Machine Learning
Handwritten Digits Classification

Overview:

In this project, a Multilayer Perceptron (MLP) Neural Network was implemented to classify handwritten digits and facial images, allowing an exploration of neural networks fundamental mechanisms and their application in image classification tasks. The primary objectives were to understand and implement feedforward and backpropagation in neural networks, to optimize model performance through hyperparameter tuning, and to analyze the role of regularization in mitigating overfitting.

The project employed two datasets:

- MNIST Dataset - a widely used dataset of handwritten digits with 60,000 training and 10,000 test examples, each 28x28 pixels. The data was split further into training and validation subsets to optimize hyperparameters.
- CelebA Dataset - a subset of facial images, with two classes distinguishing whether individuals were wearing glasses. Each image was resized and flattened into vectors for model compatibility.

Key aspects of this project included:

- Neural Network Design: Building an MLP with a single hidden layer and incorporating backpropagation and regularization.
- Hyperparameter Tuning: Using the validation set to tune the regularization parameter (λ) and the number of hidden units. Different values of λ and the number of hidden units were tested to find an optimal balance between bias and variance.
- Performance Comparison: Testing the trained neural network on both datasets, comparing results with a deep neural network (DNN) and convolutional neural network (CNN) model, highlighting differences in accuracy and training time.

The report includes a detailed examination of the hyperparameter selection process, and an analysis of experimental results across models. Regularization effects were explored by adjusting λ, providing insights into the tradeoff between underfitting and overfitting.

Hyperparameters Tuning for Neural Network:

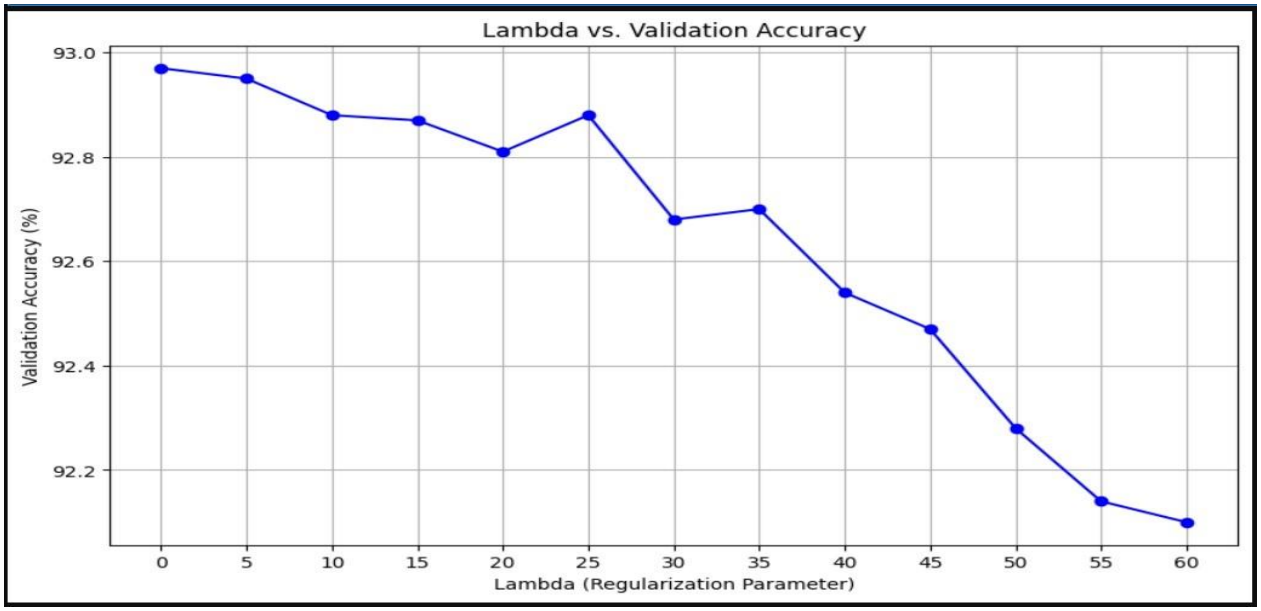
Hyperparameter tuning is a crucial part of training a neural network. In this project, we focused on two key hyperparameters: the regularization term (λ) and the number of hidden units in the network. These hyperparameters significantly impact the model's performance, and our goal was to find the optimal values for both in order to maximize the accuracy and efficiency of the network.

1. **Regularization Parameter (λ):**

Regularization is introduced to prevent overfitting and improve the generalization ability of the model. The regularization term λ controls the magnitude of the weights, penalizing large weights and ensuring that the model does not rely too heavily on any particular feature. A higher λ value results in stronger regularization, which can lead to underfitting, while a lower λ value may result in overfitting.

To determine the optimal value of λ, we varied it between 0 and 60, in increments of 5. We trained the neural network on the MNIST dataset and evaluated its performance on the validation set using different λ values. The relationship between λ and model accuracy was plotted to visualize the trade-off between underfitting and overfitting.

```
Lambda: 0, Validation Accuracy: 92.97%, Runtime: 44.08s
Lambda: 5, Validation Accuracy: 92.95%, Runtime: 40.73s
Lambda: 10, Validation Accuracy: 92.88%, Runtime: 33.29s
Lambda: 15, Validation Accuracy: 92.87%, Runtime: 34.48s
Lambda: 20, Validation Accuracy: 92.81%, Runtime: 33.16s
Lambda: 25, Validation Accuracy: 92.88%, Runtime: 33.96s
Lambda: 30, Validation Accuracy: 92.68%, Runtime: 34.75s
Lambda: 35, Validation Accuracy: 92.70%, Runtime: 34.89s
Lambda: 40, Validation Accuracy: 92.54%, Runtime: 34.81s
Lambda: 45, Validation Accuracy: 92.47%, Runtime: 33.89s
Lambda: 50, Validation Accuracy: 92.28%, Runtime: 37.12s
Lambda: 55, Validation Accuracy: 92.14%, Runtime: 34.33s
Lambda: 60, Validation Accuracy: 92.10%, Runtime: 34.75s
```



Analysis of Results:

As λ increased, the model's accuracy initially improved due to better regularization. However, beyond a certain threshold, further increases in λ led to a decrease in accuracy, indicating that the network became too constrained to learn the data effectively. The optimal value of λ was found where the accuracy plateaued before it started to degrade.

- Lambda = 0 (no regularization) results in the highest accuracy of 92.97%. However, this might indicate overfitting, as the model can fit the training data very closely, but may fail to generalize well to new, unseen data.
- As Lambda increases, we observe a slight decrease in validation accuracy, with the accuracy plateauing at Lambda = 10 (92.88%). Further increases in Lambda (to 15, 20, etc.) begin to show a noticeable decline in performance.
- Lambda = 10 offers the best balance between regularization and model accuracy, as it prevents overfitting without penalizing the model too harshly. It is the point where the accuracy appears to stabilize, and further increases in Lambda only lead to diminishing returns in terms of accuracy.

Choosing the Optimal Lambda:

Based on the observed trend, the optimal Lambda value is chosen to be Lambda = 10. This is because:

- Lambda = 10 shows minimal drop in accuracy compared to Lambda = 0 (from 92.97% to 92.88%).
  At Lambda = 10, the validation accuracy stabilizes, and further increases in Lambda lead to a gradual decline in performance.

- Lambda = 10 ensures the model is appropriately regularized to avoid overfitting while maintaining high accuracy.

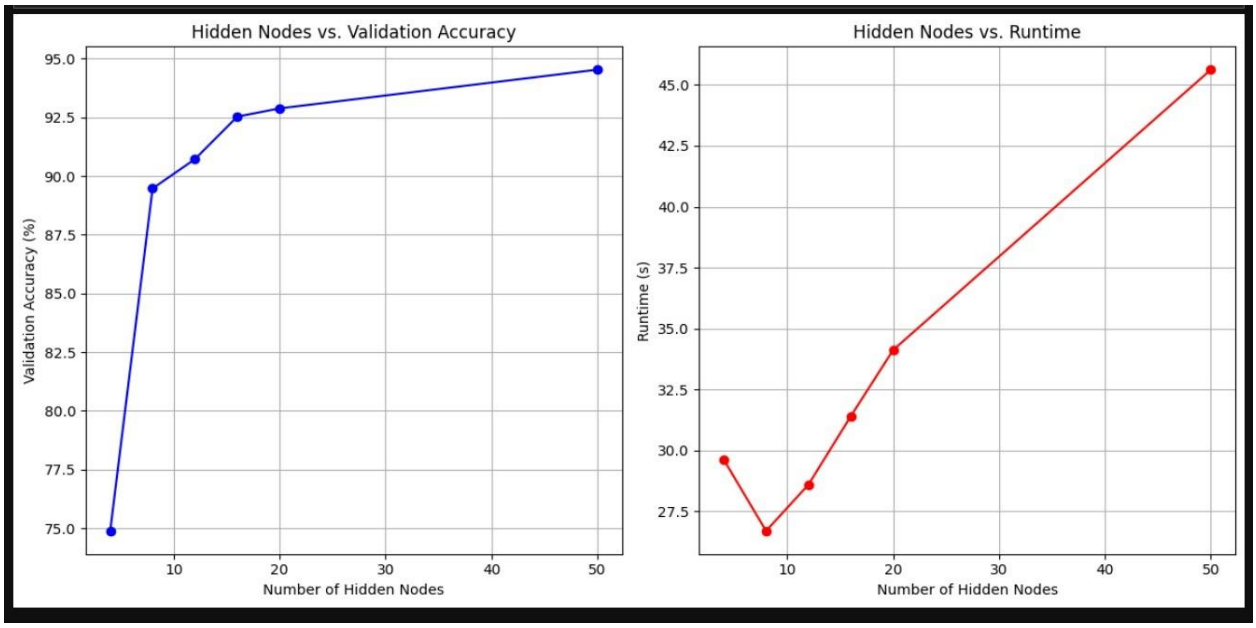
Thus, Lambda = 10 was selected as the optimal regularization parameter as it provides a good trade-off between model complexity and generalization.

