Group Assignment 2: Deep Learning

Table of Contributions

Leon Roe	Najib Al Awar
Equal participation - Both completed code and merged work and both worked through report	Equal participation - Both completed code and merged work and both worked through report

Data Visualisation (Task 1)

The Dataset and Principal Component Analysis

The MNIST dataset, consisting of 784-dimensional handwritten digit images, was analysed using Principal Component Analysis (PCA) to reduce dimensionality and explore digit separability. PCA is effective for visualising high-dimensional data as it reduces the original 784 dimensions into a manageable 2D space, capturing the directions of highest variance.

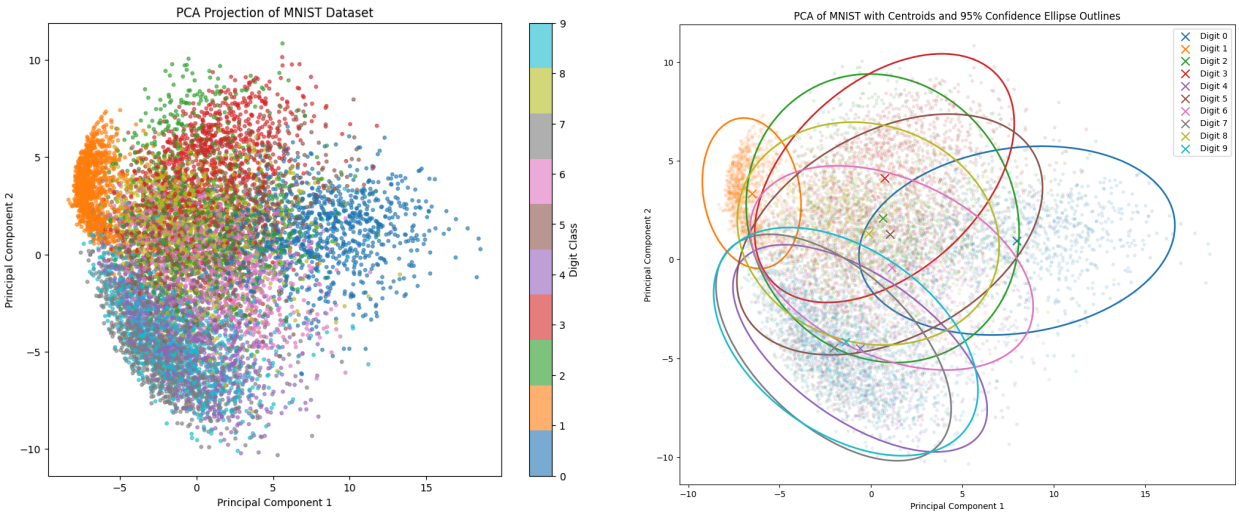
Mathematically, PCA involves computing the covariance matrix of the centered data X:x

$$Cov(X) = \frac{1}{n-1} X^T X$$

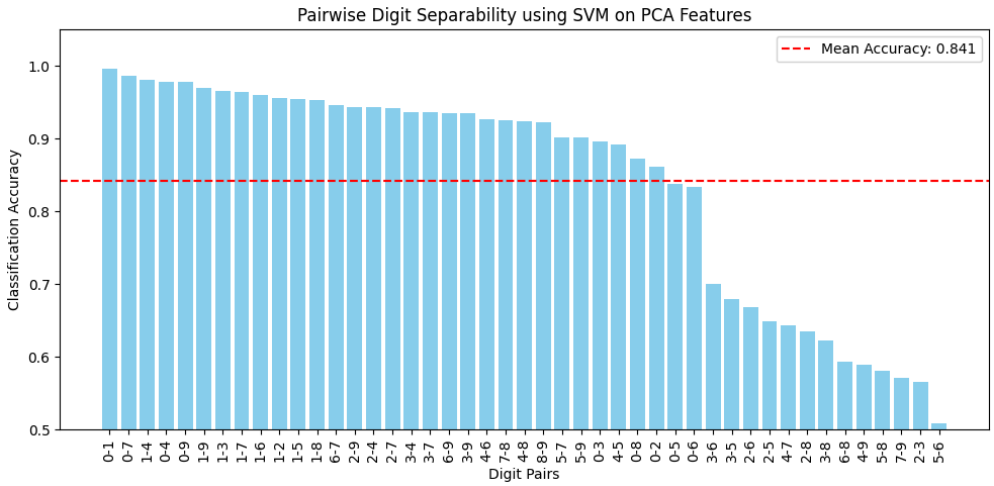
where **X** is an n×784 data matrix after mean-centering (each feature has zero mean). PCA then applies eigendecomposition to this covariance matrix, to obtain principal components:

$$X_{PCA} = XV_2$$

Below are the plots from this PCA on the MNIST data set. The left shows the raw PCA plotted for 10,000 random samples, the left shows the same 10,000 samples but with centroids and 95% confidence ellipse outlines.



Below is a histogram of the data from a LinearSVC that we implemented to assess the separability of MNIST digit pairs in 2D PCA space. It trains the classifier with 10,000 iterations for each pair, calculates their separation accuracy, showing a mean accuracy of 0.841.



Observations from the Plots and Data

The histogram illustrates the classification accuracy of digit pairs using an SVM classifier on 2D PCA projected data, with a mean accuracy of 0.841. Pairs such as (0, 1), (0, 7), and (1, 4) achieve accuracies near 1.0, indicating strong separability due to distinct visual features, such as 0s circular shape versus 1s vertical line. Conversely, pairs like (3, 5), (3, 8), and (7, 9) have lower accuracies, suggesting similarities, like the curves of 3 and 8, make them harder to distinguish.

The table of PCA results shows digit 1s centroid at (6.71, 3.38) with low PC1 variance (1.32), indicating a tight cluster, while digit 0s centroid at (7.77, 1.75) with higher PC1 variance (13.70) suggests a broader spread. Digits 4, 7, and 9 have similar PC2 values (4.37 to 4.63), hinting at overlap, unlike the distant digits 1 and 0.

Digit	Centroid_PC1	Centroid_PC2	Var_PC1	Var_PC2	N_Samples
0	7.77	1.75	13.70	3.17	958
1	-6.71	3.38	1.32	3.60	1122
2	0.78	2.81	9.77	7.09	971
3	0.61	2.86	8.22	6.11	1012
4	-0.33	-4.37	8.33	4.61	960
5	0.83	0.77	11.14	5.43	967
6	1.29	0.68	11.03	4.18	1021
7	-1.96	-4.63	6.56	6.55	1024
8	0.04	1.29	8.95	5.08	931
9	-1.09	-4.63	7.99	5.44	1034

Classes That Can Be Linearly Separated

Digits 1 and 0 are linearly separable in the 2D PCA space, as their clusters are distinctly positioned with minimal overlap, supported by close to 1 SVM accuracies. Digit 4 shows partial separability, particularly from digits like 1 and 0, but overlaps with others like 5 and 9. Digits 3, 5, and 8, with significant overlap in both scatter plots and lower accuracies (around 0.6), cannot be linearly separated here, nor can pairs like 7 or 9 due to proximity and shared features.

PCA effectively visualises the MNIST dataset by reducing dimensionality and revealing clustering patterns. While digits 1 and 0 are linearly separable, the overlap among digits like 3, 5, and 8 underscores PCA's limitations in fully separating all classes in 2D, suggesting that higher dimensions or non linear methods might enhance separability for these challenging digits.

