Task 1. Data Visusalization

Principal Component Analysis (PCA) has been used to create a 2D representation of the dataset (Figure 1) by selecting the 2 components that explain the most variability in the data. This is useful for learning about the pixel clustering in the dataset. It enables 2D representations of high dimensional data, allowing an understanding of the clustering patterns inherent to the data.

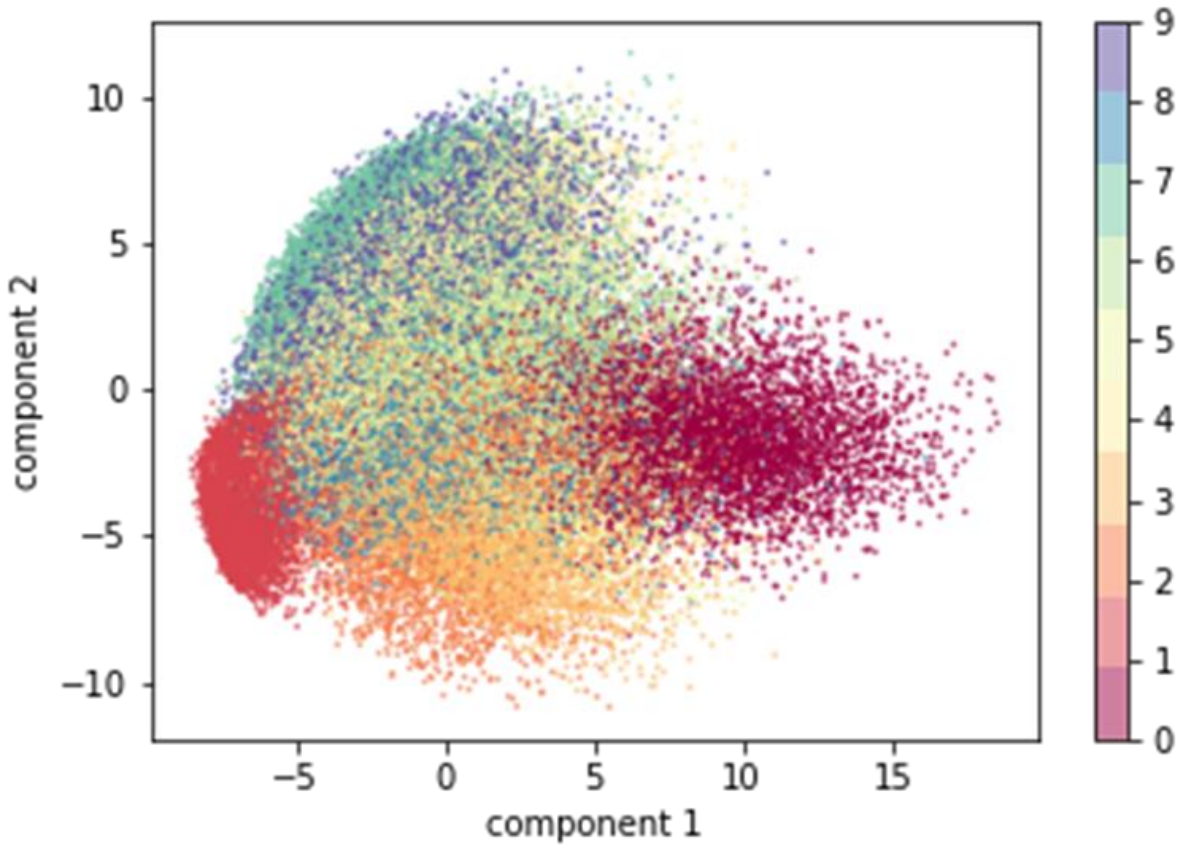

Figure 1. PCA visualisation of training data. The 2 principal components are represented in this 2D space. Components represent 9.70% and 7.10% of the total variance respectively.

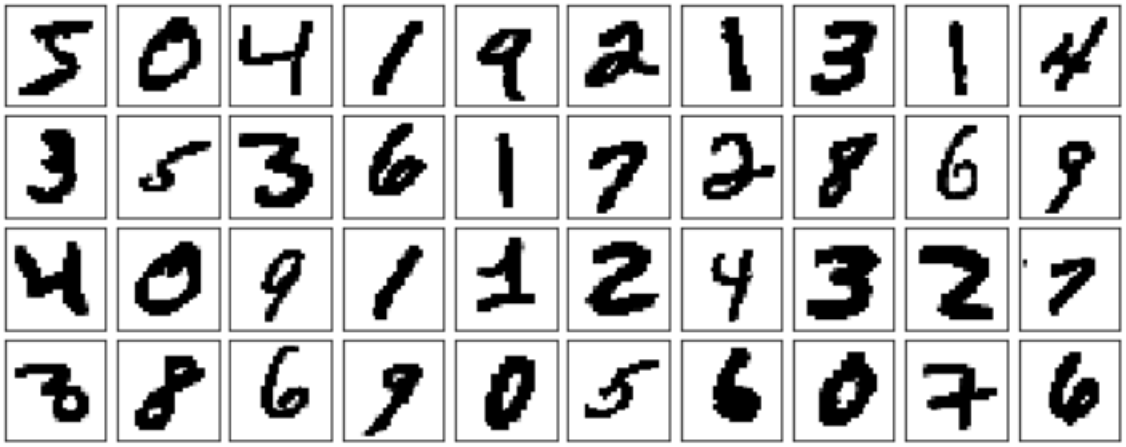

Figure 2. Sample of training data as 28x28 pixel images.