2. Number of Hidden Nodes

The number of hidden nodes in a neural network impacts the model's capacity to learn complex patterns within the data. Adding more hidden nodes allows the model to capture intricate relationships, but it also increases computational cost and runtime. For this project, we experimented with varying numbers of hidden nodes to identify an optimal balance between validation accuracy and runtime.

To evaluate the effect of different numbers of hidden nodes, we tested configurations ranging from 4 to 50 nodes, recording both validation accuracy and runtime for each configuration.

```
Hidden Nodes: 4, Validation Accuracy: 74.89%, Runtime: 29.63s
Hidden Nodes: 8, Validation Accuracy: 89.47%, Runtime: 26.70s
Hidden Nodes: 12, Validation Accuracy: 90.72%, Runtime: 28.60s
Hidden Nodes: 16, Validation Accuracy: 92.53%, Runtime: 31.41s
Hidden Nodes: 20, Validation Accuracy: 92.88%, Runtime: 34.13s
Hidden Nodes: 50, Validation Accuracy: 94.53%, Runtime: 45.63s
```



Analysis of Results:

The results show that as the number of hidden nodes increases, validation accuracy generally improves, with a few important trends:

- Sharp Initial Improvement: Moving from 4 to 8 hidden nodes resulted in a significant increase in validation accuracy (from 74.89% to 89.47%). This indicates the importance of a sufficient number of hidden nodes to capture core patterns in the data.
- Accuracy Plateau: Validation accuracy continues to improve as we add more hidden nodes, but the accuracy increase becomes marginal around 20 hidden nodes, suggesting that the network has reached a suitable level of complexity for this dataset.
- Increased Runtime: As the number of hidden nodes grows, runtime also increases substantially. For example, while 50 hidden nodes achieved the highest accuracy at 94.53%, the runtime rose to 45.63 seconds, a significant increase from the 34.13 seconds observed with 20 hidden nodes. This highlights the computational cost associated with a larger network, making it necessary to balance accuracy with runtime.

Choosing the Optimal Number of Hidden Nodes:

Based on these observations, we selected 20 hidden nodes as the optimal choice. This configuration was chosen because:

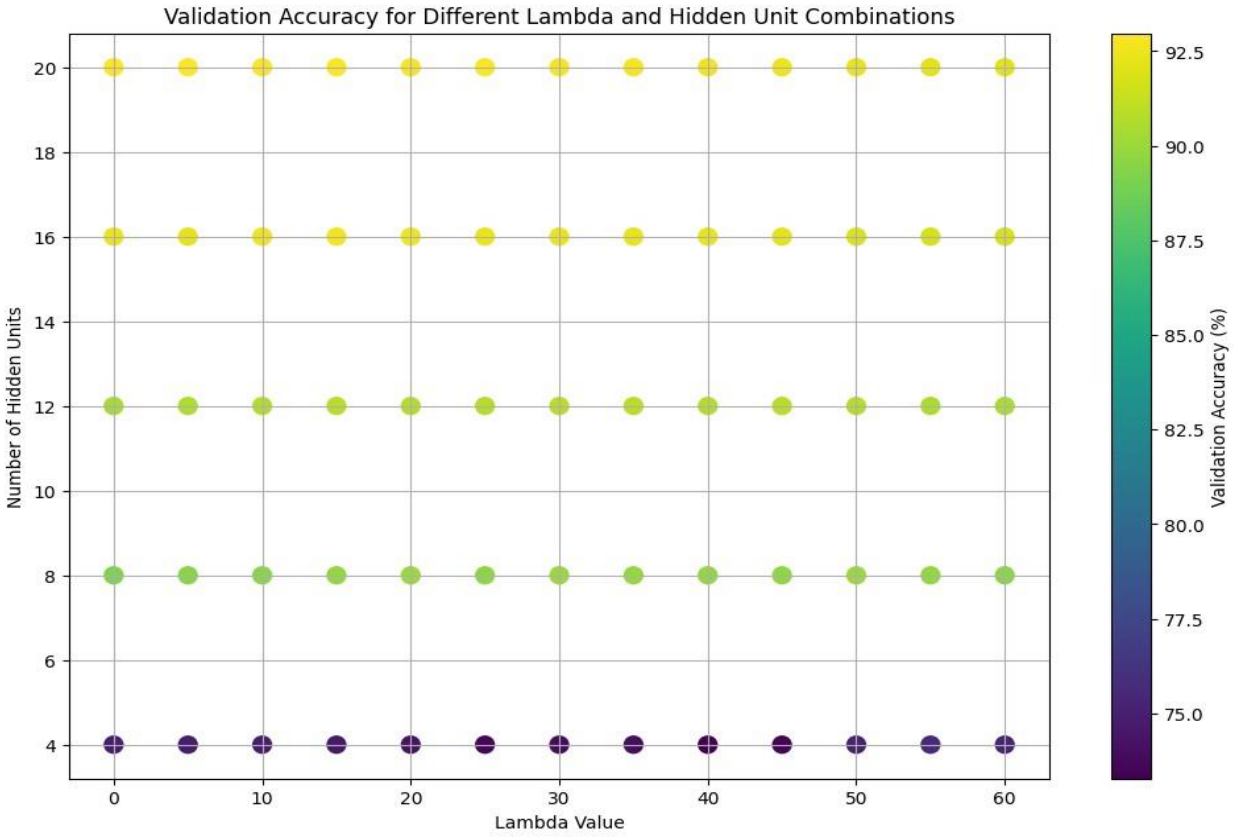
- High Accuracy: With 20 hidden nodes, the model achieves 92.88% validation accuracy, which is close to the highest observed accuracy.
- Efficient Runtime: The runtime with 20 hidden nodes (34.13 seconds) remains manageable, providing a good balance between accuracy and run time.
- Avoiding Excessive Runtime: Increasing beyond 20 hidden nodes leads to only minor accuracy improvements, while runtime continues to increase substantially.

Thus, 20 hidden nodes was selected as the optimal configuration, providing a balanced trade-off between model accuracy, computational cost, and runtime. This choice ensures that the neural network captures essential patterns in the data without incurring the high computational cost and runtime associated with a larger number of hidden nodes.

3. Validating the Optimal Hyperparameters with Combinations of Lambda and Hidden Nodes

To ensure that our selected values for lambda (10) and hidden nodes (20) were indeed optimal, we conducted an additional experiment by running combinations of different lambda values and hidden node configurations. This allowed us to assess whether our chosen values continued to offer a balanced performance in terms of both accuracy and runtime across various combinations.

In this experiment, we tested a range of values for both lambda and hidden nodes, observing the combined effect on validation accuracy and runtime. The results confirmed that the configuration of lambda = 10 and 20 hidden nodes consistently yielded high validation accuracy and efficient runtime, even when compared to other combinations.



Analysis of Combination Results:

- Consistency in Performance: The configuration of lambda = 10 and 20 hidden nodes provided a stable and high validation accuracy across different runs, reinforcing that this choice was effective in preventing overfitting while maintaining strong generalization on the validation set.
- Balanced Runtime: Among the combinations tested, the selected parameters achieved a good balance between computational cost and accuracy, with a runtime that remained manageable even as other combinations resulted in longer runtimes for similar or lower accuracy levels.

Conclusion:

This combination experiment validated our initial choice, confirming that lambda = 10 and 20 hidden nodes is an optimal configuration. The results showed that this choice not only maximizes validation accuracy but also ensures efficient runtime, providing a well-rounded approach for balancing model complexity with performance.

By testing combinations of hyperparameters, we confirmed that our selected values achieved robust results and performed effectively even in comparison with other configurations, demonstrating the reliability and effectiveness of our chosen hyperparameters.

Accuracy of Classification Method on the Handwritten Digits Test Data

After selecting the optimal values for lambda (10) and hidden nodes (20), we evaluated the neural network's accuracy on the training, validation, and test datasets to measure its performance on the MNIST handwritten digits classification task. The results were as follows:

```
Training set Accuracy: 93.74 %

Test set Accuracy: 93.47 %

Validation set Accuracy: 92.88 %
```

These results indicate that the neural network achieved a high level of accuracy across all datasets, showing effective learning and generalization. The training accuracy of 93.74% suggests that the model has successfully learned patterns in the training data, while the validation accuracy of 92.88% confirms that our chosen hyperparameters (lambda and hidden nodes) are preventing overfitting. Additionally, the test accuracy of 93.47% indicates strong generalization to unseen data.