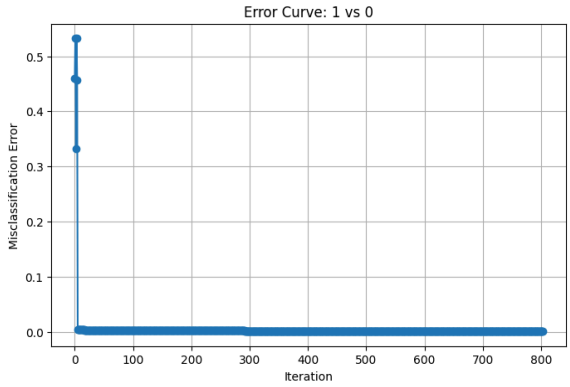
Perceptrons (Task 2)

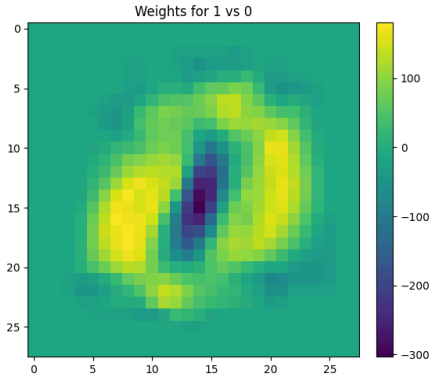
The perceptron algorithm is a fundamental supervised learning technique for binary classification. particularly for. The perceptron takes in a flattened 28x28 array of pixel values. It is also initialised with a weight vector w and bias b randomly, then iteratively adjusts them for misclassified examples, “pushing” the decision boundary towards the correct class to reduce errors progressively.

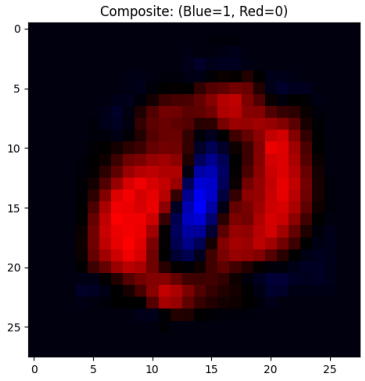
Key hyperparameters include a maximum of 1000 iterations (max_iter=1000) to limit training duration, a tolerance of 1e-3 (tol=1e-3) to halt when misclassifications stabilise, and a learning rate of 0.01 (learning_rate=0.01) to control update magnitude, ensuring stable convergence.

Below are some samples of digit pairs that were trained using this approach:

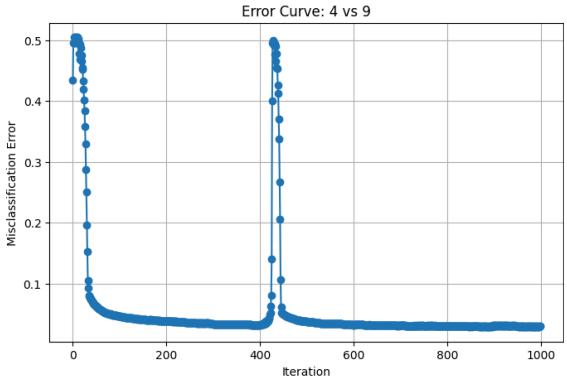
Digit 1 vs 0

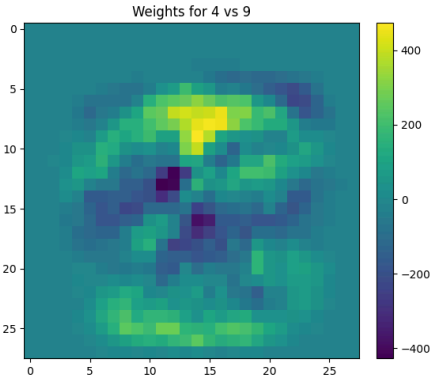


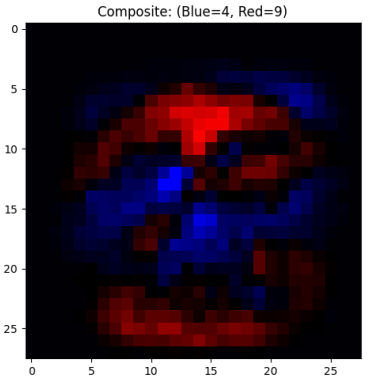




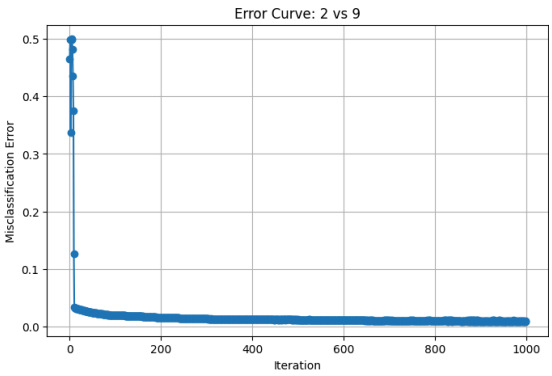
Digit 4 vs 9

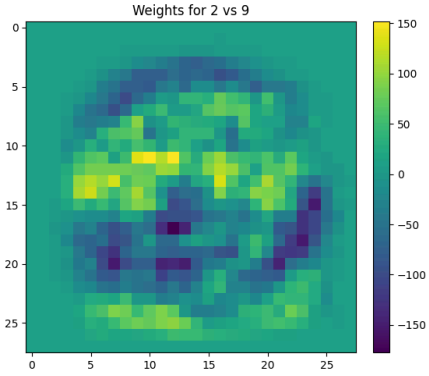


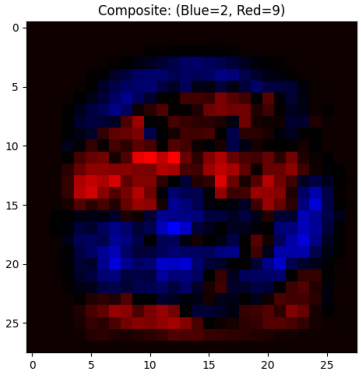




Digit 2 vs 9







Mathematically, the perceptron computes a weighted sum of inputs:

$$w \cdot x + b$$

where **w** is the weight vector, **x** is the input vector, and **b** is the bias. It then applies a step function:

$$y = sign(w \cdot x + b)$$

to classify the result as +1 (one digit) or -1 (the other), where **sign** outputs +1 for positive values and -1 for negative. For a misclassified sample (where the predicted label differs from the true label **y**), the update rule is:

$$w \leftarrow w + \eta y \cdot x,\ b \leftarrow v + \eta y$$

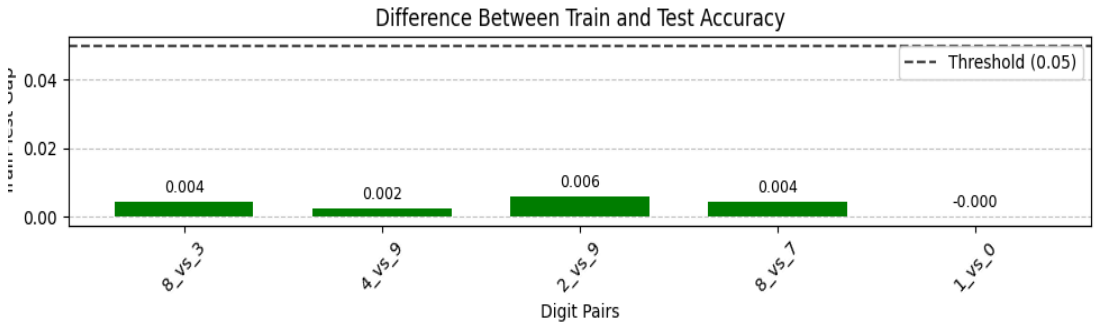
with **η**=0.01 as the learning rate. This gradient-free adjustment minimises classification errors by aligning the decision hyperplane with the data.

Analysis of the perceptrons results

A primary indicator of convergence is the training misclassification error curve over iterations. Initially high due to random initialisation, the error for pairs like 1 vs 0 drops sharply within dozens of iterations, flattening near zero as a linear separator emerges. Even for trickier pairs like 8 vs 3 or 4 vs 9, the error stabilises low within a few hundred iterations, demonstrating convergence to a good local minimum.

To visualise what the perceptron learns, its final weight vector w is reshaped into a 28x28 grid, forming a heatmap. Large positive weights highlight regions favouring one digit, while negative weights favour the other. For 8 vs 7, bright positives mark 8s loops, and negatives emphasise 7s horizontal bar. In 1 vs 0, the vertical stroke of 1 contrasts with 0s circular outline. These

heatmaps show the perceptron targeting distinctive features like loops, bars, or strokes.



