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.

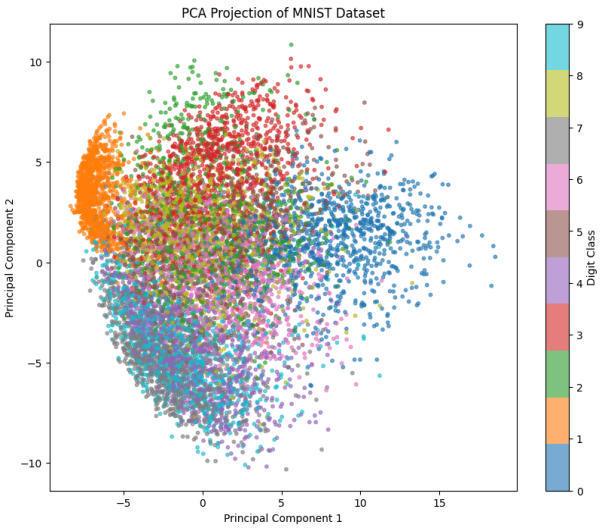
Mathematically, PCA involves computing the covariance matrix of the centered data X:x

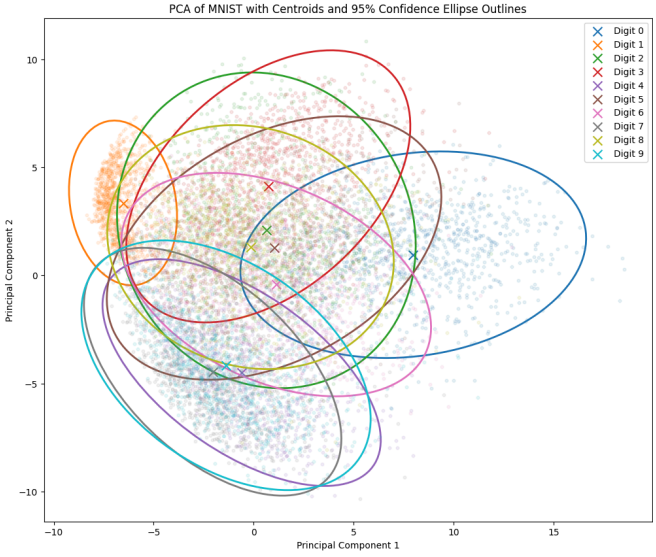
$$Cov(X) = \frac{1}{n-1} X^T X$$

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:

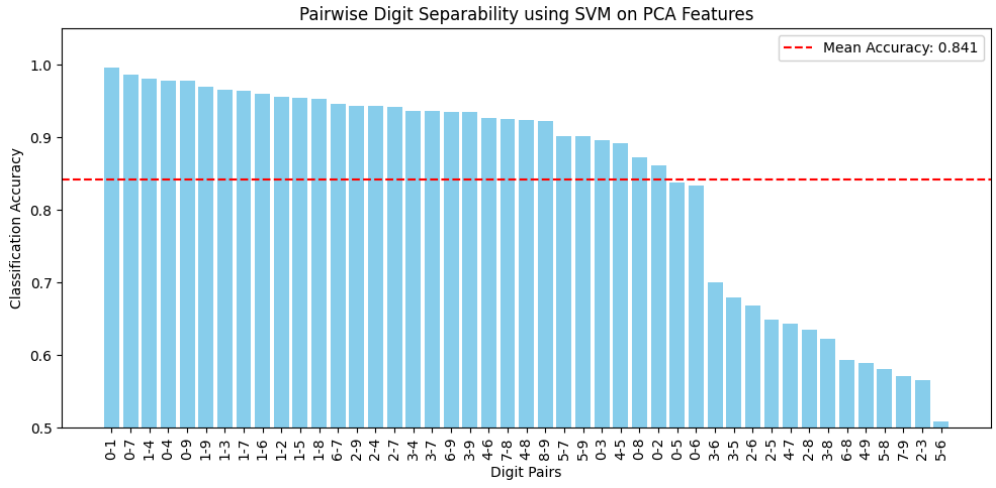
$$X_{PCA} = XV_2$$

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 like (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

The scatter plot visualises these clusters: digit 1 (orange) clusters left (PC1: 5 to 0), digit 0 (blue) right (PC1: 10 to 15), both distinct. Digit 4 (purple) is bottom right, while digits 3 (red), 5 (brown), and 8 (yellow) overlap centrally. The second scatter plot, with centroids and 95% confidence ellipses, confirms digit 1s isolation, digit 7s partial distinction, and dense overlap among digits 4, 5, 9, and others (0, 2, 3, 6, 8) on the right.

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.

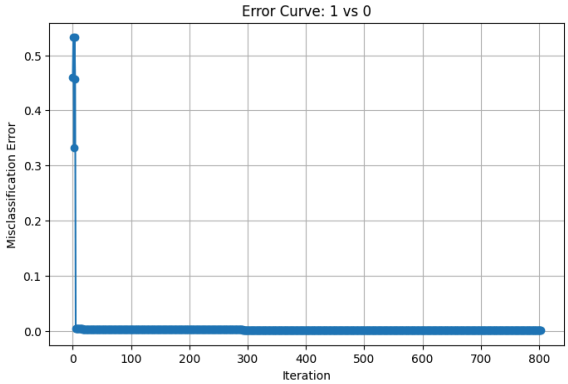
PCA effectively visualises the MNIST dataset by reducing dimensionality and revealing clustering patterns. While digits 1 and 0 are linearly separable, the overlap among digits like 3, 5, and 8 underscores PCA's limitations in fully separating all classes in 2D, suggesting that higher dimensions or non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

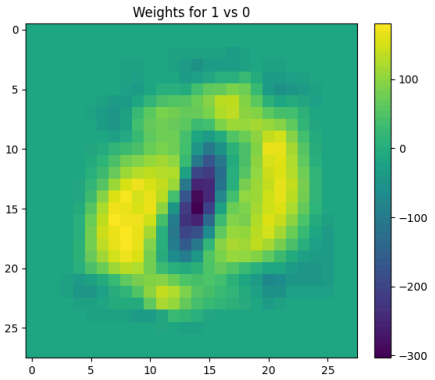
Perceptrons (Task 2)

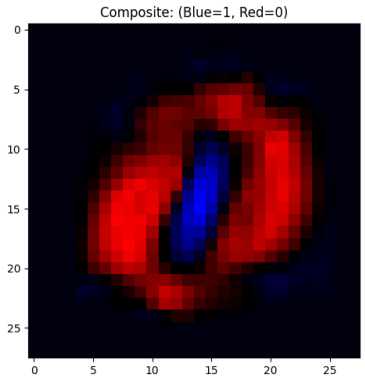
The perceptron algorithm, a fundamental supervised learning technique for binary classification, excels at distinguishing pairs of handwritten digits in the MNIST dataset. Each training image, a 28x28 array of pixel values, is flattened into a high dimensional vector. The perceptron update rule initialises 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:

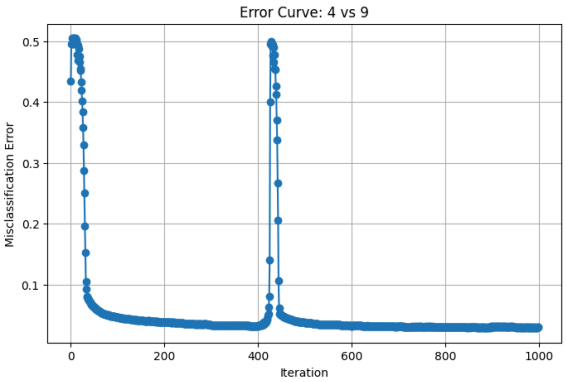
Digit 1 vs 0

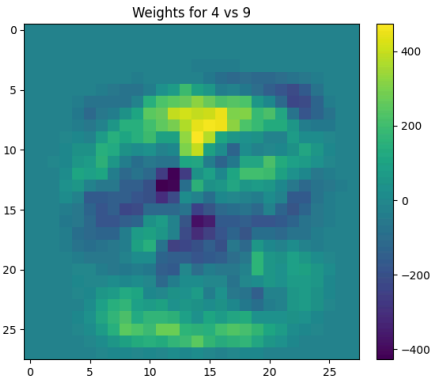


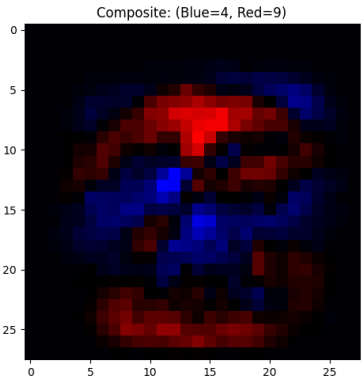




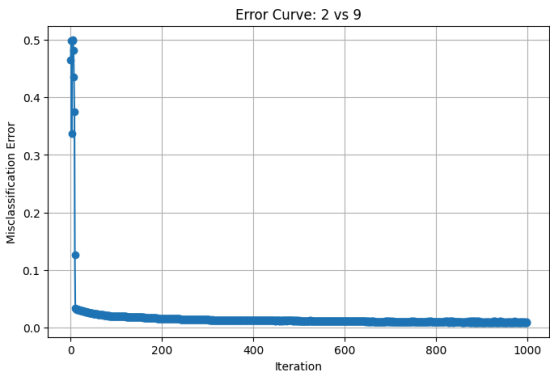
Digit 4 vs 9

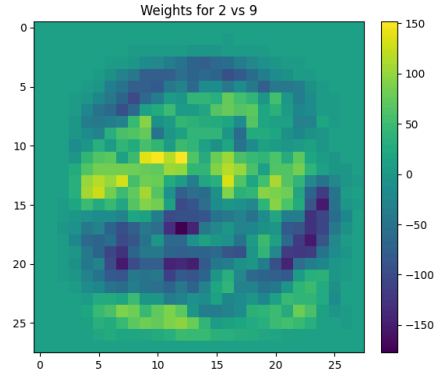


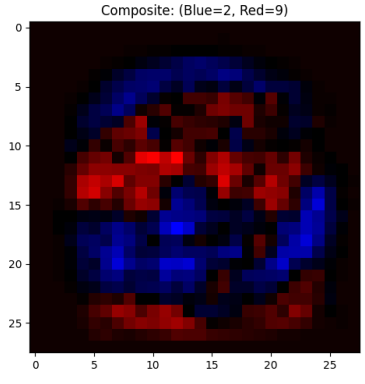




Digit 2 vs 9







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.

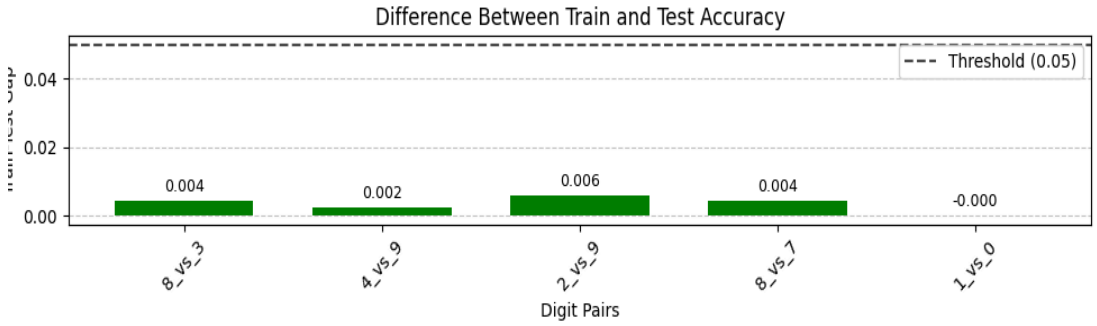
Analysis of the perceptrons results

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.

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.

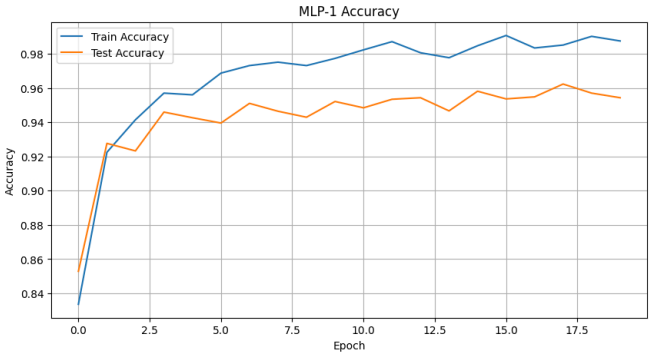


These results highlight key insights. A simple linear model, updated via the perceptron rule, learns meaningful pixel-based features without complex architecture. Pairs with distinct visuals converge faster and score higher, while structural similarities demand more iterations and yield slightly lower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

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

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:

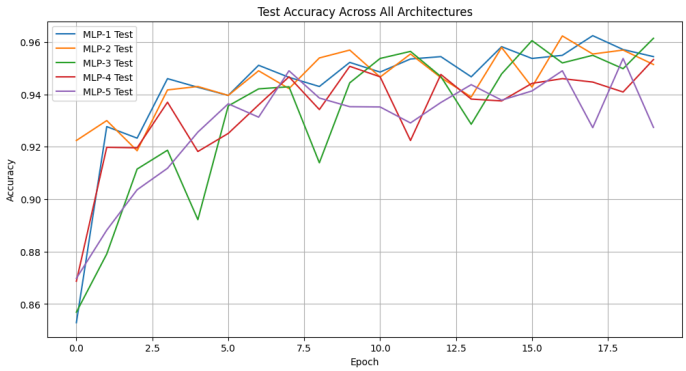
$$a^{(l)} = \sigma(z^{(l)})$$

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

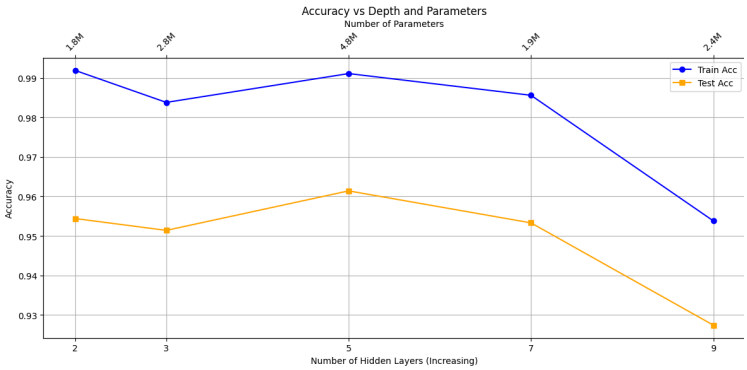
$$y = softmax(z^{(l)})$$

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)