To gain further insights into the model's performance on the test set, we analyzed the confusion matrix:

```
Confusion Matrix on Test Set:
[[ 960    0    1    0    2    5    6    3    3    0]
 [   0 1119    2    2    1    0    4    2    5    0]
 [  12    3  925   17   10    2   14   16   25    8]
 [   4    2   22  923    1   20    3   14   14    7]
 [   2    4    4    0  926    0    9    3    4   30]
 [  11    3    2   46    8  774   16    6   19    7]
 [  14    4    4    1    6   13  914    1    1    0]
 [   4   19   23    2   10    1    2  936    1   30]
 [   3   10    7   16    9   29   10    8  873    9]
 [  13    4    2   11   27    8    1   13   10  920]]
```

Analysis of the Confusion Matrix:

- Correct Classifications: The diagonal values represent the number of correct classifications for each digit. For instance, the model correctly classified 960 images of the digit 0, 1119 images of the digit 1, and 925 images of the digit 2. The high values along the diagonal reflect the model’s strong performance for each class.

- Misclassifications: Off-diagonal values indicate instances where the model misclassified one digit as another. For example, 12 instances of the digit 2 were misclassified as digit 0, and 46 instances of the digit 5 were misclassified as digit 3. These errors often occur due to similarities in shape or features between certain digits, which can make it challenging for the model to distinguish them accurately.

Overall Performance:

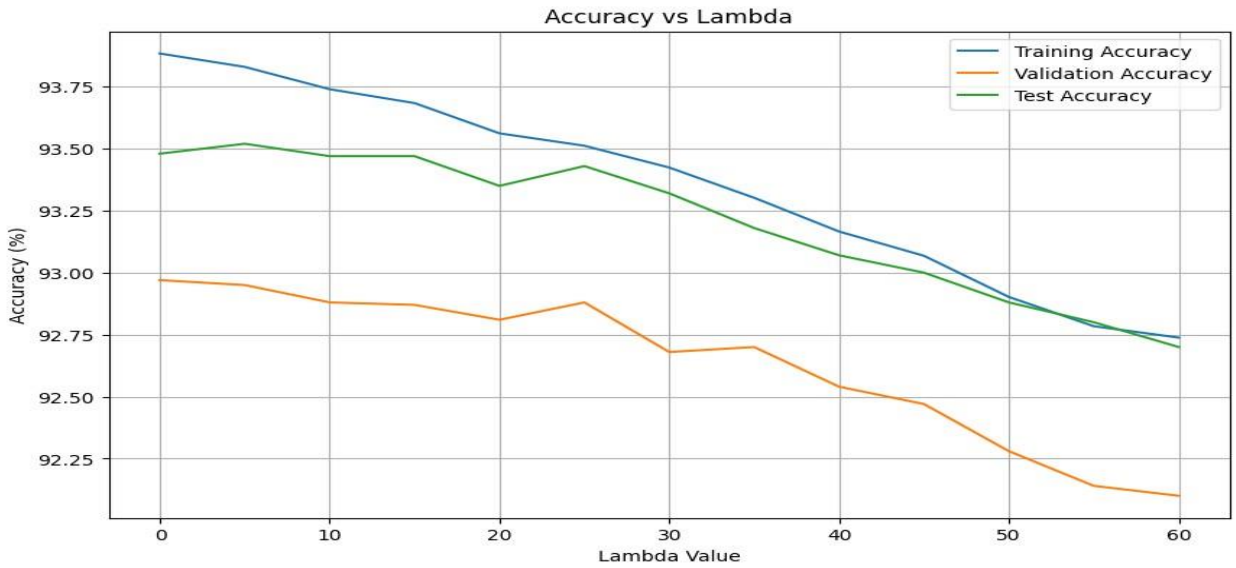
- The model performs particularly well on digits like 1 and 0, which have relatively few misclassifications.
- The confusion matrix shows that digits with similar shapes (e.g., 3 and 5, or 4 and 9) tend to have higher misclassification rates, highlighting areas where the model could benefit from further tuning or feature enhancement.
- Overall, these accuracy values and the confusion matrix analysis demonstrate that our model, with the chosen hyperparameters, performs well in classifying handwritten digits. The high accuracy across training, validation, and test sets, along with the distribution of errors in the confusion matrix, reinforces the model's ability to generalize effectively and achieve reliable classification on the MNIST dataset.

Extra Analysis

To further evaluate the robustness of our model configuration, we conducted an additional analysis to observe how the training, validation, and test accuracies vary across different values of lambda and number of hidden nodes. This extra analysis allowed us to verify the effectiveness of our chosen hyperparameters (lambda = 10 and hidden nodes = 20) and assess how accuracy is influenced by changes in these parameters.

Lambda Vs Accuracies:

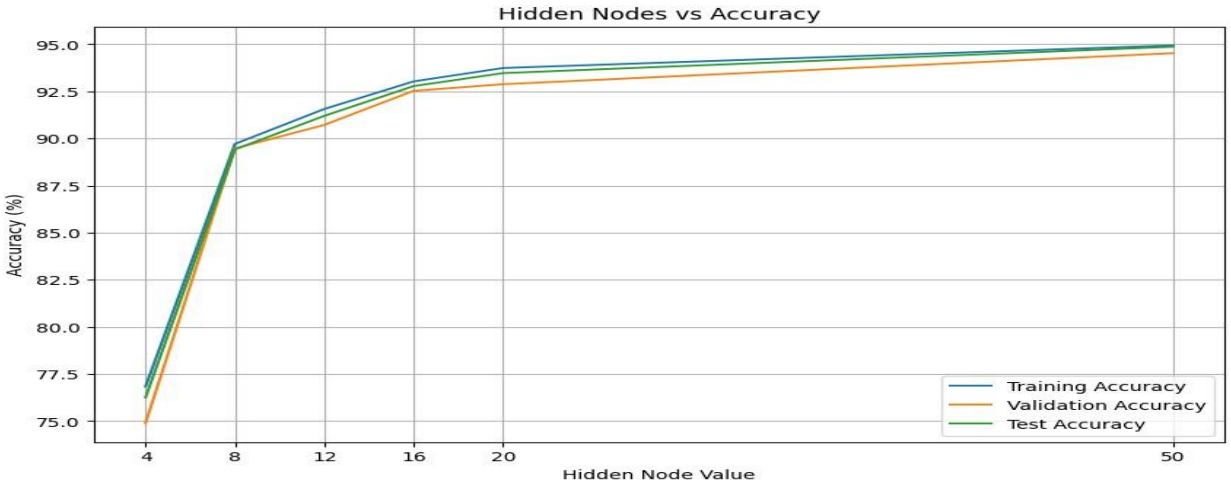
- We examined the model’s performance at multiple values of lambda (0, 5, 10, 20, 30, etc.), measuring accuracy across the training, validation, and test sets. This helped us understand how regularization impacts model learning and generalization.
- This analysis reinforced that lambda = 10 provides an optimal level of regularization, ensuring the model generalizes well without compromising accuracy on the training data.

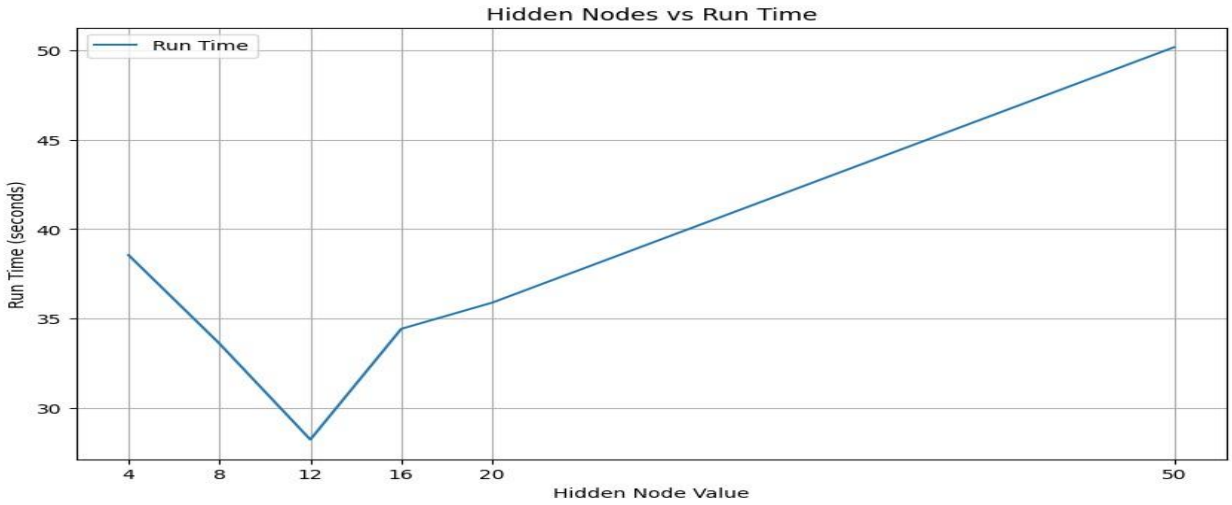


Hidden nodes Vs Accuracies and Run time:

- We also experimented with different numbers of hidden nodes (4, 8, 12, 16, 20, and 50) to observe how model capacity affects accuracy and run time on the training, validation, and test sets.
- This analysis validated that 20 hidden nodes offer a suitable balance between model complexity and accuracy, ensuring effective learning while avoiding unnecessary increases in computational cost.

```
Hidden Nodes values, corresponding train, validation, test accuracies, and runtime:
Hidden Nodes: 4, Train Accuracy: 76.83%, Validation Accuracy: 74.89%, Test Accuracy: 76.25%, Runtime: 38.55s
Hidden Nodes: 8, Train Accuracy: 89.71%, Validation Accuracy: 89.47%, Test Accuracy: 89.42%, Runtime: 33.60s
Hidden Nodes: 12, Train Accuracy: 91.56%, Validation Accuracy: 90.72%, Test Accuracy: 91.20%, Runtime: 28.22s
Hidden Nodes: 16, Train Accuracy: 93.03%, Validation Accuracy: 92.53%, Test Accuracy: 92.78%, Runtime: 34.42s
Hidden Nodes: 20, Train Accuracy: 93.74%, Validation Accuracy: 92.88%, Test Accuracy: 93.47%, Runtime: 35.89s
Hidden Nodes: 50, Train Accuracy: 94.94%, Validation Accuracy: 94.53%, Test Accuracy: 94.87%, Runtime: 50.18s
```





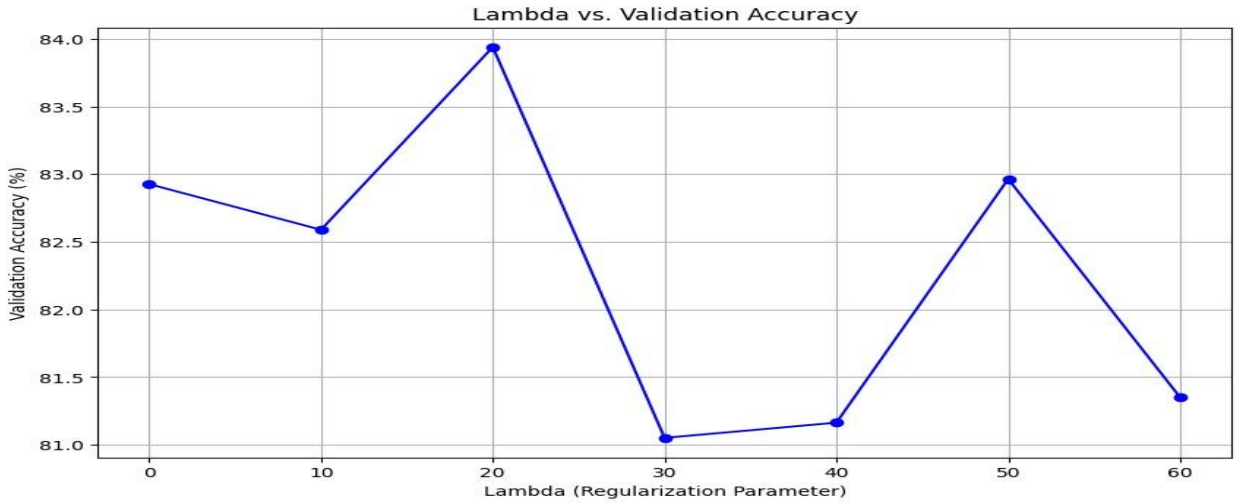
CelebA data set

Hyperparameter Tuning on the CelebA data set

In addition to the MNIST dataset, we applied hyperparameter tuning on the CelebA Dataset to evaluate the model's performance on a more complex, real-world dataset. We tuned the lambda (regularization parameter) to observe how it impacts the validation accuracy of the model on this dataset. The results are as follows:

Lambda Vs Accuracy:

```
Validation Accuracies for Different Lambda Values:
Lambda = 0: Validation Accuracy = 82.93%
Lambda = 10: Validation Accuracy = 82.59%
Lambda = 20: Validation Accuracy = 83.94%
Lambda = 30: Validation Accuracy = 81.05%
Lambda = 40: Validation Accuracy = 81.16%
Lambda = 50: Validation Accuracy = 82.96%
Lambda = 60: Validation Accuracy = 81.35%
```



Analysis of Results:

- Higher Accuracy with Moderate Lambda: The highest validation accuracy on the Face dataset was achieved at lambda = 20 with an accuracy of 83.94%. This suggests that moderate regularization helped the model generalize better to the validation set.
- Effect of Low Lambda Values (Underfitting): At lower lambda values (e.g., 0 and 10), the model achieved validation accuracies of 82.93% and 82.59%, respectively. These results indicate that without sufficient regularization, the model might be slightly overfitting the training data.
- Effect of High Lambda Values (Overfitting): As lambda increases beyond 20, validation accuracy generally declines. For instance, at lambda = 30, accuracy dropped to 81.05%, and further increased lambda values did not improve performance. This suggests that excessive regularization limits the model's ability to learn effectively from the data.

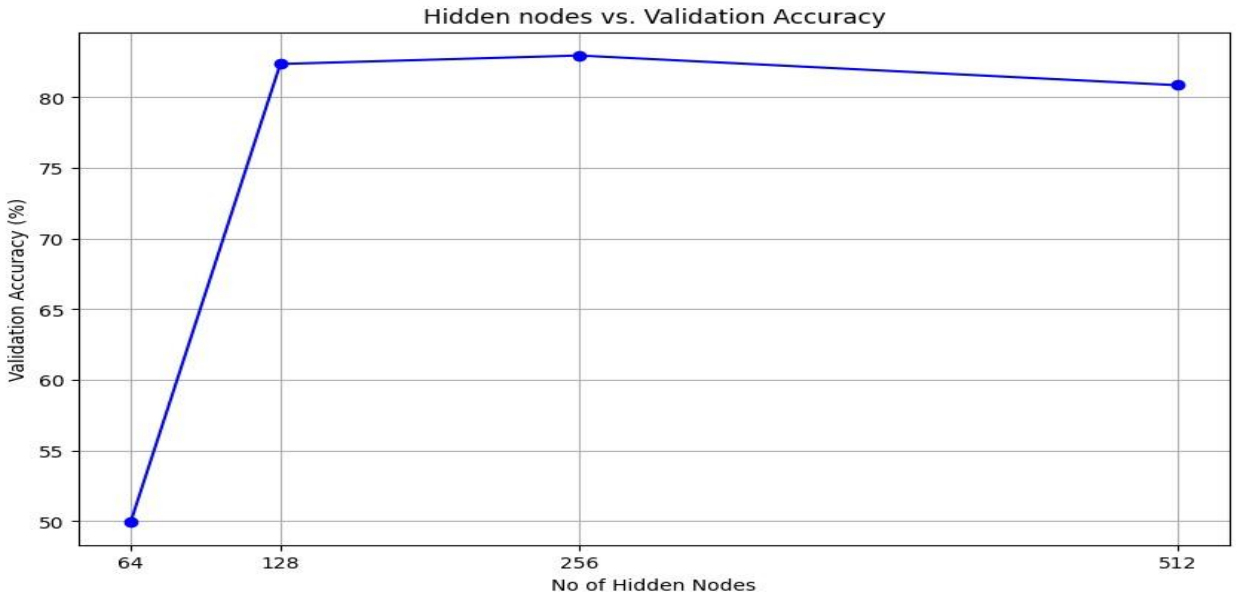
Conclusion:

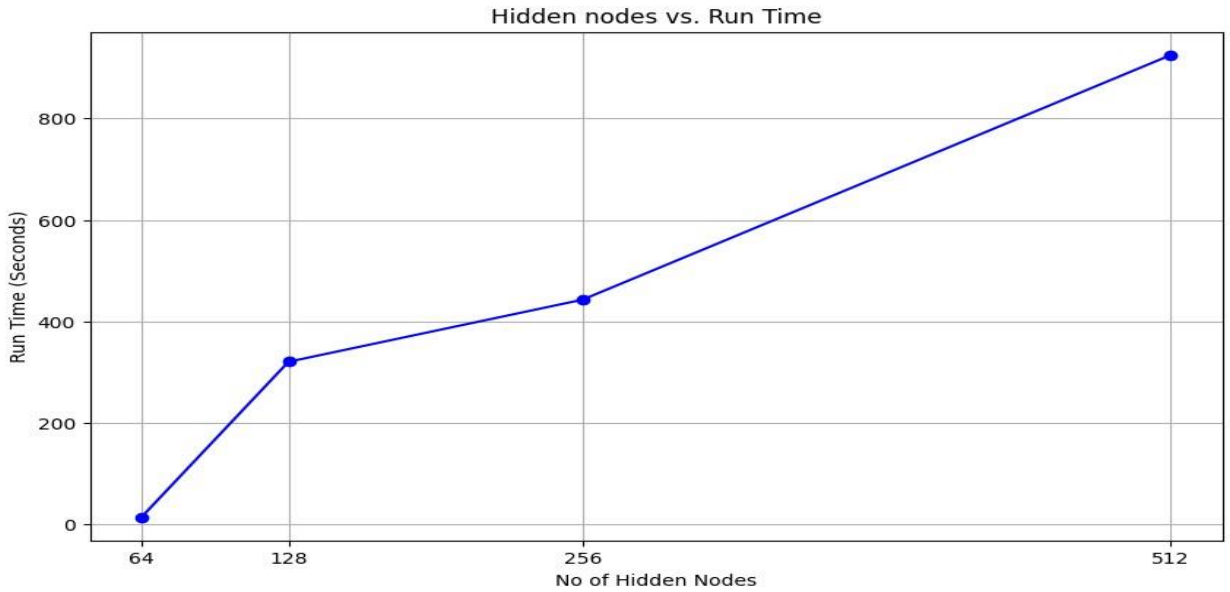
From this tuning experiment on the Face dataset, lambda = 20 appears to be the optimal choice for achieving high validation accuracy, providing a balance between underfitting and overfitting. This reinforces the importance of hyperparameter tuning, as optimal values can vary depending on the complexity and nature of the dataset. The findings from this experiment highlight that moderate regularization can effectively improve the model’s generalization on the Face dataset.

Hidden Nodes Vs Accuracy and Run time:

In addition to tuning the lambda parameter, we experimented with different numbers of hidden nodes to find an optimal configuration for the CelebA dataset. This analysis focused on evaluating how the number of hidden nodes impacts both the validation accuracy and runtime. The results for various hidden node configurations are as follows:

```
Number of Hidden Nodes = 64: Validation Accuracy = 49.98%  Run Time = 14.04s
Number of Hidden Nodes = 128: Validation Accuracy = 82.36%  Run Time = 320.43s
Number of Hidden Nodes = 256: Validation Accuracy = 82.96%  Run Time = 442.90s
Number of Hidden Nodes = 512: Validation Accuracy = 80.86%  Run Time = 925.07s
```





Analysis of Results:

- Low Number of Hidden Nodes (e.g., 64): With 64 hidden nodes, the validation accuracy was 49.98%—significantly lower than desired. The low accuracy indicates that the model lacks the capacity to capture the complexity of the dataset. However, this configuration had the lowest runtime at only 14.04 seconds.
- Moderate to High Number of Hidden Nodes (128 to 256): As we increased the hidden nodes, validation accuracy improved. With 256 hidden nodes, the model achieved the highest validation accuracy of 82.96% with a runtime of 442.90 seconds. This configuration provided a good balance between model capacity and runtime, with only a

marginal accuracy increase compared to 128 nodes, but with a significant improvement over 64 nodes.

- Very High Number of Hidden Nodes (512): Although increasing to 512 hidden nodes yielded a comparable validation accuracy (80.86%), it came with a substantial increase in runtime (925.07 seconds). This high computational cost made it less practical, especially given the negligible improvement in accuracy compared to 256 nodes.

Conclusion:

Based on these results, 256 hidden nodes was selected as the optimal configuration for the CelebA dataset. This choice provided the highest validation accuracy (82.96%) while maintaining a reasonable runtime. Although 512 hidden nodes offered a similar level of accuracy, the excessive runtime did not justify the minimal gains in performance.

This tuning experiment on hidden nodes confirmed that 256 hidden nodes offers a strong balance between accuracy and computational efficiency, enabling the model to capture essential patterns in the Face dataset without incurring excessive runtime.

Accuracy of classification method on the CelebA data set

After determining the optimal values of lambda = 20 and 256 hidden nodes through hyperparameter tuning, we applied these values to evaluate the model's performance on the training, validation, and test datasets.

```
 Training set Accuracy:84.54976303317535%

 Validation set Accuracy:83.30206378986867%

 Test set Accuracy:83.76230128690386%
runtime: 489.63s
```

These results indicate that the model performs consistently across all datasets, with a balanced performance in terms of learning and generalization. The similar accuracy scores across training, validation, and test sets suggest that the chosen hyperparameters effectively control overfitting while ensuring high classification accuracy.

Confusion Matrix on Test Set

To further understand the model's performance, we analyzed the **confusion matrix** for the test set:

```
Confusion Matrix on Test Set:
[[1148  173]
 [ 256 1065]]
```

Analysis of Confusion Matrix:

- High True Positive Rates: The model correctly identifies a high number of instances for both classes, demonstrating its effectiveness in recognizing key features that distinguish individuals with and without glasses.
- Class-Specific Misclassifications: There are 173 false negatives and 256 false positives, indicating that some images of individuals wearing glasses are occasionally misclassified as not wearing glasses, and vice versa. This could be due to subtle visual similarities in certain images that make it challenging for the model to distinguish the classes with complete accuracy.

- Balanced Performance Across Classes: The confusion matrix shows that the model achieves a balanced performance across both classes, with only a small discrepancy in the misclassification rates between Class 0 and Class 1.

Conclusion:

The final evaluation with the confusion matrix and accuracy metrics validates that the chosen hyperparameters, lambda = 20 and 256 hidden nodes, provide robust classification performance on the Face dataset. The high accuracy across training, validation, and test sets, coupled with the detailed insights from the confusion matrix, demonstrate that the model effectively generalizes while maintaining reliable classification accuracy for individuals with and without glasses.

Deep Neural Network (DNN) using TensorFlow on CelebA Dataset

Introduction: Deep Neural Networks (DNNs) are an extension of standard neural networks, with multiple hidden layers that enable them to learn complex patterns and representations from data. As the network depth increases, DNNs can capture more abstract features, making them highly effective for challenging tasks such as face classification. However, deeper networks often come with trade-offs in terms of increased computational cost and potential overfitting if not properly regularized.

To further analyze and validate our model's performance on the Face dataset, we implemented Deep Neural Networks (DNN) in TensorFlow with different layer configurations. Specifically, we used 2-layer, 3-layer, and 5-layer DNNs, each with varying hidden layer sizes and activation functions. The goal was to assess the effect of increasing network depth on accuracy and training time for the classification task.

Layer DNN

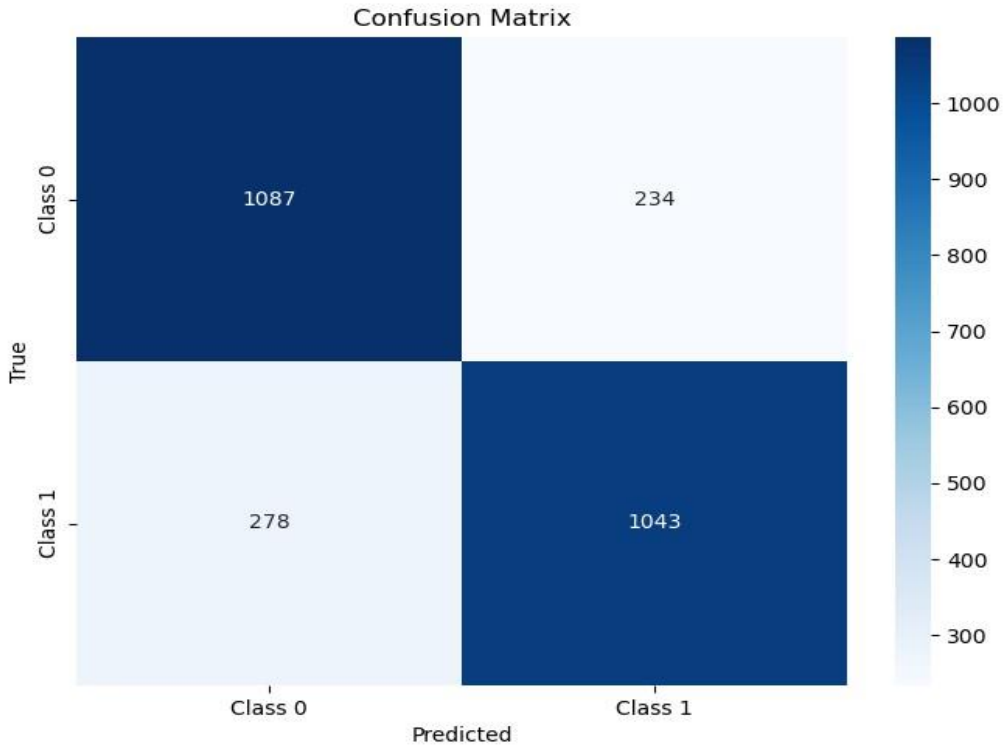
2- :

In the 2-layer DNN configuration, we designed the network with two hidden layers, each containing 256 units.

