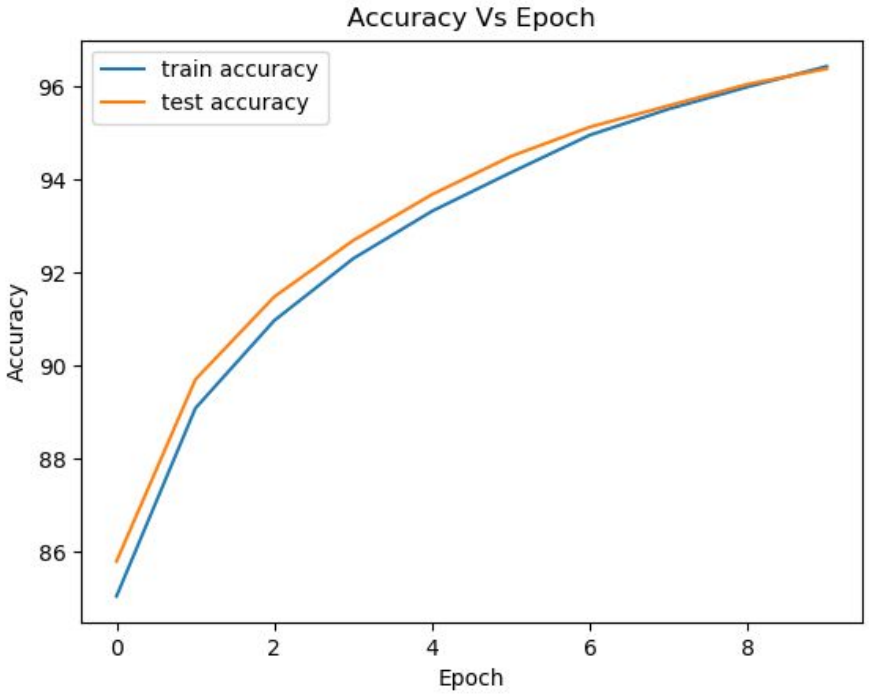
Recall rate: The amount of times a given number is detected while still including the correct and incorrect predictions. I.e. Detected the total number of 9's in all numbers (i.e. correct + incorrect).

$$\frac{True\ Positives}{True\ Positives + False\ Negatives}$$

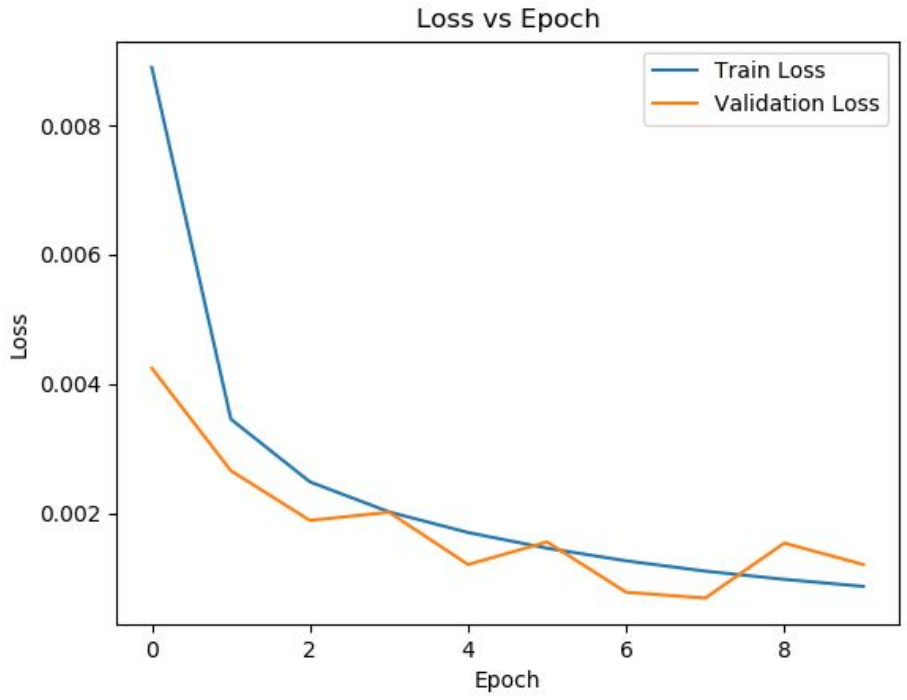
F1-score: The measure of a test's accuracy where it is the harmonic average of precision and recall.