There are some clear clusters shown in Figure 1, particularly in the case of the digits 0 and 1. Tight clustering shows that the representations of these digits have low variability, i.e are consistent in the

way they are drawn. The points for the 1 digit are the most tightly clustered, which makes sense as it is the most simple digit to draw as it is made up of a simple vertical line. Because it is such a simple digit and takes up less space compared to all other digits, the images of the 1 have more similarity in the pixel values, hence less variability. The 0 digit is well clustered but covers a slightly larger area of pixels than the 1, which means there is more variability. The 0 digit is in a well-defined zone on the edge of the overall cluster, because it is quite different to other digits in that it is entirely curved, whereas the others tend to be a mixture of curved and straight lines. The 1 on the other hand is completely straight and there would be little overlap if you overlay a 1 and a 0 because of their shapes, which is why the 1 is clustered at the other side 2D space. This figure can thus help to form an immediate idea of where the challenges will lie in creating a machine learning solution to recognising digits. 1s and 0s are linearly separable, so even the most basic algorithm should not mistake one for the other, but where digit clusters are less well defined and there is more overlap one can expect to encounter misclassifications. This is true for the 9, 8, 6 and 5 digits. None of these digits seem to have a clearly defined cluster and they all overlap significantly with little separation, due to their similarities in shape. All these digits generally involve curves towards the centre, with lots of overlap between them.

Task 2. Perceptrons

A single perceptron model has been created to perform a binary classification of the MNIST dataset. This model learns a single layer of weights, with 1 weight for each pixel via its training algorithm. Once the training is complete the model can be used to make predictions by performing an element-wise product of the weights and inputs, adding on a learned bias value, and passing this through a nonlinear activation function. The output from the activation is then classified. For training, weights are adjusted over 1000 iterations. Within each iteration, a random single training data sample is chosen, and the current weights are used to predict its output. If the prediction is wrong, all the weights are updated using this formula:

$$W(t+1) = W(t) + \eta(y_n - y_{pred})x_n$$

This has the effect of increasing weights for which the input pixels have a high value and decreasing the weights for which pixel values are low in the input sample. As the input data was classified using the labels 1 and -1 for the digits 1 and 0 respectively, this means the weights converge to positive values where the pixels exhibit more activity for the 1 digit, and negative values where the pixels are mainly dominated by the 0 digit. This is clearly represented in the visualisation of the learnt weights (Figure 3), which shows a clear blue circle with a hole in the centre of the image. These blue values show where the pixels in which the 0 digits in the training data have been drawn. It is reassuring to be able to see the clarity of the 0 shape. The weights for 1, shown in red can be seen in the middle of the circle, since the shape of a 1 is a vertical line going through the middle.

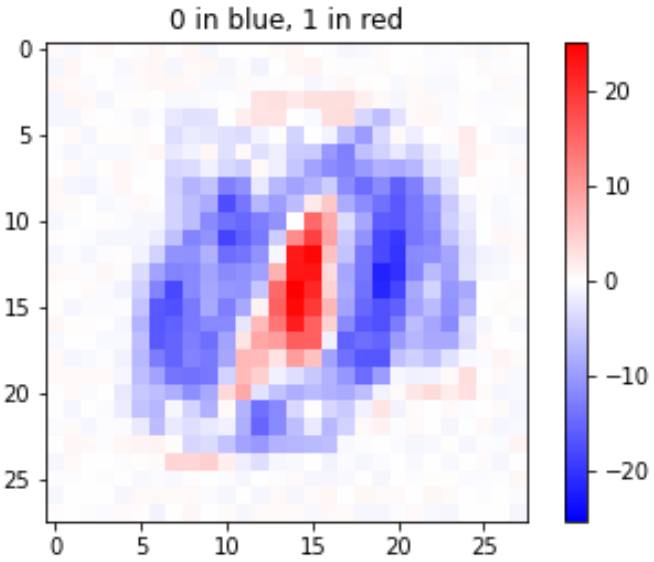

Figure 3. Learnt weights for [0,1] classification. The values are the pixel weight + bias.

This classifier produced an accuracy of 0.9991 on the test set, showing that the weights soundly represent the differences between pixel values for the 0 and 1, to the extent whereby approximately only 1 in 1000 images are misclassified, based on the test dataset of 10,000 images.

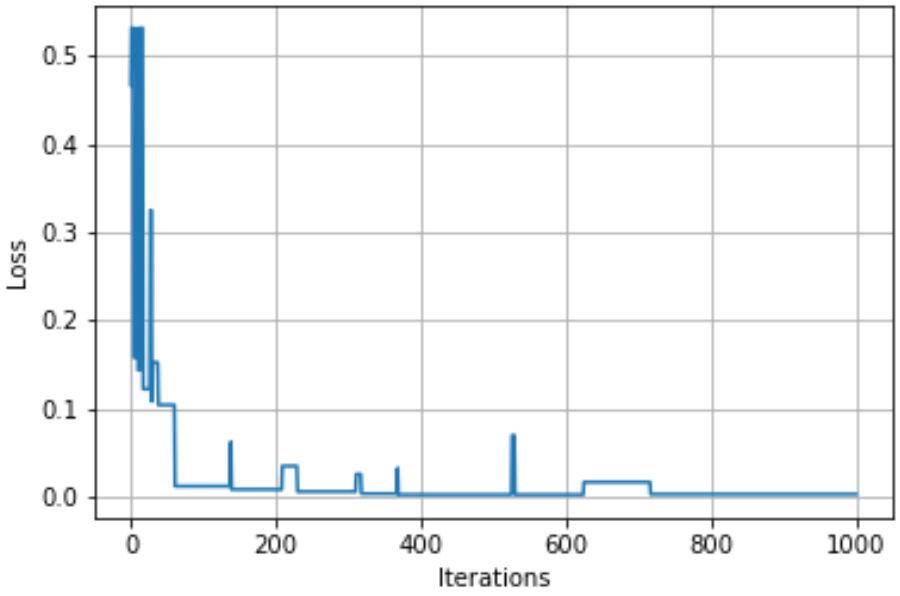

Figure 4. Training error curve for binary [0,1] classifier.

