**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.**

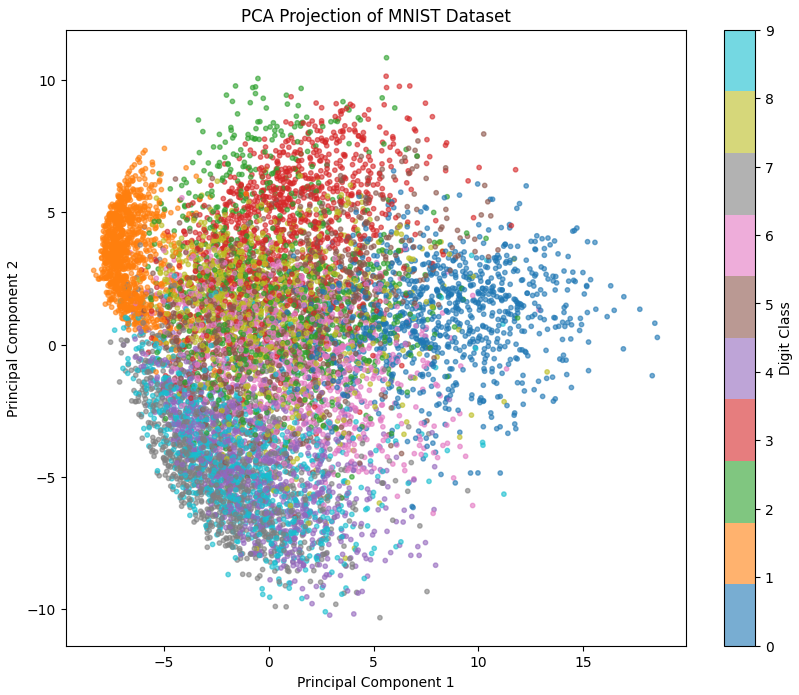
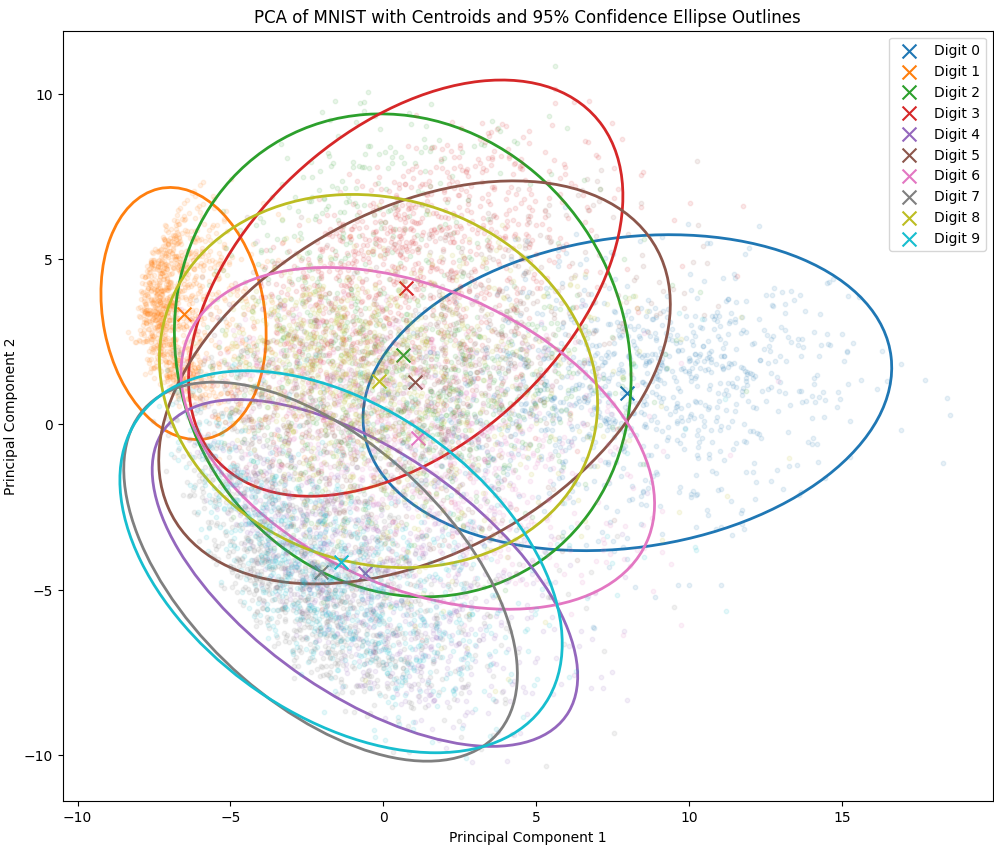
**Mathematically, PCA involves computing the covariance matrix of the centered data X:x**

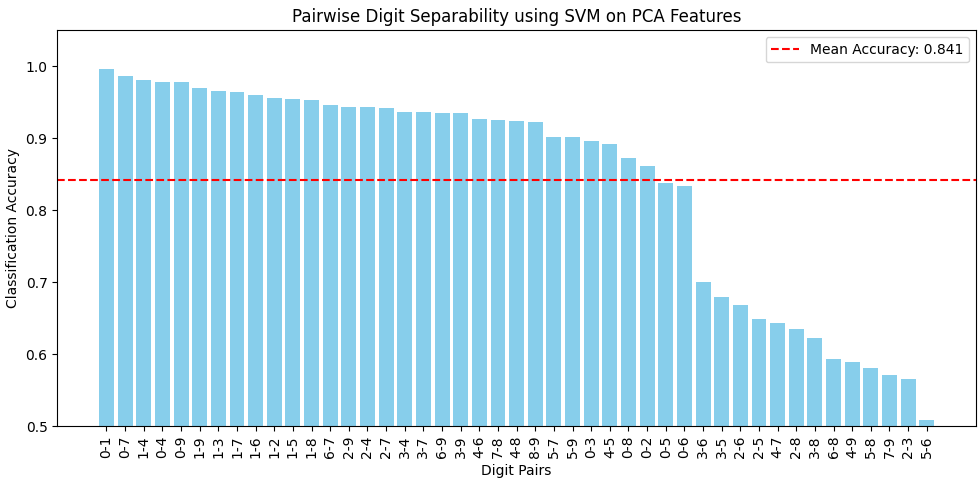
**Cov(X) = 1 n-1XTX**

**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:**

**XPCA=XV2**

**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.**

**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.**

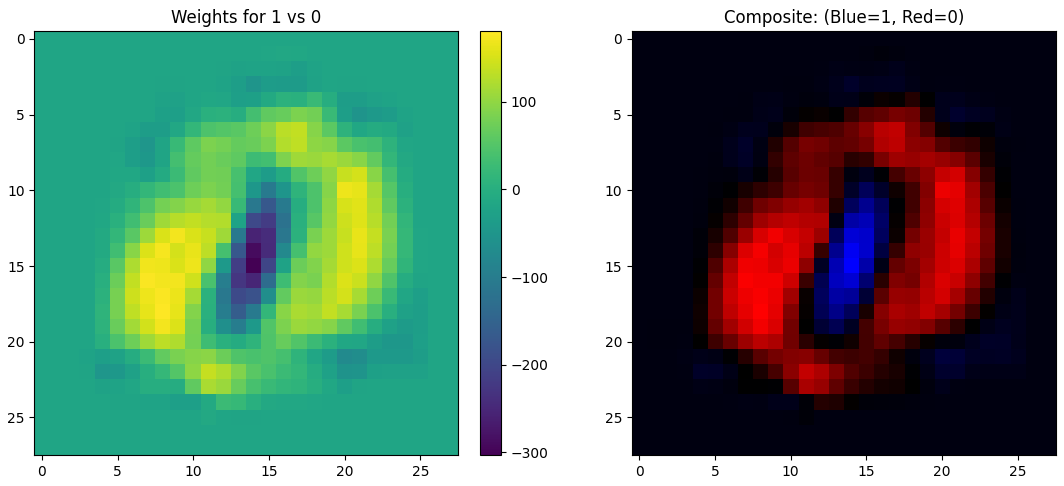
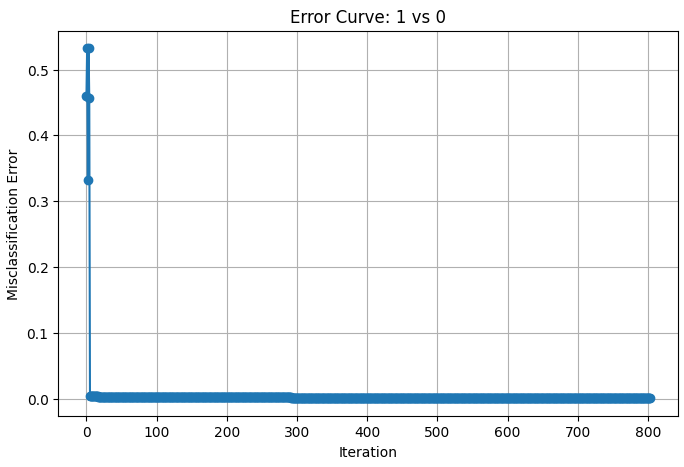
**PCA effectively visualises the MNIST dataset by reducing dimensionality and revealing clustering patterns. While digits 1 and 0 are linearly separable, the overlap among digits like 3, 5, and 8 underscores PCA’s limitations in fully separating all classes in 2D, suggesting that higher dimensions or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