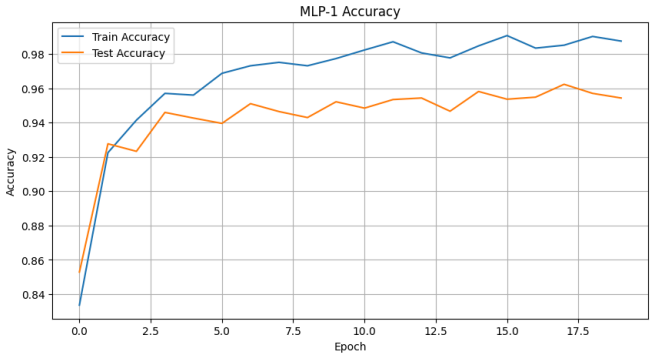
Digit Pair	Train Accuracy	Test Accuracy	Iterations
1_vs_0	0.999	0.999	781
8_vs_3	0.970	0.967	1000
4_vs_9	0.970	0.969	1000
8_vs_7	0.979	0.969	1000
2_vs_9	0.990	0.983	1000

Performance metrics affirm its efficacy. Easier pairs, such as 1 vs 0 or 2 vs 9, achieve over 99% accuracy on training and test sets, reflecting clear visual differences. Pairs like 4 vs 9 and 8 vs 3, with overlapping shapes, settle at 97% to 98% test accuracy. Occasional error spikes may stem from data reshuffling or tough samples, yet the perceptron adapts, converging stably.

The plot of train-test accuracy differences below, shows minimal gaps (e.g., 0.006 for 2 vs 9, 0.004 for 8 vs 3, below a 0.05 threshold), indicating low overfitting risk. This suggests the model generalises well, with training and test accuracies closely aligned across digit pairs, reinforcing the perceptrons robustness.

These results highlight key insights. A simple linear model, updated via the perceptron rule, learns meaningful pixel-based features without complex architecture. Pairs with distinct visuals converge faster and score higher, while structural similarities demand more iterations and yield slightly lower accuracy. The weight matrices offer an interpretable glimpse into critical pixel regions, proving linear models can pinpoint discriminative image parts.

Multi-layer Perceptrons (Task 3)



The experimentation with Multilayer Perceptron (MLP) architectures highlights significant insights into how varying depth and the number of parameters affect classification performance. Initial tests began with the simplest architecture ([1000, 1000]), consisting of two hidden layers totaling approximately 1.8 million parameters. The training and test accuracies of this baseline model reached 99.19% and 95.44%, respectively. Subsequent architectures, featuring more layers or parameters, offered mixed improvements, clearly illustrating the complex relationship between model complexity and performance.

Mathematically, an MLP is described by equations governing forward propagation:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

Followed by the activation function:

$$a^{(l)} = \sigma(z^{(l)})$$

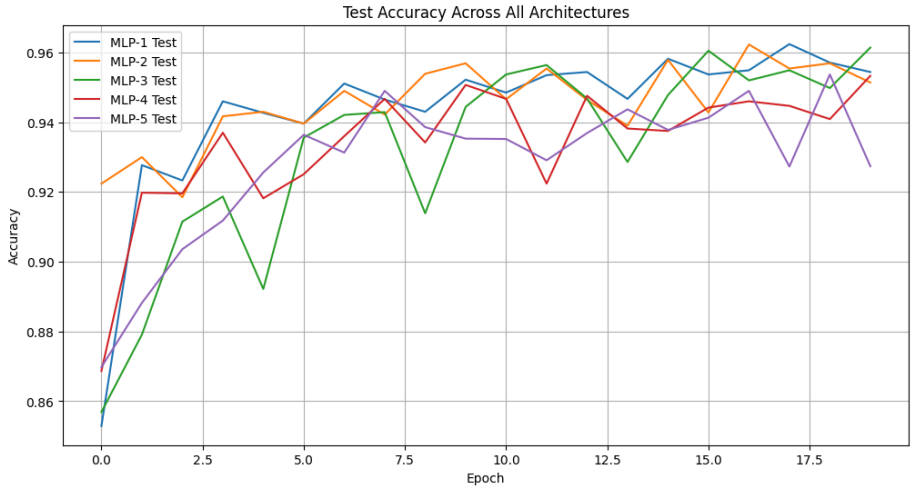
The final output layer commonly compute via softmax for classification tasks:

$$y = softmax(z^{(l)})$$

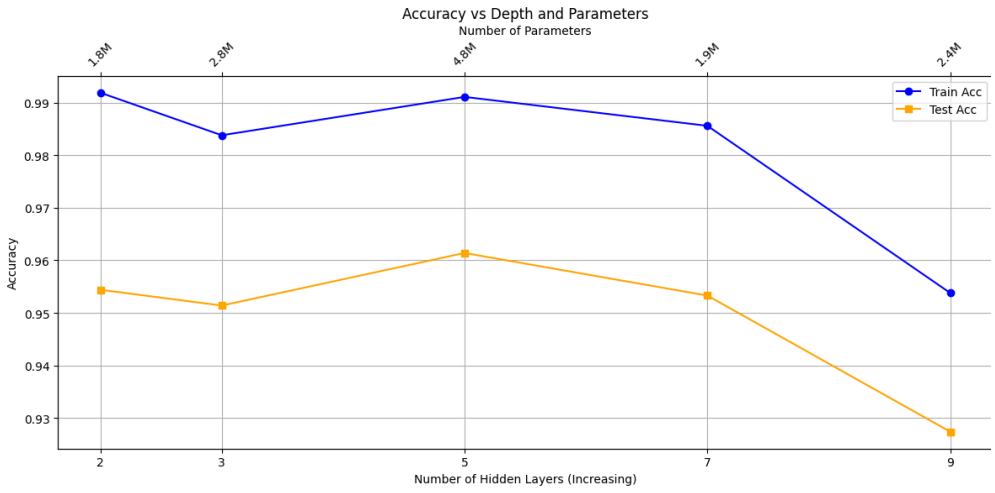
MLP Architecture Comparison

Expanding to a deeper network ([1000, 1000, 1000]) increased the total parameter count to around 2.8 million. Interestingly, despite the added complexity, the training accuracy slightly decreased to 98.38%, although test accuracy improved marginally to 95.14%. This pattern persisted with the even larger 5-layer network ([1000, 1000, 1000, 1000, 1000]) containing about 4.8 million parameters, which achieved a notable improvement in both training accuracy

(99.11%) and test accuracy (96.14%), suggesting a beneficial impact from deeper architectures at this scale.



Contrastingly, models with more layers but fewer neurons per layer, such as the 7-layer ([500, 500, 500, 500, 500, 500, 500]) and 9-layer ([500, 500, 500, 500, 500, 500, 500, 500, 500]) configurations, resulted in declining performance. Despite having 1.9 million and 2.4 million parameters respectively, both showed reduced accuracies (training: 98.56% and 95.38%; testing: 95.33% and 92.74%), suggesting a detrimer model demonstrated robust performance relative to the deeper and more parameter-intensive networks. Although not always the best performer, it remained competitive, indicating a nonlinear relationship between depth, width, and generalisation capability. Indeed, models closer in structure to the initial implementation ([1000, 1000]) generally exhibited balanced performances, reinforcing the notion that moderate complexity can offer optimal generalisation.



The results corroborate findings by Neyshabur et al. (2018), who showed that increased network parameters often improve generalisation, even when capable of memorising random labels, emphasising the beneficial impact of over-parametrisation up to a point.