Figure 4 shows the model's error (the inverse of accuracy) over the course of 1000 iterations of training. Within the first ~25 iterations the error jumps up and down constantly, since very few samples have been used for training at this point, so the weights are fluctuating a lot and are biased to the few examples they have seen. In addition, the amount of change to the weights is affected by the learning rate. The learning rate uses a step decay schedule, whereby it starts off at a value of 1,

and then decreases by 0.1 every 100 iterations (*Setting the learning rate of your neural network.*, 2018). This means when a sample is misclassified early in the process, the weights are adjusted much more than they would be for a misclassification towards the end of the training process. This is used so that at the beginning of training, when loss is high, the weights will move much faster, whereas towards the end when the model is approaching the local minima, the learning rate decreases to ensure there are no big changes which could cause high error. After the initial fluctuations, at around 50 iterations the error goes down to ~0.2, and then there is a gradual decline in error from this point on, with some brief smaller fluctuations, such as the spike at just over 500 iterations up to an error of ~0.8. This is likely to be an outlier sample that has caused the weights to be adjusted in a way that is counter productive.

This [0,1] classifier is the best performing out of the 5 combinations tested, which aligns with the PCA analysis which suggested the 0 and 1 were most easily separable. The model performs well across the board though, performing better on the unseen data for 3 out of 5 models. The high test accuracies show that the model certainly is not overfitting. The model has the most difficulty classifying between 5s and 6s. These digits were expected to be difficult as they are very similar in shape, and the visualisation of weights is very ambiguous (Figure 5). These are also 2 of the digits which do not seem to have any discernible clusters in Figure 1.

Table 1. Accuracies for binary classifiers trained on 6 different pairs of digits.

Binary classifier	Train accuracy	Test accuracy
0,1	0.9977	0.9991
1,2	0.9797	0.9811
3,4	0.9900	0.9895
5,6	0.9683	0.9714
7,8	0.9823	0.9780
9,0	0.9877	0.9854

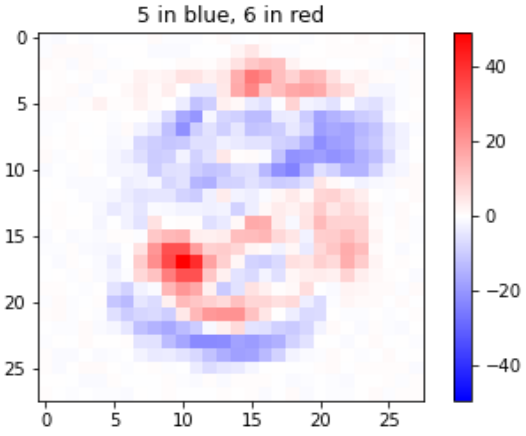

Figure 5. Weights visualised for [5,6] binary classifier.

Task 3: Multi-Layer Perceptron

A multi-layer perceptron is used to classify the digits 0-9. It requires a much deeper training process with orders of magnitude greater parameters when compared to the single perceptron. It uses 2 hidden layers, each with 1000 fully connected neurons. Tensorflow was used to create the MLP model, with the Rectified Linear Unit (ReLU) activation function used for the dense layers, and a softmax used on the final layer of 10 to obtain probabilities for classifying the input as each digit 0-9. A large batch size of 100 is used, as this decrease computation time and gives a smoother

convergence, as can be seen in Figure 6. The results for the final chosen MLP model is shown in Figure 7.

