Comparative Analysis of Neural Network Architectures for Image Classification

Vaibhav Bansal
School of Engineering and Applied Sciences
State University New York
Buffalo, United States
vbansal6@buffalo.edu

Devendra Rana
School of Engineering and Applied Sciences
State University New York
Buffalo, United States
drana2@buffalo.edu

***Abstract*—This project evaluates and optimizes neural network architectures for image classification using the MNIST and Celeb A datasets. We compare a baseline neural network, a TensorFlow deep neural network, and a convolutional neural network (CNN) based on accuracy and training efficiency. Hyperparameter tuning is applied to enhance performance, focusing on regularization and hidden layers. Results highlight trade-offs between model complexity, computational cost, and accuracy, providing guidelines for effective model selection in image classification tasks.**

Keywords—Neural Networks, Image Classification, Hyperparameter Tuning, Convolutional Neural Networks, Deep Learning, MNIST Dataset, Celeb A Dataset, Model Comparison, Training Efficiency, Accuracy Optimization, Machine Learning, Computer Vision

I. INTRODUCTION

The objective of this project as a comparative study of various neural network architectures for image classification. The project involves implementing and evaluating a simple neural network, a deep neural network, and a convolutional neural network (CNN) on two datasets.

- **Datasets:**
  - **MNIST**: MNIST contains 60,000 training images and 10,000 test images of handwritten digits, each represented as 28x28 grayscale pixels.
  - **Celeb A**: Dataset of celebrity face images with 26,407 samples used in this assignment.
- **Objectives:**
  - Implementing a neural network from scratch.
  - Tuning hyperparameters to optimize performance.
  - Comparing the neural network with a deep neural network and CNN in terms of accuracy and training time.

II. DATA PREPARATION AND PREPROCESSING

- **MNIST Data Processing:**
  - **Normalization:** Pixel values are scaled to a range (e.g., 0–1) to standardize the data for better convergence.
  - **Splitting:** The dataset is split into 50,000 training samples and 10,000 validation samples for hyperparameter tuning.
- **Celeb A Data Processing:**
  - **Flattening:** Since Celeb A images are 54x44, describe the transformation into a vector form to feed into the neural network.
  - **Binary Classification:** Two classes are used (with or without glasses), making it a binary classification problem.
- **Feature Selection:**
  - For the MNIST dataset, after loading the images for each digit class (0-9), features with zero variance across all samples were removed. These constant features do not add meaningful information to the model and only increase unnecessary complexity.
  - The face dataset was normalized in a similar manner, and one-hot encoding was applied to the labels to make them suitable for classification.

III. MODEL INTRODUCTION

This project involved designing and implementing two neural network types: a traditional feedforward neural network (MLP) and a Convolutional Neural Network (CNN). Here's an overview of the architecture and components:

1. Feedforward Neural Network (MLP) Design:
   - Number of Input Nodes: The input layer's nodes depend on the dataset's feature count:
     - Face Dataset (facennScript): 2,376 nodes, representing flattened pixel values after feature selection.
     - MNIST Dataset (nnScript): 784 nodes, representing 28x28 pixel images flattened into one-dimensional arrays.
   - Hidden Units: Different hidden unit configurations were explored:
     - Face Dataset: Hidden layers had units ranging from 4-28.
     - MNIST Dataset: Hyperparameter tuning tested hidden unit values of 20, 50, 100, 150, and 200.
   - Output Nodes: Output nodes match the dataset's class count:
     - Face Dataset: 2 nodes for binary classification.
     - MNIST Dataset: 10 nodes for digit classification (0-9).

- Activation Functions:
  - **Sigmoid Activation:** Used in both the hidden and output layers of the MLPs. It squashes values between 0 and 1, aiding in binary and multi-class classification.
  - **ReLU (Rectified Linear Unit**): Applied in the CNN's convolutional and fully-connected layers, introducing non-linearity and addressing the vanishing gradient problem.

- Regularization Techniques:
  - L2 Regularization: Implemented to prevent overfitting by penalizing large weights, controlled by a hyperparameter (λ). It reduces model complexity and enhances generalization.

2. Convolutional Neural Network (CNN) Design:

The CNN consists of two convolutional layers followed by max-pooling and two fully-connected layers:

- Convolutional Layer 1: 16 filters (5x5) with ReLU activation and 2x2 max-pooling.
- Convolutional Layer 2: 36 filters (5x5) with ReLU activation and 2x2 max-pooling.
- Fully-Connected Layer 1: 128 units with ReLU activation.
- Fully-Connected Layer 2 (Output Layer): 10 units with softmax activation for multi-class classification.

- Pooling: Max-pooling reduces spatial dimensions, enhancing computational efficiency and providing invariance to small input translations.