MLP	Architecture	Hidden Layers	Parameters	Train Accuracy	Test Accuracy	Train Loss	Test Loss	Training Time (s)
MLP-1	[1000, 1000]	2	1,796,010	0.9919	0.9544	0.0383	0.2104	41.07
MLP-2	[1000, 1000, 1000]	3	2,797,010	0.9838	0.9514	0.0565	0.2133	63.43
MLP-3	[1000, 1000, 1000, 1000, 1000]	5	4,799,010	0.9911	0.9614	0.0864	0.2222	100.95
MLP-4	[500, 500, 500, 500, 500, 500, 500]	7	1,900,510	0.9856	0.9533	0.0878	0.2233	86.08
MLP-5	[500, 500, 500, 500, 500, 500, 500, 500, 500]	9	2,401,510	0.9538	0.9274	0.0856	0.3235	103.04

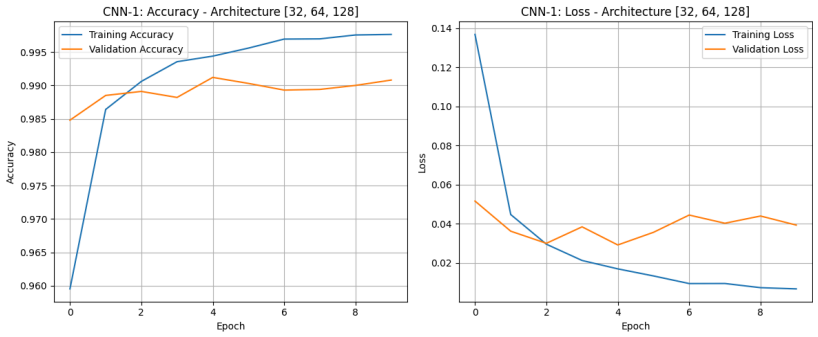
Learning curves revealed a common trend: test accuracy rapidly increased during early epochs before plateauing or slightly oscillating. This indicates that networks initially learn generalisable features quickly, after which improvements become marginal and oscillatory, potentially due to noise in gradient updates or minor overfitting. Interestingly, deeper models (MLP-3 and beyond) exhibited more pronounced oscillations, suggesting deeper structures might introduce training instabilities.

In conclusion, these experiments affirm the nuanced relationship between neural network depth, parameter count, and classification accuracy. Moderate network depth and sufficient width (number of neurons per layer) appear optimal for performance, balancing the benefits of complexity against the risks of instability and overfitting. Future studies could further explore these trade-offs using advanced regularisation techniques.

CNN (Task 4)

Base CNN Training [32, 64, 128]

The base recommended CNN-1, constructed using three convolutional layers using filters of shape 32, 64 and 128 obtained a test accuracy of 99.08% after 20 epochs.



Convolution enables the network to extract features from input data. It involves sliding a small matrix called a filter over the image to produce a feature map. For a 2D input, convolution is defined as:

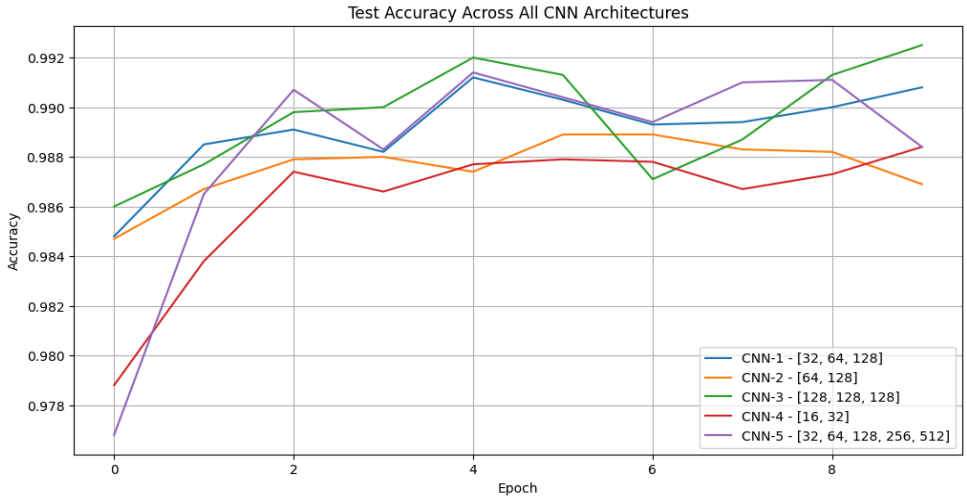
$$(f * g)(i,j) = \sum_m \sum_n f(m,n) \cdot g(i-m, j-n)$$

Here, **f** represents the input data (image pixels), and **g** is the filter. The operation involves element-wise multiplication between the filter and a region of the input, followed by summing the results to produce a single value in the feature map.

CNN Architecture Comparison

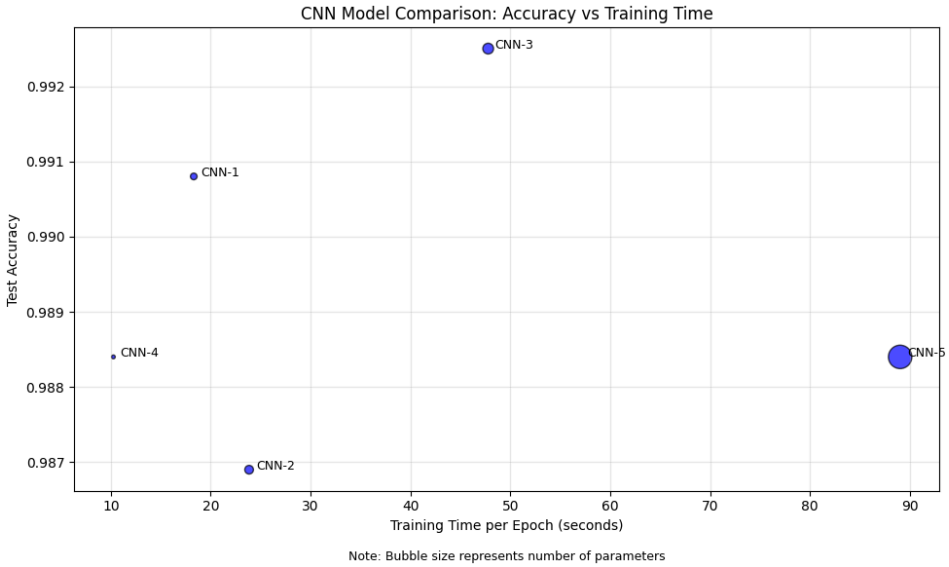
Four additional convolutional neural networks (CNN) were trained with different architectures. Models were compared based on test accuracy, training time, and parameter count. The table of results can be seen below, along with all 5 CNN architectures test accuracies.

CNN	Architecture	Layers	Parameters	Train Accuracy	Test Accuracy	Training Time (s)
CNN-1	[32, 64, 128]	3	227,306	0.9987	0.9908	183.05
CNN-2	[64, 128]	2	383,178	0.9982	0.9869	238.47
CNN-3	[128, 128, 128]	3	589,450	0.9984	0.9925	477.84
CNN-4	[16, 32]	2	71,226	0.9989	0.9884	102.57
CNN-5	[32, 64, 128, 256, 512]	5	2,807,274	0.9963	0.9884	890.32



The comparison of five convolutional neural network (CNN) architectures reveals distinct trade offs between model depth, parameters, and test accuracy. CNN-3, with a depth of three layers [128, 128, 128] and ~600k parameters, achieves the highest test accuracy of 0.9925, striking an optimal balance between complexity and performance.

The scatter plot reinforces CNN-3's efficiency, showing high accuracy with moderate training time, while CNN5 lags with prolonged training. Thus, CNN-3 excels in balancing depth, parameters, and accuracy, underscoring that deeper, parameter heavy models do not always enhance performance, and optimal design is key.