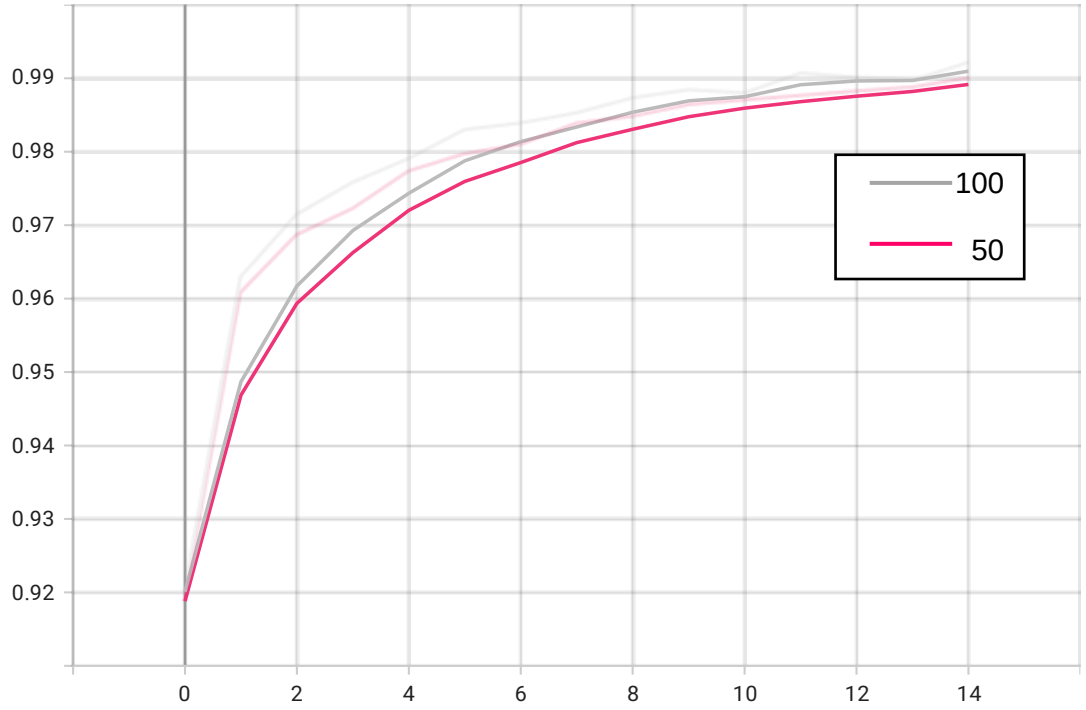

Figure 6: Accuracy curve for training data using batch sizes of 100 and 50, across the 15 training epochs. Curves are smoothed using an exponential moving average, at a rate of 0.5. The faint curves show the true values.

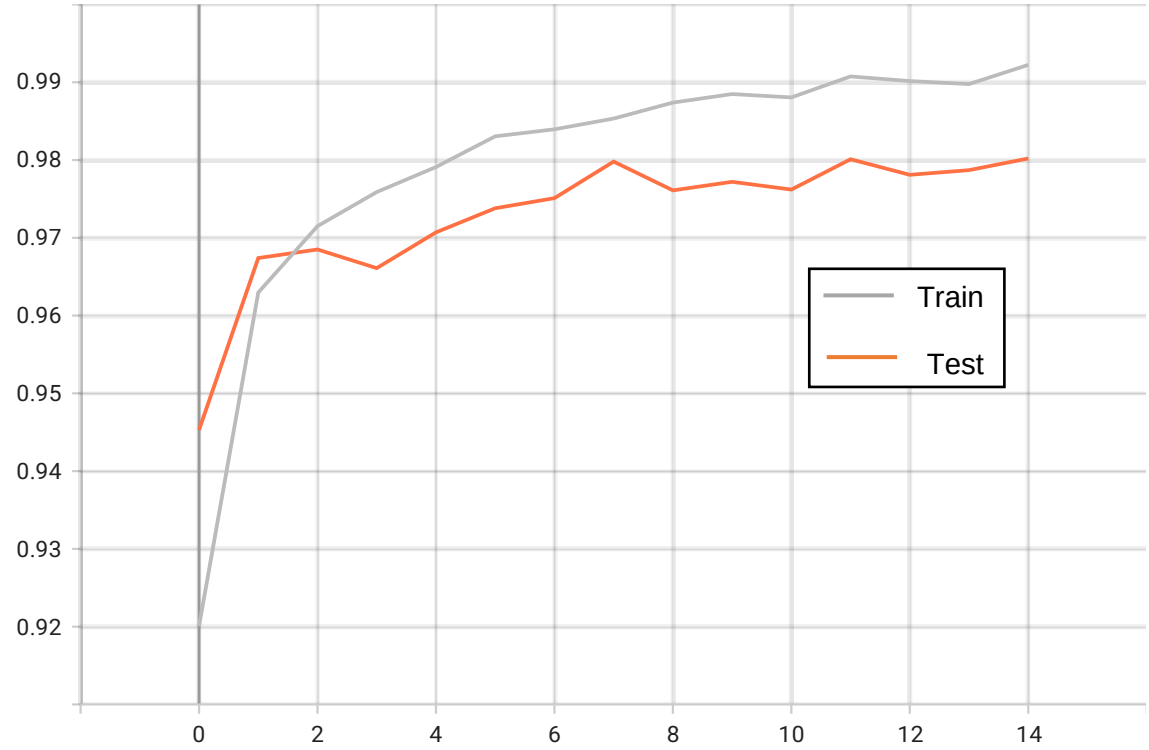

Figure 7. Accuracy curve for the final chosen model trained over 15 epochs (zero indexing). MLP with 2 hidden layers of 1000 neurons each. Batch size = 100.

The model shows a classic convergence, with training accuracy following a logarithmic pattern, improving rapidly over the first few epochs and then more slowly as it reaches its maximum of 0.9922 (Table 2). Interestingly, the accuracy on the unseen data is higher for the first two epochs, but then the training data sits firmly above the test accuracy. Perhaps there is just slightly more variability in the training data compared to the test data, which means that the performance on the test data is higher to start with. The gain in accuracy from the beginning of training to the end is more than double for the training data, which follows, as the model is actively fitting to this data. Importantly the model is not overfitting, exemplified by the high test accuracy, which is only 1 percentage point lower than the train accuracy. 15 epochs seems to be a good balance between generating a high performance model without overtraining and extending the it takes to train the model too much.

Table 2. Classification accuracy values for 6 MLP models of varying complexity. Width describes the number of neurons in each of the hidden layers.

Layers	Width	Parameters	Time	Train Accuracy	Test Accuracy
2	1000	1,796,010	2m 47s	0.9922	0.9802
3	1000	2,797,010	4m 44s	0.99	0.9802
4	500	1,149,010	2m 22s	0.9885	0.9782
6	500	1,650,010	3m 35s	0.9873	0.9776
8	250	638,010	1m 15s	0.987	0.9755

4 more MLP models were trained with varying numbers of hidden layers and widths. Their results are detailed in Table 2. The layers are all made up of the same number of neurons, which is given in the width column. These models fail to clearly improve upon the base 2 layer model. The model with an extra layer of 1000 neurons achieves the exact same test accuracy, to 4s.f, showing that this is needless added complexity.