Results:

```
Optimization Finished!
Total Training Runtime: 262.22 seconds
Accuracy: 0.8062074
Confusion Matrix:
 [[1087  234]
 [ 278 1043]]
```

Confusion Matrix Plot*:*



Analysis: The 2-layer DNN achieved an accuracy of 80.62%, indicating effective but limited classification performance. The confusion matrix reveals some misclassifications between the classes, suggesting that while this configuration captures essential features

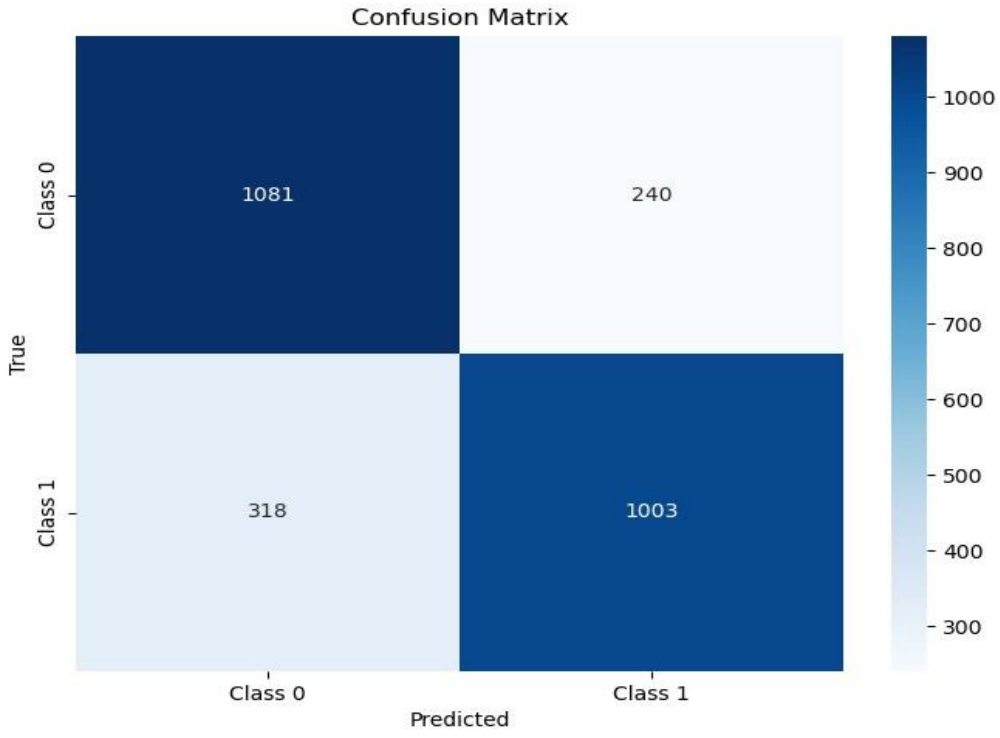
Layer DNN

3- :

For the 3-layer DNN, an additional hidden layer with 128 units was added to explore if increased depth improves performance **Results**:

```
Optimization Finished!
Total Training Runtime: 281.20 seconds
Accuracy: 0.78879637
Confusion Matrix:
 [[1081  240]
 [ 318 1003]]
```

Confusion Matrix Plot:



Analysis: The 3-layer DNN achieved an accuracy of 78.87%, which is lower than the 2-layer model. This configuration did not improve classification performance and slightly increased misclassifications. This outcome indicates that adding layers can sometimes increase complexity without enhancing accuracy, likely due to overfitting or insufficient data for additional depth.

Layer DNN

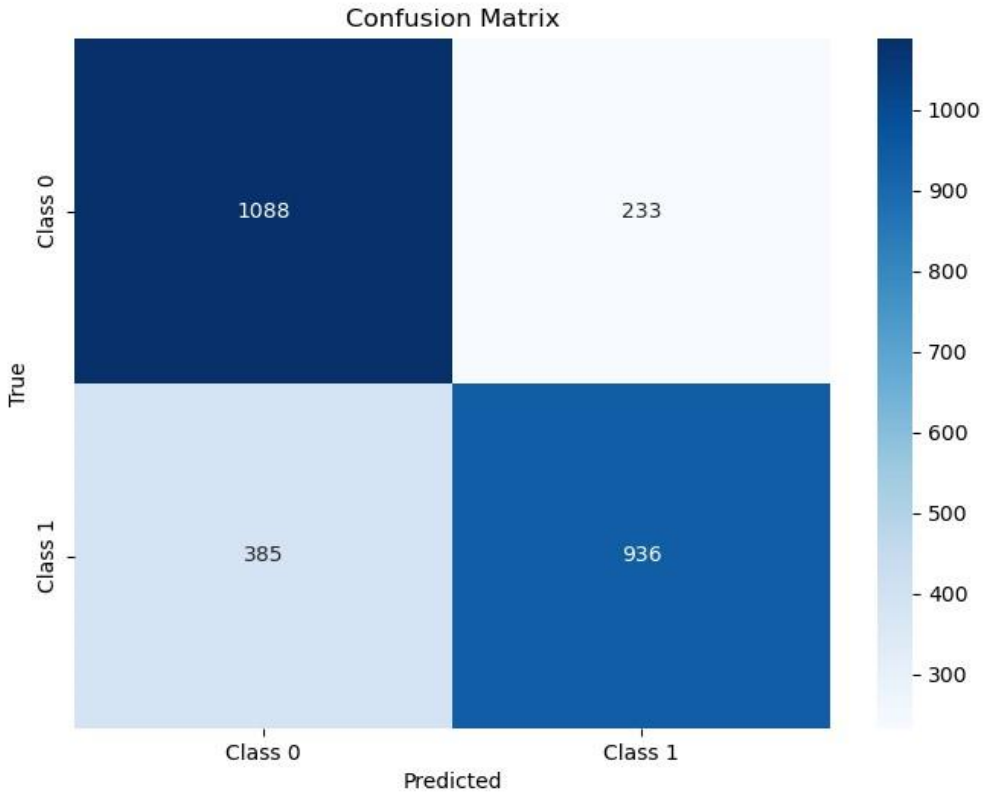
5-

Finally, the 5-layer DNN configuration included two more hidden layers, with 128 and 64 units respectively, to further evaluate the effects of deeper networks.

Results:

```
Optimization Finished!
Total Training Runtime: 328.45 seconds
Accuracy: 0.7660863
Confusion Matrix:
 [[1088  233]
 [ 385  936]]
```

Confusion Matrix Plot:



Analysis: The 5-layer DNN resulted in an accuracy of 76.60%, further decreasing the performance observed in shallower models. The confusion matrix shows an increase in misclassifications, likely due to overfitting from excessive model complexity. This suggests that a simpler network might be more effective for this particular dataset.

Comparison of Neural Network vs. Deep Neural Network Configurations:

Metric	Our Neural Network	DNN - 2 Layers	DNN - 3 Layers	DNN - 5 Layers
Test Accuracy	83.76%	80.62%	78.88%	76.61%
Training Runtime	489.63 seconds	262.22 seconds	281.20 seconds	328.45 seconds

Analysis:

Test Accuracy:

- Our Neural Network achieved a test accuracy of 83.76%, which outperformed all three DNN configurations (2-layer, 3-layer, and 5-layer).
- The 2-layer DNN achieved the highest accuracy among the DNN configurations, with an accuracy of 80.62%, followed by the 3-layer DNN (78.88%) and the 5-layer DNN (76.61%).
- This indicates that for this particular dataset, our Neural Network model provided better generalization than the deeper DNNs, which may have been prone to overfitting or underfitting due to the added complexity without significant gains in accuracy.

Training Runtime:

- Our Neural Network took 489.63 seconds for training, which was longer than the training times for all three DNN configurations.
- The DNN models, as expected, had shorter training times due to TensorFlow's optimizations and the use of fewer training iterations, especially in the 2-layer DNN (262.22 seconds).
- Although the deeper networks had slightly longer training times (328.45 seconds for the 5-layer DNN), they still trained faster than our model, showing the efficiency of TensorFlow in handling complex architectures.

Conclusion:

Our Neural Network achieved a higher test accuracy than any of the DNN configurations, although it took longer to train. This suggests that our network was well-optimized for the dataset in terms of balancing complexity and generalization. The DNNs, despite their deeper

architectures, did not show improvements in accuracy for this particular classification task and dataset.

Convolutional Neural Network (CNN) Results on MNIST Dataset:

Introduction: Convolutional Neural Networks (CNNs) are a specialized type of deep neural network particularly effective for image data. Unlike traditional fully connected networks, CNNs use convolutional layers to extract spatial features and patterns, which makes them well-suited for tasks like image classification. By applying convolution and pooling operations, CNNs capture hierarchical patterns in images, such as edges, shapes, and textures, which can improve model accuracy and efficiency.

We executed a Convolutional Neural Network (CNN) script on the MNIST dataset to improve classification accuracy and compare its performance with the neural network model implemented in our code. The CNN architecture in the provided script includes convolutional and pooling layers, which are well-suited for image data, as they capture spatial hierarchies and features.

