Graded Assignment 2: Deep learning

Data Visualisation

Principal component analysis (PCA) is a technique that can be applied to a dataset in order to express the data in a more meaningful way, and has applications such as "dimensionality reduction, data compression, feature extraction, and data visualization" (Kurita, 2019).

PCA is good option for visualising high dimensional data, allowing data to be projected and visualised easily in 2 or 3 dimensions. PCA transforms the data into linear combinations of the variables creating a new set of variables called principal components (Zhang and Castelló, 2017). This technique maximises variance in the data which is good for visualisation, and is one of the simplest approaches to dimensionality reduction (Plant and Schaefer, 2009), making it computationally simple to run.

Figure 1 shows a PCA representation of the MNIST handwritten digits, where we have applied PCA from scikit-learn to reduce a dataset with 784 features down to 2 dimensions, with each colour representing a class of handwritten digit from 0 to 9. Many of the classes overlap and few

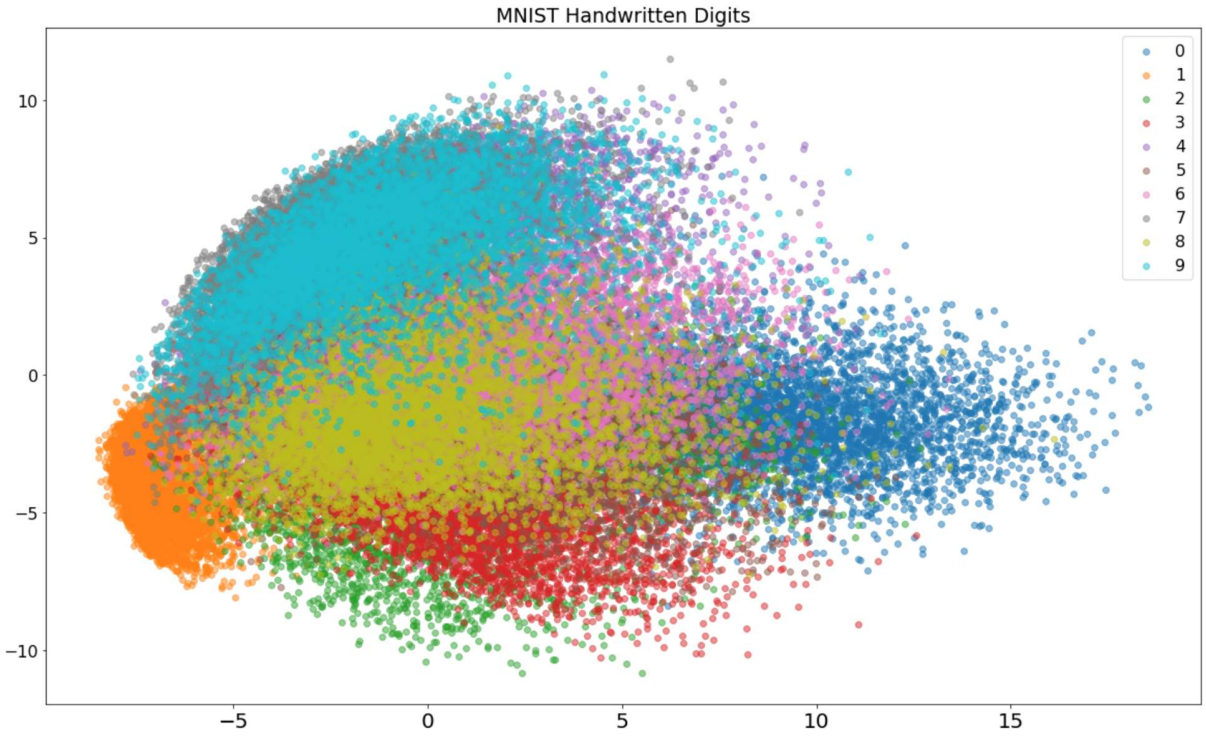

Figure 1

can be separated out, demonstrating that few of the classes are linearly separable. The more the classes overlap the more features they share, therefore numbers that look similar will be more difficult to classify as they are not linearly separable and share similar features, for example 8 and 6 or 9 and 5. We consider the classes to be linearly separable if they can be

separated by a straight line. From observation classes than could be mostly separated include 0 and 1, 2 and 4, 2 and 7, 2 and 9, 0 and 4, 0 and 7, 0 and 9.

Perceptrons

We will train a single layer perceptron to classify handwritten digits from the MNIST dataset into classes of 0 and 1. We need to construct a 'predict' function to classify our data, which will be modelled as an artificial neuron, and will work as an activation function and provide a prediction based on its inputs. The function takes a number n of features x, weights w and a bias term b, applies the dot product operator to aggregate x multiplied by w and adds b. This value is passed into the activation function, which is a step function which classifies the input with a binary label of -1 or 1.

To train the model, we use a method known as back-propagation to continually adjust w, until the predictions become more accurate and the loss tends towards 0. We construct an 'optimise' function which takes in the training dataset and the binary training labels, initialises random w and b and begins iterating and incrementally making continual adjustments to w by calculating the loss function between the predictions and the training labels, using partial derivatives to calculate the gradient with respect to w, and applying gradient descent to tend towards a global minimum which will give the optimum w. We select the learning rate here to be 0.01, as this is noted to be within the typical default range for a starting value (Brownlee, 2019a) and we want to strike a balance between accuracy and training time.

Where we would consider the global minimum to be the optimal value of w where the loss would be 0, in the interest of running time we set a condition that the function will stop once the error reaches an acceptable local minimum of 0.003. As demonstrated in figure 2 we see that the