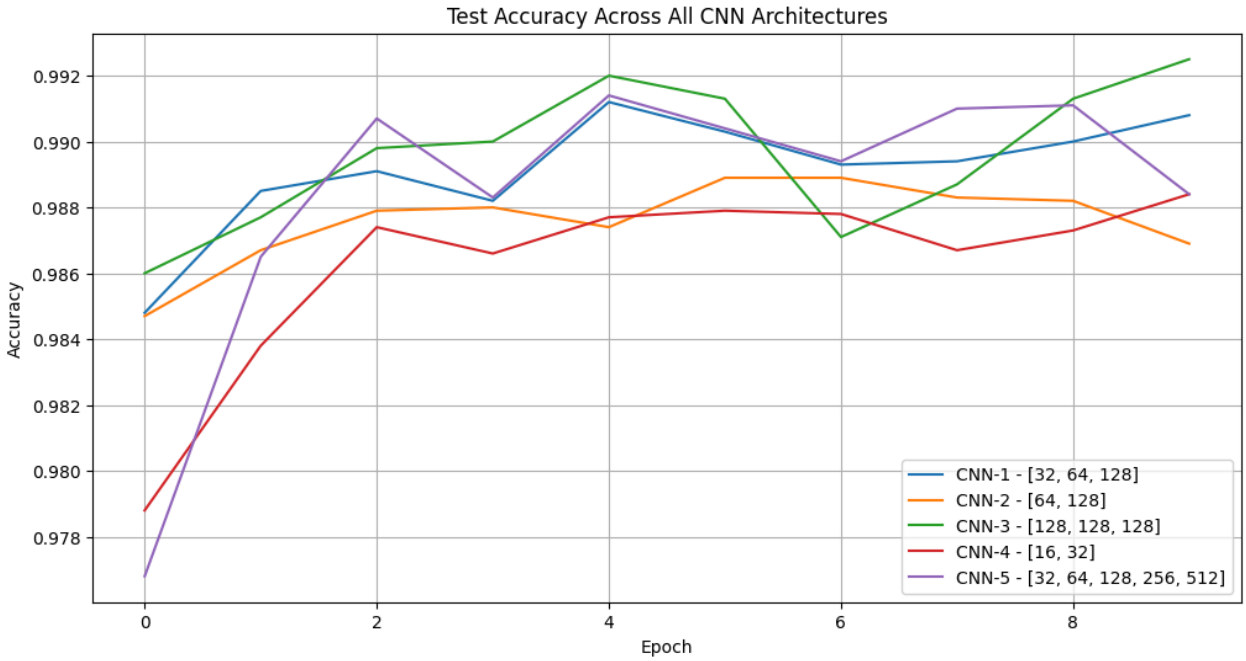
The CNN model is a major step above all MLP architectures that have been used and tested. This is no surprise as the principal application of CNNs is to pool neighboring pixels and extract meaning from them, thus working more like a human brain would rather than a sequential computer.

Base CNN Training [32, 64, 128]
The base recommended CNN-1 is constructed using 3 convolution layers followed by a classification layer composed of a flat and a dense layer. The three convolutional layers use filters of shape 32, 64 and 128 respectively and the second and third layers use a stride of two to downsize the image and avoid using pooling layers thus reducing the number of parameters.

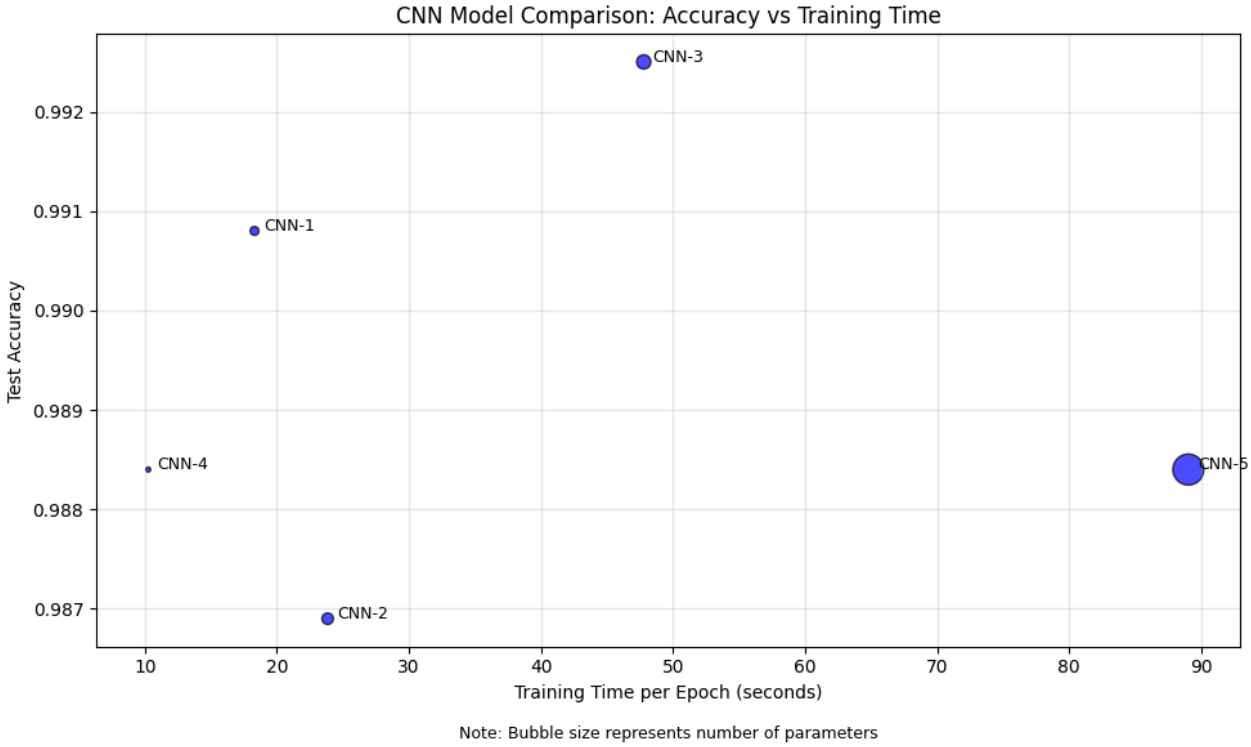
The figure below shows the improvement in accuracy and reduction of loss as the training progresses through the epochs. The training data classification accuracy climbs steadily to reach 99.69% while the testing data classification fluctuates and reaches 98.85%. The CNN-1 model is a major step above all MLP architectures that have been used and tested. This is no surprise as the principal application of CNNs is to pool neighboring pixels and extract meaning from them, thus working more like a human brain would rather than a sequential computer.

CNN Architecture Comparison
To further test this architecture, a set of CNNs with different number and size of layers will be trained on the data. CNN-2 and CNN-4 both have only two convolutional layers. CNN-2 has filters with size 64 and 128 and have almost 1.5 times the number of parameters of our original CNN-1 architecture. With a test accuracy of 98.56%, this model is more complex and less accurate than the original model which is due to the fact that the first convolution layer has a large size filter and not enough layers to capture the complexities that these large filters highlight. CNN-4 on the other hand has filters of size 16 and 32 and have about a third of the parameters compared to the original model. With a test accuracy of 98.66% and a training run time of about a half of the original time, this is a good candidate for applications that require quick training at the cost of some negligible loss of accuracy.

The CNN-3 architecture utilises 3 convolution layers with 3 filters of size 256. Although the final testing accuracy sits at 99.12%, the ten fold number of parameters and eightfold training time makes it a very slow and inefficient model compared to the base CNN-1, but that is a price to pay to reach this high level of accuracy. Another possibility to reach a higher accuracy is to make the CNN deeper by adding more convolution layers like for CNN-5 with 5 convolutional layers and filters of size 32, 64, 128, 256 and 512. With the highest accuracy of 99.19%, the CNN-5 has the most parameters of all architectures, twelve times more than the base model, but requires only five times the training time.