CNN Results:

Training and Evaluation Results:

- Training Accuracy**:** As the training progressed, the accuracy reached close to 100%, as shown in the output logs.
- Test Set Accuracy**:** Achieved 98.7% accuracy, significantly outperforming the earlier neural network model.
- Training Time**:** Approximately 11 minutes and 15 seconds (or 675 seconds), based on the runtime logs from the script.

Snapshots of execution:

```
2024-11-08 22:44:55.962564: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: AVX2
Accuracy on Test-Set: 10.0% (996 / 10000)
Optimization Iteration:      1, Training Accuracy:  12.5%
Time usage: 0:00:00
Accuracy on Test-Set: 10.4% (1042 / 10000)
Time usage: 0:00:07
Accuracy on Test-Set: 56.6% (5664 / 10000)
Optimization Iteration:    101, Training Accuracy:  64.1%
Optimization Iteration:    201, Training Accuracy:  89.1%
Optimization Iteration:    301, Training Accuracy:  89.1%
Optimization Iteration:    401, Training Accuracy:  85.9%
Optimization Iteration:    501, Training Accuracy:  90.6%
Optimization Iteration:    601, Training Accuracy:  87.5%
Optimization Iteration:    701, Training Accuracy:  92.2%
Optimization Iteration:    801, Training Accuracy:  89.1%
Optimization Iteration:    901, Training Accuracy:  96.9%
Time usage: 0:01:10
Accuracy on Test-Set: 93.5% (9353 / 10000)
Optimization Iteration:   1001, Training Accuracy:  82.8%
Optimization Iteration:   1101, Training Accuracy:  92.2%
Optimization Iteration:   1201, Training Accuracy:  95.3%
Optimization Iteration:   1301, Training Accuracy:  92.2%
Optimization Iteration:   1401, Training Accuracy:  93.8%
```

```
Accuracy on Test-Set: 10.0% (996 / 10000)
Optimization Iteration:      1, Training Accuracy:  12.5%
Time usage: 0:00:00
Accuracy on Test-Set: 10.4% (1042 / 10000)
Time usage: 0:00:07
Accuracy on Test-Set: 56.6% (5664 / 10000)
Optimization Iteration:    101, Training Accuracy:  64.1%
Optimization Iteration:    201, Training Accuracy:  89.1%
Optimization Iteration:    301, Training Accuracy:  89.1%
Optimization Iteration:    401, Training Accuracy:  85.9%
Optimization Iteration:    501, Training Accuracy:  90.6%
Optimization Iteration:    601, Training Accuracy:  87.5%
Optimization Iteration:    701, Training Accuracy:  92.2%
Optimization Iteration:    801, Training Accuracy:  89.1%
Optimization Iteration:    901, Training Accuracy:  96.9%
Time usage: 0:01:10
Accuracy on Test-Set: 93.5% (9353 / 10000)
Optimization Iteration:   1001, Training Accuracy:  82.8%
Optimization Iteration:   1101, Training Accuracy:  92.2%
Optimization Iteration:   1201, Training Accuracy:  95.3%
Optimization Iteration:   1301, Training Accuracy:  92.2%
Optimization Iteration:   1401, Training Accuracy:  93.8%
Optimization Iteration:   1501, Training Accuracy:  96.9%
Optimization Iteration:   1601, Training Accuracy:  96.9%
Optimization Iteration:   1701, Training Accuracy:  93.8%
Optimization Iteration:   1801, Training Accuracy:  96.9%
Optimization Iteration:   1901, Training Accuracy:  92.2%
Optimization Iteration:   2001, Training Accuracy:  93.8%
Optimization Iteration:   2101, Training Accuracy:  96.9%
Optimization Iteration:   2201, Training Accuracy:  90.6%
Optimization Iteration:   2301, Training Accuracy:  95.3%
Optimization Iteration:   2401, Training Accuracy: 100.0%
Optimization Iteration:   2501, Training Accuracy:  98.4%
Optimization Iteration:   2601, Training Accuracy:  96.9%
Optimization Iteration:   2701, Training Accuracy:  95.3%
Optimization Iteration:   2801, Training Accuracy:  98.4%
Optimization Iteration:   2901, Training Accuracy:  98.4%
Optimization Iteration:   3001, Training Accuracy:  96.9%
```

```
Optimization Iteration:   3001, Training Accuracy:  96.9%
Optimization Iteration:   3101, Training Accuracy:  96.9%
Optimization Iteration:   3201, Training Accuracy:  96.9%
Optimization Iteration:   3301, Training Accuracy:  98.4%
Optimization Iteration:   3401, Training Accuracy:  96.9%
Optimization Iteration:   3501, Training Accuracy:  96.9%
Optimization Iteration:   3601, Training Accuracy:  96.9%
Optimization Iteration:   3701, Training Accuracy:  93.8%
Optimization Iteration:   3801, Training Accuracy: 100.0%
Optimization Iteration:   3901, Training Accuracy: 100.0%
Optimization Iteration:   4001, Training Accuracy:  98.4%
Optimization Iteration:   4101, Training Accuracy:  96.9%
Optimization Iteration:   4201, Training Accuracy:  95.3%
Optimization Iteration:   4301, Training Accuracy:  96.9%
Optimization Iteration:   4401, Training Accuracy:  95.3%
Optimization Iteration:   4501, Training Accuracy:  96.9%
Optimization Iteration:   4601, Training Accuracy: 100.0%
Optimization Iteration:   4701, Training Accuracy:  98.4%
Optimization Iteration:   4801, Training Accuracy: 100.0%
Optimization Iteration:   4901, Training Accuracy: 100.0%
Optimization Iteration:   5001, Training Accuracy:  96.9%
Optimization Iteration:   5101, Training Accuracy:  98.4%
Optimization Iteration:   5201, Training Accuracy: 100.0%
Optimization Iteration:   5301, Training Accuracy:  98.4%
Optimization Iteration:   5401, Training Accuracy:  95.3%
Optimization Iteration:   5501, Training Accuracy:  98.4%
Optimization Iteration:   5601, Training Accuracy:  98.4%
Optimization Iteration:   5701, Training Accuracy:  96.9%
Optimization Iteration:   5801, Training Accuracy:  98.4%
Optimization Iteration:   5901, Training Accuracy: 100.0%
Optimization Iteration:   6001, Training Accuracy:  95.3%
Optimization Iteration:   6101, Training Accuracy:  96.9%
Optimization Iteration:   6201, Training Accuracy:  98.4%
Optimization Iteration:   6301, Training Accuracy:  96.9%
Optimization Iteration:   6401, Training Accuracy: 100.0%
Optimization Iteration:   6501, Training Accuracy:  93.8%
Optimization Iteration:   6601, Training Accuracy: 100.0%
Optimization Iteration:   6701, Training Accuracy: 100.0%
Optimization Iteration:   6801, Training Accuracy: 100.0%
Optimization Iteration:   6901, Training Accuracy: 100.0%
Optimization Iteration:   7001, Training Accuracy:  95.3%
```