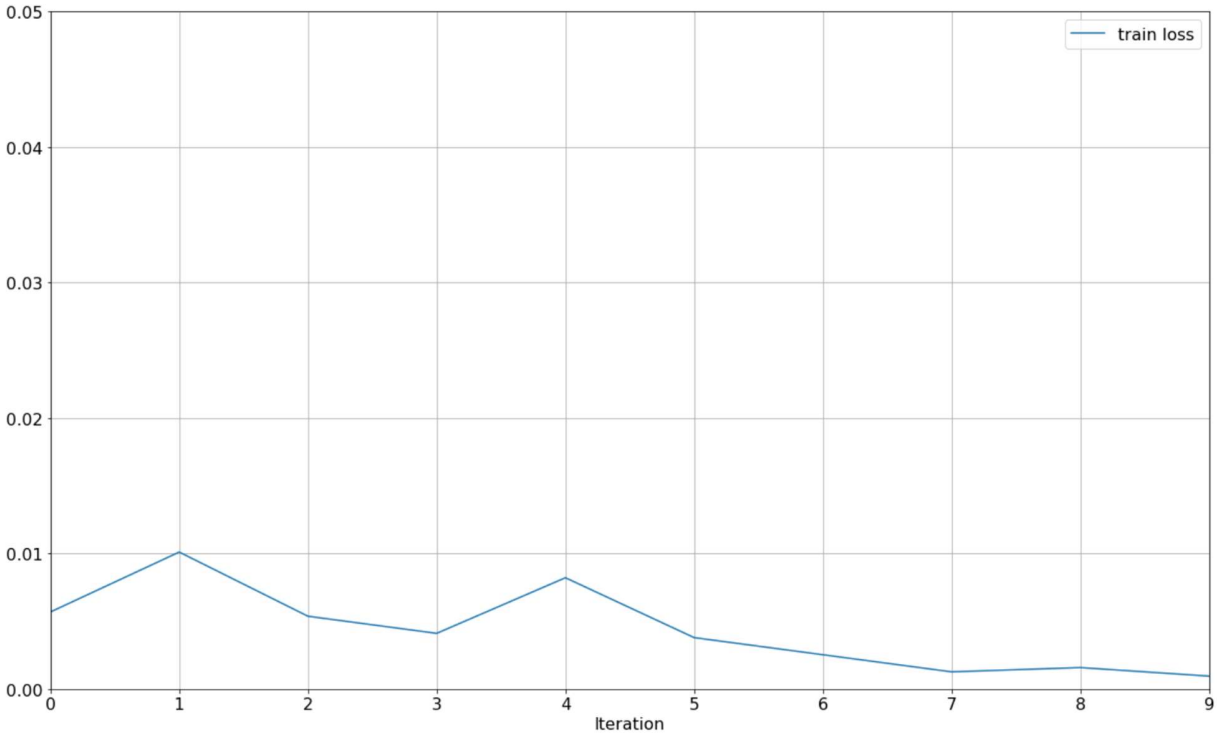

Figure 2

error (loss) gradually decreases until we converge on a local minimum. For the binary classes of 0 and 1 the model converges on the local minimum within 9 iterations.

We can display the parameter w we have been adjusting visually, as seen in figure 3. We see that w appears to be shaped like a 0. This demonstrates that features of the class 0 have been learned. The darker colours represent negative weights, meaning that when a 0 is passed through the predict function these negative weights will be activated and produce a negative prediction and be classified with a -1 label, similarly when a 1 is passed through more positive weights will activated leading to a classification of 1.

Figure 4 shows the training and testing accuracy for the binary classification of the classes 0 and 1. The model produces very high accuracies, with 99.8% on test data and only 4 incorrect classifications.

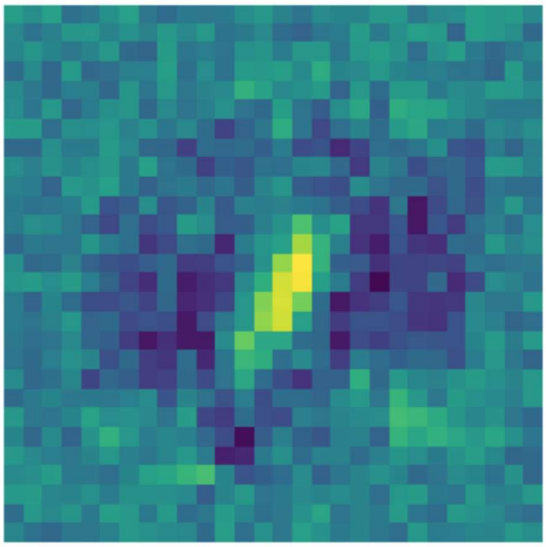

Figure 3

	Accuracy
Training	100.0%
Test	99.8%

Figure 4

We repeat the training and testing process with another 5 pairs of numbers which yield the test accuracies displayed in figure 5.

Pair	Accuracy
(1, 9)	99.5%
(2, 8)	96.2%
(3, 6)	99.3%
(4, 5)	98.1%
(0, 9)	99.0%

Figure 5

All accuracies are above 96% which shows that the model performs well. We can see that some pairs have higher accuracies than others, this is evident in the pairs that are more linearly separable such as 0 and 1 as they have more distinguishable features which make them easier to classify. 2 and 8 and 4 and 5 both have an accuracy of below 99% and it is visible from the PCA visualisation that these numbers do share some cross over of features. This is evident not only in accuracy scores but in training time, the classes with more cross-over took longer to train compared to 0 and 1.

Multi-Layer Perceptron

After training the multi-layer perceptron (MLP) has the following classification accuracy:

- Training data: 98.19%
- Test data: 97.36%

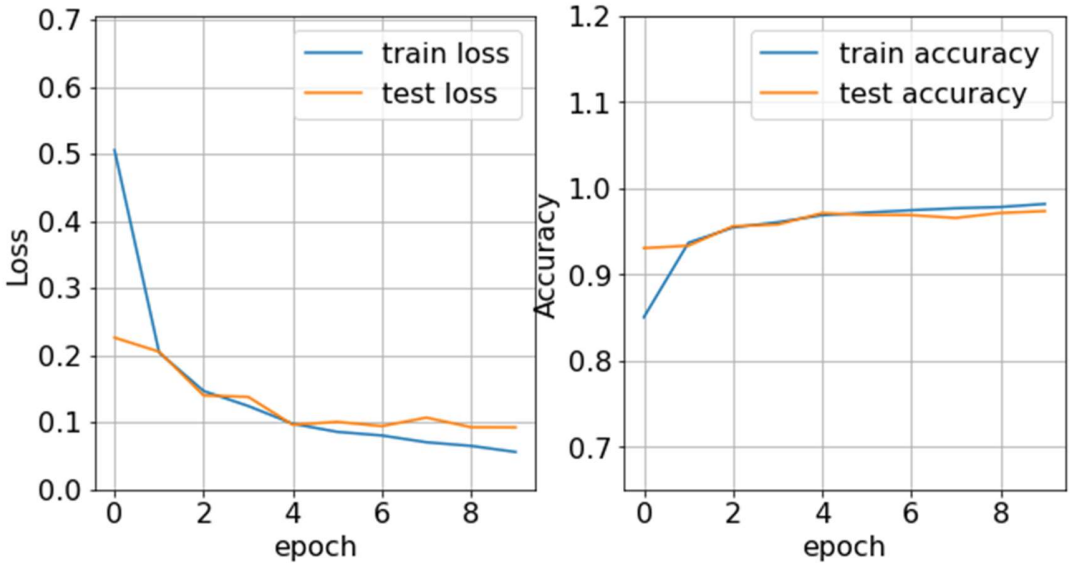

Figure 6

Figure 6 shows the loss and accuracy of the model through the epochs. As the number of epochs increases, the loss on the training and testing data decreases and the accuracy on both datasets increases. The accuracy starts to converge by 10 epochs. The accuracy on both datasets is high and similar so this is a good model and does not look to be overfitted.