$$2 * \frac{precision * recall}{precision + recall}$$

LeNet-5



Train and test accuracy very similar so can be viewed as a good fit



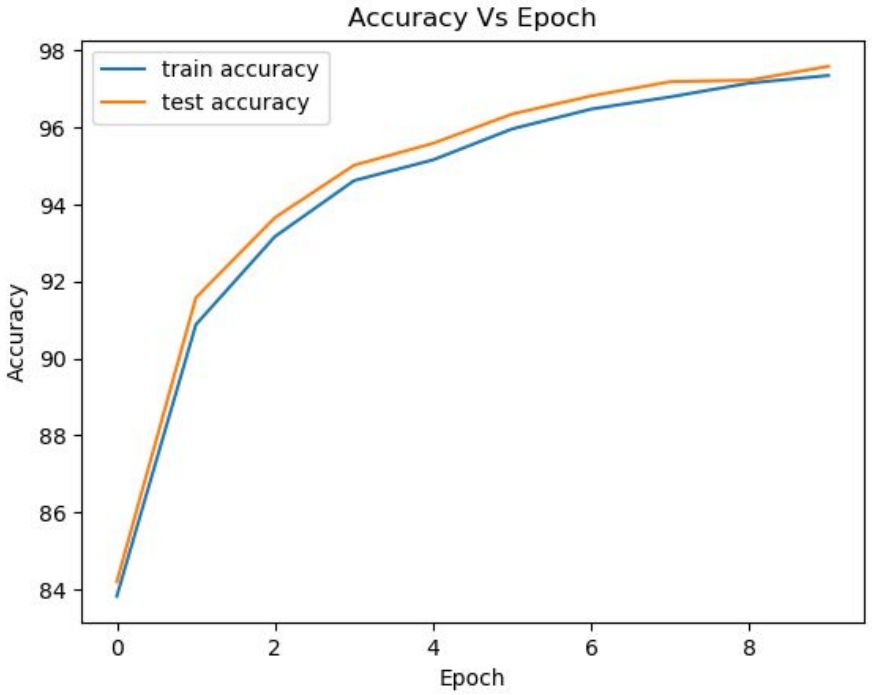
Validation loss a bit higher than Train loss, causing a little bit overfitting but can be viewed as a good fit

```
[[ 970    0    2    0    0    2    3    1    2    0]
 [   0 1123    2    3    0    1    3    0    3    0]
 [   5    1  993    6    3    0    3    9   11    1]
 [   0    1    8  970    0    9    0   10    8    4]
 [   1    1    6    0  952    0    5    2    2   13]
 [   4    1    0   14    2  857    5    1    5    3]
 [   9    3    1    0    5    9  928    0    3    0]
 [   3    8   20    5    3    0    0  980    1    8]
 [   4    1    4   10    4    7    4    7  931    2]
 [   3    9    0   11   28    7    1    8    7  935]]
              precision    recall  f1-score   support

           0      0.971     0.990     0.980       980
           1      0.978     0.989     0.984      1135
           2      0.958     0.962     0.960      1032
           3      0.952     0.960     0.956      1010
           4      0.955     0.969     0.962       982
           5      0.961     0.961     0.961       892
           6      0.975     0.969     0.972       958
           7      0.963     0.953     0.958      1028
           8      0.957     0.956     0.956       974
           9      0.968     0.927     0.947      1009

    accuracy                          0.964     10000
   macro avg      0.964     0.964     0.964     10000
weighted avg      0.964     0.964     0.964     10000

Epoch: 10 of 10, Train Acc: 96.444, Test Acc: 96.390, Loss: 0.131
6.34 minutes

Process finished with exit code 0
```

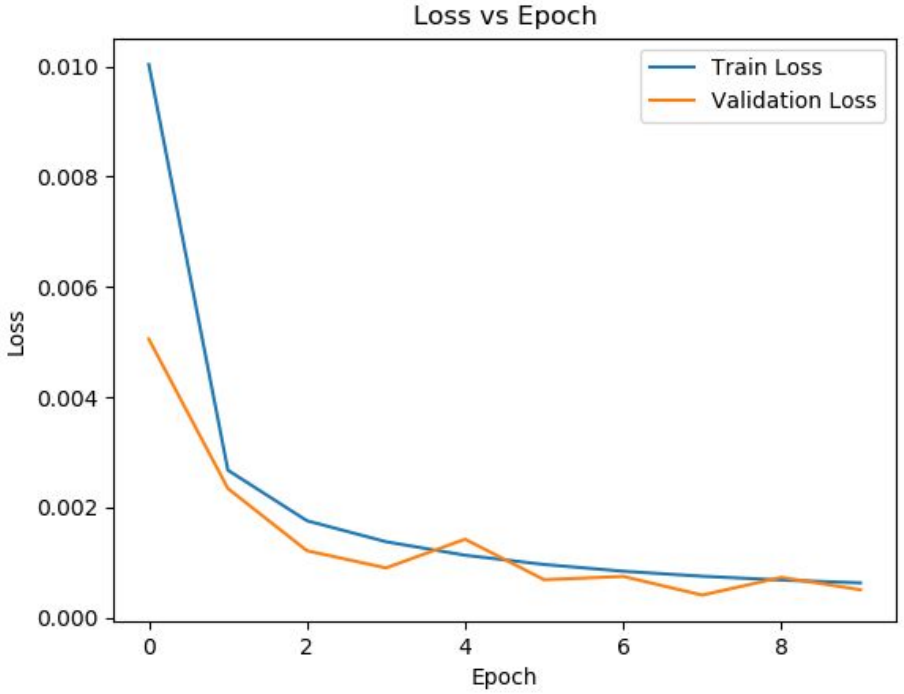
Highest precision, recall and f1-score is 97.8% (digit ‘1’), 99% (digit ‘0’), 98% (digit ‘0’)