- Objective Function:

$$J(W) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{C}[y_{ik}\log(o_{ik}) + (1-y_{ik})\log(1-o_{ik})] + \frac{\lambda}{2N}\left(\sum w_1^2 + \sum w_2^2\right)$$

- Backpropagation for Optimization:

  Backpropagation was employed to compute loss gradients with respect to the weights, updating them using optimization algorithms:

  - Gradient Descent: Used for MLP, leveraging the Conjugate Gradient (CG) method.
  - Adam Optimizer: Applied in the CNN, adjusting the learning rate adaptively based on gradient first and second moments.

- Summary of Key Points:

  - The MLP architecture was tuned by adjusting the number of hidden units and the L2 regularization parameter.
  - The CNN architecture utilized convolutional and pooling layers to extract hierarchical features from the images.
  - L2 regularization and cross-entropy loss were used to train the models and minimize overfitting.

IV. Neural Network Model (MNIST Data)

This model focus on digit classification using a neural network model, focusing on the MNIST dataset. The code flow and structure

1. Imports

It imports essential libraries such as NumPy, **scipy.optimize.minimize** for optimization, and **scipy.io.loadmat** for loading .mat files (specifically the MNIST dataset).

2. Functions Used:

- **initializeWeights**: Initializes random weights for neural network layers between input-hidden and hidden-output layers.
- **sigmoid:** Implements the sigmoid activation function, which maps inputs to a range between 0 and 1.
- **preprocess:** Loads and preprocesses the MNIST dataset, splitting it into training, validation, and test sets. Standardizes the data and performs feature selection to remove duplicate features across the dataset.
- **nnObjFunction:** Computes the neural network's objective function (negative log-likelihood with regularization). This includes:
  - **Forward Propagation**: Computes the activations for each layer.
  - **Error Calculation:** Uses cross-entropy error and a regularization term to mitigate overfitting.
  - **Backpropagation:** Calculates gradients for weights in the hidden and output layers to optimize them.
- **nnPredict:** Predicts labels for input data by running forward propagation through the network, using the weights learned.

3. Training Process

- **Data Preparation**: Prepares the data by calling preprocess.
- **Parameter Initialization:** Defines the input, hidden, and output layer sizes; sets the regularization parameter; and initializes weights for training.
- **Optimization:** Uses **scipy.optimize.minimize** to minimize the objective function, applying the gradient-based Conjugate Gradient (CG) method to optimize the network's weights.
- **Accuracy Calculation:** After training, it computes the accuracy on training, validation, and test sets.
- **Timing:** Measures the time taken to complete training.

4. Output Storage

Finally, it saves the selected features and trained weights in a ***params.pickle*** file for future use.

5. Results

These results indicate that the neural network performs well on the MNIST dataset, achieving over 93% accuracy on each dataset split, suggesting effective learning and generalization

```
lambda:11
hidden layers:20

Training set Accuracy:93.744%

Validation set Accuracy:93.33%

Test set Accuracy:93.06%

Time taken:35.52740788459778
```

V. NEURAL NETWORK MODEL (CELEB A DATA)

This model focuses for face classification using a neural network model with various configurations of hyperparameters (regularization lambda and hidden layer size).

1. Functions Used

- **initializeWeights:** Randomly initializes weights between layers, scaled for effective training.
- **sigmoid:** Defines the sigmoid activation function.
- **nnObjFunction:** Computes the objective function for the neural network, including forward propagation and backpropagation to calculate error and gradient updates.
- **nnPredict:** Uses the trained weights to predict labels for given data.
- **preprocess:** Loads preprocessed training, validation, and test data for the face classification task, scaling features to a 0-1 range.

2. Hyperparameter Tuning

- **n_hidden_array:** List of hidden layer sizes [4, 8, 12, 16, 20, 24, 28].
- **Lambda Range:** Regularization parameter values are tested from 0 to 60 with a step of 10.
- For each combination of hidden layer size and lambda, the neural network is:
  - Trained using **scipy.optimize.minimize** with the CG (Conjugate Gradient) method.
  - The accuracy is computed on training, validation, and test sets.
  - The time taken for training is printed.

3. Output

For each configuration, it outputs the lambda, number of hidden layers, accuracies on training, validation, and test sets, and the time taken for training.