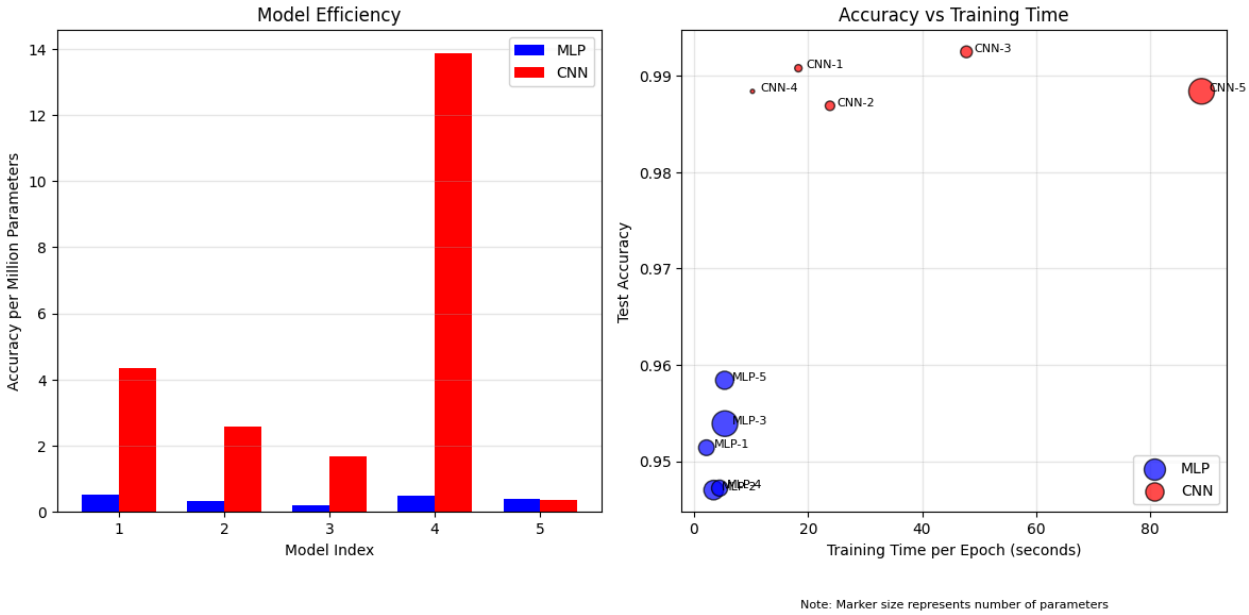


CNN	Architecture	Layers	Parameters	Train Accuracy	Test Accuracy	Training Time (s)
CNN-1	[32, 64, 128]	3	227,306	0.9987	0.9908	183.05
CNN-2	[64, 128]	2	383,178	0.9982	0.9869	238.47
CNN-3	[128, 128, 128]	3	589,450	0.9984	0.9925	477.84
CNN-4	[16, 32]	2	71,226	0.9989	0.9884	102.57
CNN-5	[32, 64, 128, 256,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



One important aspect to note is that as the models train, their accuracy varies and does not steadily increase. Although the CNN-5 architecture got the highest accuracy by the end of the training, it is clear that both CNN-1 and CNN-3 got to similar heights at earlier epochs. If known, a target accuracy can be set and training can be stopped earlier saving time. That said, the test accuracy is only a part of the full picture and although important, the model might not perform as expected on real data and live data depending on the project.

Comparing the CNNs to MLPs

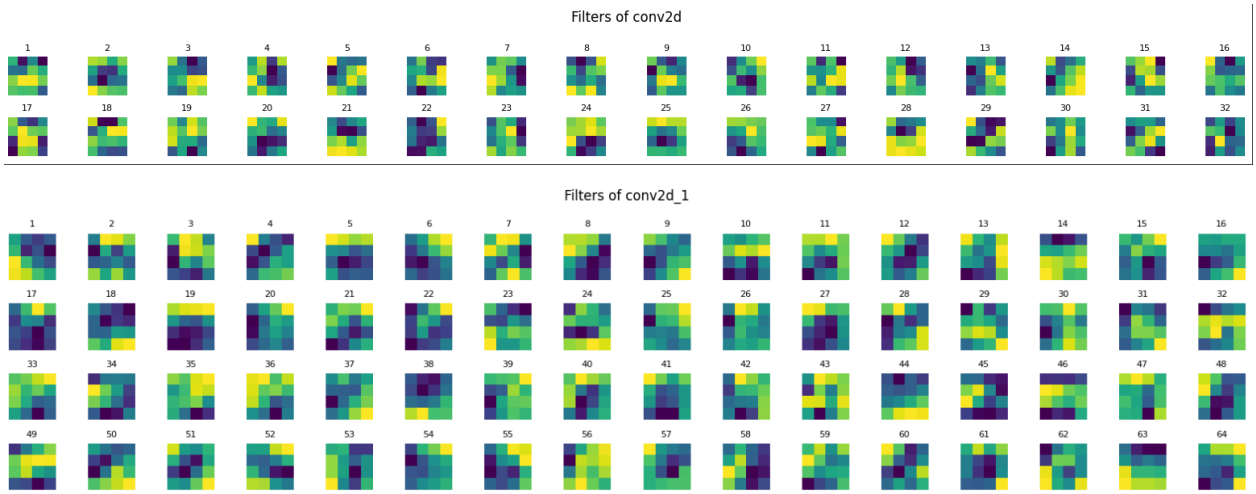


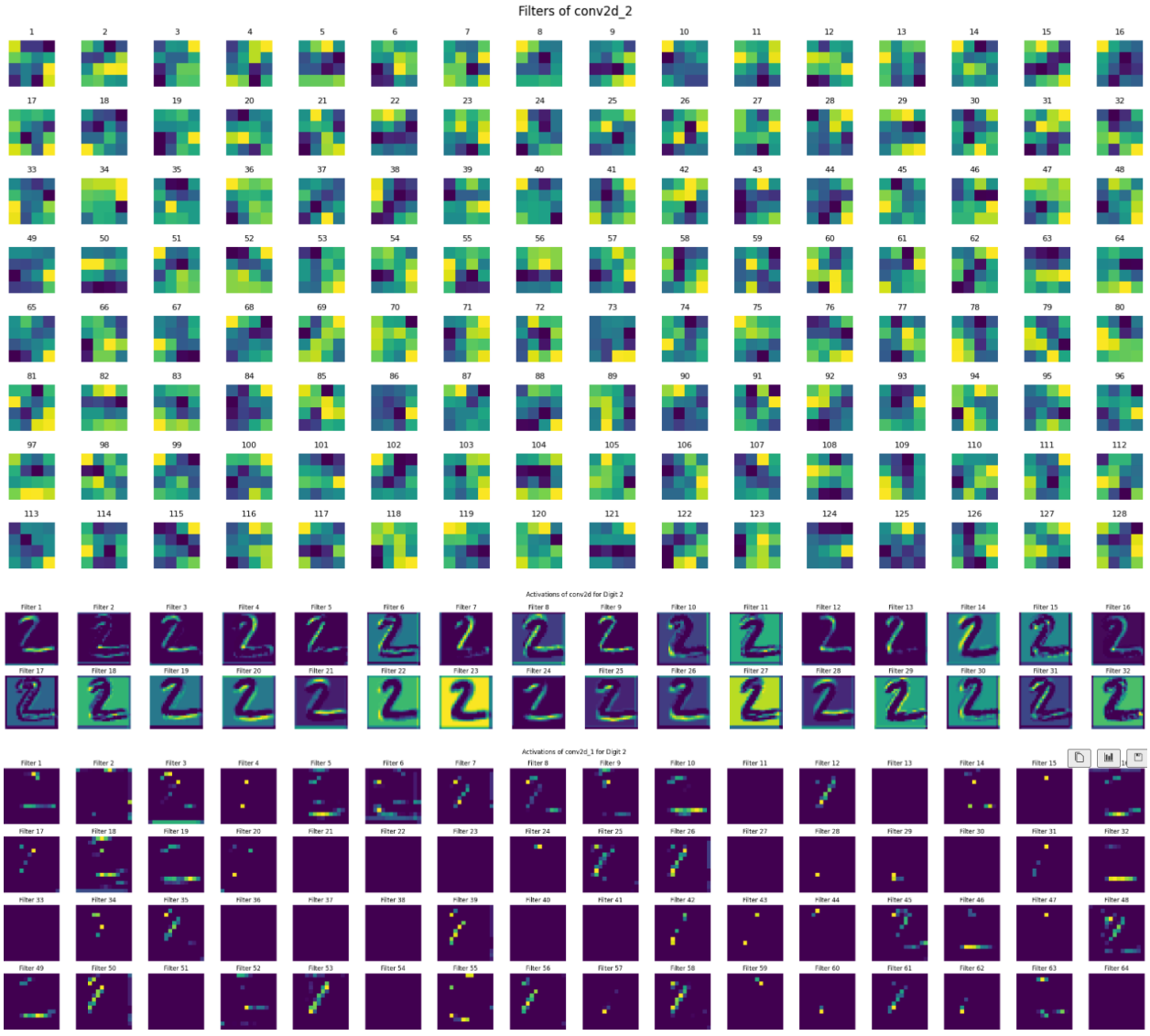
MLP Architecture	MLP Test Accuracy	MLP Parameters	MLP Training Time (s)-MLP	Epochs	Epoch Train Time (s)	MLP Epoch Train Time (s) / 1m param-MLP	CNN Architecture	CNN Test Accuracy	CNN Parameters	CNN Training Time (s)-CNN	Epochs	Epoch Train Time (s)	CNN Epoch Train Time (s) / 1m param-CNN
MLP-1	0.9514	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