LeNet predicts ‘1’ digit 1123 times correctly but predicts ‘5’ digit 857 times incorrectly

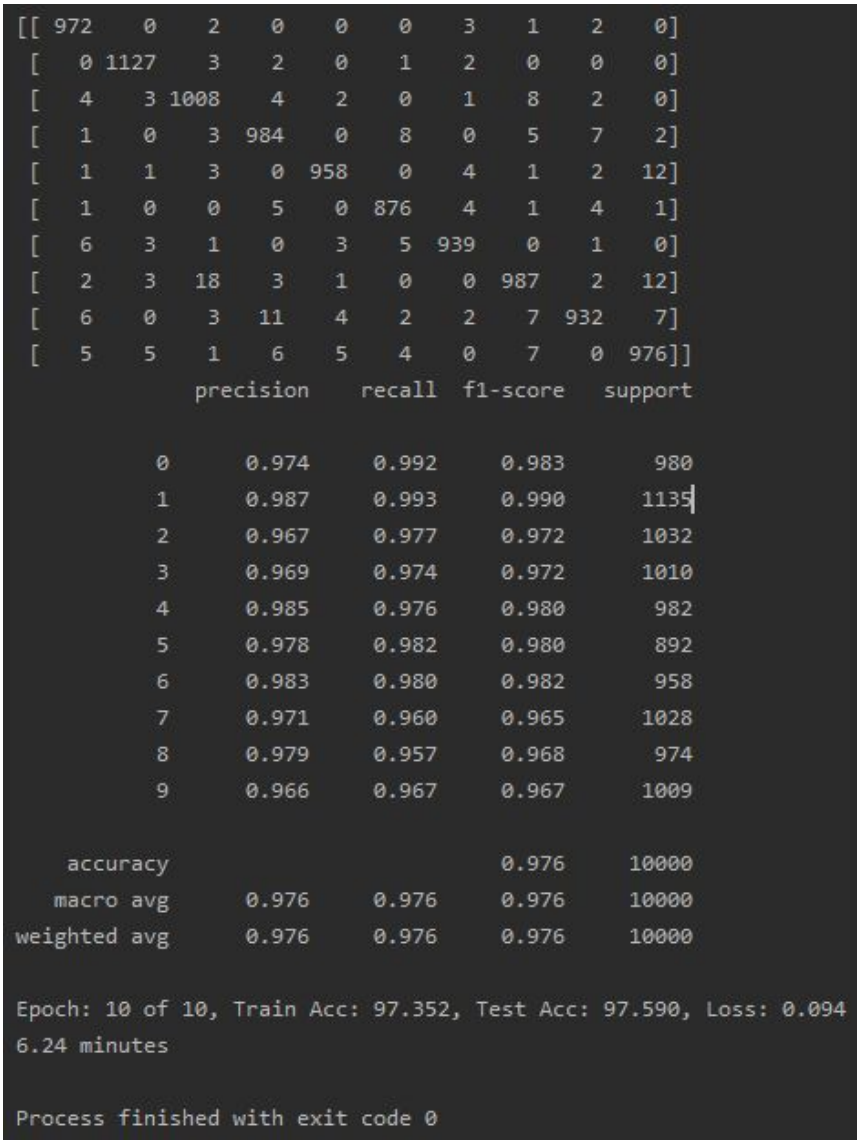
LeNet-Improved



Train and test accuracy very similar so can be viewed as a good fit



Validation loss very similar to Train loss, so can be viewed as a good fit



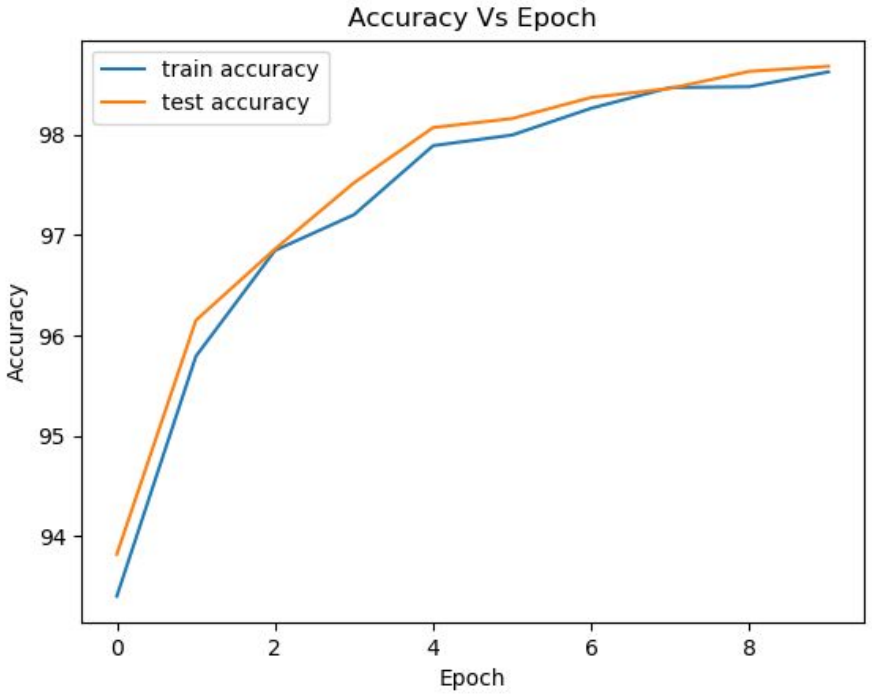
```
[[ 972    0    2    0    0    0    3    1    2    0]
 [   0 1127    3    2    0    1    2    0    0    0]
 [   4    3 1008    4    2    0    1    8    2    0]
 [   1    0    3  984    0    8    0    5    7    2]
 [   1    1    3    0  958    0    4    1    2   12]
 [   1    0    0    5    0  876    4    1    4    1]
 [   6    3    1    0    3    5  939    0    1    0]
 [   2    3   18    3    1    0    0  987    2   12]
 [   6    0    3   11    4    2    2    7  932    7]
 [   5    5    1    6    5    4    0    7    0  976]]
              precision    recall  f1-score   support

           0      0.974     0.992     0.983       980