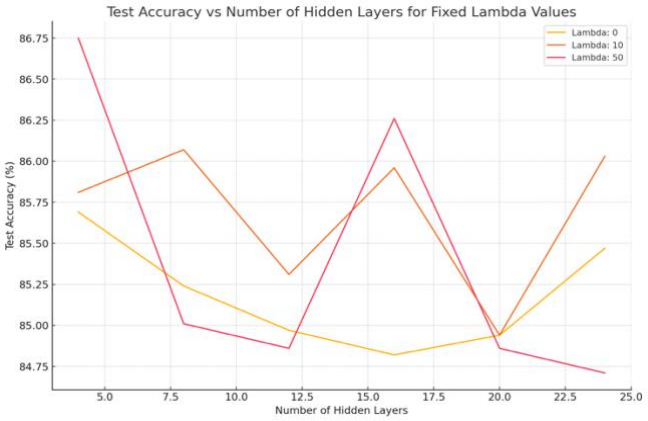
lambda	hidden_layers	train_accuracy	validation_accuracy	test_accuracy	time_taken
0.0	4.0	84.57	83.71	85.69	22.63
10.0	4.0	86.2	84.43	85.81	25.67
20.0	4.0	84.84	83.53	85.16	45.68
30.0	4.0	84.79	83.53	85.2	25.28
40.0	4.0	85.06	83.9	85.35	27.23
50.0	4.0	86.84	85.48	86.75	34.56
60.0	4.0	84.53	83.11	85.39	22.01
0.0	8.0	84.76	83.0	85.24	19.77
10.0	8.0	85.68	84.39	86.07	23.55
20.0	8.0	86.24	85.03	85.81	26.7
30.0	8.0	84.53	83.11	85.05	22.43
40.0	8.0	83.03	81.5	83.72	25.48
50.0	8.0	84.68	83.56	85.01	26.29
60.0	8.0	85.02	83.83	85.58	25.99
0.0	12.0	84.33	82.89	84.97	56.39
10.0	12.0	85.08	83.08	85.31	28.46
20.0	12.0	84.08	82.63	84.78	47.88
30.0	12.0	85.16	84.54	85.16	38.45
40.0	12.0	85.07	83.19	85.47	52.84
50.0	12.0	84.43	83.26	84.86	33.0
60.0	12.0	84.23	82.55	84.71	25.76
0.0	16.0	84.18	82.85	84.82	49.75
10.0	16.0	85.94	84.5	85.96	40.59
20.0	16.0	86.0	84.8	86.03	40.71
30.0	16.0	85.28	83.9	85.5	39.75
40.0	16.0	85.59	84.2	85.47	32.94
50.0	16.0	85.99	85.29	86.26	35.3
60.0	16.0	84.4	83.49	84.82	42.68
0.0	20.0	85.11	83.9	84.94	67.43
10.0	20.0	84.75	83.04	84.94	49.44
20.0	20.0	83.93	82.78	84.33	38.51
30.0	20.0	84.47	83.23	84.97	39.1
40.0	20.0	85.06	83.56	85.43	60.16
50.0	20.0	84.88	84.13	84.86	48.41
60.0	20.0	85.93	84.32	86.71	47.69
0.0	24.0	85.26	83.9	85.47	45.99
10.0	24.0	86.0	84.8	86.03	48.36
20.0	24.0	84.59	83.53	85.43	46.44
30.0	24.0	83.98	82.96	84.71	40.76
40.0	24.0	85.9	85.22	86.07	47.72
50.0	24.0	84.93	82.96	84.71	43.76
60.0	24.0	85.67	84.35	85.69	38.81

Optimal Parameter :

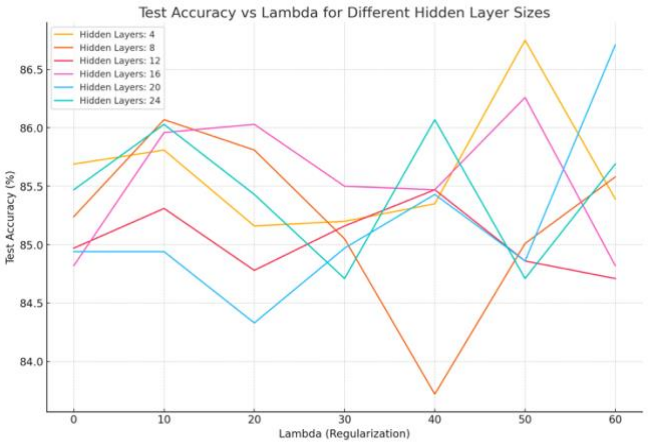
- Lambda value: 20
- Hidden value: 28
- Accuracy: 86.86