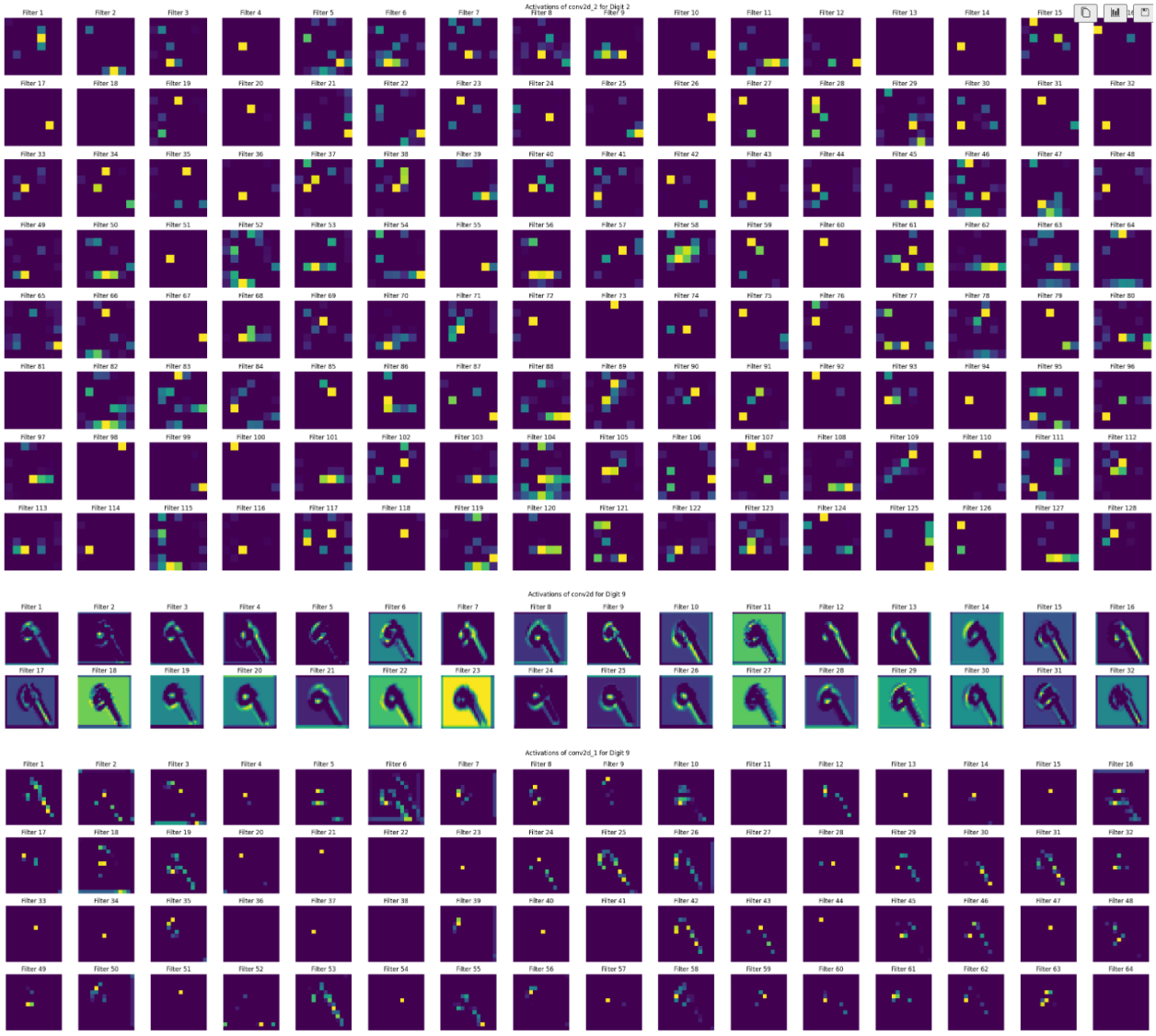
Visualising CNN Outcomes (Task 5)

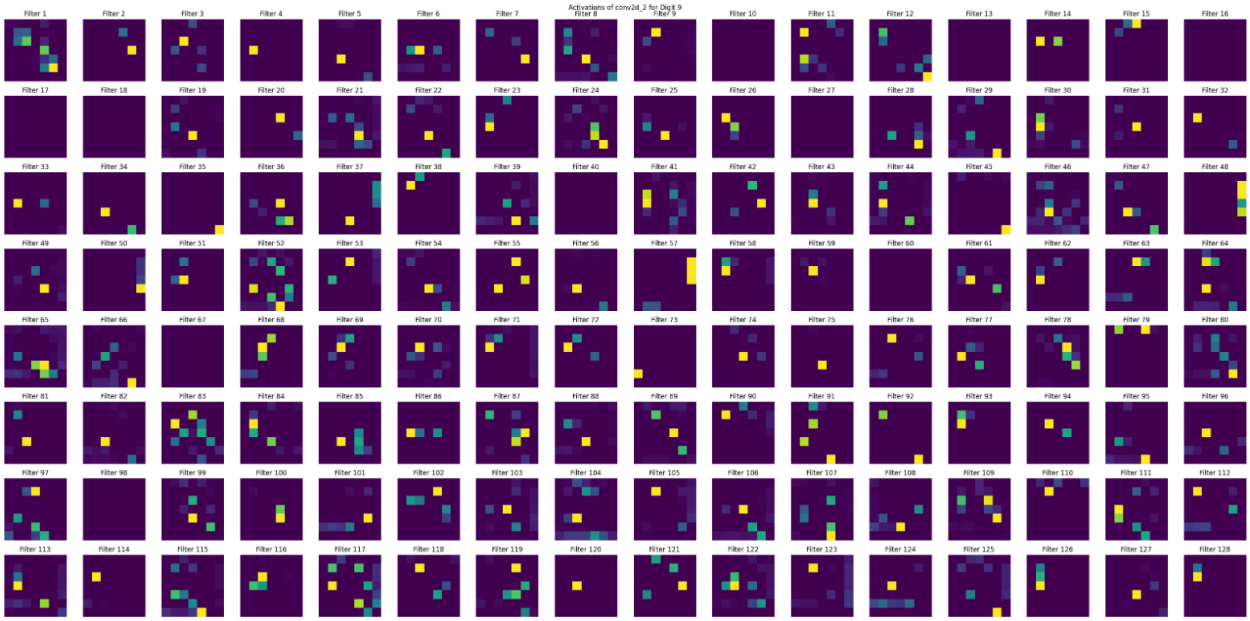
- Once your CNN (in Question 4) is trained, access its filters via, e.g.‘tf.get_collection’ and plot them on a grid for each layer.