           1      0.987     0.993     0.990      1135
           2      0.967     0.977     0.972      1032
           3      0.969     0.974     0.972      1010
           4      0.985     0.976     0.980       982
           5      0.978     0.982     0.980       892
           6      0.983     0.980     0.982       958
           7      0.971     0.960     0.965      1028
           8      0.979     0.957     0.968       974
           9      0.966     0.967     0.967      1009

    accuracy                          0.976     10000
   macro avg      0.976     0.976     0.976     10000
weighted avg      0.976     0.976     0.976     10000

Epoch: 10 of 10, Train Acc: 97.352, Test Acc: 97.590, Loss: 0.094
6.24 minutes

Process finished with exit code 0
```

Highest precision, recall and f1-score is 98.7% (digit ‘1’), 99.3% (digit ‘1’), 99% (digit ‘1’)

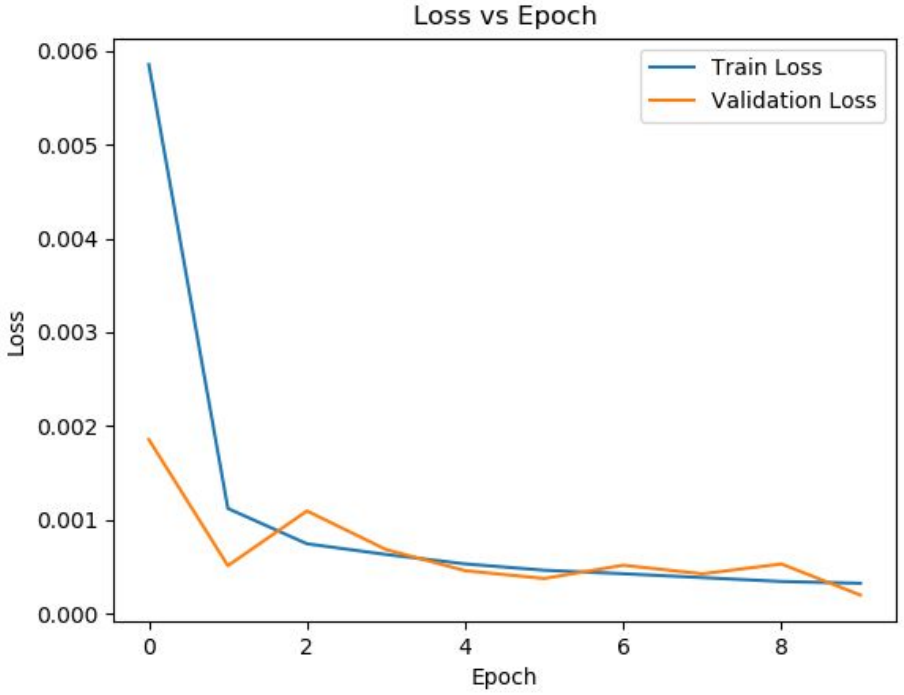
LeNet_Improved predicts ‘1’ 1127 times correctly but predicts ‘5’ 876 times incorrectly