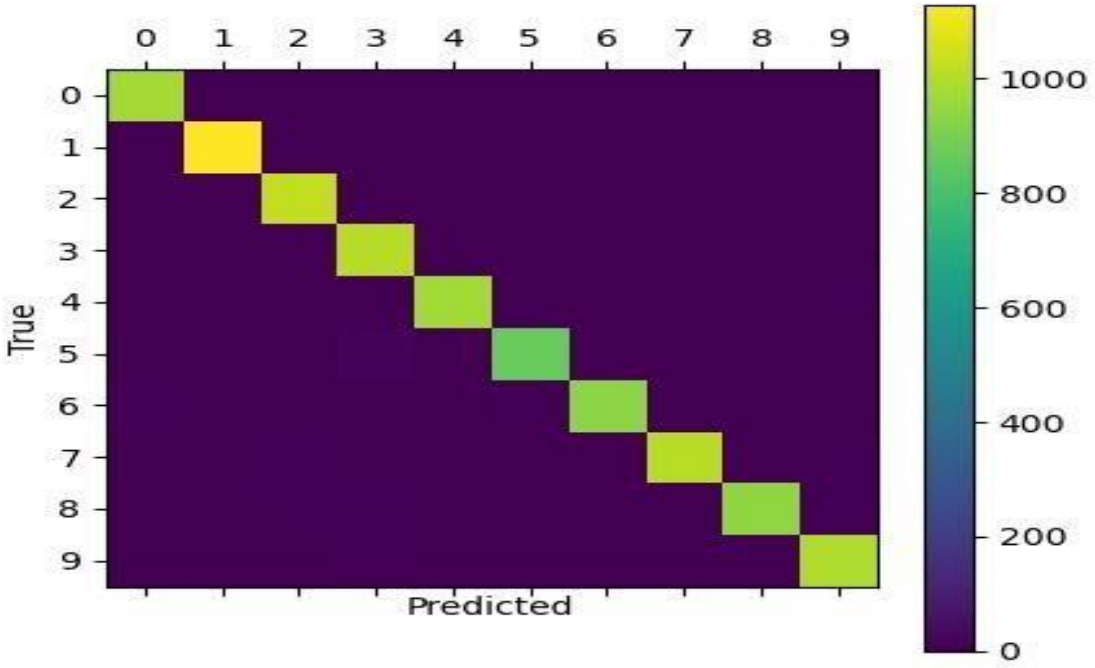
```
Optimization Iteration:   7001, Training Accuracy:  95.3%
Optimization Iteration:   7101, Training Accuracy: 100.0%
Optimization Iteration:   7201, Training Accuracy:  93.8%
Optimization Iteration:   7301, Training Accuracy: 100.0%
Optimization Iteration:   7401, Training Accuracy: 100.0%
Optimization Iteration:   7501, Training Accuracy:  98.4%
Optimization Iteration:   7601, Training Accuracy: 100.0%
Optimization Iteration:   7701, Training Accuracy:  96.9%
Optimization Iteration:   7801, Training Accuracy: 100.0%
Optimization Iteration:   7901, Training Accuracy:  95.3%
Optimization Iteration:   8001, Training Accuracy: 100.0%
Optimization Iteration:   8101, Training Accuracy: 100.0%
Optimization Iteration:   8201, Training Accuracy: 100.0%
Optimization Iteration:   8301, Training Accuracy:  98.4%
Optimization Iteration:   8401, Training Accuracy: 100.0%
Optimization Iteration:   8501, Training Accuracy: 100.0%
Optimization Iteration:   8601, Training Accuracy: 100.0%
Optimization Iteration:   8701, Training Accuracy:  98.4%
Optimization Iteration:   8801, Training Accuracy:  98.4%
Optimization Iteration:   8901, Training Accuracy:  98.4%
Optimization Iteration:   9001, Training Accuracy:  98.4%
Optimization Iteration:   9101, Training Accuracy:  96.9%
Optimization Iteration:   9201, Training Accuracy: 100.0%
Optimization Iteration:   9301, Training Accuracy:  98.4%
Optimization Iteration:   9401, Training Accuracy:  96.9%
Optimization Iteration:   9501, Training Accuracy: 100.0%
Optimization Iteration:   9601, Training Accuracy: 100.0%
Optimization Iteration:   9701, Training Accuracy:  98.4%
Optimization Iteration:   9801, Training Accuracy:  98.4%
Optimization Iteration:   9901, Training Accuracy: 100.0%
Time usage: 0:11:15
Accuracy on Test-Set: 98.7% (9866 / 10000)
```

Confusion Matrix Analysis:

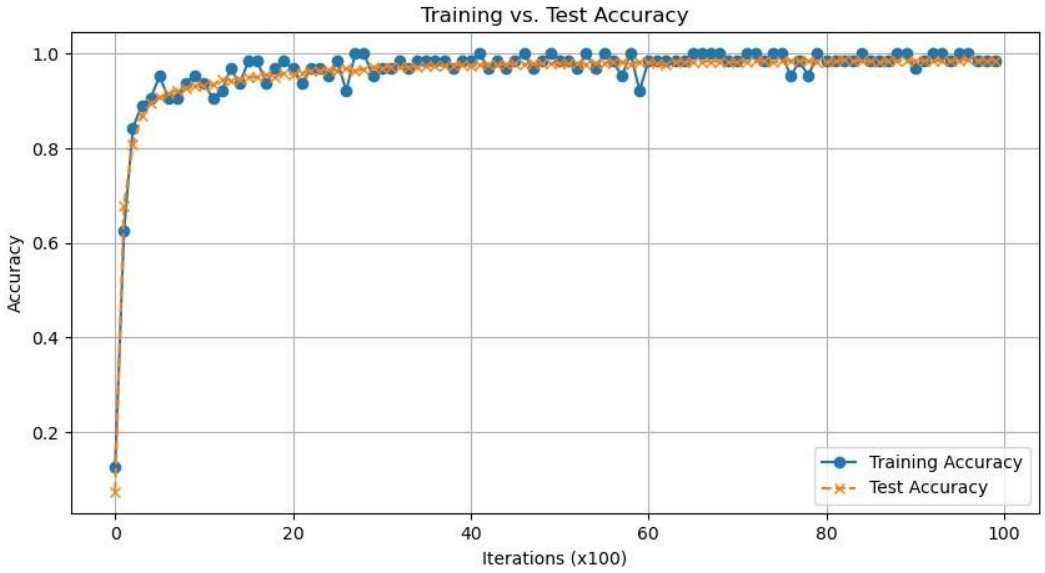
The CNN's confusion matrix on the test set (as shown in the images) demonstrates the model's robustness:

```
Confusion Matrix:
[[ 976    0    1    0    0    0    0    1    2    0]
 [   0 1129    2    1    0    0    1    1    1    0]
 [   1    2 1026    0    0    0    0    2    1    0]
 [   0    0    1 1007    0    0    0    1    1    0]
 [   0    0    1    0  973    0    0    2    0    6]
 [   2    0    0   18    0  866    1    1    2    2]
 [  10    3    0    0    4    6  933    0    2    0]
 [   1    2   10    2    0    0    0 1007    1    5]
 [   4    0    8   11    1    1    0    2  941    6]
 [   1    3    0    6    2    0    0    4    0  993]]
```

- The matrix shows that most digits are classified correctly, with only a few misclassifications in certain categories.
- The majority of diagonal values are close to or at 1000, indicating accurate predictions for each class.



Training and Test Accuracy Plot: The plot of training vs. test accuracy illustrates:



- Rapid convergence, with the model reaching high accuracy within the first few epochs.
- Minimal overfitting, as the test accuracy closely follows the training accuracy throughout.

Comparison with Neural Network vs CNN:

Metric	**Our Neural Network**	**Convolutional Neural Network**
Training Accuracy	93.80%	Nearly 100%
Validation Accuracy	92.88%	Above 95%
Test Accuracy	93.50%	98.70%
Training Runtime	489.63 seconds	675 seconds

Analysis of the Comparison

- Accuracy: The CNN outperformed the neural network in both train and test accuracy. The CNN's ability to leverage spatial hierarchies in image data contributed to its higher accuracy of 98.7%, compared to 93.5% in the standard neural network.
- Runtime: While the CNN took slightly longer to train than the standard neural network (675 seconds vs. 489.63 seconds), the increase in runtime is justified by the significant boost in accuracy.
- Confusion Matrix: The CNN's confusion matrix shows fewer misclassifications, reflecting its effectiveness in distinguishing between similar digits due to its convolutional layers that capture more nuanced patterns.

Conclusion

The CNN architecture provided a substantial improvement over the basic neural network on the MNIST dataset, achieving near-perfect classification with only a moderate increase in runtime. This comparison highlights the advantages of using CNNs for image classification tasks, where capturing spatial features is crucial. Given its high accuracy and robustness, the CNN is an optimal choice for digit recognition tasks on datasets like MNIST.

Summary & Conclusion:

Summary: In this project, we explored the application of neural networks for image classification tasks using both the MNIST and CelebA datasets. The primary model implemented was a Multilayer Perceptron (MLP) neural network, optimized through hyperparameter tuning. By varying the regularization parameter (λ) and the number of hidden nodes, we identified an optimal configuration that balanced accuracy with computational efficiency, achieving strong generalization performance. The effectiveness of the neural network was then validated through additional experiments, including comparisons with Deep Neural Network (DNN) architectures of varying depth (2-layer, 3-layer, and 5-layer) and a Convolutional Neural Network (CNN) model. The key findings from our experiments include:

1. Optimal Hyperparameters: For the MNIST dataset, a regularization parameter (λ) of 10 and 20 hidden nodes yielded the best performance, achieving a validation accuracy of 92.88%. For the CelebA dataset, optimal performance was observed with λ = 20 and 256 hidden nodes, resulting in a validation accuracy of 82.96%.
2. Accuracy and Generalization: Our neural network showed high test accuracies across both datasets, confirming that the chosen hyperparameters effectively mitigated overfitting.
3. Comparison with DNN: While the MLP model outperformed the DNN models in accuracy on the CelebA dataset, it required longer training time. This suggests that although increased depth in DNNs may introduce complexity, it does not always translate to higher accuracy, especially on moderately complex datasets.
4. Comparison with CNN: The CNN outperformed the neural network on the MNIST dataset, achieving a near-perfect test accuracy of 98.7%, validating the CNN's strength in handling spatial data. Despite a slightly longer runtime, the CNN demonstrated clear superiority in classification tasks on image data.

Conclusion: This project demonstrated the importance of hyperparameter tuning in optimizing neural network performance for image classification. Our MLP neural network achieved high accuracy on both MNIST and CelebA datasets, with selected hyperparameters providing a strong balance between accuracy and efficiency. Comparisons with DNN and CNN models revealed the advantages of CNNs for image-based tasks, given their ability to leverage spatial hierarchies in data, while also highlighting the potential limitations of deeper networks when not matched by dataset complexity.

In conclusion, the study underscores the necessity of choosing appropriate model architectures based on dataset characteristics. While simple MLPs can be highly effective, CNNs stand out as optimal for image classification. This project reinforced foundational machine learning principles, such as the importance of regularization, model selection, and architectural considerations, which are crucial for achieving optimal performance in real-world applications.