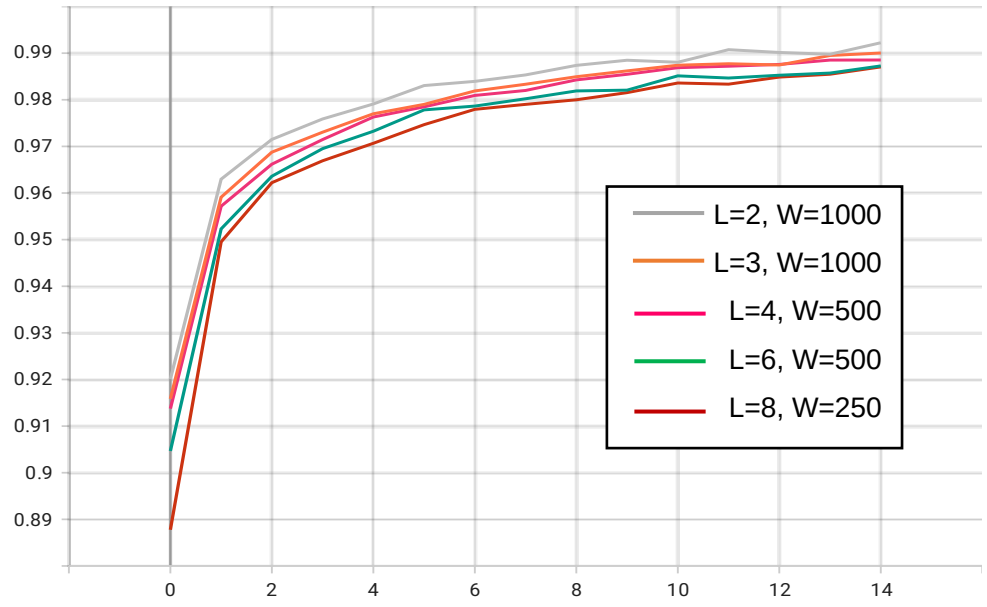

Figure 8. Training accuracy curves for all MLP models.

Having said this, the accuracies of all of these models are within 0.02 of eachother, that is to say, they all perform well. Figure 8 shows that the original model with 2 hidden layers performs the best across all epochs, and that each model with an added layer performs slightly worse than the one before it. This suggests that having fewer layers with a high number of neurons is the way to achieve the best performance. Despite the model with 6 layers of 500 neurons having a similar number of parameters to the original model, it does perform worse, as well as taking longer to train, due to the increased computational demands of doing backpropagation with more layers.

Task 4: Convolutional Neural Network

CNNs are the state-of-the-art models for computer vision, due to the 'human-like' way they recognise images using the spatial relationships between pixels, building up from small level structures like lines and curves into bigger shapes and structures to allow it to label images. The strength of this spatial approach is that CNNs are robust to changes in scaling, positioning, rotation, and weight. In contrast, the MLP relies on digits being drawn in very similar ways so that the size, position and rotation are not too different. It relies on the lighting up of specific pixel locations, rather than the identification of characteristic shapes within the entire image.

The CNN model trained here has 3 layers, with 32, 64, and 128 kernels for each respective layer. This architecture means that the number of features that the network is 'trying' to recognise doubles for each successive layer, starting off with a small number of kernels, and putting these together in different combinations to understand the bigger diverging structures inherent to the data, which is necessary for classification. The model achieves a very high training accuracy of 0.9974. This is a significant improvement over the MLP model of 0.9922. The model generalises very well for unseen data, achieving an accuracy of 0.9902 (or 99.02%) for the test data. This is a net classification improvement of 1% over the MLP model which achieved 98.02%, so it is a similar proportional improvement as the training accuracy improvement (because the training accuracy was already higher for the MLP model). It is significant as it approximately halves the number of real misclassifications that would be made if the model was used to classify new data. This is hugely significant when thinking about classifying large datasets.

The accuracy curve shows the standard logarithmic increase for the training data, but there is a lack of change in the test data over the 10 epochs (increase of 0.005). A reduced 10 epochs were used for the CNNs due to the increased computation time (this model took over 8 minutes to train). The returns are diminishing for additional epochs, so 10 seems to be a good compromise.

A number of difference CNNs were trained, with the results shown in Table 3. These have 2, 3, 4, and 5 convolutional layers each with the aim of exploring what the best level of complexity is for this CNN architecture and the data. Figure 10 shows the training curves for all 5 models together, and it suggests that there is no benefit to adding more than 3 convolutional layers to this architecture, and it is actually a hindrance in some cases. In fact, the most simple model, with only 2 convolution layers with 32 and 64 kernels, respectively, has the best training curve. It does not perform as well on the

test data, but this should not be considered when thinking about choosing a model. Ultimately, more testing using validation sets would be necessary to evaluate whether this 2 model or the original 3 layer model is optimal. Clearly the worst model is the 3 layer model with reversed kernel sizes, shown in orange. This has the lowest training accuracy and takes much longer to train than the other models, at over 20 minutes. Due to the stride of 2 used in training all models, the feature maps shrink at each layer, thus the earlier layers take longer to compute, so having the first layer use the most features leads to this significant increase in computation. In addition, this structure of descreasing the number of kernels throughout the convolutional layers is contrary to the intuition of the way CNNs abstract patterns in the data, starting off with simple patterns like edges, corners and combining them into much more complex patterns (which require more combinations of the simple patterns i.e filters).