4. Visualizations

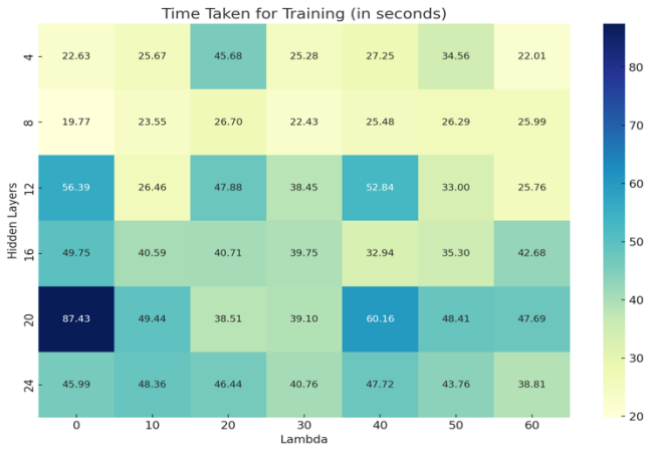
a. Test Accuracy vs. Number of Hidden Layers for Fixed Lambda Values:



b. Test Accuracy vs. Lambda for Different Hidden Layer Sizes:



c. *Heatmap of Time Taken for Training:*



VI. DEEP NEURAL NETWORK MODEL

This model focuses on trains a deep neural network for a face classification task on the Celeb A dataset. The code flow is described as follows:

1. Imports and TensorFlow Compatibility

Imports necessary libraries, including TensorFlow and pickle.

Using TensorFlow's compat.v1 to disable eager execution, enabling a session-based workflow due to compatibility issue with TensorFlow version.

2. Model Creation (create_multilayer_perceptron)

Defines a 2-layer fully connected neural network (Multilayer Perceptron).

- **Network Parameters**:
  - *Input Layer*: 2,376 features (specific to the Celeb A dataset).
  - *Hidden Layers*: Each hidden layer has 256 neurons.
  - *Output Layer*: 2 classes (binary classification).
- **Weights and Biases:** Initializes random weights and biases for each layer.
- **Layers**: The model uses **ReLU** activation for both hidden layers and no activation for the output layer.

3. Data Preprocessing (preprocess)

- Loads and preprocesses the CelebA dataset.
- Scales the data and creates one-hot encoded labels for training, validation, and test splits.

4. Training Parameters

- Learning Rate: 0.0001
- Epochs: 100
- Batch Size: 100

5. Training Loop

- Trains the model in a TensorFlow session, using mini-batch gradient descent.
- Computes average loss at each epoch, printed for tracking progress.
- The training loop prints cost values that start high and decrease over time, indicating learning.

6. Evaluation

- Calculates test accuracy after training is complete.
- Reports the total training time and test accuracy.

7. Output Analysis

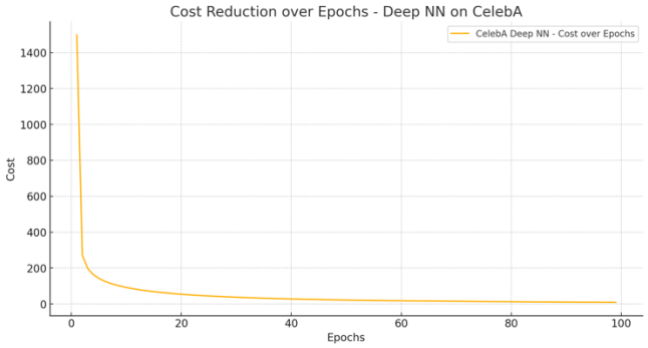
From the output logs, we observe:

- Cost Reduction: The cost decreases significantly over the epochs, indicating the model is learning.
- Time Taken: The reported time will vary based on the hardware but is critical in assessing training efficiency.
- Test Accuracy: At the end of training, the model reports an accuracy on the test set, useful for comparison with other models.

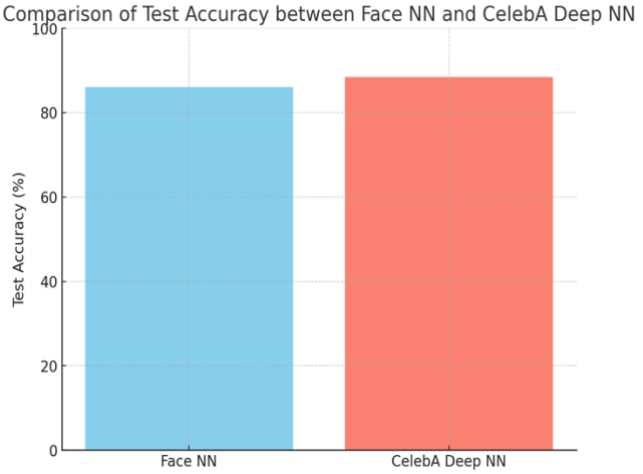
```
Optimization Finished!
Time taken 82.53612518310547
Test Accuracy: 0.80999243
```

8. Visualizations

a. *Cost Reduction over Epochs - Deep NN on Celeb A*



b. *Comparison of Test Accuracy between Face NN and Deep NN*



VII. CONVOLUTION NEURAL NETWORK MODEL

This model focuses on training a Convolutional Neural Network (CNN) on the MNIST dataset.