**Perceptrons (Task 2)**

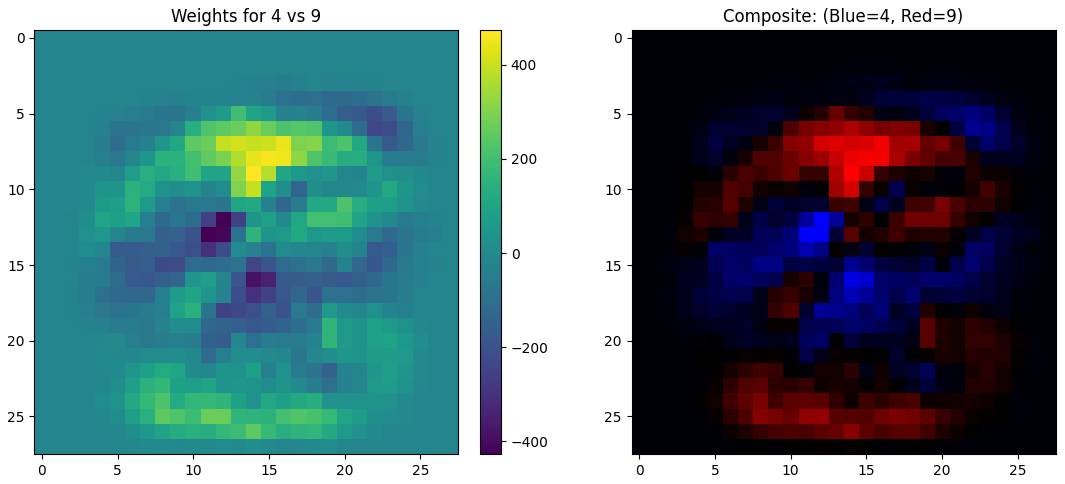
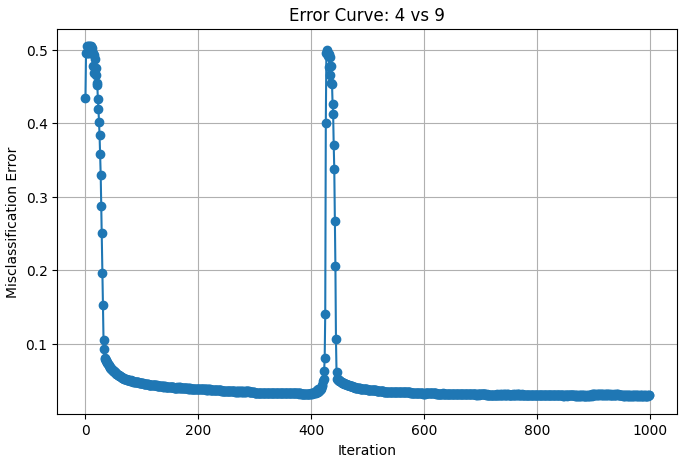
**The perceptron algorithm, a fundamental supervised learning technique for binary classification, excels at distinguishing pairs of handwritten digits in the MNIST dataset. Each training image, a 28x28 array of pixel values, is flattened into a high dimensional vector. The perceptron update rule initialises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

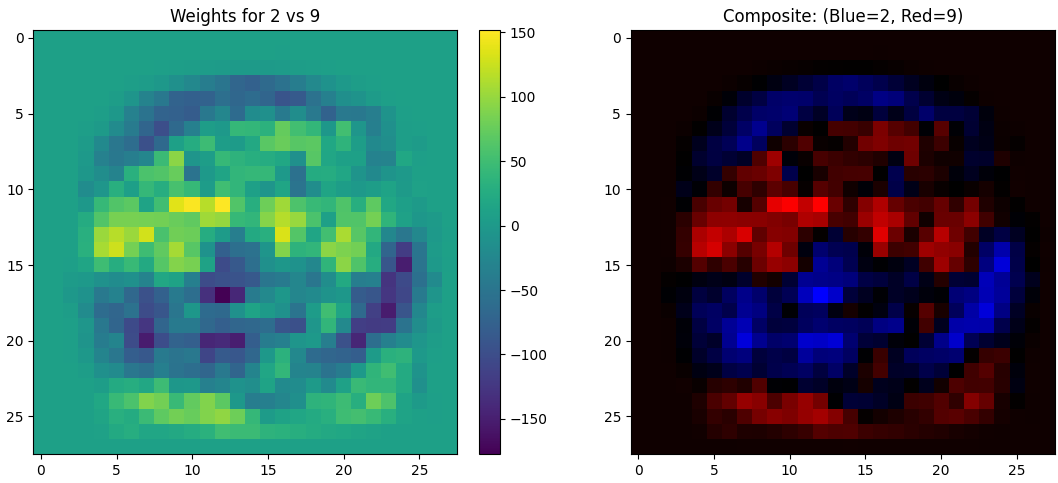
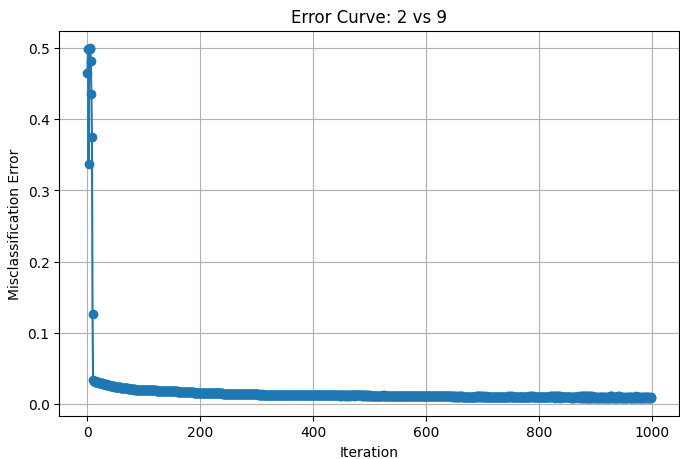
**Digit 1 vs 0**

****

**Digit 4 vs 9**

****

**Digit 2 vs 9**

****

**Mathematically, the perceptron computes a weighted sum of inputs:**

**w x+b**

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

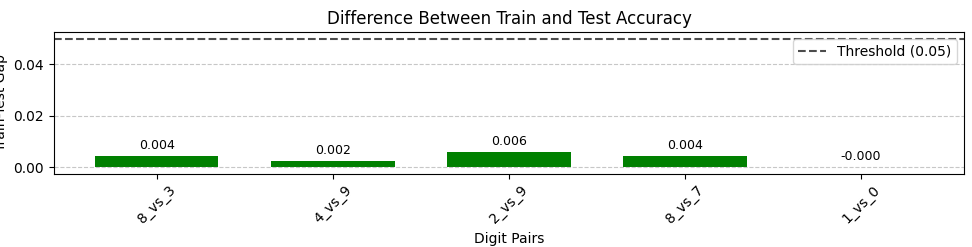
**y=sign(wx+b)**

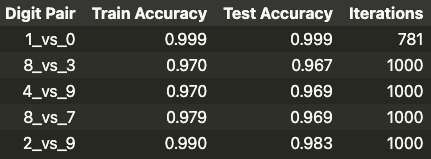
**to classify the result as +1 (one digit) or -1 (the other), where sign outputs +1 for positive values and -1 for negative. For a misclassified sample (where the predicted label differs from the true label y), the update rule is:**

**w ← w+ƞyx, b ← v +ƞy**

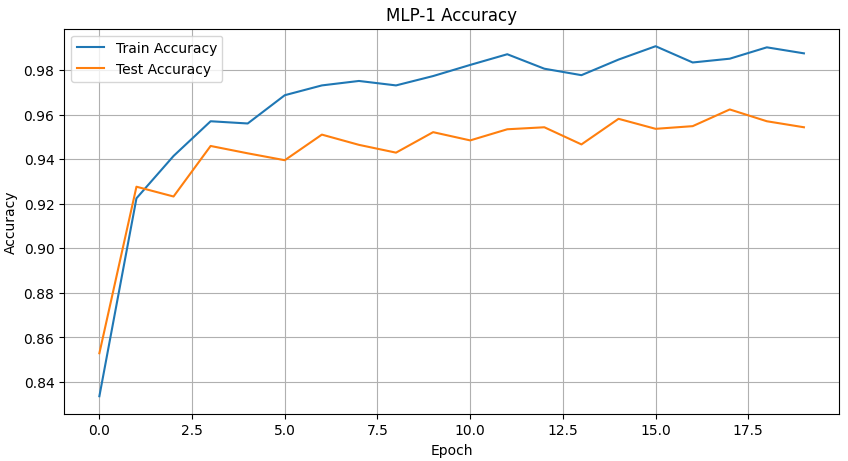
**with η=0.01 as the learning rate. This gradient-free adjustment minimises classification errors by aligning the decision hyperplane with the data.**