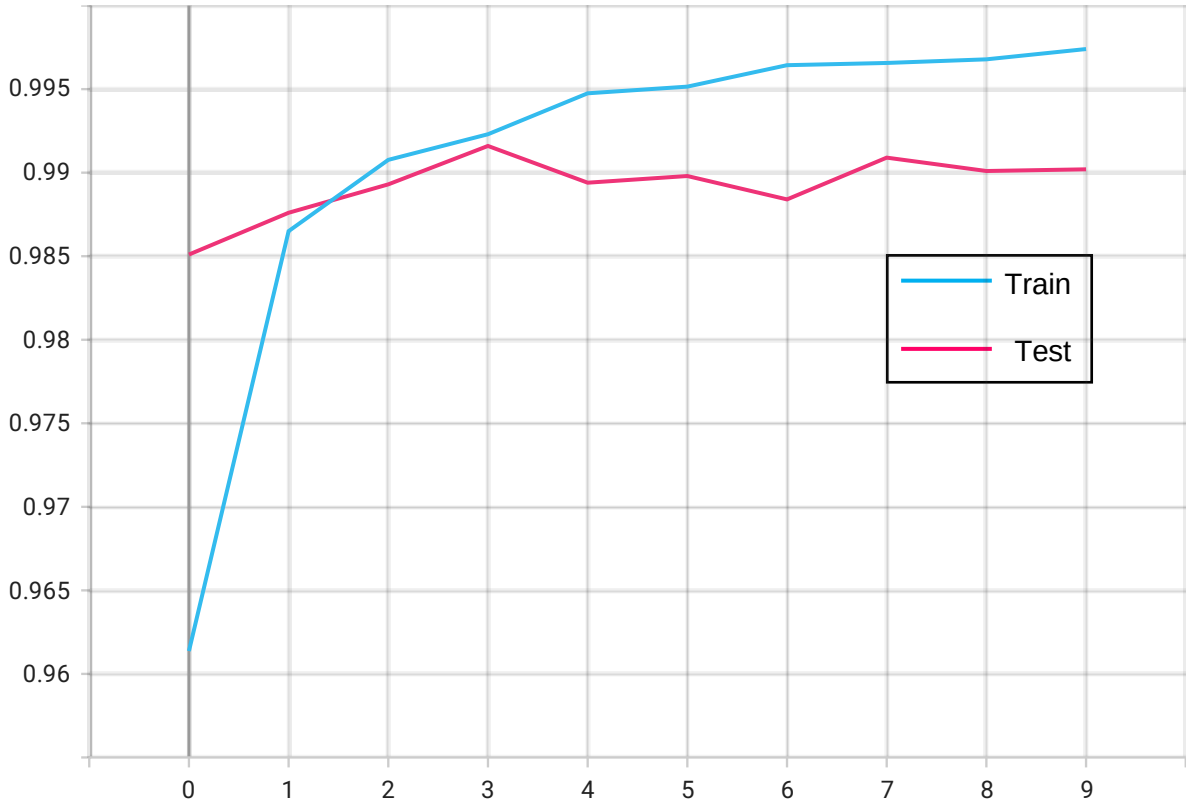

Figure 9. Classification accuracy for 3 layer CNN with 32,64 and 128 kernels respectively.

Table 3. Classification accuracies for CNN models.

Layers	Kernels	Parameters	Time	Train Accuracy	Test Accuracy
3	[32,64,128]	185,066	8m 23s	0.9974	0.9902
2	[32,64]	110,826	6m 16s	0.9981	0.9885
3	[128,64,32]	171,242	20m 32s	0.9963	0.9861
4	[16,32,64,128]	173,818	4m 21s	0.9973	0.9909
5	[16,32,64,128,256]*	699,642	12m 36s	0.9966	0.9913