- What pat-terns do you observe, and why?
- In addition, plot the activations of each layer (i.e., the output of each conv layer in your CNN) for two images chosen from digit-classes ‘2’ and ‘9’. Discuss your observations.
- Create deep dream images for digit classes 2 and 9 and show these in your report. Discuss how each deep dream image shows the model’s sensitivity to the patterns that appear in the input images relative to each output class (category).

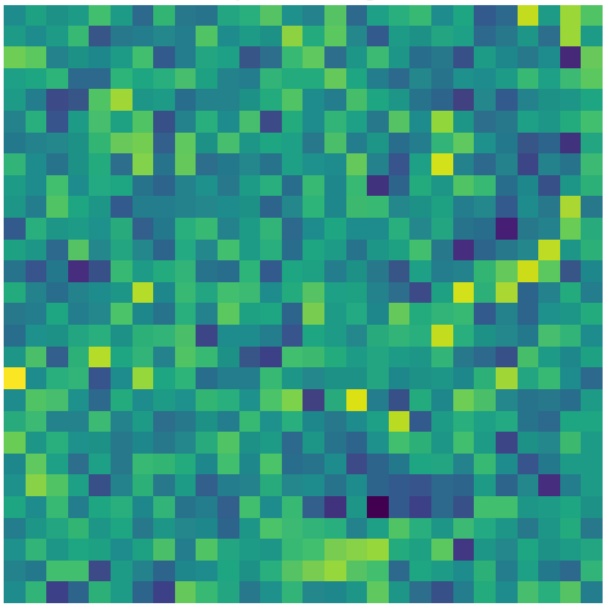



Filters of conv2d_2
Activations of conv2d for Digit 2
Activations of conv2d_1 for Digit 2

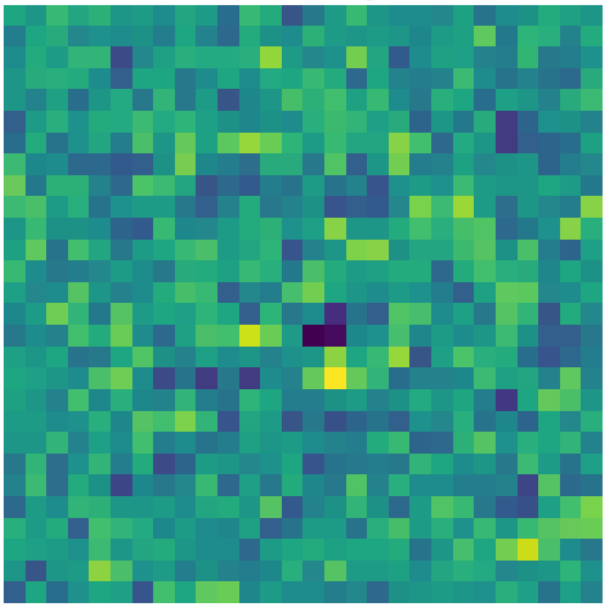

Activations of conv2d_2 for Digit 2
Activations of conv2d for Digit 9
Activations of conv2d_1 for Digit 9



Deep Dream: Digit 2



Deep Dream: Digit 9



Multi-task learning - Fashion MNIST (Task 6)

- Comment on your results and explain what makes the cases of and λ=1 particularly special? λ=0
- Compare the performance of MTL models to single-task networks. Discuss important considerations when using MTL and its pros and cons

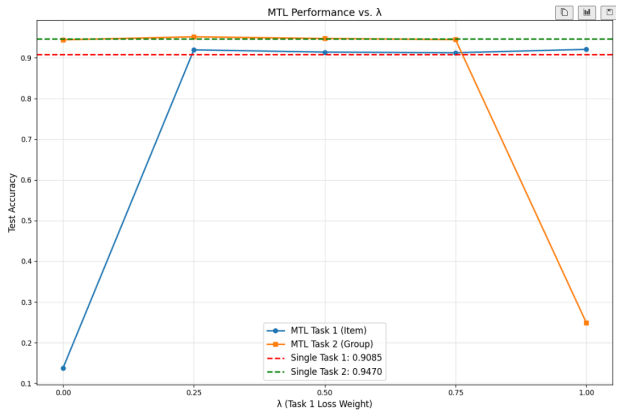
In this analysis, we evaluated the performance of Multi-Task Learning (MTL) models designed to handle two tasks, (Item Classification and Group Classification), across five different values of the hyperparameter λ: 0.0, 0.25, 0.5, 0.75, and 1.0. The parameter λ controls the balance between the losses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

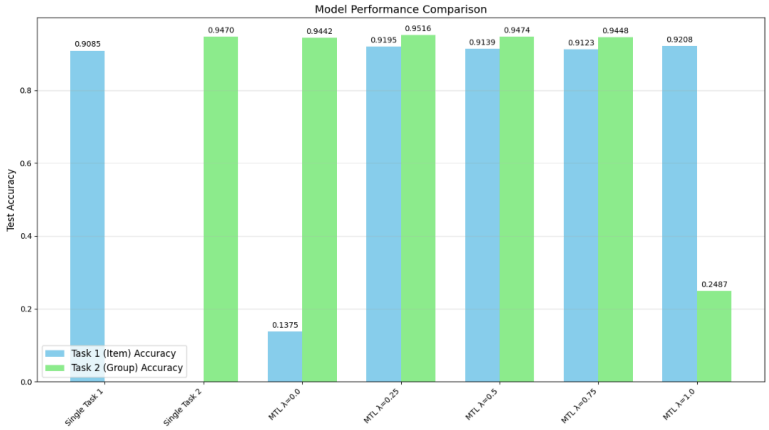
$$\text{Total Loss} = \lambda \cdot \text{Loss}_{\text{Task1}} + (1 - \lambda) \cdot \text{Loss}_{\text{Task2}}$$

We also trained two single-task models: one optimised solely for Task 1 and another for Task 2, the results table can be seen 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

Coupled with plots of the accuracies across all the training runs:





MTL Performance comparison to single-task networks

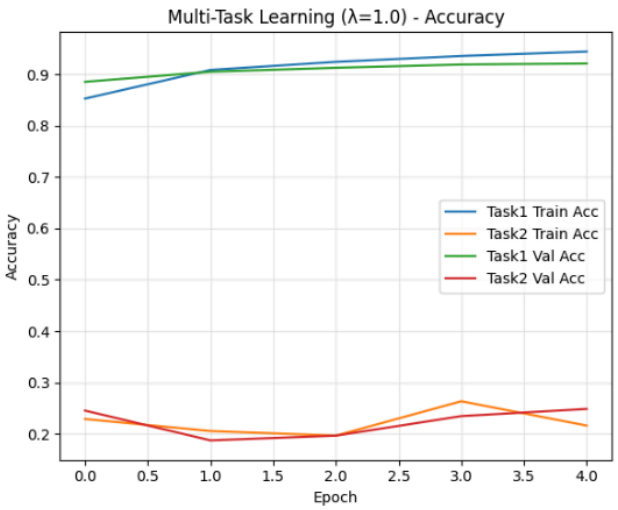
λ = 0 Analysis

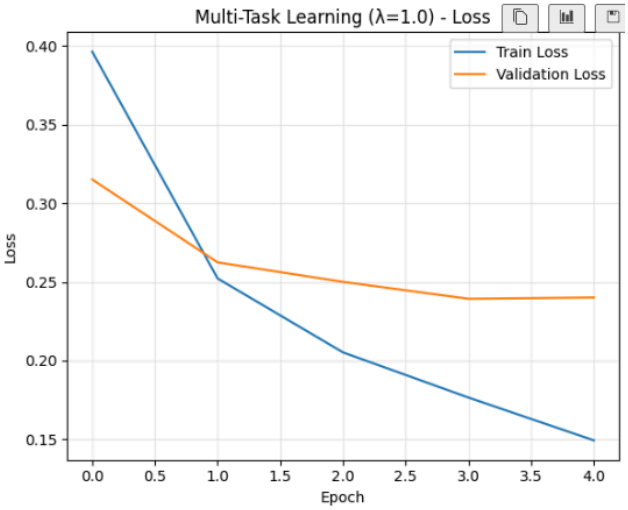
λ = 1 Analysis

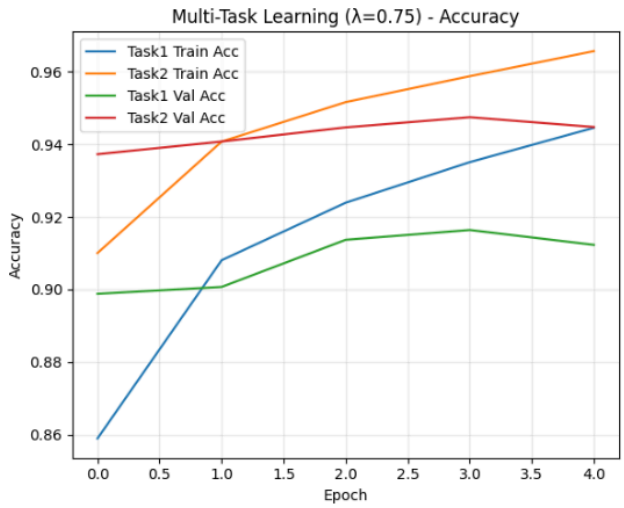
λ = (0.25, 0.5, 0.75) Analysis

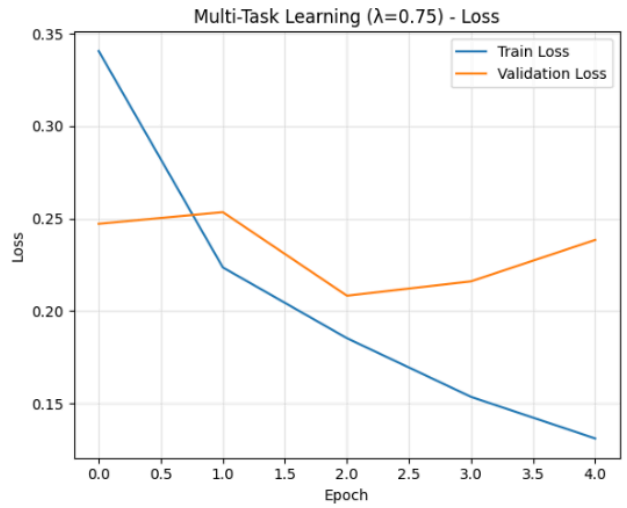
Important Considerations, Pros, and Cons of MTL

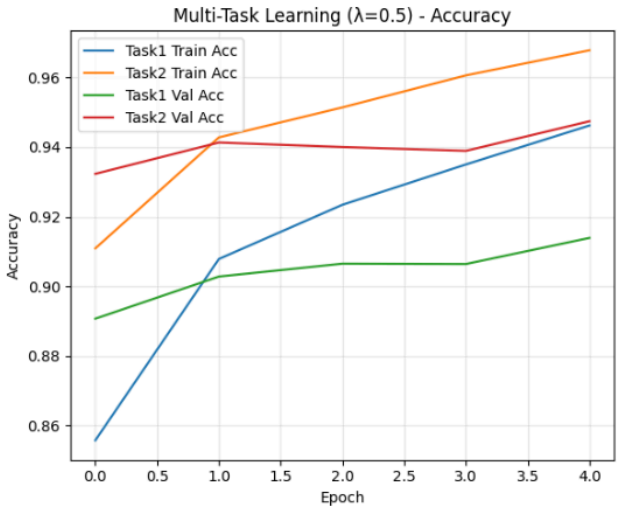
- Parameter Savings with MTL: 19,764,800 (42.82%)
- Best MTL Avg Accuracy (λ=0.25): 0.9356 vs. Single Avg: 0.9278
- MTL Improves Both Tasks Simultaneously: Yes

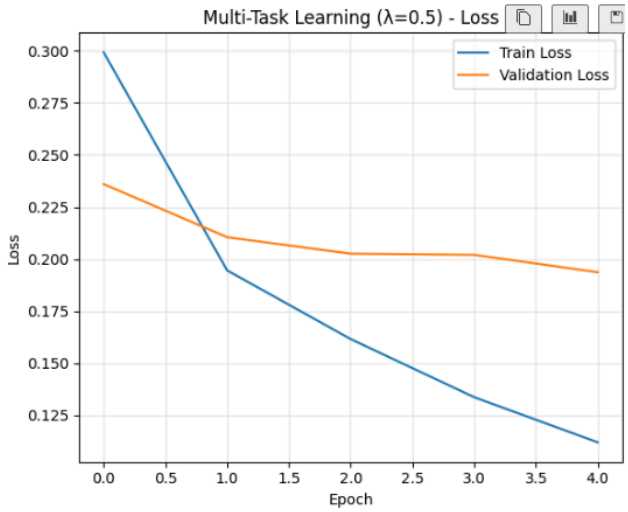

Multi-Task Learning (λ=1.0) - Accuracy
Task1 Train Acc
Task2 Train Acc
Task1 Val Acc
Task2 Val Acc
Accuracy
Epoch

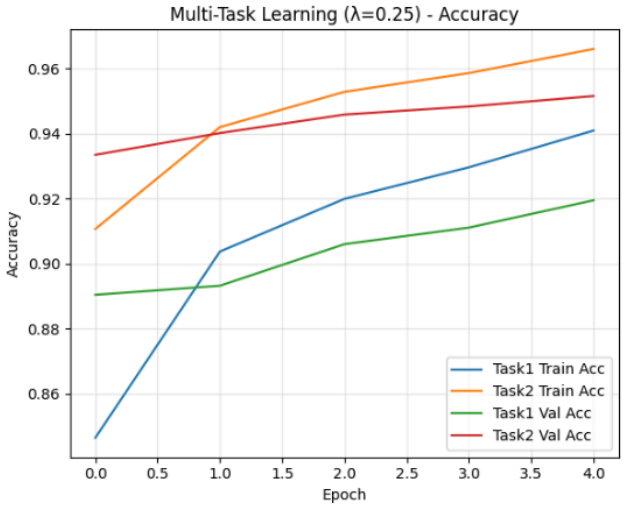

Multi-Task Learning (λ=1.0) - Loss
Train Loss
Validation Loss
Loss
Epoch

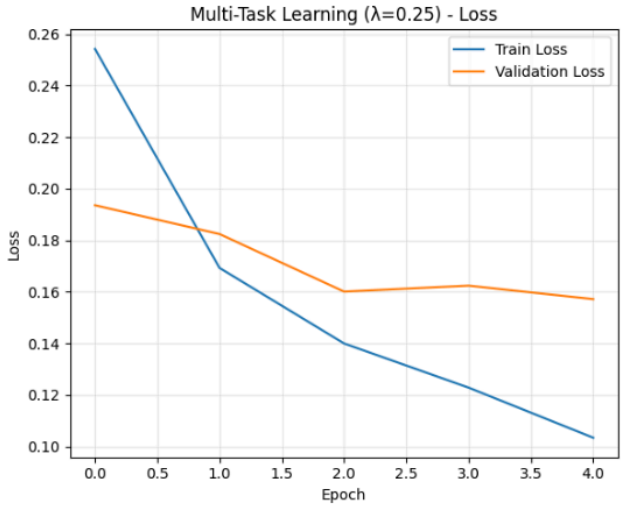

Multi-Task Learning (λ=0.75) - Accuracy
Task1 Train Acc
Task2 Train Acc
Task1 Val Acc
Task2 Val Acc
Accuracy
Epoch