**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.**

**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)**

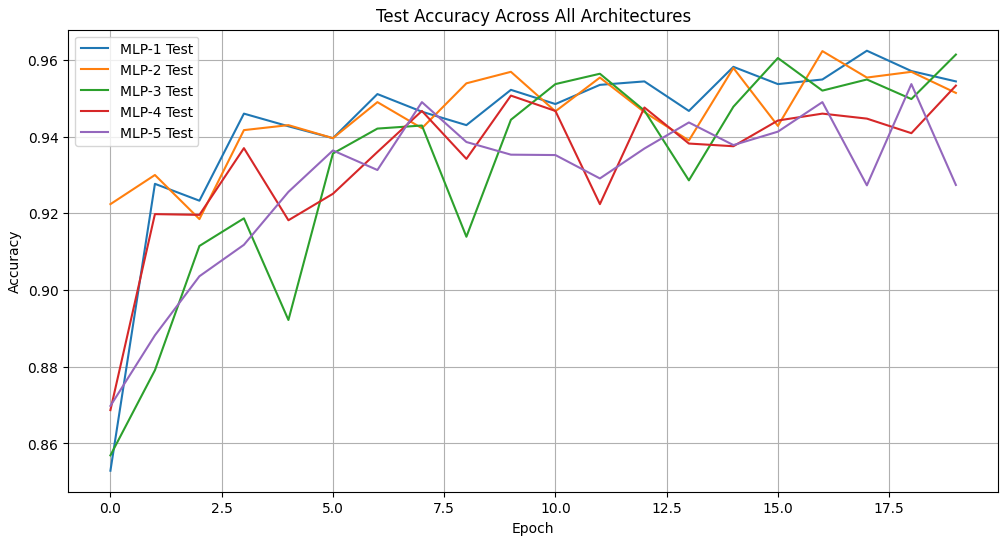
**Followed by the activation function:**

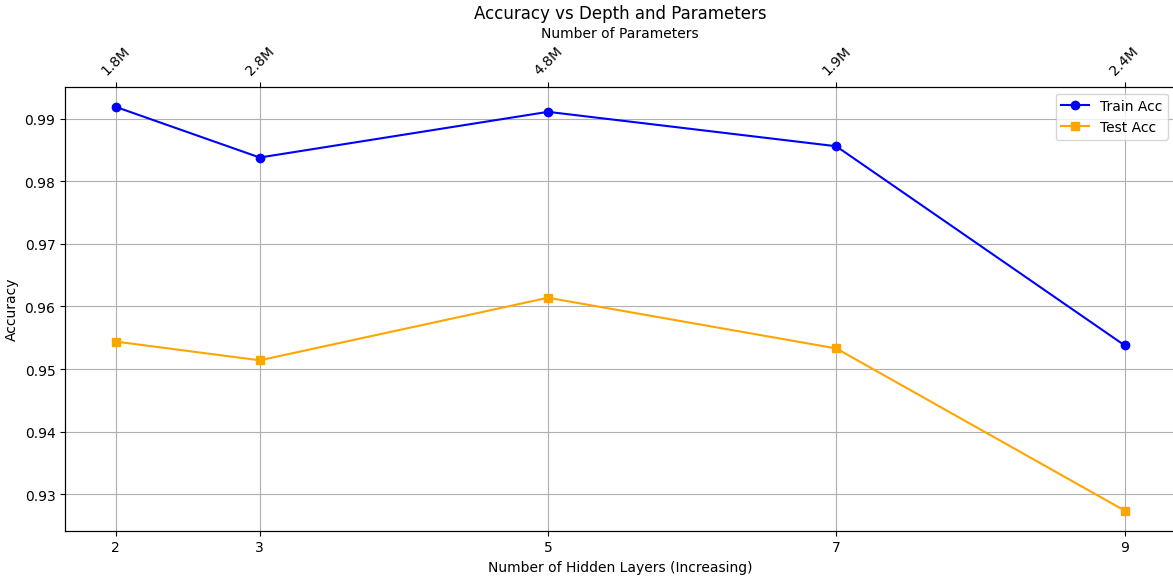
**a(l)=(z(l))**

**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.  
  
**

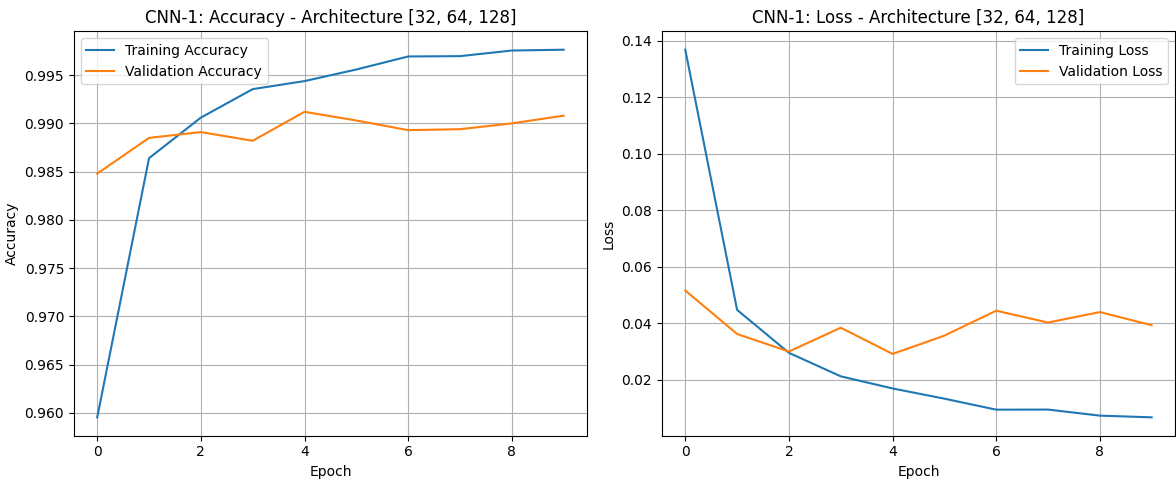
**The results corroborate findings by Neyshabur et al. (2018), who showed that increased network parameters often improve generalisation, even when capable of memorising random labels, emphasising the beneficial impact of over-parametrisation up to a point.**

****

**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.**

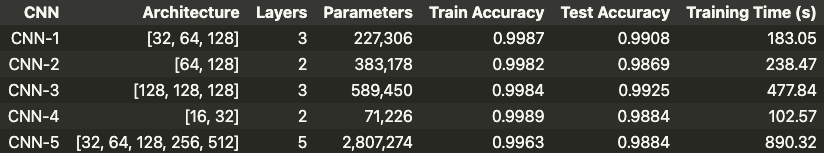
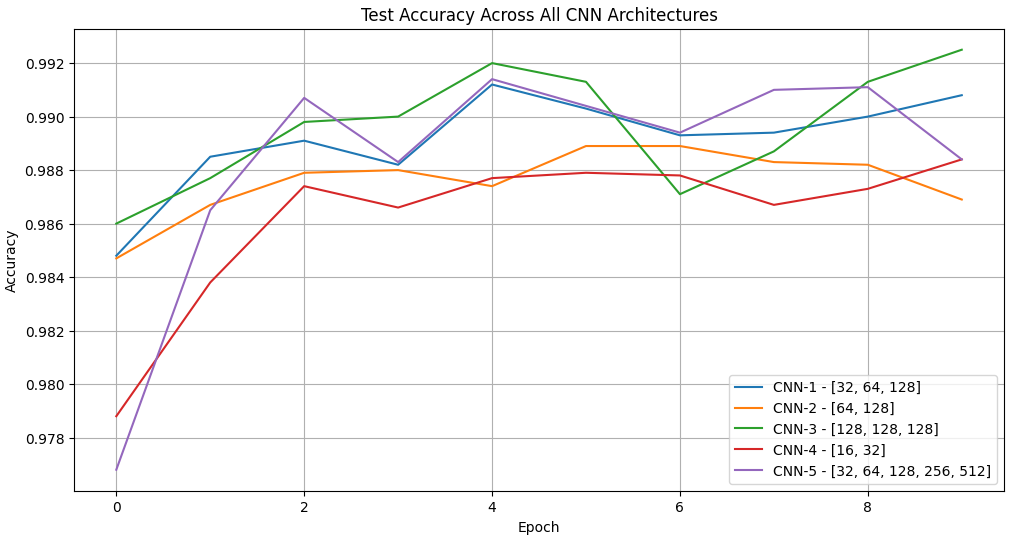
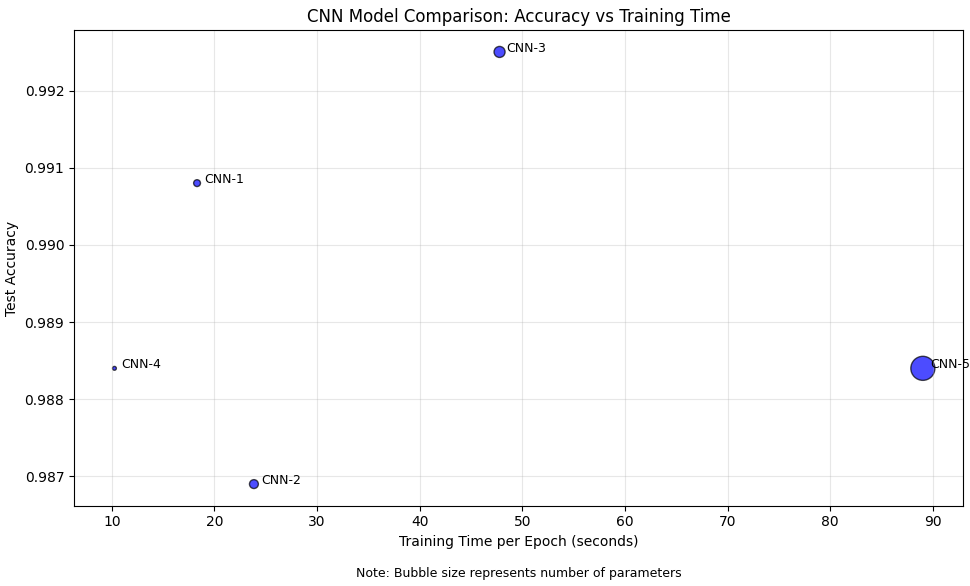
**CNN (Task 4)**