To train the model, we configured the parameters. For the learning rate, we kept this at the default value of 0.001. With a higher value, the accuracy was lower after 10 epochs as the model converged to a less optimal solution. With a lower value, the model had lower accuracy after 10 epochs as it was making smaller changes to the weights each time. To reach the same level of accuracy, it required more epochs so the model took longer to train. Using the learning rate of 0.001 meant that the model was more accurate and did not take as long to train.

When looking at the number of epochs used, the model has converged to a solution by 10 epochs. Running more epochs does not significantly change the results but using fewer does give a slightly lower accuracy. Given that running more epochs takes longer, we have chosen to train the model over 10 epochs.

The batch size defines the number of samples that are used to make a weight update in the model. We have used a batch size of 300, so the model weights update after every 300 samples. In each epoch the weights are updating 200 times as there are a total of 60,000 samples in the training data. A smaller batch size means the model takes longer to train as it updates the weights more frequently and here it did not significantly improve on the accuracy of the model. For a larger batch size, the accuracy of the model after 10 epochs was lower as it had not updated the weights as many times or converged to an optimal solution after 10 epochs. The batch size of 300 was chosen to get an acceptable level of accuracy without the model taking longer to train.

When adding more hidden layers, we initially looked at keeping 1,000 neurons in each layer. Without a GPU the model can take up to 50s longer to train with 10 hidden layers instead of 2. With more layers, the number of parameters in the model increases so it takes longer to update the weights and train the model. With 1,000 neurons per layer, the accuracy of the model does not improve and is best for the model with 2 hidden layers as shown in Figure 7. As the accuracy has not improved and the models take longer to train, we have reduced the number of neurons in the hidden layers.

When reducing the number of neurons in each layer, we chose to keep the number of neurons between 10 and 784, the number of neurons in the output and input layers respectively. Having tried different combinations, we ended up using 300 neurons for the first hidden layer and reducing this number for the following layers. The summary of each model used is shown in the Jupyter Notebook. Figure 8 shows the accuracy on test data and the complexity (number of parameters) for each model.

Using fewer neurons per layer, the complexity of the model reduces although when using a GPU all the models take a similar time to train. As the number of hidden layers increases, the accuracy of the model decreases. This shows that the extra hidden layers are not necessary for this model. The model with 2 hidden layers is the best model in this case as it gives the highest accuracy.

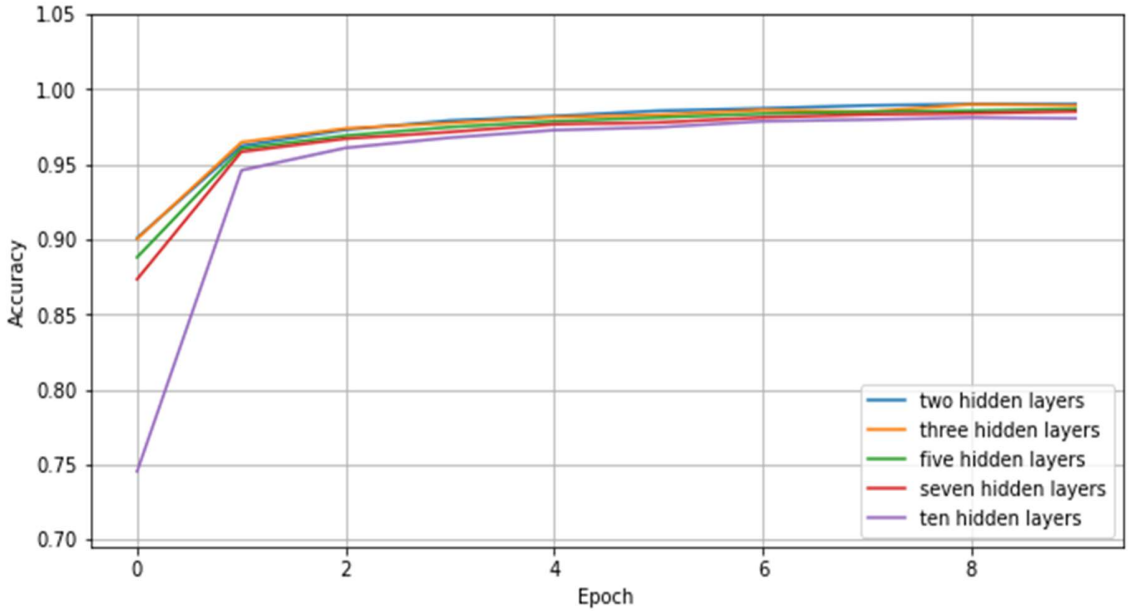

Figure 7

Depth (hidden layers)	Complexity	Accuracy
2	1,796,010	97.70%
3	362,960	96.24%
5	407,210	96.50%
7	552,035	96.25%
10	590,335	94.99%

Figure 8

Convolutional Neural Network

After training the convolutional neural network has the following classification accuracy:

- Training data: 99.37%
- Test data: 98.61%

Figure 9 shows the loss and accuracy of the model through the epochs. As the number of epochs increases, the loss on the training and testing data decreases and the accuracy on both datasets increases. The accuracy starts to converge after 10 epochs. The accuracy on both

datasets is high and similar so this is a good model and does not look to be overfitted. These results are similar to those for the MLP but with a higher level of accuracy.