High values for precision, recall and f1-score means model performs very well

AlexNet



Train and test accuracy very similar so can be viewed as a good fit



Validation loss very similar to Train loss, so can be viewed as a good fit

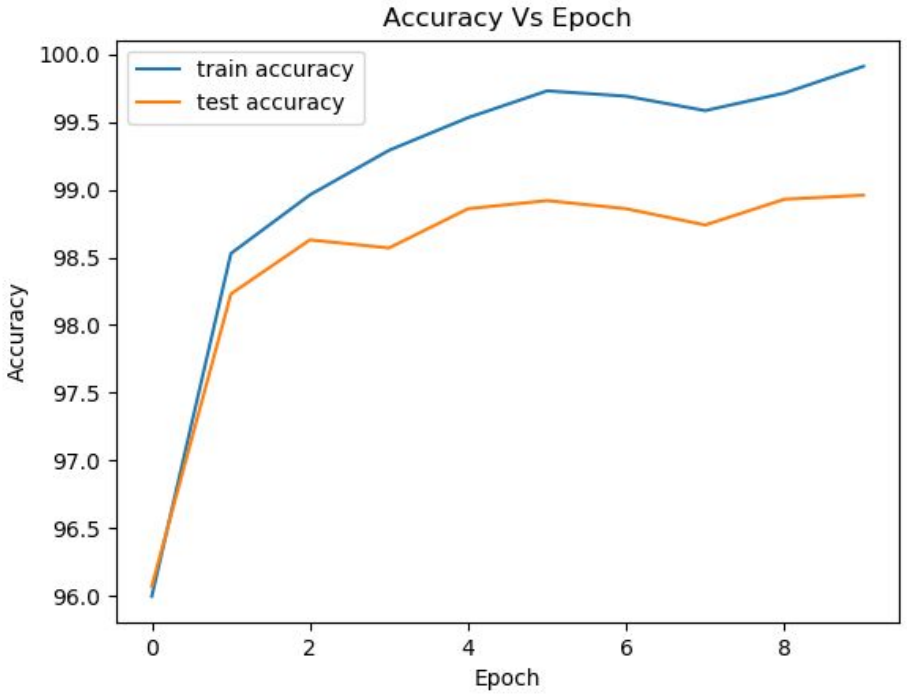
```
[[ 977    0    0    0    0    0    1    1    1    0]
 [   0 1127    3    0    2    0    1    2    0    0]
 [   4    0 1017    0    0    0    0    3    8    0]
 [   2    0    0 1000    0    3    0    3    1    1]
 [   0    0    0    0  966    0    0    0    2   14]
 [   2    0    0    3    0  880    2    1    0    4]
 [   9    2    0    0    2    4  936    0    5    0]
 [   0    1    5    1    0    0    0 1018    1    2]
 [   4    1    3    0    1    0    1    0  957    7]
 [   2    0    1    2    6    1    0    5    2  990]]
              precision    recall  f1-score   support

           0      0.977     0.997     0.987       980
           1      0.996     0.993     0.995      1135
           2      0.988     0.985     0.987      1032
           3      0.994     0.990     0.992      1010
           4      0.989     0.984     0.986       982
           5      0.991     0.987     0.989       892
           6      0.995     0.977     0.986       958
           7      0.985     0.990     0.988      1028
           8      0.980     0.983     0.981       974
           9      0.972     0.981     0.977      1009

    accuracy                          0.987     10000
   macro avg      0.987     0.987     0.987     10000
weighted avg      0.987     0.987     0.987     10000

Epoch: 10 of 10, Train Acc: 98.625, Test Acc: 98.680, Loss: 0.048
29.69 minutes

Process finished with exit code 0
```

Highest precision, recall and f1-score is 99.6% (digit ‘1’), 99.7% (digit ‘0’), 99.5% (digit ‘1’)