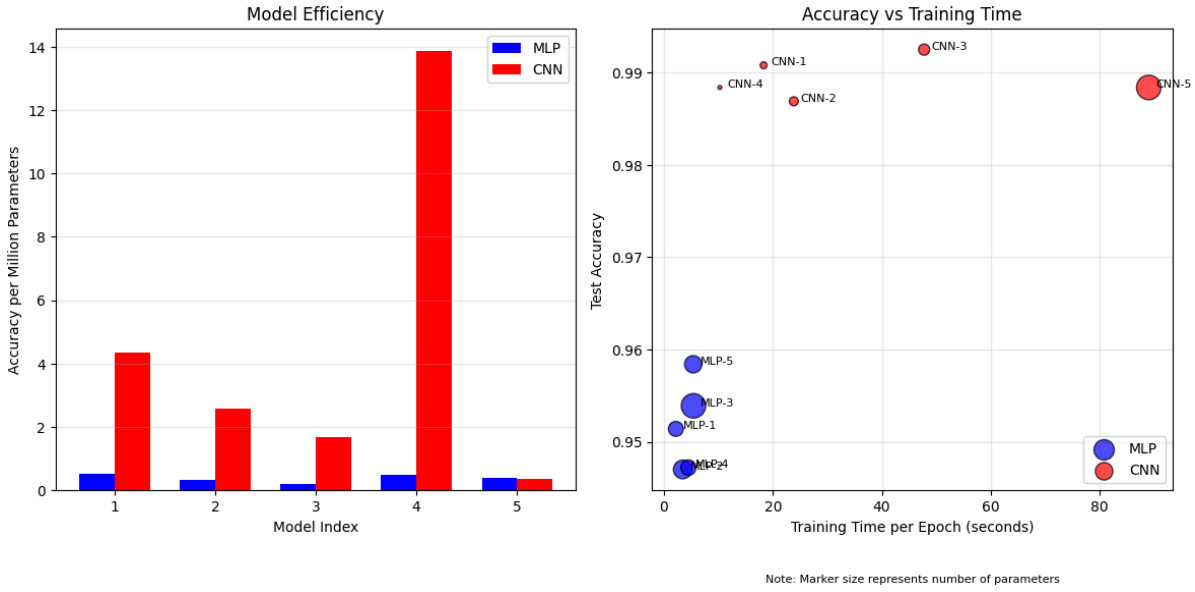
In contrast, CNN5, the deepest model with five layers [32, 64, 128, 256, 512] and ~2.8 million parameters, yields a lower test accuracy of 0.984. This in fact matches CNN-4, which has only two layers [16, 32] and ~70k parameters. This suggests that excessive depth and parameters, as in CNN-5, may lead to diminishing returns, potentially due to overfitting or computational inefficiency, as evidenced by its 890.32 second training time versus CNN4's 102.57 seconds.

Another interesting observation can be found when comparing CNN-2, with two layers [64, 128] and ~380k parameters, with CNN-5, which also has two layers [16, 32] buta much smaller parameter count of ~70k. It can be seen that CNN-4 archives a stronger test accuracy, indicating raw parameter count does not lead to a higher accuracy.

Comparing the CNNs to MLPs

CNNs demonstrated much stronger accuracies of ~99%, compared to MLPs at approximately ~95%. This is due to their ability to capture spatial hierarchies via convolutional layers. See the results below:

MLP Architecture	MLP Test Accuracy	MLP Parameters	MLP Training Time (s)-MLP	Epochs	Epoch Train Time (s)	MLP Epoch Train Time (s) / 1m param-MLP	CNN Architecture	CNN Test Accuracy	CNN Parameters	CNN Training Time (s)-CNN	Epochs	Epoch Train Time (s)	CNN Epoch Train Time (s) / 1m param-CNN
MLP-1	0.9514	1,796,010	43.66	20	2.1830	1.22	CNN-1	0.9908	227,306	183.05	10	18.305	80.53
MLP-2	0.9470	2,797,010	68.85	20	3.4425	1.23	CNN-2	0.9869	383,178	238.47	10	23.847	62.23
MLP-3	0.9539	4,799,010	108.42	20	5.4210	1.13	CNN-3	0.9925	589,450	477.84	10	47.784	81.07
MLP-4	0.9472	1,900,510	89.85	20	4.4925	2.36	CNN-4	0.9884	71,226	102.57	10	10.257	144.01
MLP-5	0.9584	2,401,510	107.40	20	5.3700	2.24	CNN-5	0.9884	2,807,274	890.32	10	89.032	31.71

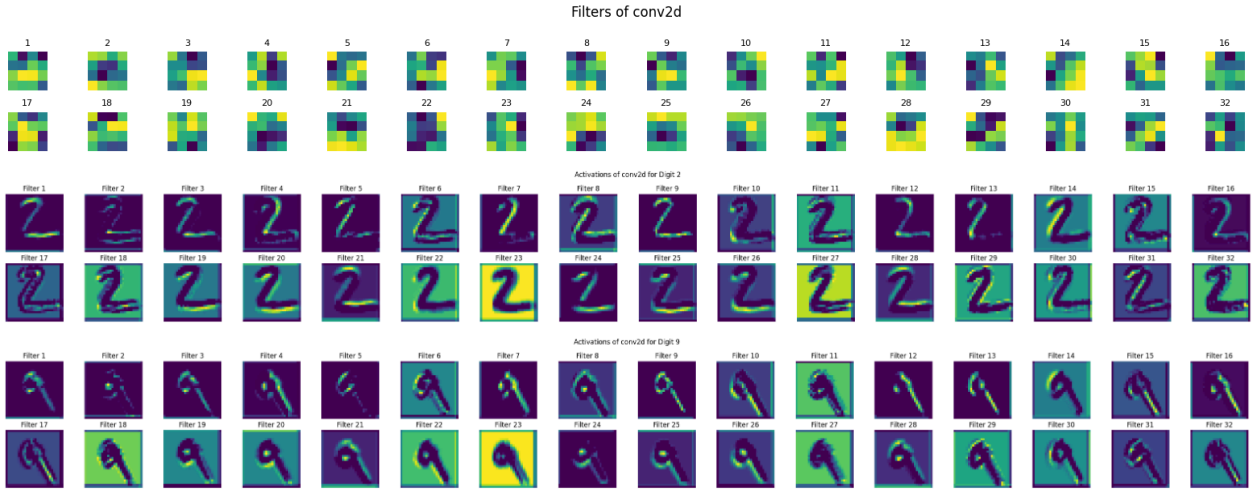


The y-axis on the plot of the left shows the accuracy of a model per 1 million parameters. We can see that CNNs dominate on a per parameter basis. However, this performance advantage of CNNs comes with increased computational demands. If you look at the x-axis on the right-hand plot, you can see that all CNNs Time per epoch count is higher (10.257 to 80.53 seconds) than all of the various MLP architectures (1.22 to 2.36 seconds), relating to the lower parameter count (~1.8 - 2.4 million). This reflects the computational intensity of convolutional operations.

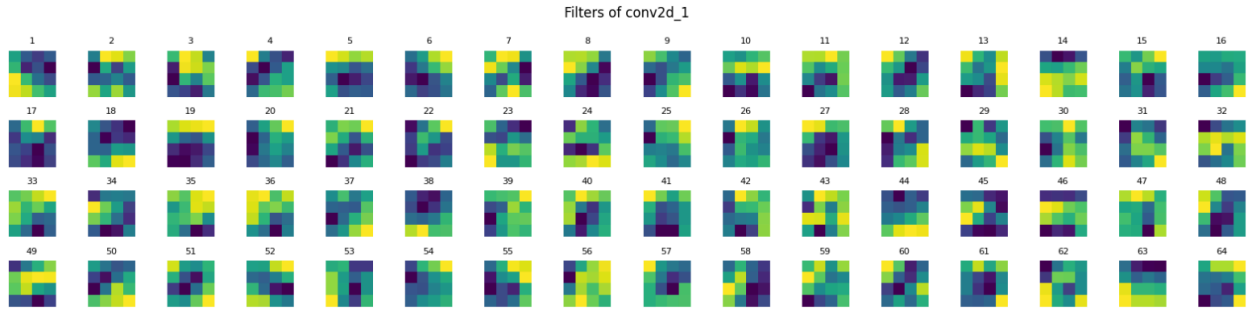
In conclusion, CNNs offer superior accuracy for spatial tasks, justifying their use despite greater parameter counts and longer training times, whereas MLPs, being simpler and quicker to train, suit less complex tasks or resource-limited settings. The choice depends on balancing accuracy needs with computational constraints.