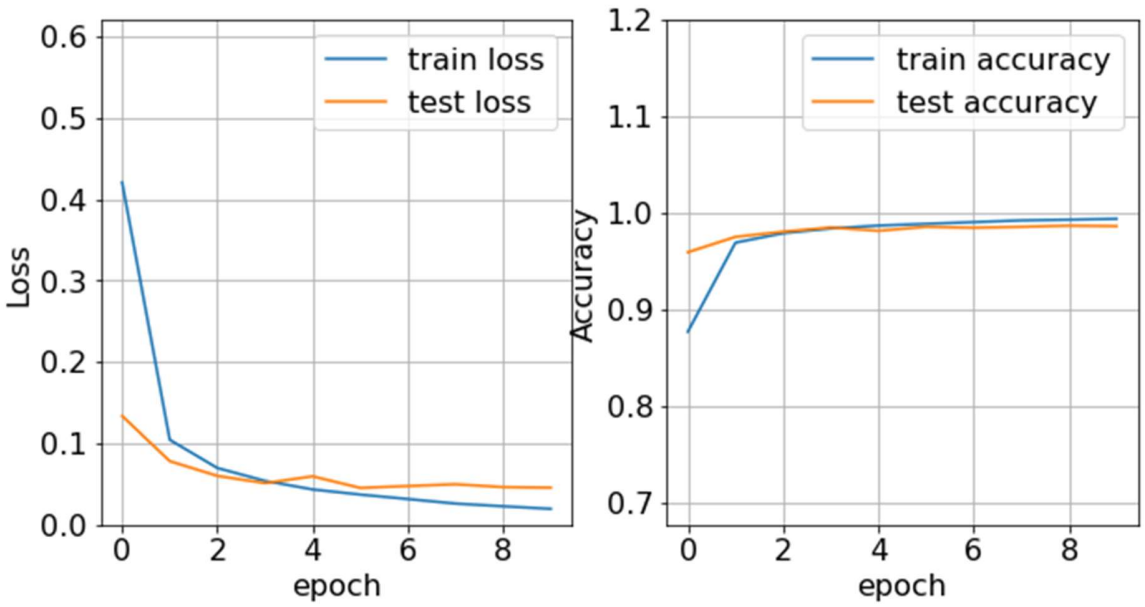

Figure 9

As with the MLP, I have configured the learning rate, batch size and number of epochs. For all three parameters I came to the same conclusions as for the MLP. Therefore, the learning rate is 0.001, the batch size is 300 and the number of epochs is 10.

When creating additional CNNs of different depths and widths I have also revisited the parameters being used, mostly the batch size. Using a batch size of 300 as before led to longer training times, so I increased the batch size to decrease the training time on the models with 4 and 5 hidden Conv2D layers. Even with the larger batch sizes, the models with more layers take significantly longer to train unless using a GPU.

Conv2D Layers	MaxPooling2D Layers	Complexity	Accuracy
3	1	185,066	98.61%
4	1	182,282	97.86%
5	2	247,882	98.40%
2	1	85,226	98.72%
1	1	46,634	97.90%

Figure 8

Figure 10 shows the number of layers in the models, the accuracy on test data and the complexity (number of parameters) for each model.

Increasing the number of layers achieves slightly lower levels of accuracy to the initial CNN, and when training without a GPU they take significantly longer to train. The increase in training time does not correspond to an increase in accuracy so these are not improved models. Using fewer layers, the training time is lower and the model with 2 Conv2D layers and 1 MaxPooling2D layer has a higher accuracy than the initial model. Although often using more convolutional layers improves the accuracy as it reduces the number of input features, it is not worth the training time so is better to use 2 or 3.

To compare the CNNs and MLPs, I have selected the default model and the best of the 4 I created. The results of these are given in Figure 11.

Model Type	Hidden Layers	Parameters	Training Time	Accuracy
MLP	2	1,796,010	13s	97.70%
MLP	3	362,960	10s	97.68%
CNN	4	185,066	17s	98.61%
CNN	3	85,226	10s	98.72%

Figure 11

The CNNs give a higher level of accuracy than the MLPs across all variations that I have tried. The training time is similar for all models and is shown here when using a GPU. Therefore, the CNN models are the best to use, with the accuracy of the two here being very similar although the model with 3 total hidden layers gives the highest accuracy.

Visualising CNN outcomes

As the filters and feature maps will only be shown for the convolutional layers, I have created a new CNN that is the same as the initial CNN in task 4 but without the MaxPooling2d layer. I have then updated the stride of the first convolutional layer to be 2 to help with downsampling. The convolutional layers make up layers 0, 1 and 2 of the model and I have plotted the filters and feature maps for each layer based on the method by Brownlee (2019b). For layer 0, the filters are in Figure 12, the feature maps for a ‘2’ are in Figure 13 and the feature maps for a ‘9’ are in Figure 14.

As this is the first layer and is close to the input the feature map captures a lot of the detail and it is possible to see that these are the digits ‘2’ and ‘9’. There is no pattern seen in the filters.