**Base CNN Training [32, 64, 128]**

**The base recommended CNN-1 is constructed using three convolutional layers using 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 model obtained a test accuracy of 99.08% after 20 epochs.**

**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.**

**  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**

**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.**

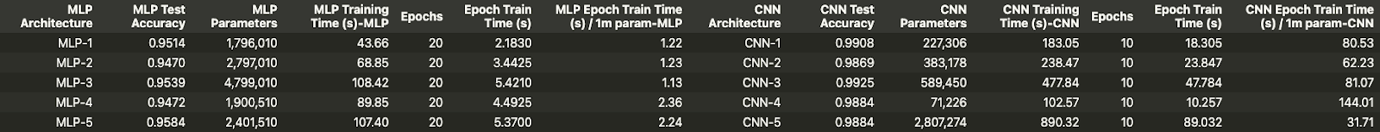
**The scatter plot reinforces CNN-3’s efficiency, showing high accuracy with moderate training time, while CNN5 lags with prolonged training. Thus, CNN-3 excels in balancing depth, parameters, and accuracy, underscoring that deeper, parameter heavy models do not always enhance performance, and optimal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

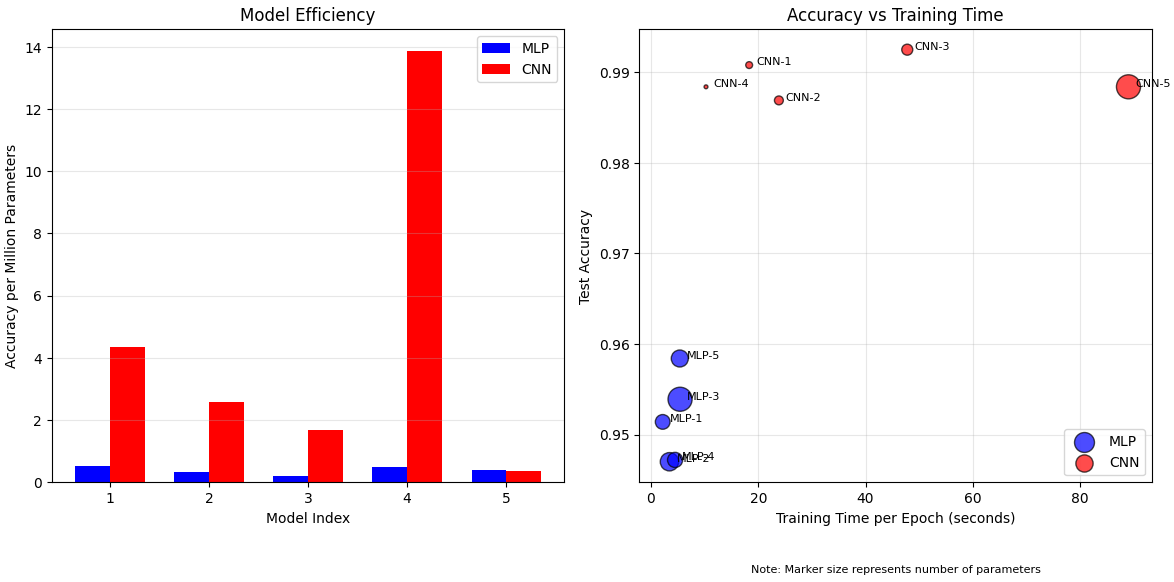
**Another interesting observation can be found when comparing CNN-2, with two layers [64, 128] and ~380k parameters, with CNN-5, which also has two layers [16, 32] but a much smaller parameter count of ~70k. It can be seen that CNN-4 archives a stronger test accuracy, indicating raw parameter count does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

**Comparing the CNNs to MLPs**

**CNNs achieved accuracies around 0.99, compared to MLPs at approximately 0.95, due to their ability to capture spatial hierarchies via convolutional layers. See the results table below:**

****

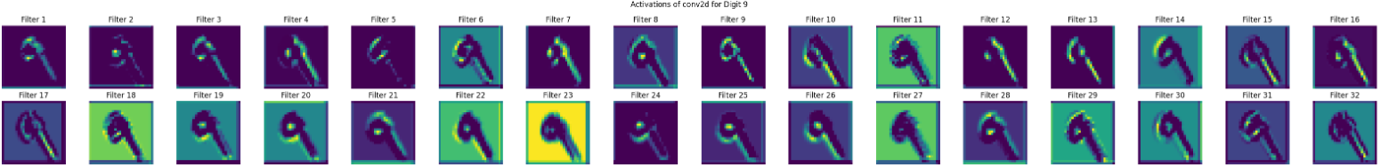
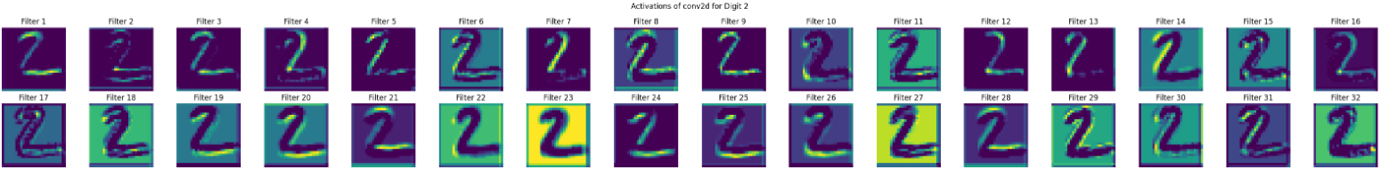
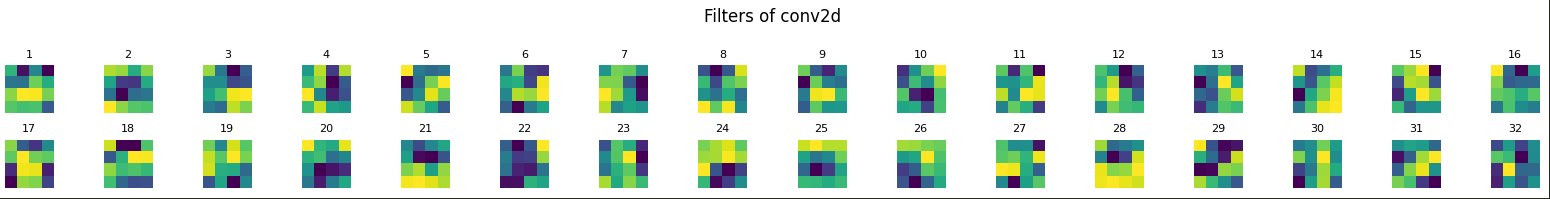
**The "Model Efficiency" graph supports this, with CNNs constantly displaying higher accuracies across model indices, notably model 4, which excels significantly. However, this performance advantage comes with increased computational demands. While some CNNs, like CNN-1, are parameter efficient with ~220k parameters against MLP-1’s ~1.8m, the parameter count for CNNs can escalate, as seen with ranges up to ~2.8m in the table, with up to 14m.**