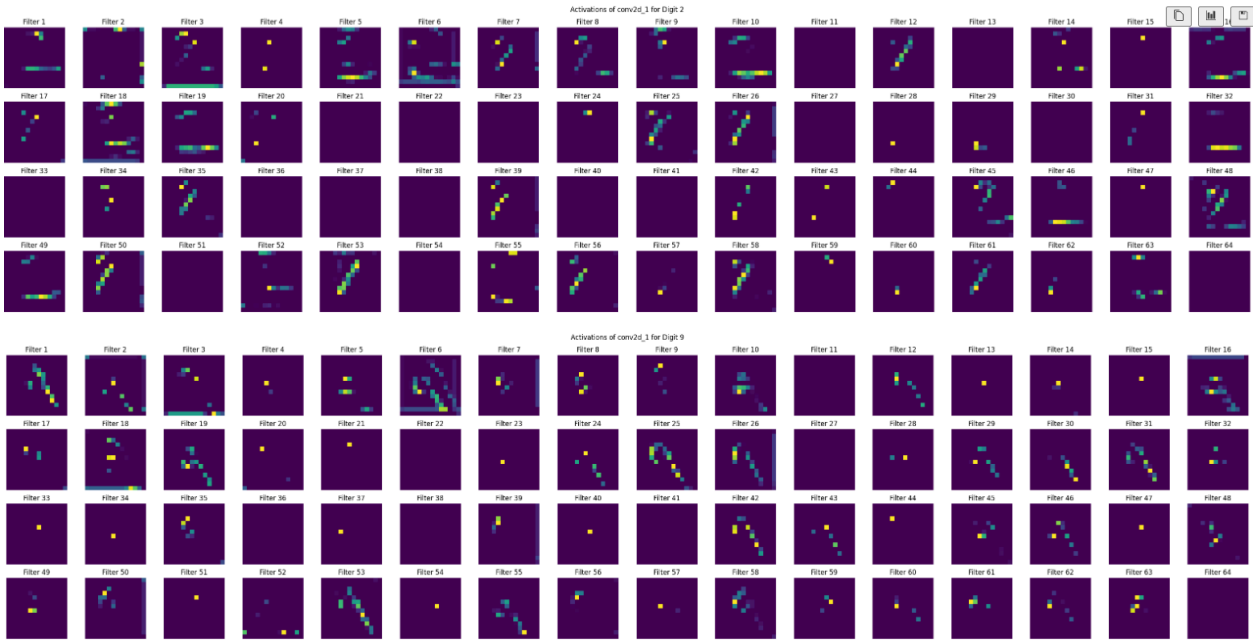
Visualising CNN Outcomes (Task 5)

Peering into the filters of the network, the different layers show the evolution of how the model “sees” the digits and understands them. The first convolutional layer applies 32 filters, each will interact with the image in a unique way to extract information. The activation function of this second layer also indicates that. For example, from the activation of filter 7, we can see that this filter has an affinity to detect vertical edges, while the activation of filter 24 shows that this filter has an affinity to detect horizontal edges. It is very difficult to interpret what the filters will do and how they will interact with the image without the activation graph.