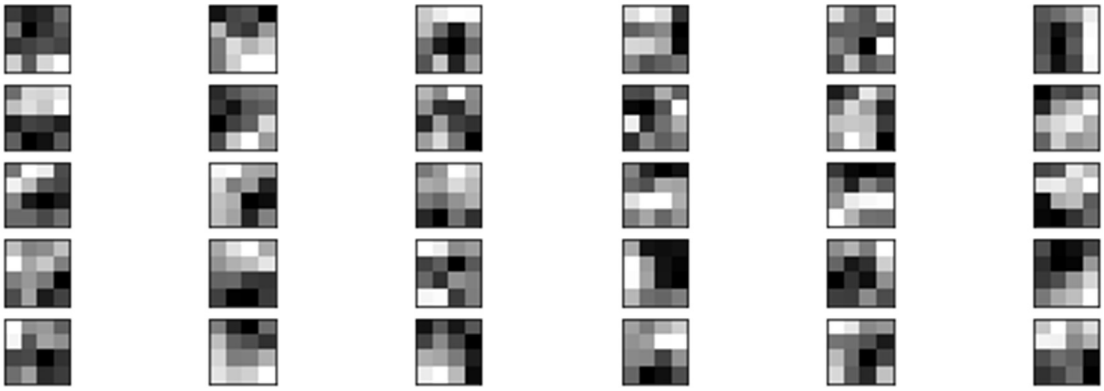

Figure 12

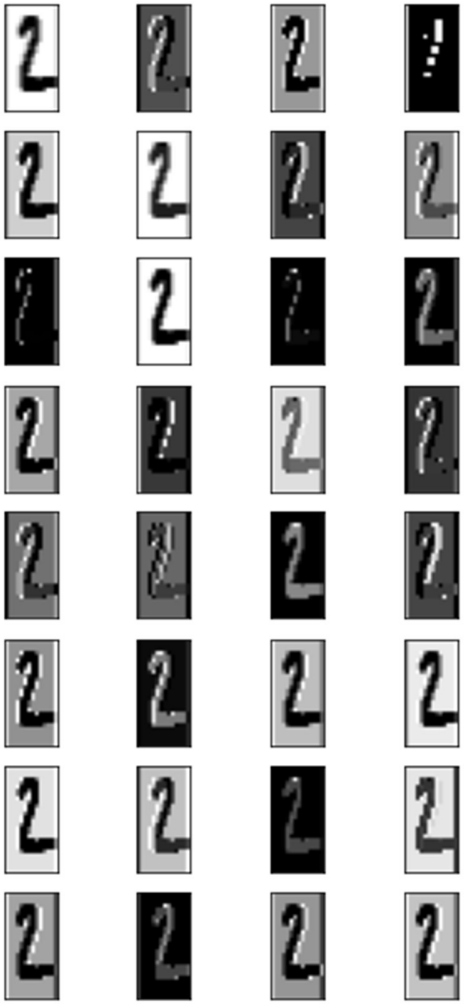

Figure 13

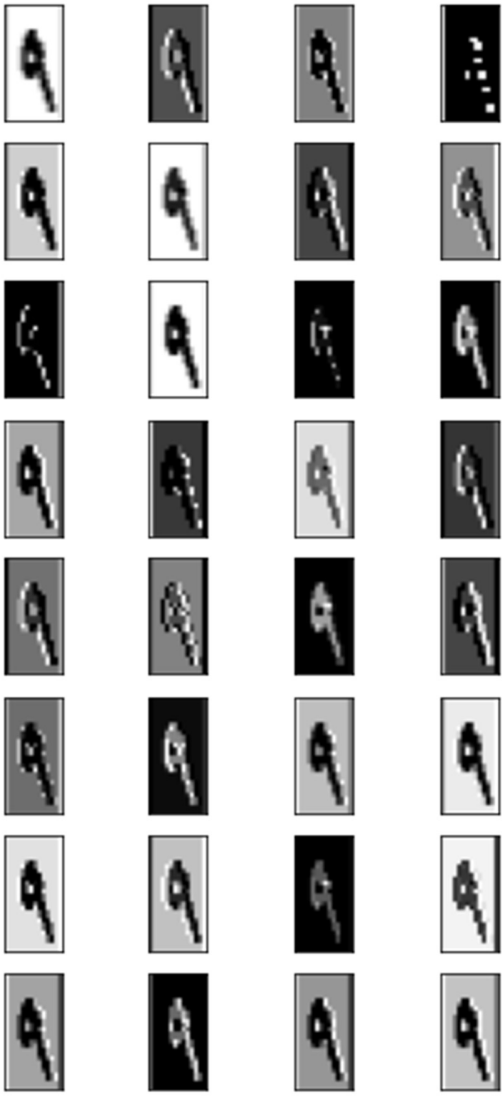

Figure 14

For layer 1, the filters are in Figure 15, the feature maps for a ‘2’ are in Figure 16 and the feature maps for a ‘9’ are in Figure 17.

The feature maps are still recognizable as the digits ‘2’ and ‘9’ for most of the images but are not as clear. Again, we don’t see a pattern in the filters for this layer.