****

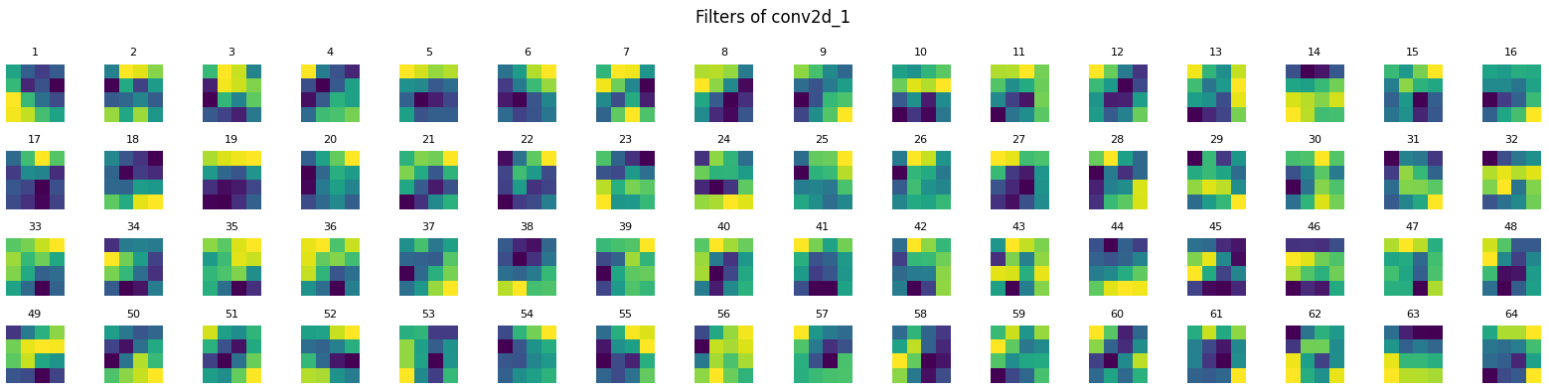
**In contrast, MLPs range from ~1.8 - 2.4 million parameters, generally requiring fewer weights and biases. Training times further highlight differences: the "Accuracy vs Training Time" graph indicates CNNs need up to 80 seconds per epoch, while MLPs train faster, often below 20 seconds. The table specifies CNN training times from 183.05 to 890.32 seconds, vastly exceeding MLPs’ 63.66 to 107.40 seconds. Moreover, epoch train time per million parameters is higher for CNNs (10.257 to 80.53 seconds) than MLPs (1.22 to 2.36 seconds), reflecting 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.**

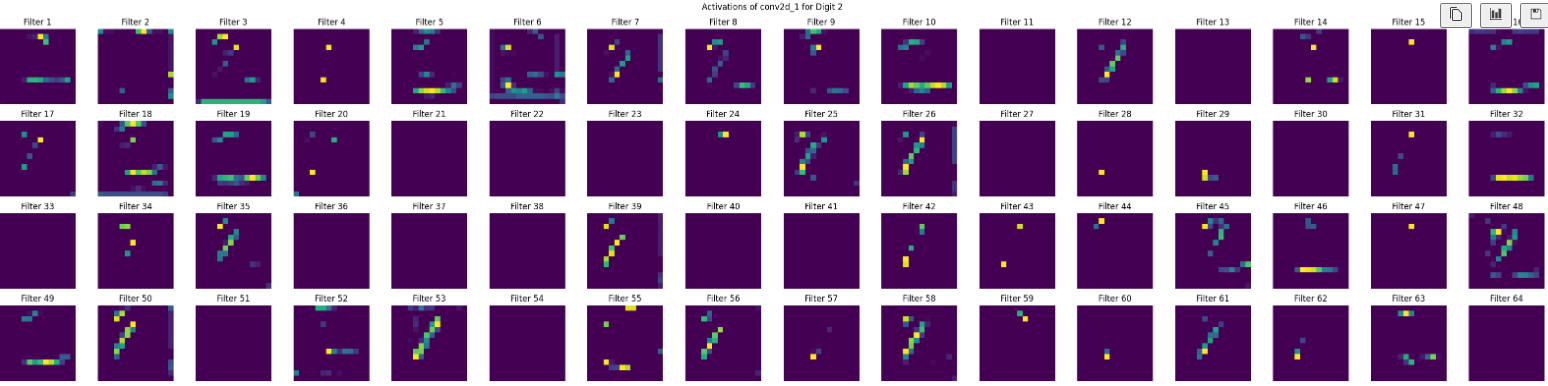
**Visualising CNN Outcomes (Task 5)**

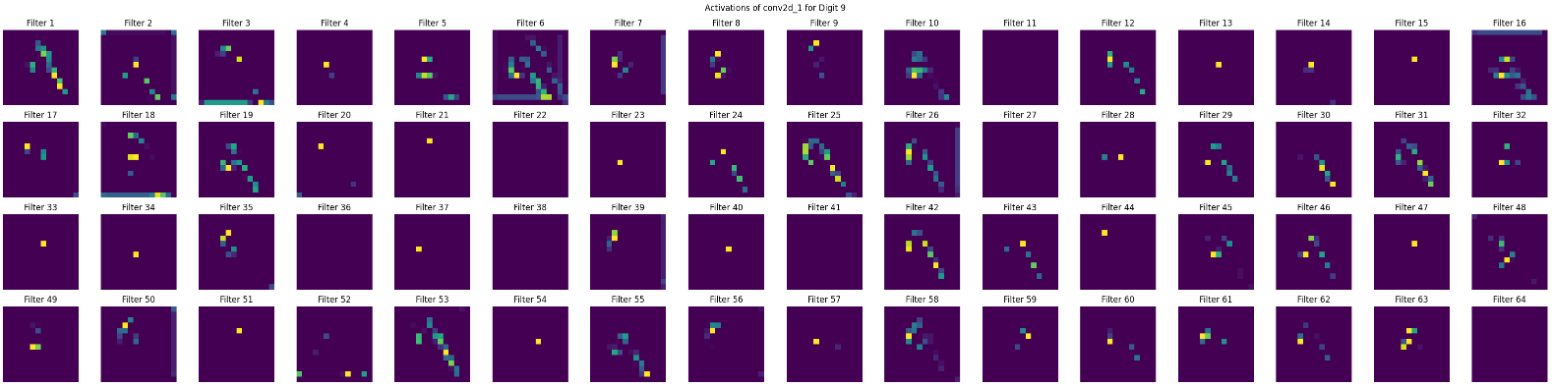
**Peering into the filters of the network, the different layers show the evolution of how the model “sees” the digits and understands them. The first convolutional layer applies 32 filters, each will interact with the image in a unique way to extract information. The activation function of this second 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.**