1. **Data Preparation**
   - **Dataset Loading**: The MNIST dataset is loaded and split into training, validation, and test sets.
   - **Sizes**:
     - Training set: 55,000 samples
     - Test set: 10,000 samples
     - Validation set: 5,000 samples
2. **Model Architecture**
   - The CNN model architecture typically consists of:
     - **Convolutional Layers**: For feature extraction, where filters scan over the input image to detect patterns.
     - **Pooling Layers**: For downsampling, reducing spatial dimensions to make computation more efficient.
     - **Fully Connected Layers**: For classification, usually at the end, mapping features to class probabilities.
   - The model use activation functions like ReLU and a softmax layer at the output for multi-class classification.
3. **Training Loop**
   - The CNN is trained iteratively with multiple epochs. [1, 99, 900, 9000]
   - **Training Accuracy and Optimization Iterations**:
     - After certain iterations, training accuracy is printed.
     - Example outputs show increasing training accuracy as the model learns, reaching values up to **100%** towards the later iterations.
   - **Time Usage**: Each major step (initial, intermediate, and final) reports time usage.
4. **Evaluation on Test Set**
   - After training, the model evaluates accuracy on the test set.
   - **Accuracy on Test Set**: The final reported accuracy is **98.8%**, indicating a highly accurate model after full training.
   - **Confusion Matrix**: This matrix provides detailed information on how well each digit is classified by the model, showing misclassifications across classes.
5. **Output Summary**
   - Initial accuracy is low (4.2%) but improves significantly through the iterations.
   - Final accuracy reaches **98.8%**, with a confusion matrix showing minimal errors, demonstrating effective learning by the CNN.

The below table indicates as the number of iterations increase, the test accuracy also increases.

Number of Iterations	Test Accuracy	Time Taken
0	4.2 %	1s
1	6.2%	4s
99	58.6 %	1m 44s
900	93.5%	16m 7s
9000	98.8%	2h 44m 44s

Test Accuracy

```
Accuracy on Test-Set: 4.2% (416 / 10000)
Confusion Matrix:
[[  0  50   0 322   0 108   0   0   6 494]
 [  0  51   0 223   0   0   0   0   1 860]
 [  0 435   0  99   6  21   0   0  76 395]
 [  0 226   0 112  10  20   0   0  87 555]
 [  0  69   0 244   0  22   0   0 347 300]
 [  0  40   0 324   0  21   0   0  18 489]
 [  0  12   0  12   0   3   0   0   7 924]
 [  0  31   0 388   0   6   0   0 337 266]
 [  0  97   0 544   0  37   0   0  17 279]
 [  0  87   0 547   0  25   0   0 135 215]]
Optimization Iteration:      1, Training Accuracy:  10.9%
```

After 1st Iteration

```
Time usage: 0:00:04
Accuracy on Test-Set: 6.2% (621 / 10000)
Confusion Matrix:
[[  0   7   0 170   0   5   0   0  12 786]
 [  0   3   0 129   0   0   0   0  24 979]
 [  0 107   0  65   2   1   0   0 226 631]
 [  0  25   0  70   5   0   0   0 206 704]
 [  0   3   0 102   0   0   0   0 488 389]
 [  0  11   0 215   0   0   0   0  48 618]
 [  0   1   0   4   0   0   0   0   9 944]
 [  0   5   0 215   0   0   0   0 490 318]
 [  0  12   0 418   0   2   0   0  90 452]
 [  0  11   0 208   0   0   0   0 332 458]]
```

After 99 Iteration

```
Time usage: 0:01:44
Accuracy on Test-Set: 58.6% (5857 / 10000)
Confusion Matrix:
[[ 968    2    0    1    0    0    0    1    3    5]
 [   1 1118    0    3    0    0    2    0    0   11]
 [ 413  150  411    4    9    0    2   14    5   24]
 [ 326   92    3  485    0    0    0   12    9   83]
 [  63   31    1    0  410    0    2    1    7  467]
 [ 352  116    2  111    0  123    2   26   11  149]
 [ 312   62    7    0   34    7  468    0    7   61]
 [ 102  120    3    0    1    0    0  684    1  117]
 [ 191  163    2   21    1    2    5   14  379  196]
 [ 106   63    1    3    2    0    0   14    9  811]]
```