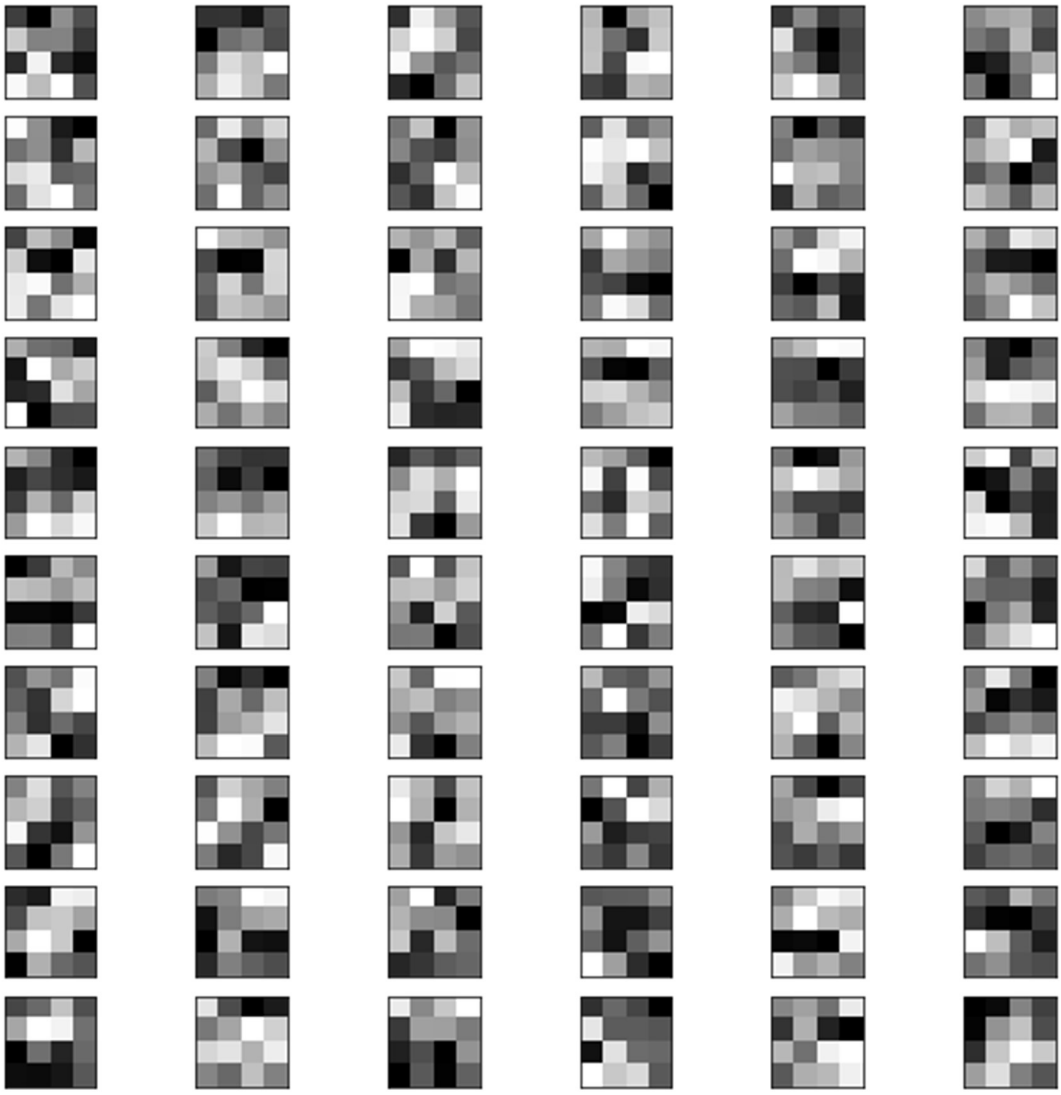

Figure 15

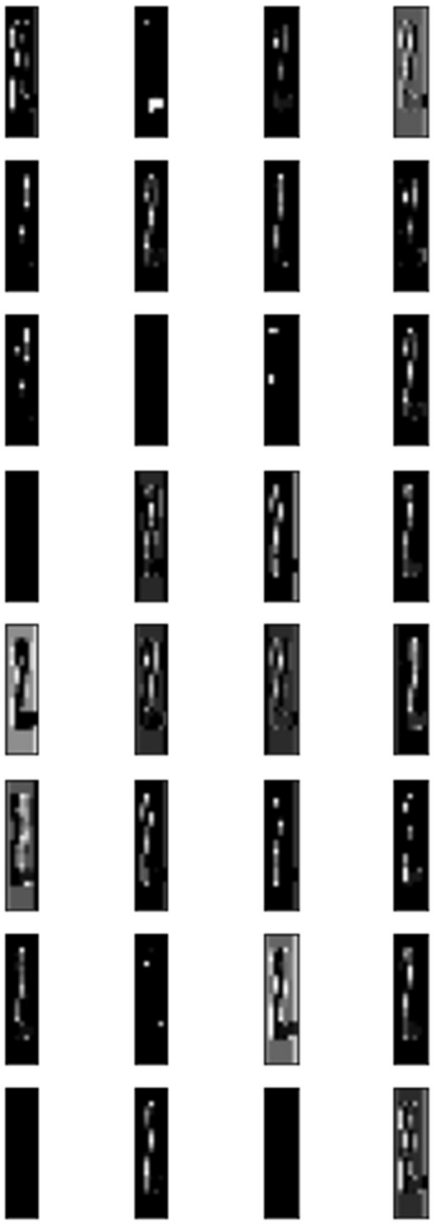

Figure 16

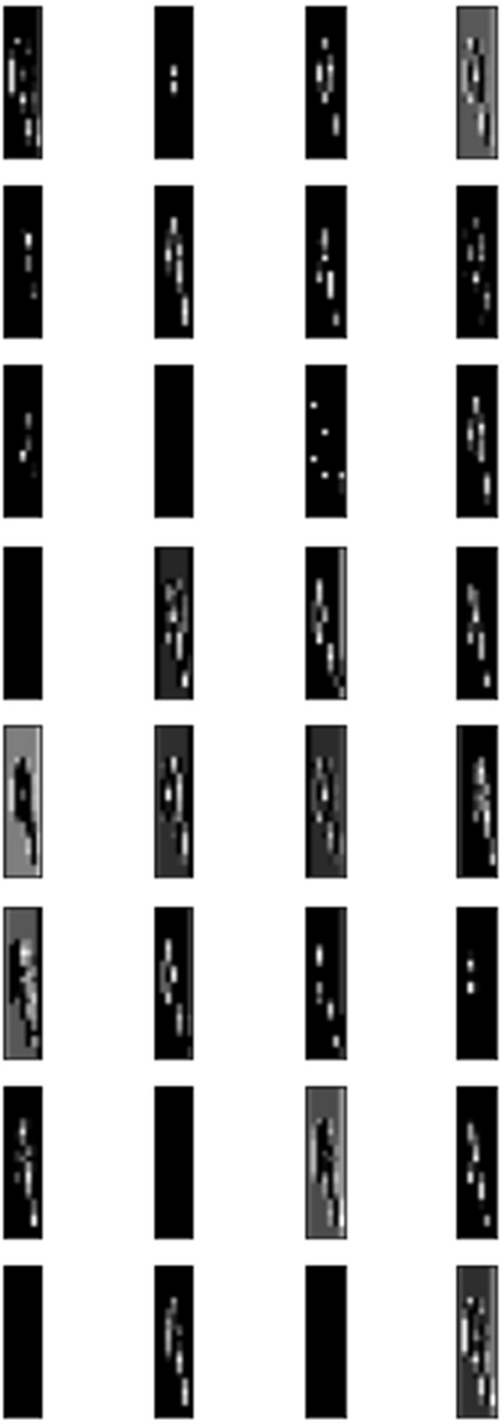

Figure 17

For layer 2, the filters are in Figure 18, the feature maps for a ‘2’ are in Figure 19 and the feature maps for a ‘9’ are in Figure 20.

The feature maps are now not recognizable for as the digits ‘2’ and ‘9’ for most of the images as we are a lot further from the input, so the feature map captures less detail. Again, we don’t see a pattern in the filters for this layer.