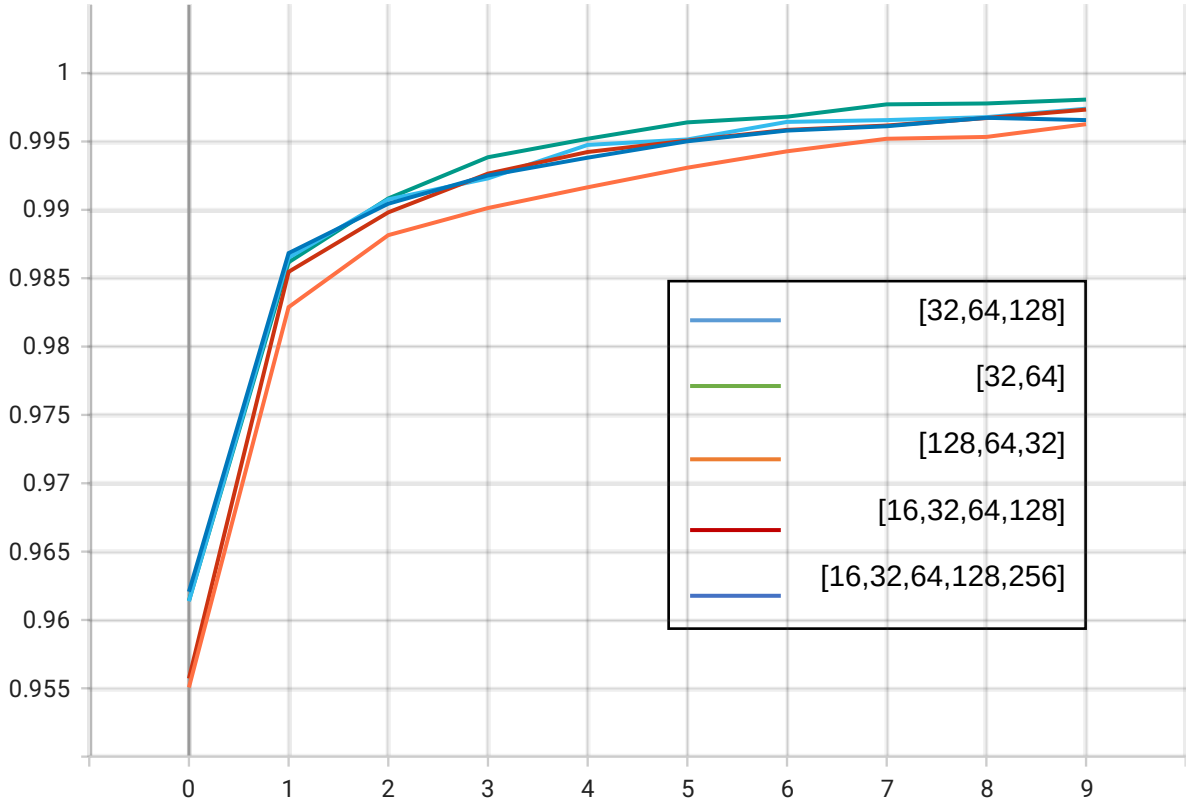

Figure 10. Training accuracy for CNN models.

Task 5: Visualising CNN outcomes

Figures 11 and 12 show the 32 feature maps for a 2 and a 9 sample, respectively. They show how powerful the CNN classifier is, as it is clearly able to pick up so many broad features of the digits. It shows how different edges of the digits are abstracted from different kernels.

Figure 13 shows a plot of the kernels from the first layer of the CNN. It is difficult to interpret, because in the input there is a lot of noise so the kernels used for the first convolution are normally mainly used to reduce noise and abstract some initial small features.

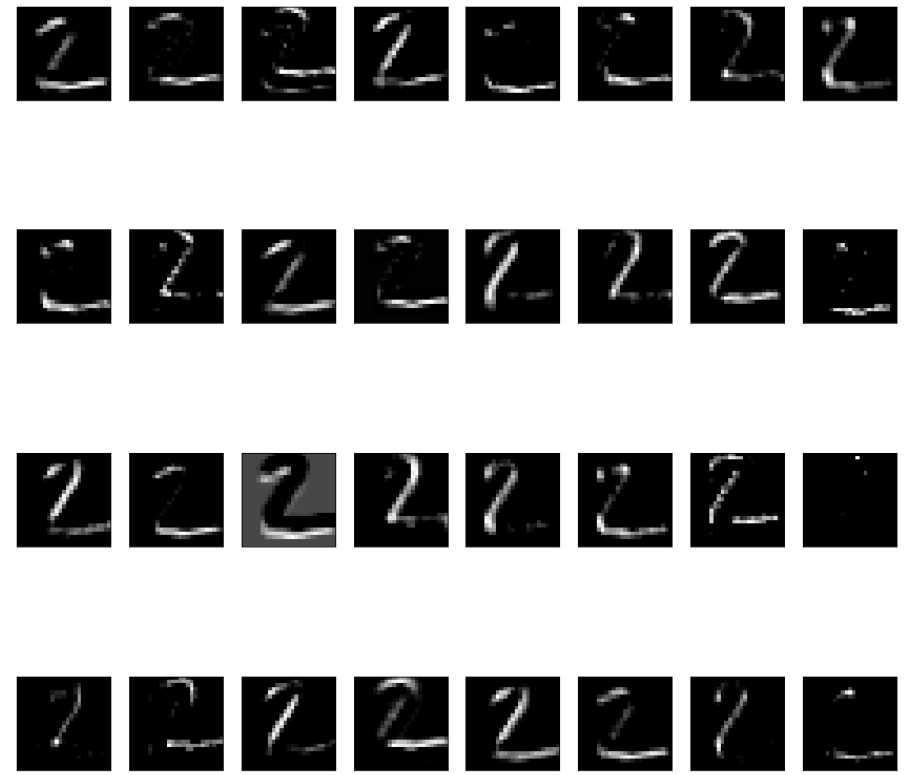

Figure 11. Plot of feature maps after 1 convolution for a sample of a 2.