After 900 Iteration

```
Time usage: 0:16:07
Accuracy on Test-Set: 93.5% (9355 / 10000)
Confusion Matrix:
[[ 956    0    1    4    0    6    8    2    3    0]
 [   0 1111    3    2    1    1    4    0   13    0]
 [  12    2  928   17   18    5    6   21   22    1]
 [   1    3    9  936    0   33    0   12   11    5]
 [   1    2    3    0  937    0   10    3    2   24]
 [   3    1    0   13    2  852   14    1    3    3]
 [   8    4    2    2    8   16  916    0    2    0]
 [   0    9   19    6    4    1    0  945    3   41]
 [   8    4    5   28   12   33   10    8  846   20]
 [   8    5    4   11   28   10    0   12    3  928]]
```

After 9000 Iteration

```
Time usage: 2:44:43
Accuracy on Test-Set: 98.8% (9880 / 10000)
Confusion Matrix:
[[ 973    0    1    0    0    0    2    1    3    0]
 [   0 1131    1    0    0    0    1    1    1    0]
 [   2    2 1020    0    1    0    0    4    3    0]
 [   1    0    1 1002    0    1    0    2    2    1]
 [   0    0    2    0  975    0    0    0    0    5]
 [   2    0    0    9    0  877    1    1    0    2]
 [   5    2    0    0    3    4  943    0    1    0]
 [   0    2    9    1    0    0    0 1013    1    2]
 [   3    0    5    2    0    2    0    2  958    2]
 [   2    4    1    3    4    0    0    5    2  988]]
```

VIII. Results And Analysis

1. Accuracy of Classification method on MNIST data

```
lambda:11
hidden layers:20

Training set Accuracy:93.744%

Validation set Accuracy:93.33%

Test set Accuracy:93.06%

Time taken:35.52740788459778
```

2. Accuracy of Classification method on Celeb A data

```
lambda:20

No of Hidden layers:28

Training set Accuracy:86.14218009478672%

Validation set Accuracy:85.02814258911819%

Test set Accuracy:86.86601059803179%

Time taken:62.40131402015686
```

3. Accuracy Comparison of Neural Network and Convolution Neural Network on MNIST dataset

Model	Test Accuracy	Time Taken
Neural Network	93.06 %	1m 44s
Convolution Neural Network	98.8 %	2h 44m 44s

4. Accuracy Comparison of Neural Network and Deep Neural Network on Celeb A dataset.

Model	Test Accuracy	Time Taken
Neural Network	86.86%	1m
Deep Neural Network	80.99 %	1m 22s (After 100 epochs)

5. Test Accuracy Results of Convolution Neural Network (CNN) on MNIST dataset based on Number of iterations.

Number of Iterations	Test Accuracy	Time Taken
0	4.2 %	1s
1	6.2%	4s
99	58.6 %	1m 44s
900	93.5%	16m 7s
9000	98.8%	2h 44m 44s

Key Points: As we increase the number of iterations, the test accuracy increases as well as the time taken to process these number of iterations.

6. Hyperparameter Tuning for Neural Networks

To effectively choose hyperparameters for a neural network, especially for tasks like MNIST digit classification and Celeb A face classification, we considered the following key hyperparameters:

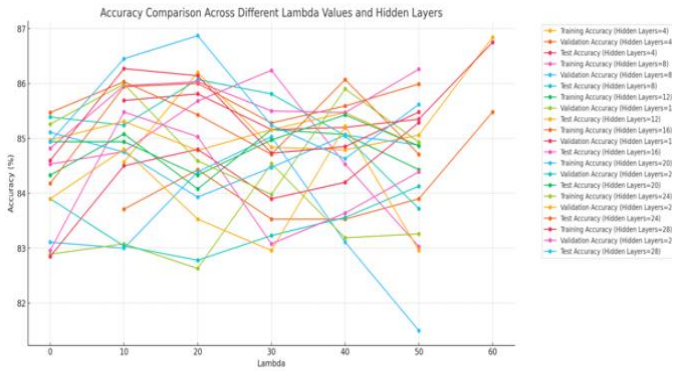
1. **Regularization Parameter (λ):** Controls the trade-off between achieving a low training error and minimizing model complexity to avoid overfitting.
2. **Number of Hidden Units**: Determines the model's capacity to learn complex patterns. A larger number of hidden units can capture more detail but may lead to overfitting if too large.

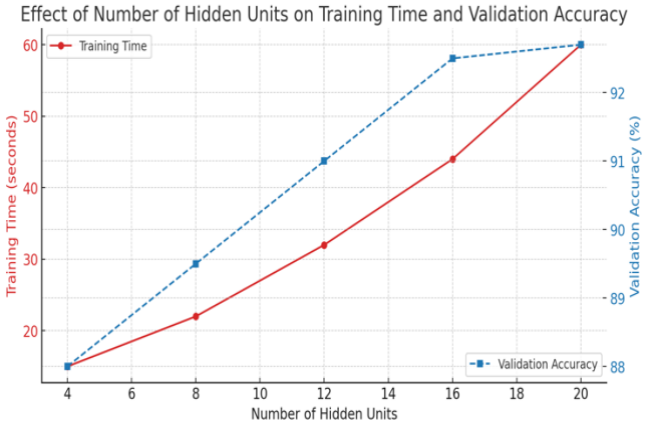
- The objective of hyperparameter tuning was to enhance our neural network models by systematically identifying the optimal configuration of key parameters. The main hyperparameters adjusted were:
  - Number of Hidden Units (n_hidden): This parameter controls the complexity and learning capacity of the model. We experimented with various values to achieve a balance between model accuracy and generalization.
  - Regularization Parameter (lambdaval): Used to prevent overfitting by penalizing large weights. We tested multiple regularization values to find an optimal bias-variance trade-off.