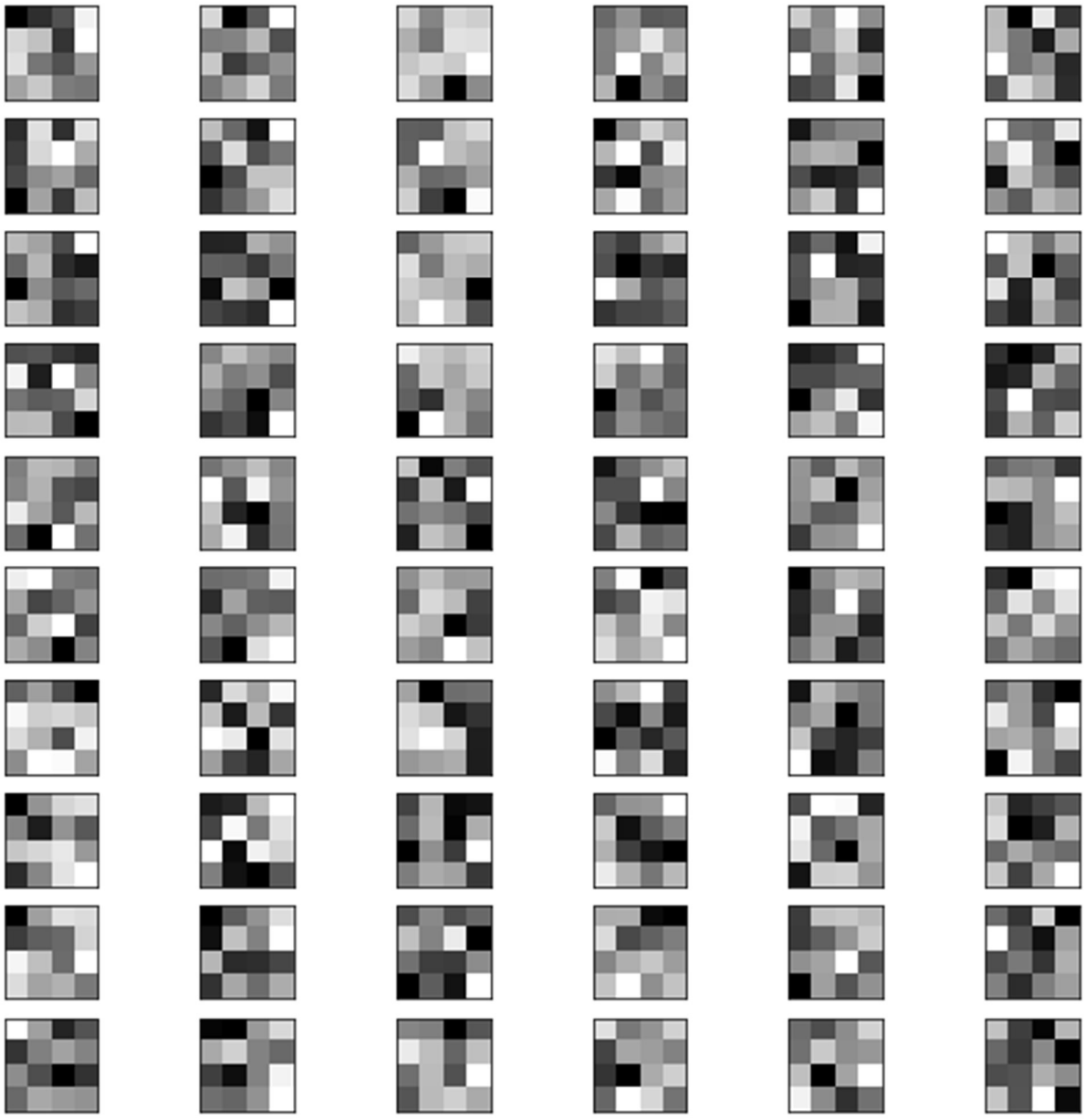

Figure 18

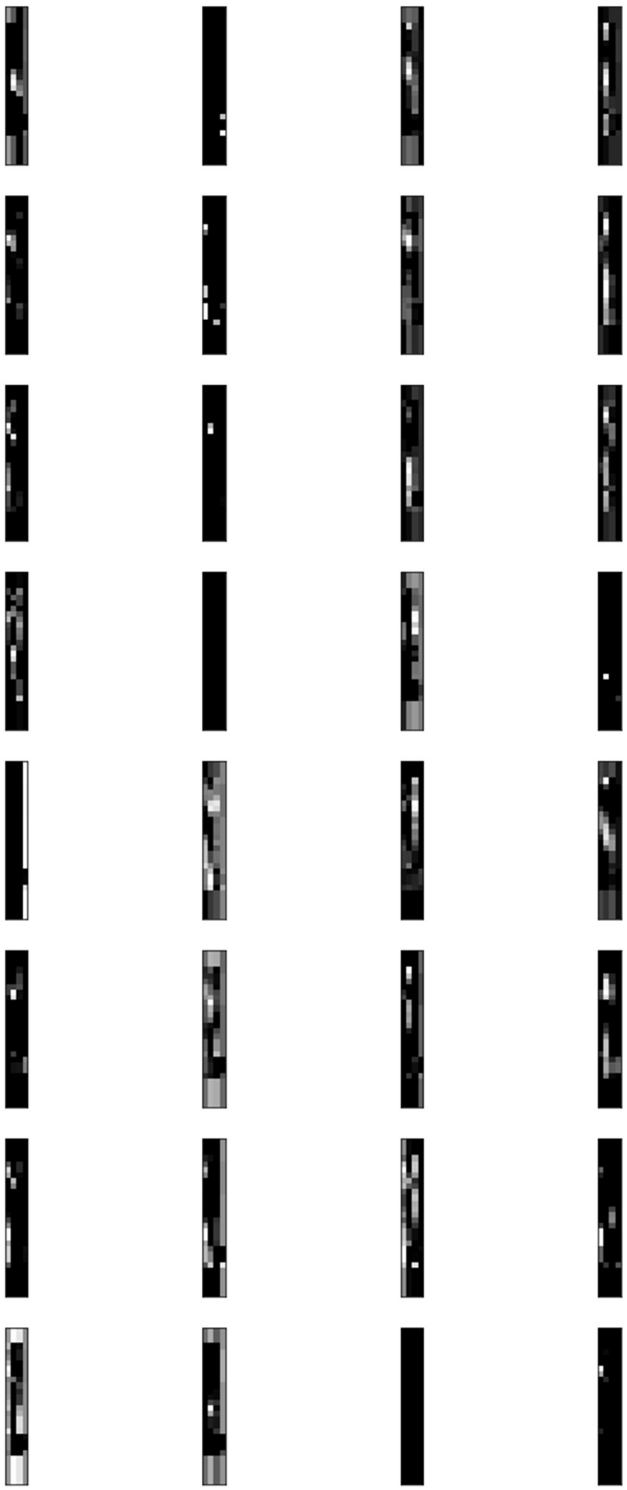

Figure 19

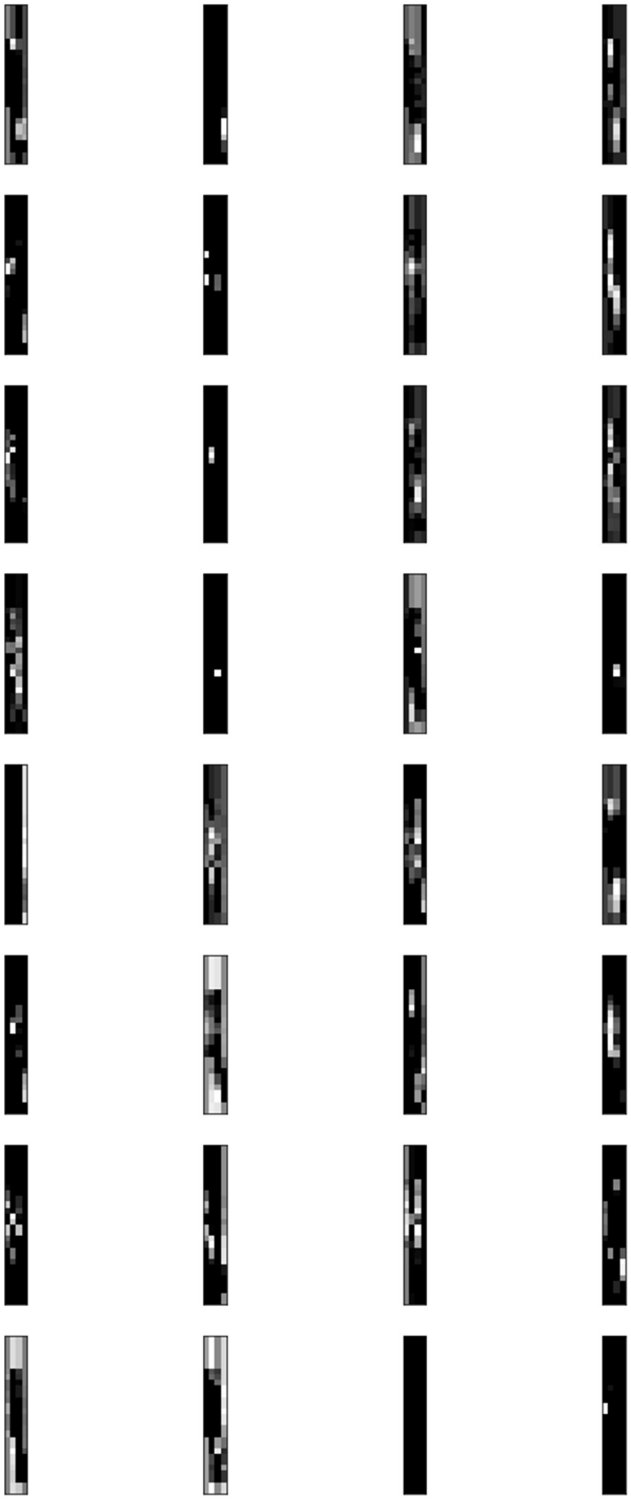

Figure 20

Multi-task Learning

Here we investigate the application of multi-task learning (MTL) for the classification of items in the MNIST fashion dataset. In MTL we utilise information from related tasks to improve performance on all or at least one of the tasks. Here, we will use a form of MTL called hard parameter sharing, which uses the same hidden layers for each of the tasks, meaning the model learns from the same representation of the input data to perform each task.

We construct two separate CNN classifiers for each of the described tasks, to compare performance with our MTL model performance.

	Task 1 accuracy	Task 2 accuracy
Training	94.1%	93.7%
Test	91.4%	92.9%

Figure 21

As shown in figure 21 our task specific CNN classifiers return good accuracies for the train and test data.

We now construct an MTL model which follows a similar format to our previous CNN's, this time we add the shared layers to a main branch which becomes our shared representation for both tasks, and then we create task specific branches to add the task specific layers to.

Gamma	Task 1 accuracy	Task 2 accuracy
0	2.0%	90.6%
0.25	89.9%	93.6%
0.5	89.9%	94.0%
0.75	89.4%	93.5%
1	89.8%	45.1%

Figure 22

The results for our MTL model are displayed in figure 22. Task 2 demonstrated a better performance than task 1 using this model and using a gamma of 0.5 yielded the highest performance for both tasks. For task 1 our performance was lower than using a task specific CNN, however with task 2 we saw an increase in accuracy of 1.1% using a gamma value of 0.5 compared to the accuracy in the task specific model. These results demonstrate that the

features learned in task 1 are useful in improving accuracy for task 2, although the reverse does not seem to be evident.

As shown in figure 22 when gamma is 0 in task 1 and when gamma is 1in task 2, the accuracy is low. This is because gamma is the parameter that weights the losses between tasks, so when gamma is 0 the model is only accounting for the loss in task 2 and is therefore only learning how to perform task 2 and vice versa.