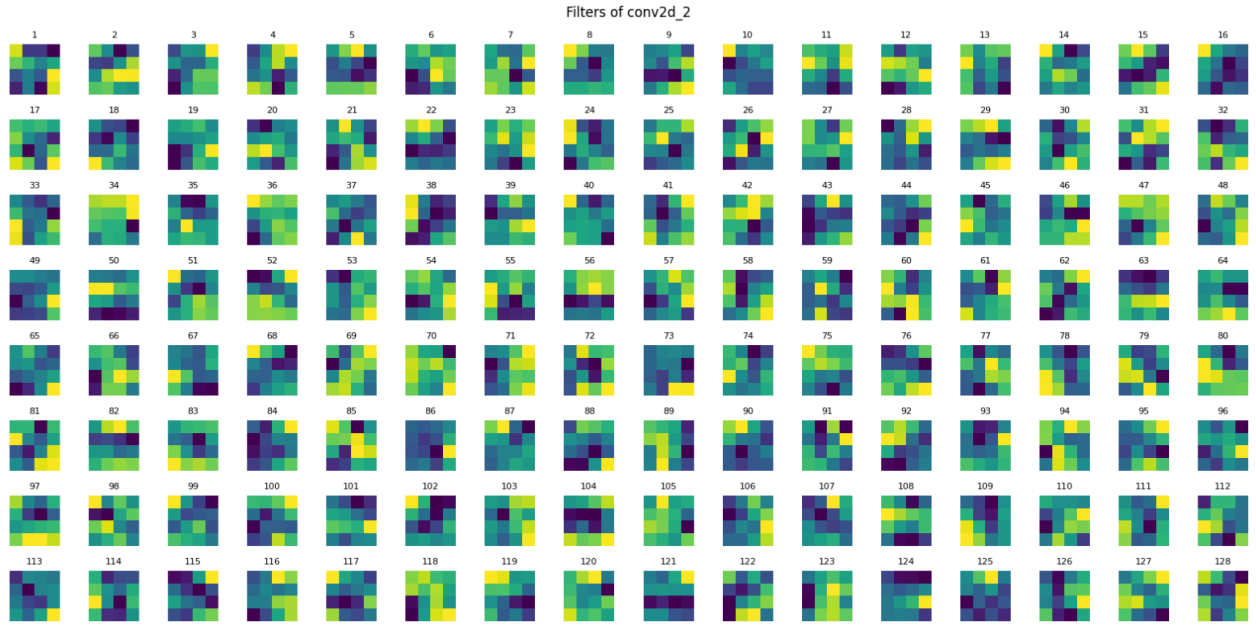


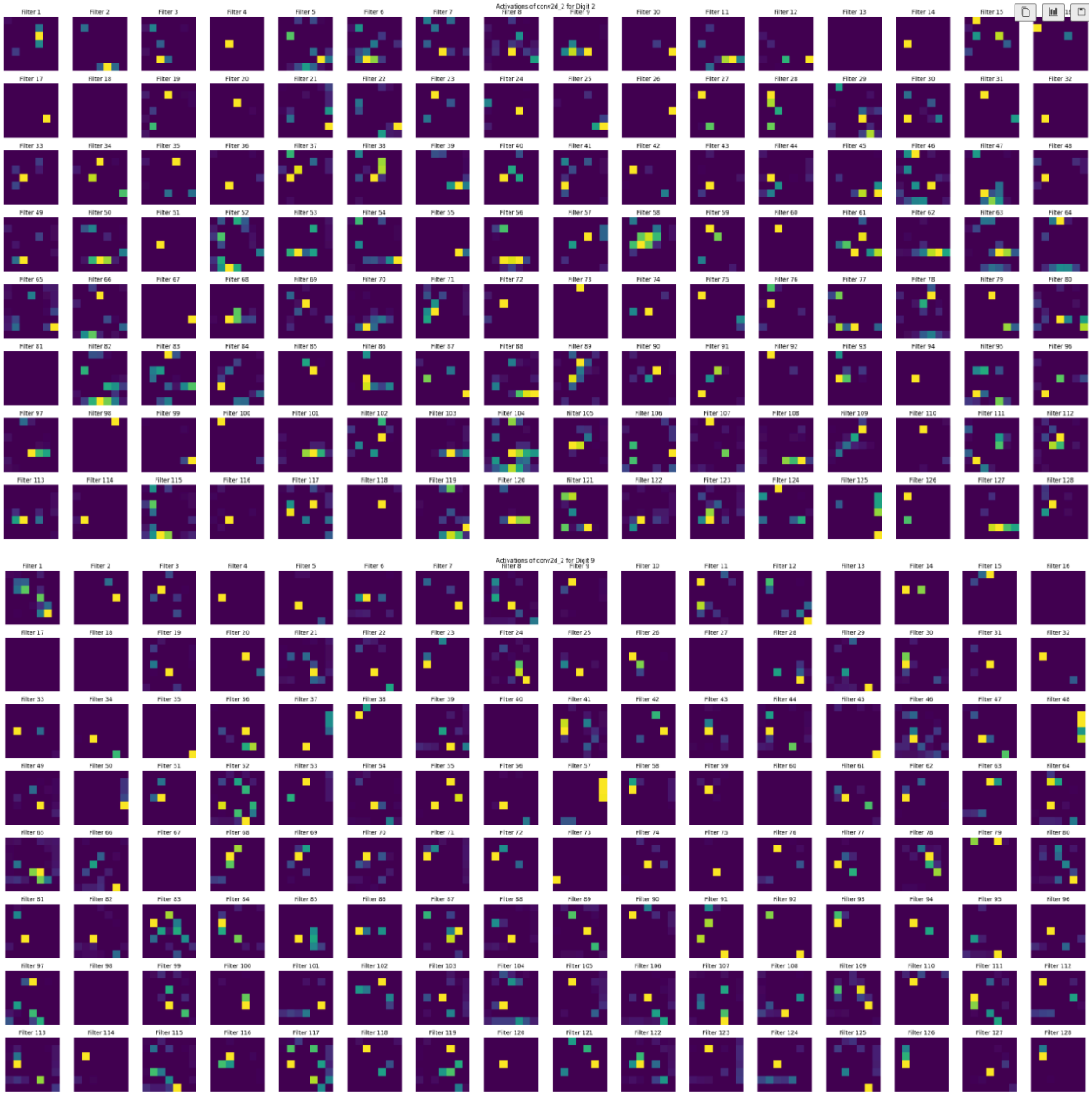
In the second convolutional layer, we have 64 filters as expected and the activation graphs show how these filters are now more specialised and able to capture many aspects of the digits 2 and 9 in these cases. For example, focusing on filter 48, the activation graph shows that this filter is able to see the 2 digit almost entirely, capturing the top curve, the diagonal and the bottom horizontal line. Similarly, filter 19 captures the features of digit 9 almost entirely. Other filters likefilter 39 focus on the parallel diagonals formed by part of the top curve and the diagonal body of the digit 2, or filter 58, which focuses on the loop in the digit 9. Some filters do not activate as they might specialise in finding characteristics not found in digit two and thus are off, like filter 11 or 27.





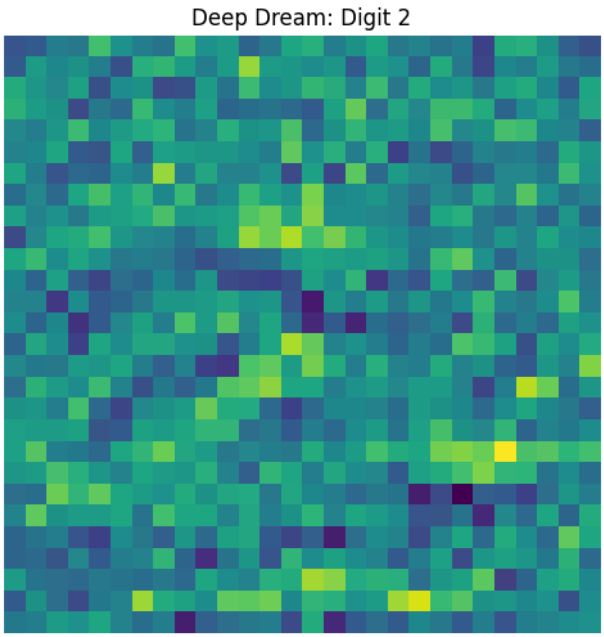
The final convolutional layer is the most specialised and most difficult to interpret as a human. Some filters, like filter 58 for digit 2 or 83 for digit 9, show a complex activation pattern indicating they might be tuned to detect features in the digits 2 and 9 respectively. Others, showing mostly dark pixels, indicate they are not activated by the features in digits 2 and 9 which indicates this third layer is very specialised.

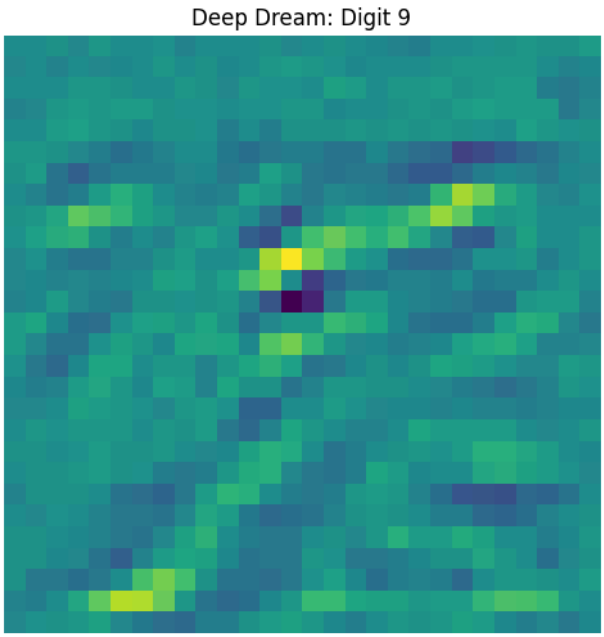




The deep dream images generated for digits 2 and 9 provide interesting insights into the features the model has learned to associate with these specific categories. For digit 2, the deep dream image reveals curved patterns at the top, followed by a diagonal structure transitioning into a horizontal element at the bottom. These features directly correspond to the characteristic shape of a handwritten 2. The image accentuates these components, exaggerating the curvature and horizontal elements to the point where they appear multiple times or in a repeating pattern throughout the image. This amplification demonstrates the model's strong sensitivity to these particular features when classifying a digit as 2. Similarly, for digit 9, the deep dream image prominently features a circular or loop-like structure at the top connected to a vertical line extending downward. This directly mirrors the defining structure of the digit 9.

The differences between these two deep dream images highlight the model's learned discriminative features. While both digits 2 and 9 share some curved elements, the deep dream images emphasise their distinct spatial arrangements. The learning rate and iteration count was manually tweaked to gain a deep dream visualisation. This is evidence that the CNN model was trained, and has learned meaningful, class-specific representations rather than memorising the training examples.





Multi-task learning - Fashion MNIST (Task 6)

In this analysis, we evaluated the performance of Multi-Task Learning (MTL) models designed to handle two tasks, (Item Classification and Group Classification), across five different values of the hyperparameter λ: 0.0, 0.25, 0.5, 0.75, and 1.0. The parameter λ controls the balance between the losses of the two tasks (10-class item classification and 3-class group classification) in the MTL framework, where the total loss is defined as:

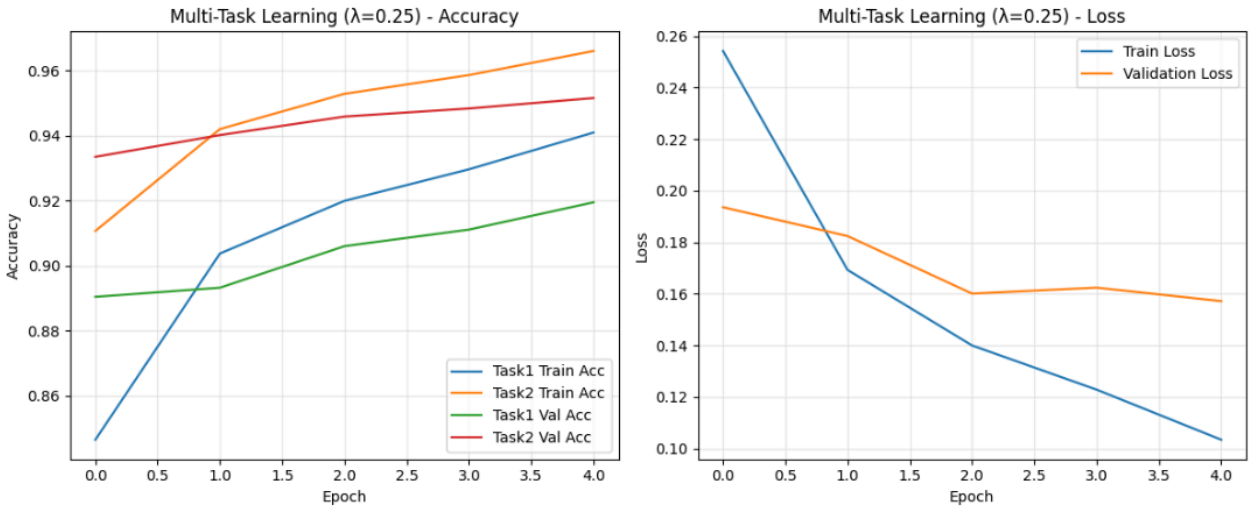
$$\textbf{Total Loss} = \lambda \cdot Loss_{Task1} + (1 - \lambda) \cdot Loss_{Task2}$$