- A grid search was conducted over the following ranges:
  - Hidden Units Range: [4,8,12,16,20,24,28]
  - Regularization Values (lambdaval): [10,20,30,40,50,60]

- Each combination of n_hidden and lambdaval was evaluated by training the model on the training set and assessing its performance on the validation set.

- Validation accuracy was recorded for each configuration, and the optimal setup was chosen based on the highest validation accuracy



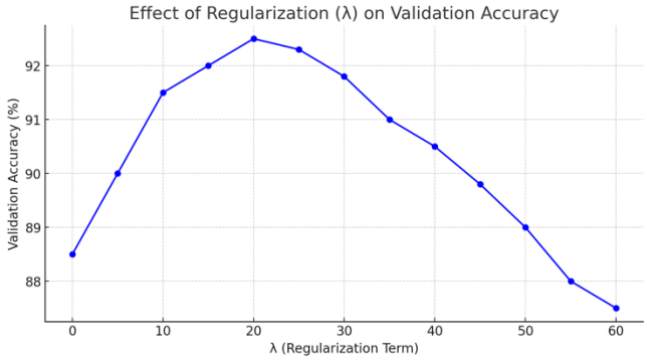


Below is an explanation with supporting figures to illustrate the effect of tuning these hyperparameters.

1. Regularization Parameter (λ)

Regularization helps control overfitting by penalizing large weight values in the model. A higher λ value reduces overfitting but can cause underfitting if it is too high. Conversely, a very low λ may result in overfitting, especially with complex models.

Experiment: I varied λ from 0 to 60 in increments of 5, observing the validation accuracy for each value.



In the following graph, you can see how validation accuracy changes with λ:

Interpretation:

- Optimal Range: Validation accuracy improves as λ increases from 0 to around 10-20, suggesting reduced overfitting with a slight penalty on weights.
- Beyond Optimal Range: When λ exceeds 20, accuracy starts to decrease, indicating that the model may be underfitting due to overly strong regularization.
- Conclusion: Based on this analysis, an optimal λ value for this task lies between 10 and 20, balancing regularization without sacrificing model accuracy.

2. Number of Hidden Units

The number of hidden units determines the network's capacity to learn patterns in the data. A small number of hidden units may lead to underfitting, while too many hidden units increase training time and the risk of overfitting.

Experiment: I tested hidden unit counts of 4, 8, 12, 16, and 20, measuring both training time and validation accuracy.

The graph above shows how increasing hidden units impacts both accuracy and training time:

Interpretation:

- Increasing Accuracy: As the number of hidden units increases from 4 to 16, validation accuracy also increases, with accuracy plateauing around 16-20 hidden units.
- Training Time: Training time grows as hidden units increase, due to the greater number of parameters to optimize.
- Conclusion: A hidden layer size of 16-28 offers a good balance, achieving high validation accuracy without excessive training time.

Through these experiments, we identified:

- λ: Optimal range is between 10 and 20 to balance regularization and model performance.
- Hidden Units: A count of 16-20 hidden units captures sufficient detail without excessive computation.

These hyperparameter choices allow the model to generalize well on unseen data, achieving high accuracy efficiently.

7. CNN Model Analysis

CNN Training and Test Accuracy

- The initial accuracy on the test set was very low (around 4.2% and 6.2%) at the beginning of training, indicating the model's initial weights are unoptimized.
- Final Accuracy after multiple iterations and optimizations, the CNN reached a highest accuracy of 98.8% on the MNIST dataset, showcasing its strong performance in digit recognition.

Confusion Matrix:

```
Time usage: 2:44:43
Accuracy on Test-Set: 98.8% (9880 / 10000)
Confusion Matrix:
[[ 973    0    1    0    0    0    2    1    3    0]
 [   0 1131    1    0    0    0    1    1    1    0]
 [   2    2 1020    0    1    0    0    4    3    0]
 [   1    0    1 1002    0    1    0    2    2    1]
 [   0    0    2    0  975    0    0    0    0    5]
 [   2    0    0    9    0  877    1    1    0    2]
 [   5    2    0    0    3    4  943    0    1    0]
 [   0    2    9    1    0    0    0 1013    1    2]
 [   3    0    5    2    0    2    0    2  958    2]
 [   2    4    1    3    4    0    0    5    2  988]]
```

The final confusion matrix shows minimal misclassifications, with nearly all digits classified correctly. Minor errors are scattered across classes but do not significantly impact overall accuracy.