One of the most important things to consider is what kind of auxiliary tasks we are using, we need to find tasks that complement each other and don't hinder each other e.g. by introducing unnecessary noise into the model. Some tasks will also be more reliable so finding the correct weightings between all our tasks to produce the maximum performance is key (Tang, 2006).

When implemented correctly MTL has benefits including improving accuracy, improving speed of learning, improving generalisation and reducing the risk of overfitting, improving model focus on relevant features in a noisy dataset and averaging through noise patterns across tasks to gain a better representation of the data (Ruder, 2017). However, there is still much work to be done to fully understand best types of auxiliary tasks to improve performance, there is no 'one size fits all' approach to MTL as all data and tasks have their own nuances and optimisation should be considered on a case by case basis (Vandenhende et al., 2020). While there are tasks that are beneficial to learning some can also hinder and lower accuracy and speed of learning.

References

Brownlee, J. (2019a). How to Configure the Learning Rate When Training Deep Learning Neural Networks. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/#:~:text=A%20traditional%20default%20value%20for.

Brownlee, J., (2019b). *How to Visualize Filters and Feature Maps in Convolutional Neural Networks* [Online]. Available from: https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/ [Accessed 31/10/2022]

Kurita, T., 2019. Principal component analysis (PCA). Computer Vision: A Reference Guide, pp.1-4.

Plant, W. and Schaefer, G. (2009). Visualising image databases. [online] IEEE Xplore. doi:10.1109/MMSP.2009.5293293.

Ruder, S. (2017). An Overview of Multi-Task Learning in Deep Neural Networks. [online] arXiv.org. Available at: https://arxiv.org/abs/1706.05098.

Tang, L. (2006). Multitask Learning. [online] Available at: http://leitang.net/presentation/multitask.pdf.

Vandenhende, S., Georgoulis, S., Proesmans, M., Dai, D. and Van Gool, L. (2020). Revisiting Multi-Task Learning in the Deep Learning Era. [online] Available at: https://homes.esat.kuleuven.be/~konijn/publications/2020/vandenhende.pdf.

Zhang, Z. and Castelló, A. (2017). Principal components analysis in clinical studies. Annals of Translational Medicine, 5(17), pp.351–351. doi:10.21037/atm.2017.07.12.

Contribution Table

Student Name	**Student Number**	**Contribution**
Sophie Wilson	159002005	50%
Ffion Harries	169206275	50%