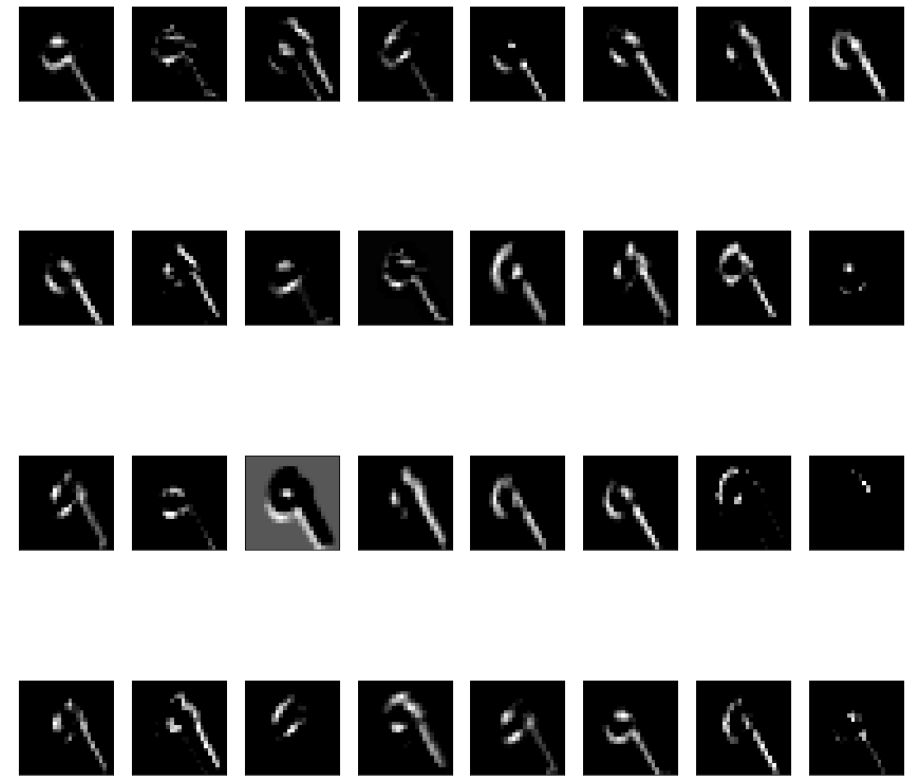

Figure 12. Plot of feature maps after 1 convolution for a sample of a 9.

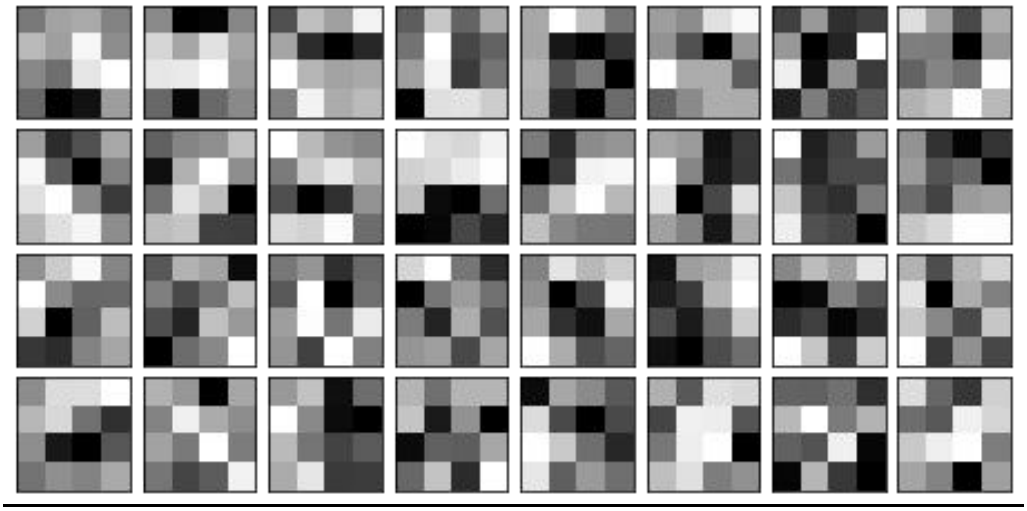

Figure 13. Plot of weights from the first layer of CNN

Task 6: Multitask learning

Table 4. Comparison of accuracies between single task models and the multitask model.

Model	Parameters		Time		Train accuracy		Test Accuracy	
	Task 1	Task 2	Task 1	Task 2	Task 1	Task 2	Task 1	Task 2
Single task	7,024,278	7,023,571	14m 35s	14m 20s	0.9263	0.9470	0.9052	0.9201
MTL	10,339,369		19m 44s		0.9258	0.9549	0.9081	0.9404

Here, the results from single task models are compared to the results for a hard-share multitask classification model. The results show that multi-task learning does allow for better generalisation to unseen data, as shown by the test accuracy scores.

The training accuracy for task 1 is slightly lower for the multitask approach, but its test accuracy is higher, showing that the model is not overfitting to the training data, but that the shared learning process helps the model perform better for unseen data. Task 1 involves classifying the input as one of 10 items of clothing. This is the more difficult task, as task 2 involves only 3 categories for the same input data. Because of this, both types of models perform significantly better on task 2. There is a significant increase in test accuracy going from a single task model to an MTL (0.02), suggesting that the shared training has allowed the model to be able to generalise for unseen data more effectively. This makes sense, as the learned parameters for task 1, classifying the individual clothing item can be used to help better understand the formation of the 3 broader categories of clothing.

The gradient descent training algorithm used categorical cross entropy loss, training to optimize the losses of both tasks equally. I was unable to implement a loss function which could alter the weighting between the two tasks. Generally, one would expect this to increase the accuracy for the task whose loss is the priority, and result in decreased accuracy for the other task. However, I think it would differ for each task. If task 1 (10 category classification) is prioritised completely, meaning that the loss

function for task 2 is not optimised in training at all, I would expect this to give better results than the inverse situation. This is because task 1 involves a deeper understanding of the item of clothing. Learning to distinguish different items of clothing implies an inherent understanding of the item of clothing and what its function is. This information can thus be used to place items more easily into the broader clothing categories. This more closely resembles how humans learn, by looking at detail in order to make conclusions at a broader scale. It seems logical that if you teach a model to understand what items of clothing look like, and then teach it which categories each item goes into, you will get better results compared to trying to categories items of clothing when the model has less understanding of the detail of the clothing items, so would be less able to accurately categorise them broadly.

References

Setting the learning rate of your neural network., 2018. Available from: https://www.jeremyjordan.me/nn-learning-rate/ [Accessed 25 June 2022].