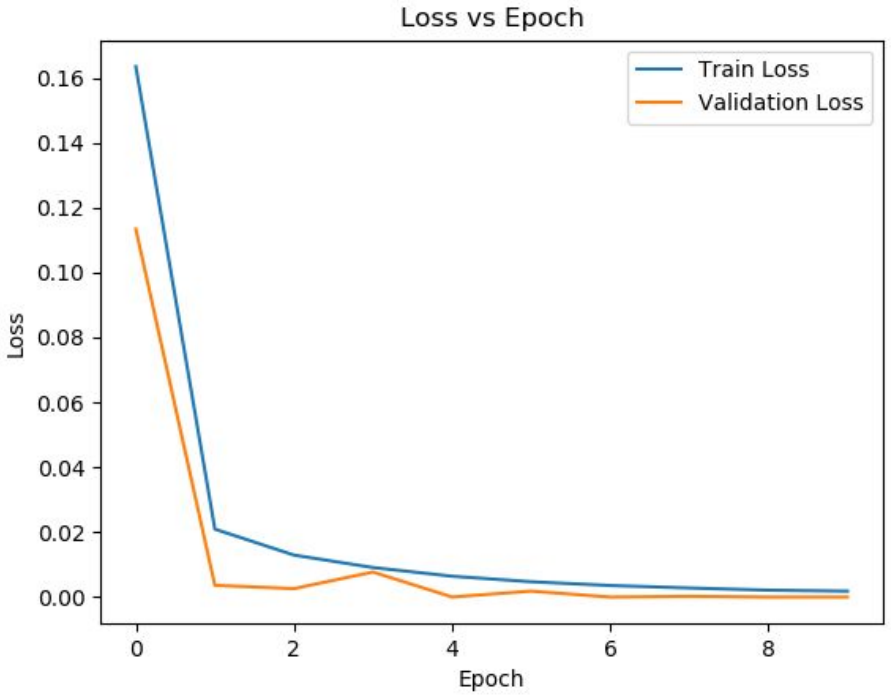
AlexNet predicts ‘1’ 1127 times correctly but predicts ‘5’ 880 times incorrectly