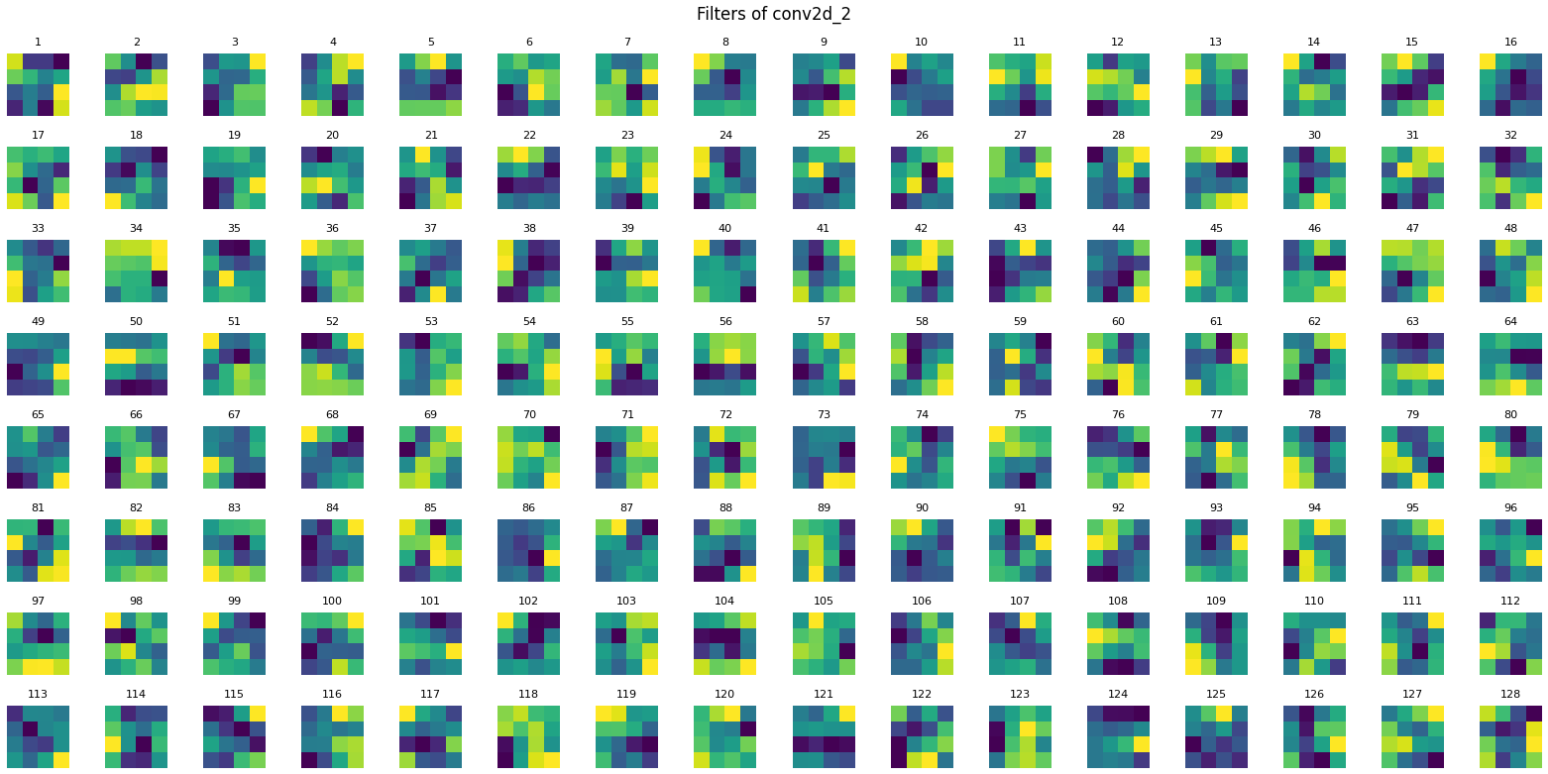
****

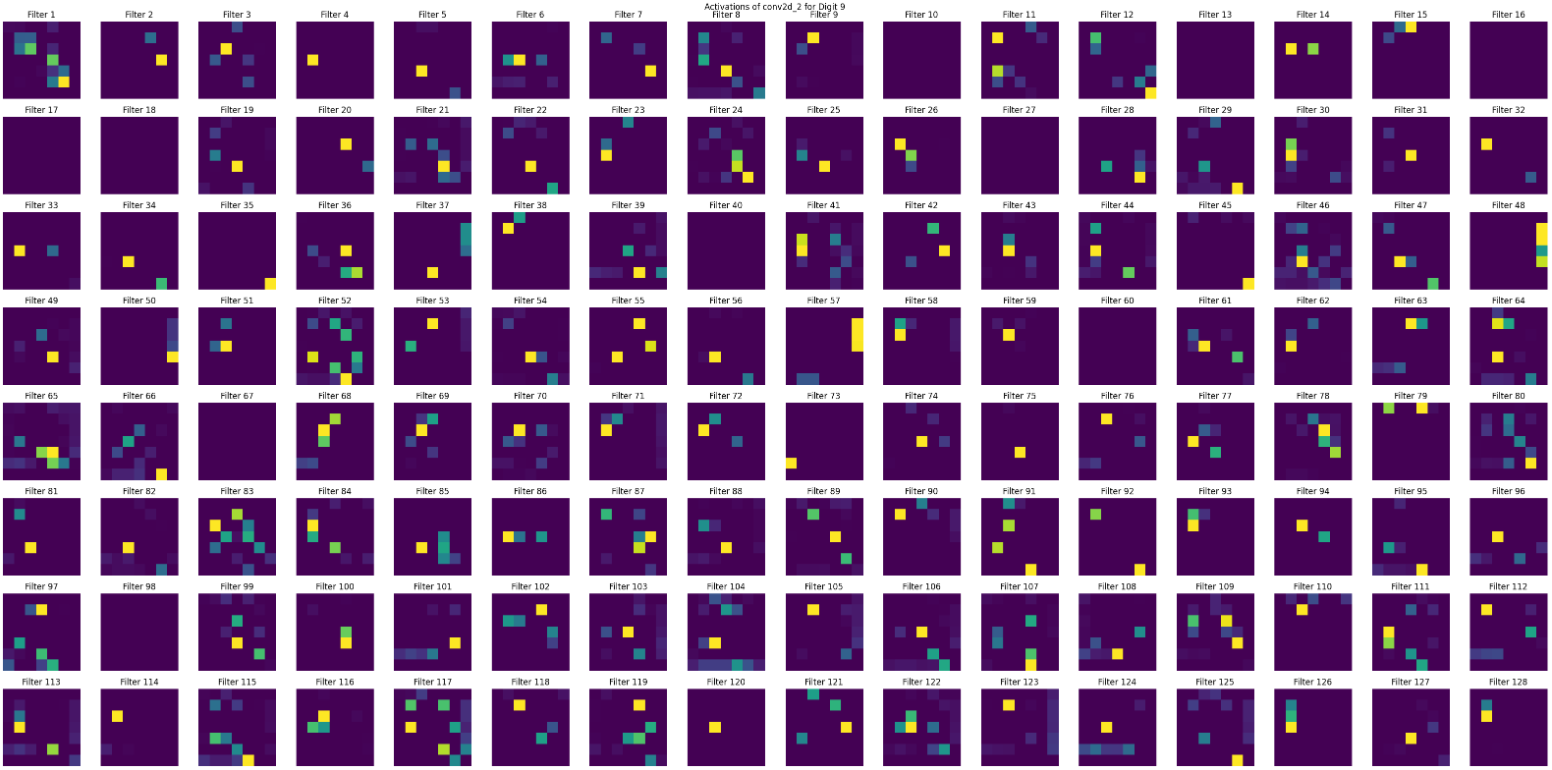
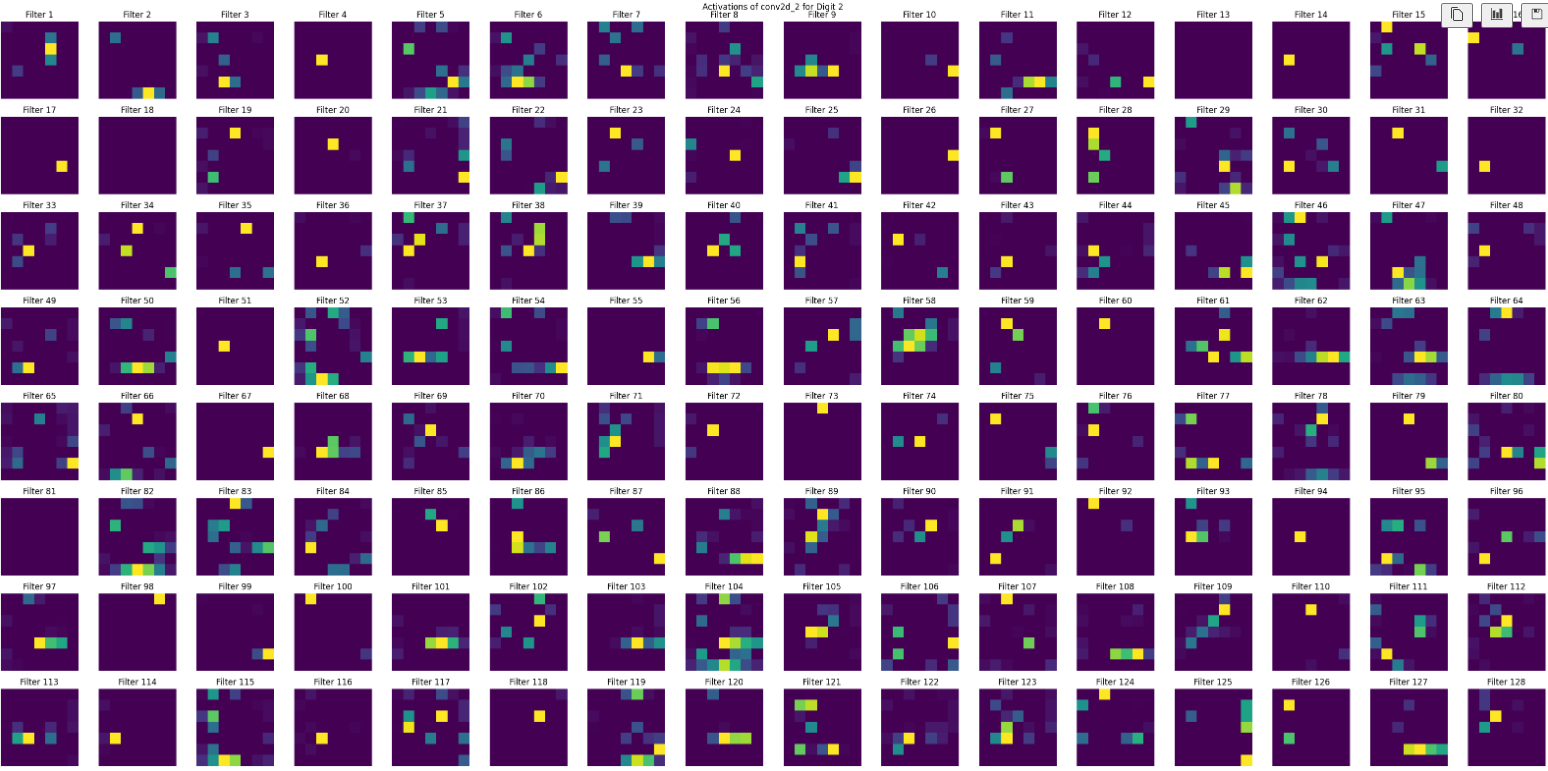
**In the second convolutional layer, we have 64 filters as expected and the activation graphs show how these filters are now more specialized and able to capture many aspects of the digits 2 and 9 in these cases. For example, focusing on filter 48, the activation graph shows that this filter is able 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.**