This means that when $\lambda = 0$ all the focus will be on Task 2, and all the updates to the network will optimise the results to improve the classification for Task 2. Conversely, $\lambda = 1$ will focus on Task 1 only, and all the updates to the model will improve the results for Task 1. We also trained two single-task models: one optimised solely for Task 1 and another for Task 2, to have a benchmark to compare our MTL to. The results table can be seen in the table below:

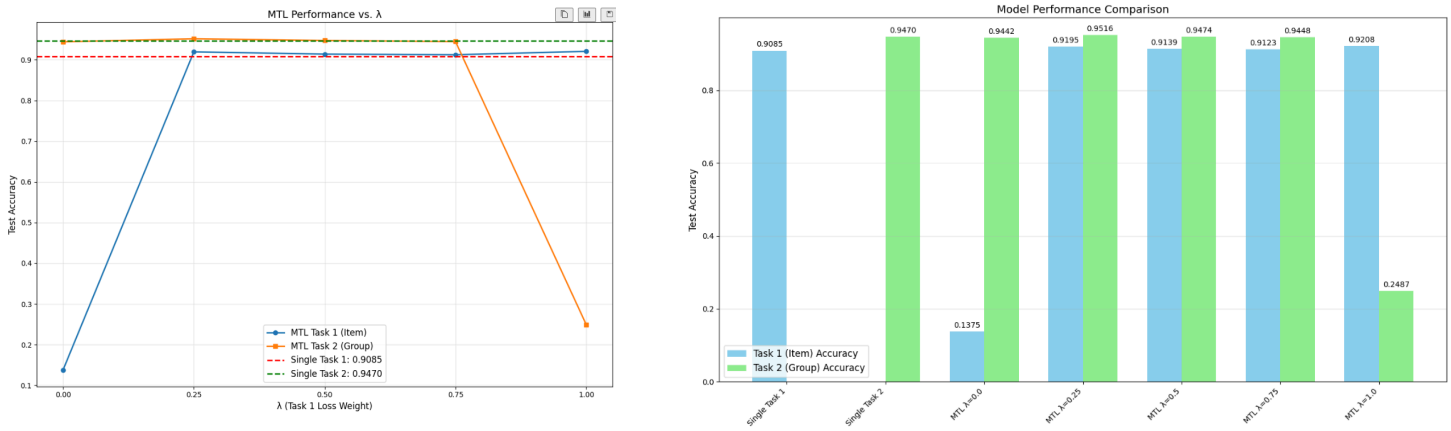
Model	Task 1 Accuracy	Task 2 Accuracy	Parameters	Training Time (s)
Single Task 1	0.9085	0.0000	23,080,598	308
Single Task 2	0.0000	0.9470	23,079,891	314
Single Total	0.0000	0.0000	46,160,489	623
MTL λ=0.0	0.1375	0.9442	26,395,689	334
MTL λ=0.25	0.9195	0.9516	26,395,689	341
MTL λ=0.5	0.9139	0.9474	26,395,689	337
MTL λ=0.75	0.9123	0.9448	26,395,689	339
MTL λ=1.0	0.9208	0.2487	26,395,689	338

Looking at the single focus models first, both models achieved a high accuracy of 90.85% for Task 1 and 94.70% for Task 2. What is important to note is that if the objective is to be able to run both tasks then the combined single tasks need over 46M parameters and just over 10 minutes of training.

The MTL optimises the process by reducing the total number of parameters and training by achieving similarly high accuracy results. As previously stated, both $\lambda = 0$ and $\lambda = 1$ cases are undesirable as they do not utilise the Multi-task nature of this architecture. Both models get high accuracy but need more space for parameters and require longer to train.



The graph below shows the interesting information about the performance of the MTL for λ values between 0 and 1. All the models seem to have similar properties but the best performing one is when $\lambda = 0.25$. This model reaches a testing accuracy of 91.95% for Task 1 and 95.16% for Task 2, better than both single task focus models. Moreover, the 26M parameters needed and 5.5 minute training time means we have achieved our goal with about half the memory space (44% improvement) and half the time (46% improvement).



Liu (2022) introduces a novel framework called Auto-Lambda, which aims to dynamically learn and adjust the relationships between tasks in a multi-task learning setting.This allows the model to adaptively balance the influence of each task, leading to improved generalisation and performance across all tasks. Yang (2022) introduces a novel approach to MTL that focuses on dynamically adjusting the learning rates for each task based on their dominance over shared parameters. By measuring the dominance degree of each task on shared parameters through the exponentially decaying average of squared updates (AU) and computing them, they identify parameters overly influenced by specific tasks and adjust their learning rates accordingly. By introducing these improvements and MTL model can be very powerful and efficient.

Finally, for our use case, the MTL performed better than single task models on all objectives: the accuracy achieved is high with a low parameter rate and reasonable training time. Now even if a third task is introduced, it can be taken into account using the following equation:

$$\textbf{Total Loss} = \lambda_1 \cdot Loss_1 + \lambda_2 \cdot Loss_2 + \lambda_3 \cdot Loss_3$$
$$\text{where } \lambda_1 + \lambda_2 + \lambda_3 = 1$$

This means that our pretrained model can now be used to solve more tasks and will require much less space for parameters and time to train than the introduction of a completely new model as we benefit from the transfer learning effect.

But MTL has many limitations as well, beginning with the fact that constructing a good and effective MTL can be very complex. The tasks chosen also need to be somewhat similar as some tasks might interfere with each other causing the model to underperform on both tasks while trying to achieve high accuracy for both (Trottier 2017).

References

- Built In (n.d.) *Step-by-step explanation of principal component analysis*. Available at: https://builtin.com/data-science/step-step-explanation-principal-component-analysis (Accessed: 28 February 2025).
- DataCamp (n.d.) *SVM classification in Scikit-Learn Python tutorial*. Available at: https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python (Accessed: 28 February 2025).
- Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y. and Srebro, N. (2018) 'The role of over-parameterisation in generalisation of neural networks'. Available at: https://openreview.net/pdf?id=BygfghAcYX (Accessed: 28 February 2025).
- Liu, S., James, S., Davison, A.J. and Johns, E. (2022) 'Auto-Lambda: Disentangling Dynamic Task Relationships', *arXiv preprint arXiv:2202.03091*. Available at: https://arxiv.org/abs/2202.03091
- Yang, E., Pan, J., Wang, X., Yu, H., Shen, L., Chen, X., Xiao, L., Jiang, J. and Guo, G. (2022) 'AdaTask: A Task-aware Adaptive Learning Rate Approach to Multi-task Learning', *arXiv preprint arXiv:2211.15055*. Available at: https://arxiv.org/abs/2211.15055
- Trottier, L., Giguère, P. and Chaib-draa, B. (2017) 'Multi-Task Learning by Deep Collaboration and Application in Facial Landmark Detection', *arXiv preprint arXiv:1711.00111*. Available at: https://arxiv.org/abs/1711.00111 (Accessed: 1 March 2025).