High values for precision, recall and f1-score means model performs very well

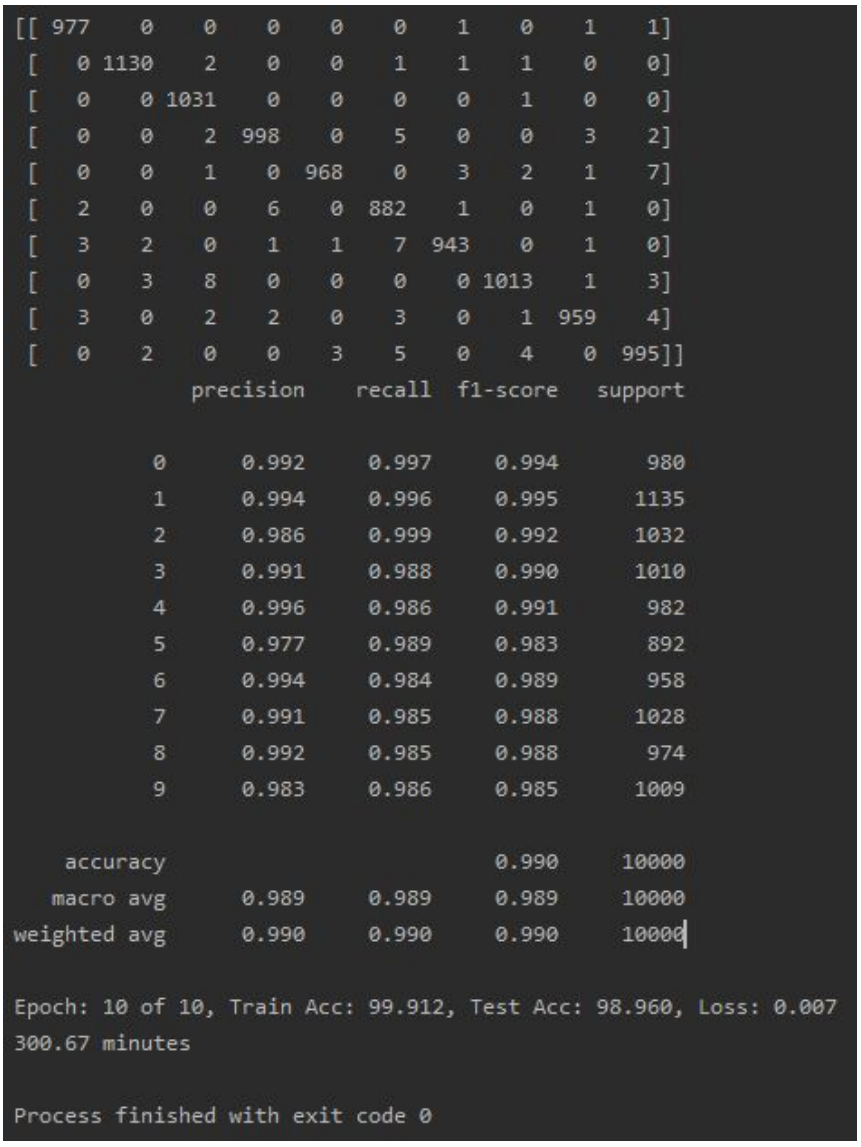
VGG-16



Train accuracy higher than test accuracy which is an expected result



Validation loss very similar to Train loss, so can be viewed as a good fit



```
[[ 977    0    0    0    0    0    1    0    1    1]
 [   0 1130    2    0    0    1    1    1    0    0]
 [   0    0 1031    0    0    0    0    1    0    0]
 [   0    0    2  998    0    5    0    0    3    2]
 [   0    0    1    0  968    0    3    2    1    7]
 [   2    0    0    6    0  882    1    0    1    0]
 [   3    2    0    1    1    7  943    0    1    0]
 [   0    3    8    0    0    0    0 1013    1    3]
 [   3    0    2    2    0    3    0    1  959    4]
 [   0    2    0    0    3    5    0    4    0  995]]
              precision    recall  f1-score   support

           0      0.992     0.997     0.994       980
           1      0.994     0.996     0.995      1135
           2      0.986     0.999     0.992      1032
           3      0.991     0.988     0.990      1010
           4      0.996     0.986     0.991       982
           5      0.977     0.989     0.983       892
           6      0.994     0.984     0.989       958
           7      0.991     0.985     0.988      1028
           8      0.992     0.985     0.988       974
           9      0.983     0.986     0.985      1009

    accuracy                          0.990     10000
   macro avg      0.989     0.989     0.989     10000
weighted avg      0.990     0.990     0.990     10000

Epoch: 10 of 10, Train Acc: 99.912, Test Acc: 98.960, Loss: 0.007
300.67 minutes

Process finished with exit code 0
```

Highest precision, recall and f1-score is 99.6% (digit ‘2’,), 99.9% (digit ‘2’), 99.5% (digit ‘1’)

VGG-16 predicts ‘1’ 1130 times correctly but predicts ‘5’ 882 times incorrectly

High values for precision, recall and f1-score means model performs very well

Result and future work

After completing the runtimes on all four of our models, LeNet-5 came last with a test accuracy of 96.39%, LeNet_improved with accuracy of 97.59%, AlexNet with an accuracy of 98.68% and finally VGG-16 came first with an accuracy of 98.96%. While the results may not match the state of art accuracies found online (e.g kaggle), we tried our best with the limited hardware we have at home and through numerous adjustments of the constants.

Initially, we tested all models with the same constants; optimiser as SGD, loss function as CrossEntropy, batch size of 150, a learning rate of 1×10^{-4} and total epoch of 10.

The first problem we encountered was training our VGG-16. Our hardware did not have enough memory to run VGG-16 due to the large batch size of 150, so we changed the value to a 4 for VGG-16. Another problem was the model did not learn at a learning rate of 1×10^{-4}, so after countless alterations of the number, a learning rate of 1×10^{-6} solved the problem.

The second problem was the optimiser. After researching online, we found that Adam achieved higher precision than SGD. The result was all models increased its accuracy by 1 - 2%, runtime decreased by ~2 minutes. For example, using SGD, LeNet-5 took 2.89 minutes longer (i.e. initially 9.28 min) and a decrease of 2.9% (i.e. initially 93.5%) compared to using Adam.