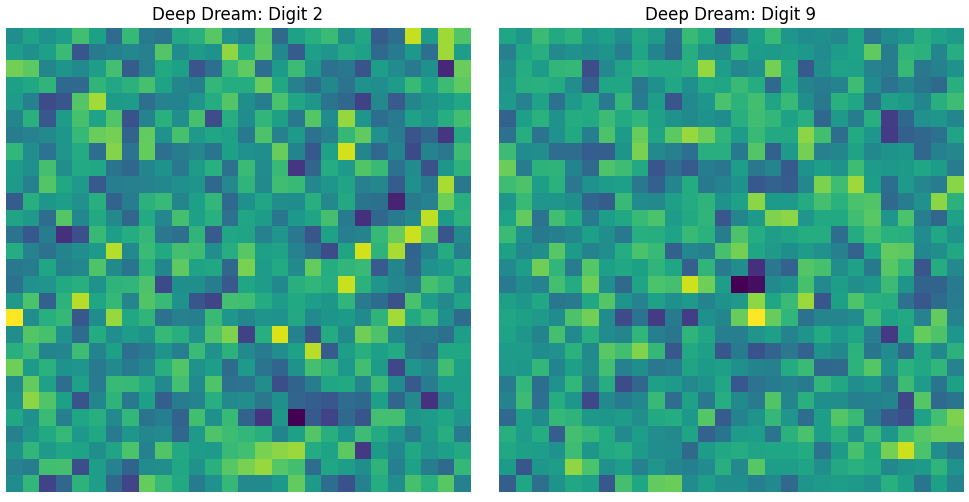
****

****

****

**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.**

****

****

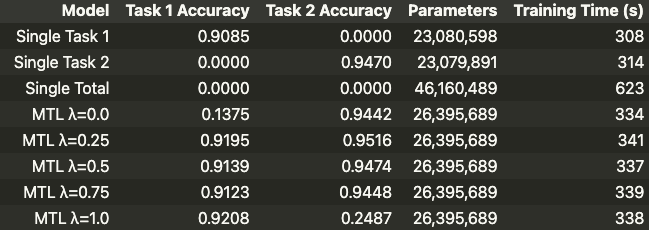
**Multi-task learning - Fashion MNIST (Task 6)**

* **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**

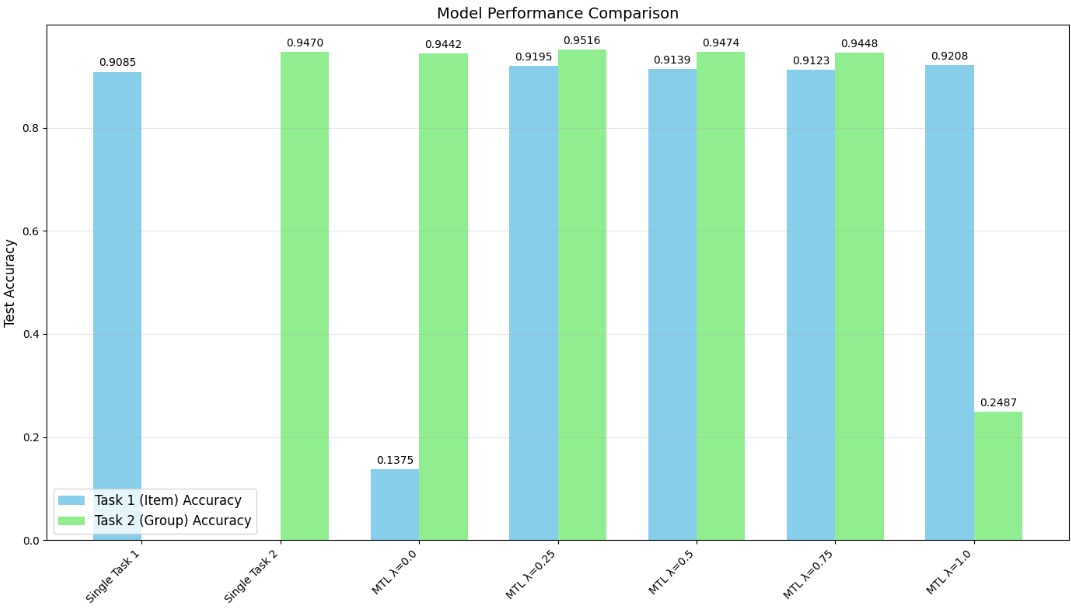
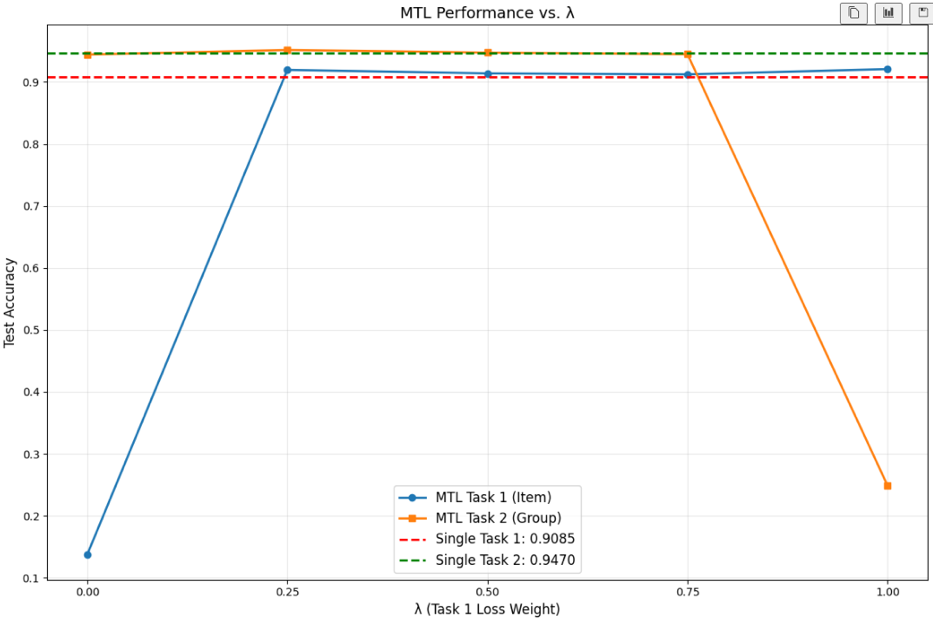
**In this analysis, we evaluated the performance of Multi-Task Learning (MTL) models designed to handle two tasks, (Item Classification and Group Classification), across five different values of the hyperparameter λ: 0.0, 0.25, 0.5, 0.75, and 1.0. The parameter λ controls the balance between the losses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