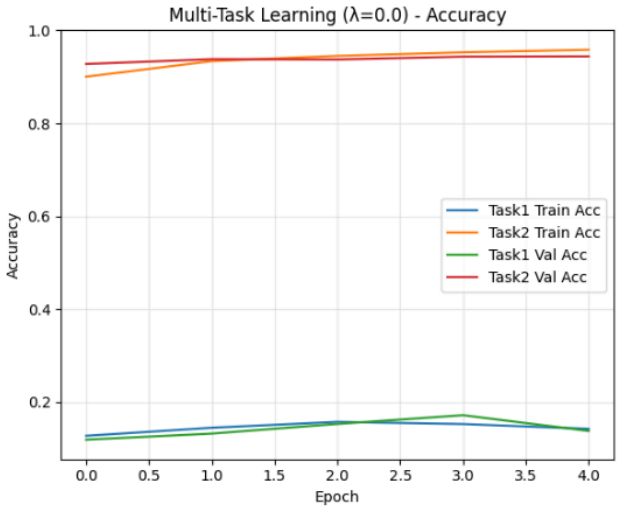

Multi-Task Learning (λ=0.75) - Loss
Train Loss
Validation Loss
Loss
Epoch

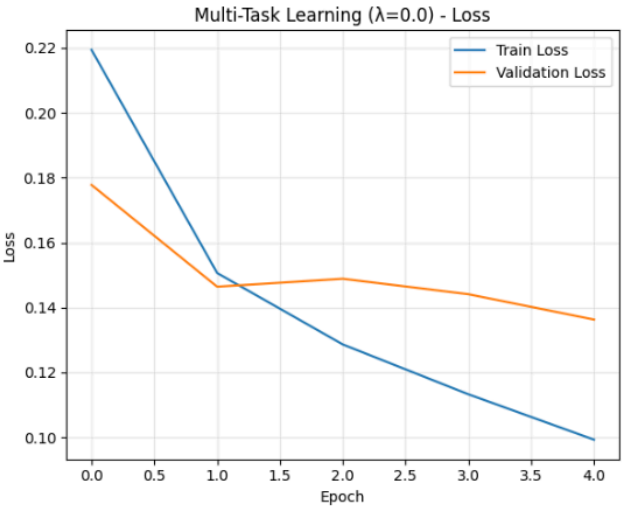

Multi-Task Learning (λ=0.5) - Accuracy
Task1 Train Acc
Task2 Train Acc
Task1 Val Acc
Task2 Val Acc
Accuracy
Epoch

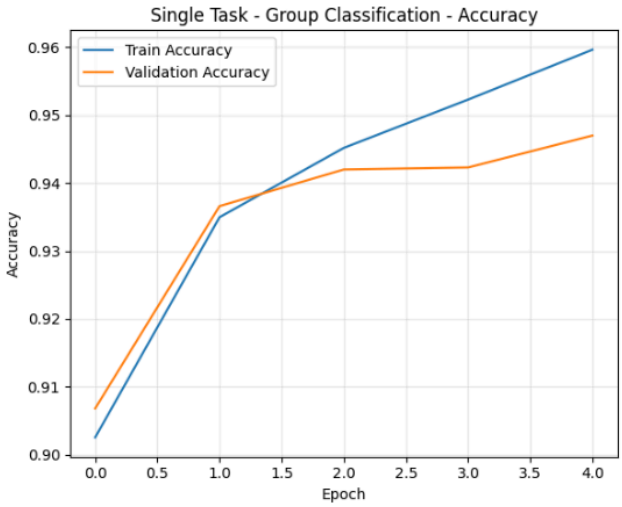

Multi-Task Learning (λ=0.5) - Loss
Train Loss
Validation Loss
Loss
Epoch

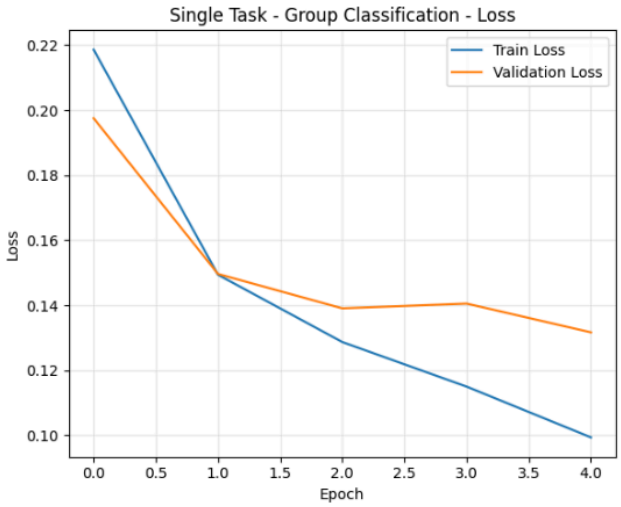

Multi-Task Learning (λ=0.25) - Accuracy
Accuracy
Epoch
Task1 Train Acc
Task2 Train Acc
Task1 Val Acc
Task2 Val Acc

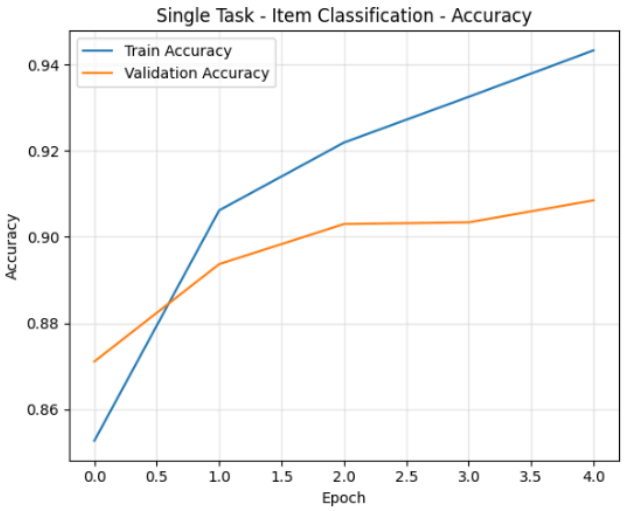

Multi-Task Learning (λ=0.25) - Loss
Loss
Epoch
Train Loss
Validation Loss

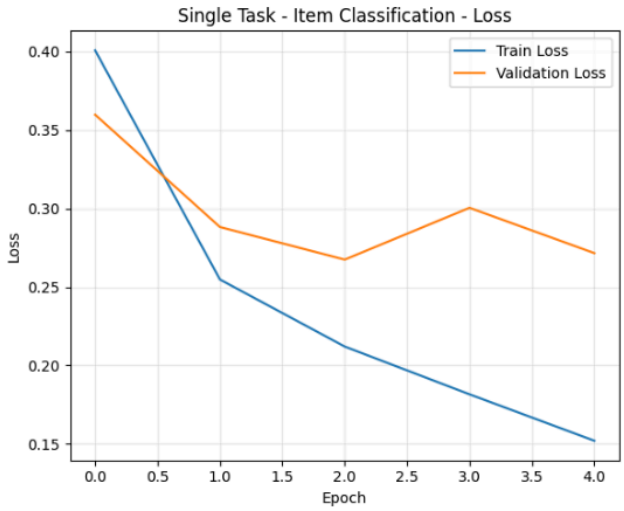

Multi-Task Learning (λ=0.0) - Accuracy
Accuracy
Epoch
Task1 Train Acc
Task2 Train Acc
Task1 Val Acc
Task2 Val Acc


Multi-Task Learning (λ=0.0) - Loss
Loss
Epoch
Train Loss
Validation Loss


Single Task - Group Classification - Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch


Single Task - Group Classification - Loss
Train Loss
Validation Loss
Loss
Epoch


Single Task - Item Classification - Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch


Single Task - Item Classification - Loss
Train Loss
Validation Loss
Loss
Epoch

References

- https://builtin.com/data-science/step-step-explanation-principal-component-analysis
- https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
- Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y. and Srebro, N., 2018. The role of over-parameterisation in generalisation of neural networks. - https://openreview.net/pdf?id=BygfghAcYX