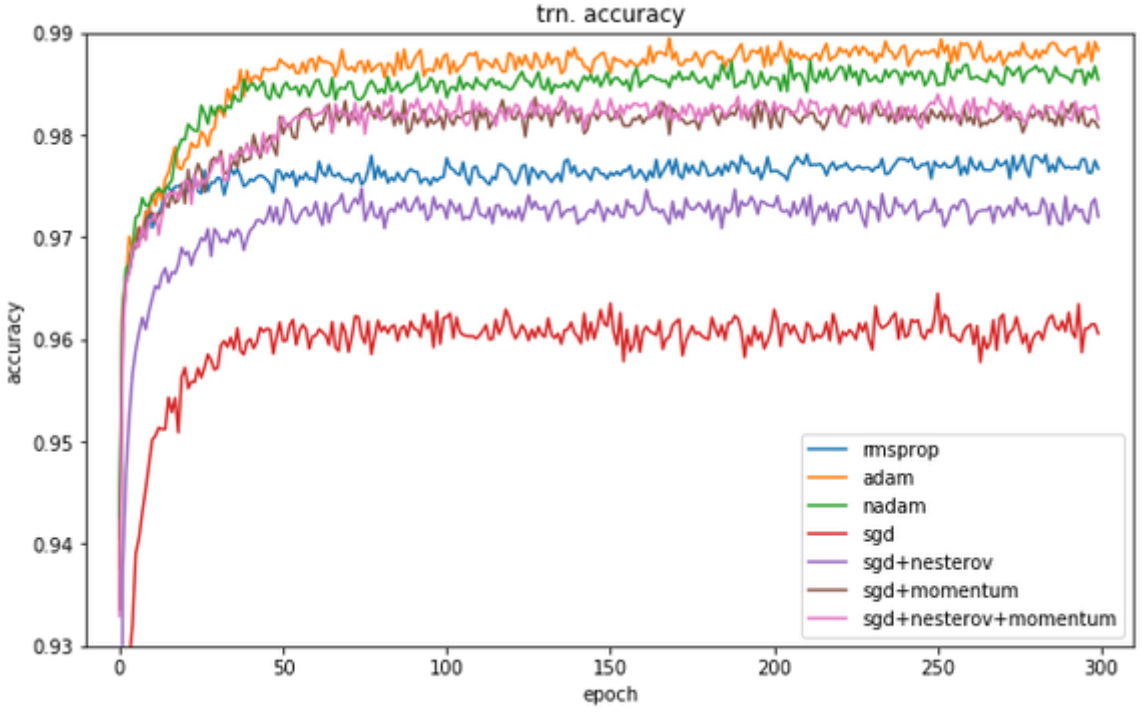


Figure 9. Optimiser comparisons (SALu, 2018)

The third problem was the fluctuation of the accuracy graph for AlexNet, so we changed the learning rate to 1×10^{-4} to stabilise the learning curve.

In the future, we would like to spend more time on the project by having more alterations of our models as well as having better hardware to increase the training time (i.e. more epoch). These changes allow us to

achieve the best possible result using the MNIST dataset we can (i.e. accuracy and loss values stop fluctuating).

In conclusion, our most accurate model was the VGG-16 that achieved a high accuracy of 98.96% and a loss of 0.007. Additionally, we have avoided the problem of overfitting or underfitting our models; thereby obtaining a ‘good fit’ for our algorithms. Lastly, I believe we have achieved passable results (i.e. accuracy greater than 95%) for our models using the MNIST dataset given our limited knowledge and hardware.

References

- *(n.d.). Retrieved from https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-1-convolution-operation*
- *Brownlee, J. (2019, August 19). When to Use MLP, CNN, and RNN Neural Networks. Retrieved from https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/*
- *Comparisons of Deep Learning Algorithms for MNIST in Real-Time Environment. (2018, June 25). Retrieved from http://www.ijfis.org/journal/view.html?uid=824&pn=lastest&vmd=Full*
- *MNIST database. (2020, March 30). Retrieved from https://en.wikipedia.org/wiki/MNIST_database*
- *Pareto principle. (2020, April 8). Retrieved from https://en.wikipedia.org/wiki/Pareto_principle*
- *Shah, T. (2017, December 10). About Train, Validation and Test Sets in Machine Learning. Retrieved from https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7*
- *TensorFlow - CNN And RNN Difference. (n.d.). Retrieved from https://www.tutorialspoint.com/tensorflow/tensorflow_cnn_and_rnn_difference.htm*
- *Training and Test Sets: Playground Exercise. (n.d.). Retrieved from https://developers.google.com/machine-learning/crash-course/training-and-test-sets/playground-exercise*
- *Yal, O. G. (2020, March 22). Image Classification in 10 Minutes with MNIST Dataset. Retrieved from https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d*
- *Karim, R. (2019, October 17). Illustrated: 10 CNN Architectures. Retrieved from https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d*
- *Hosays, K., Hosays, K., Ahernesays, I., McCloudsays, T., & ResNet. (2017, December 29). SGD > Adam?? Which One Is The Best Optimizer: Dogs-VS-Cats Toy Experiment. Retrieved from https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/*
- *Papers with Code - A Branching and Merging Convolutional Network with Homogeneous Filter Capsules. (n.d.). Retrieved from https://paperswithcode.com/paper/a-branching-and-merging-convolutional-network*
- *Koehrsen, W. (2018, March 10). Beyond Accuracy: Precision and Recall. Retrieved from https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c*
- *Rath, R. R. S. R. (2019, December 7). Deep Learning with PyTorch: Image Classification using Neural Networks. Retrieved from https://debuggercafe.com/deep-learning-with-pytorch-image-classification-using-neural-networks/*
- *torch.nn¶. (n.d.). Retrieved from https://pytorch.org/docs/stable/nn.html*