**Total Loss =LossTask1+(1-)LossTask2**

**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:**

****

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

****

**MTL Performance comparison to single-task networks**

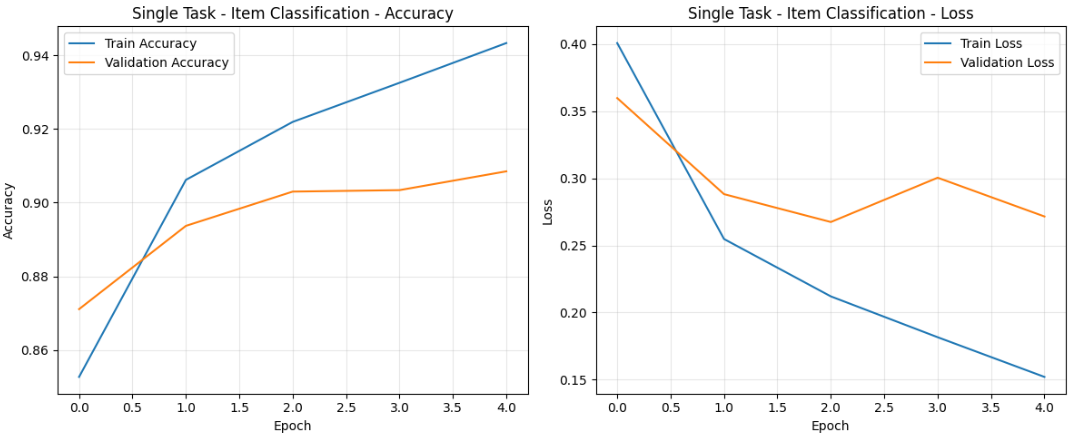
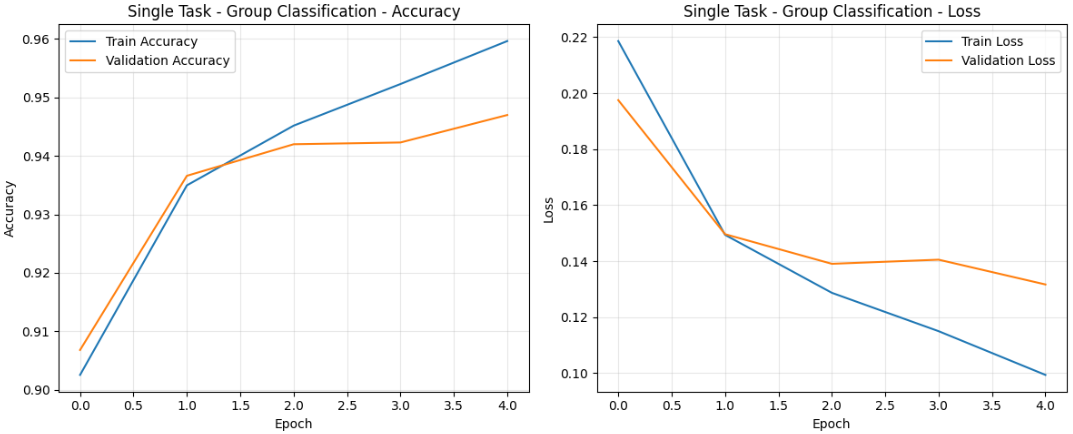
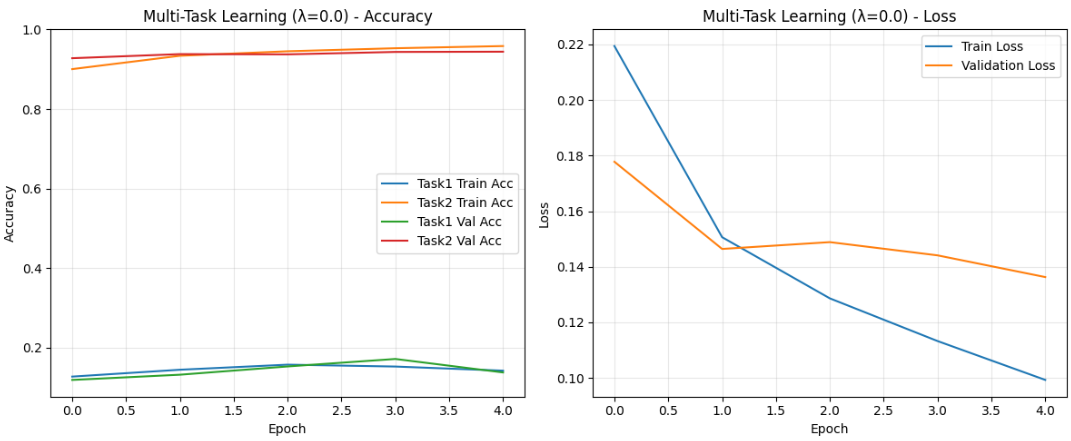
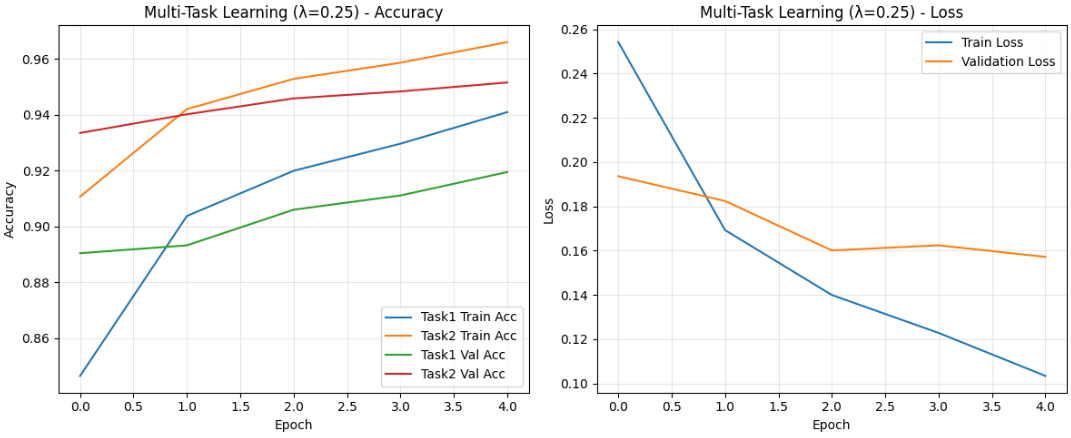
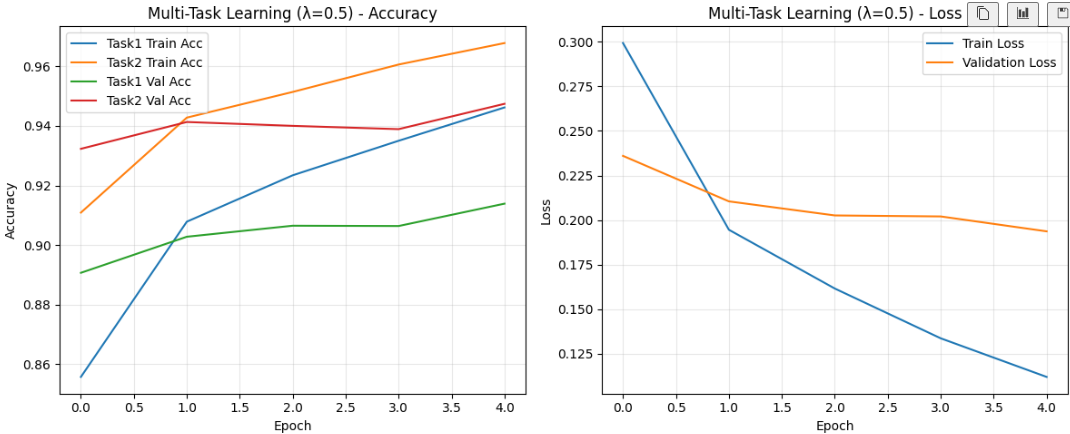
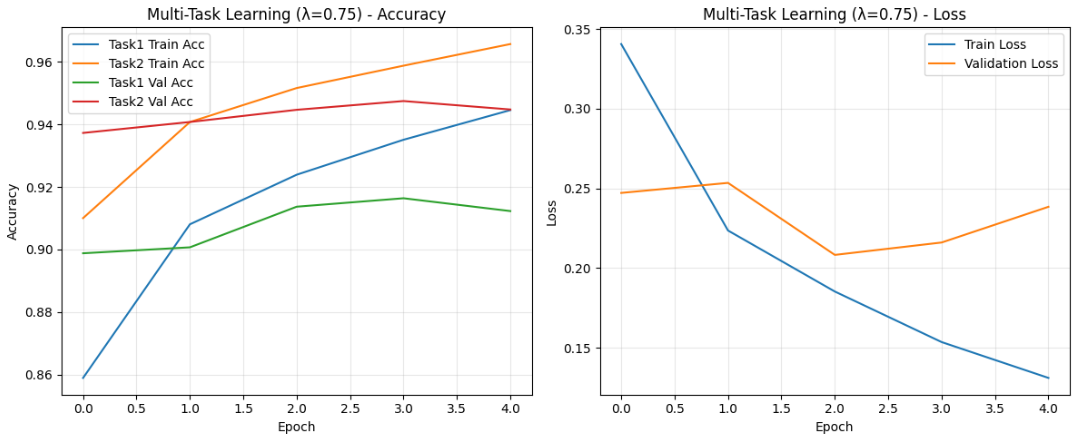
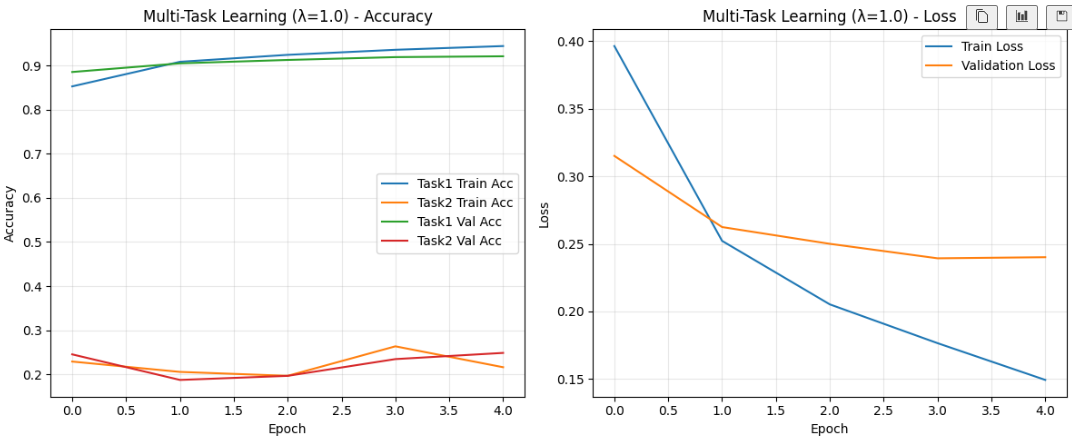
**= 0 Analysis**

**= 1 Analysis**

**= (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**

****

**References**

* [**https://builtin.com/data-science/step-step-explanation-principal-component-analysis**](https://builtin.com/data-science/step-step-explanation-principal-component-analysis)
* [**https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python**](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**](https://openreview.net/pdf?id=BygfghAcYX)