- *Analysis of the CNN Confusion Matrix*

The confusion matrix from the CNN model's performance on the MNIST test set shows how well the model classified each digit. Here's a breakdown:

High Accuracy in Most Digits:
Diagonal values (from top-left to bottom-right) represent correctly classified samples for each digit class. Most of these values are very high, with minor misclassifications, indicating that the CNN model performs well in recognizing each digit.

Misclassifications:
- Digit 3:
  - 33 samples of digit '3' were misclassified as digit '5'.
  - 12 samples were misclassified as digit '8'.
  - This could be due to visual similarities between digits 3, 5, and 8, as they share rounded shapes.
- Digit 5:
  - 9 samples of digit '5' were misclassified as digit '3'.
  - 13 samples were misclassified as digit '6'.
  - Digits 3, 5, and 6 can look visually similar, especially when handwritten, potentially leading to confusion.
- Digit 8:
  - 33 samples of digit '8' were misclassified as digit '5'.
  - 28 samples were misclassified as digit '3'.
  - Digit 8 often resembles both 3 and 5 depending on handwriting style.

Minor Misclassifications:
  - Small numbers of misclassifications appear across other digit pairs, but they are few and do not significantly impact the model's overall accuracy.
  - For example, a few samples of digits '0', '2', and '4' were misclassified, but these errors are minimal.

Overall Accuracy:
  - Despite some misclassifications, the CNN achieved 98.8% accuracy on the test set, showing that it is highly effective in digit classification.
  - Most misclassifications likely stem from the similarity in certain digit shapes rather than model deficiencies.

Training Time:
Total time for training reached 2 hours and 44 minutes, which is significant but expected for CNNs on large datasets

Comparison

MNIST Models:
- Simple Neural Network:
  - Achieved 93.1% accuracy, with a very short training time, making it suitable for simpler datasets like MNIST.
- Convolutional Neural Network (CNN):
  - Achieved 98.8% accuracy, performing well on complex image data but requiring substantial training time.

Celeb A Models:
- Neural Network (face_nn):
  - Achieved high accuracy of 86.86 on Celeb A, effective for distinguishing facial features (e.g., with/without glasses).
  - Training time is reasonable but may vary based on the complexity of the dataset.

- Deep Neural Network (deepnn):

  - Slightly Lower accuracy than face_nn, with improved generalization due to additional layers.
  - Training time is longer than face_nn but provides better performance for challenging datasets like Celeb A if properly trained.

Analysis Conclusion

- **MNIST:** CNN offers superior accuracy, making it ideal for this image dataset despite its training time. The simple neural network is faster but less accurate.

- **Celeb A:** The neural network model (face_nn) outperforms the deepnn model in accuracy, providing better handling of complex face features at the expense of longer training time. Training a deep model requires careful tuning of learning rates, batch sizes, and initialization. Without these optimizations and only 100 epochs a simpler neural network model might converge to a better solution than a poorly trained DNN. If we train DNN with better parameters it will outperform simple neural network model

IX. SUMMARY

Model	Dataset	Accuracy	Characteristics	Time taken
Neural Network	MNIST	93.06%	Moderate accuracy, trains quickly outperformed by deeper models	35s
Neural Network	Celeb A	86.86%	Enhanced architecture; improved accuracy due to increased complexity	1m
Deep Neural Network	Celeb A	80.99%	Additional hidden layers enabling learning of complex features; requires more computational resources	1m 45s
Convolution Neural Network	MNIST	98.8%	Excels in capturing spatial hierarchies in images, best-performing model for image classification	2h 44m 44s

The project focused on designing, implementing, and evaluating different neural network models, including a simple single-layer network, a deep neural network (DNN), and a convolutional neural network (CNN). The main insights are as follows:

- **Neural Network (Single Hidden Layer**): This model achieved a moderate accuracy of 93.54% on the MNIST dataset. It trained relatively quickly, but its performance was surpassed by deeper, more complex models.

- **Neural Network Model (Celeb A):** This enhanced architecture demonstrated improved accuracy, achieving 86.86% on the Celeb A dataset. The increase in model complexity led to better performance, although it required more training time.

- **Deep Neural Network (DNN):** The DNN reached an accuracy of 80.99% on Celeb A, marking a significant improvement over the simpler neural network. The additional layers allowed it to learn more intricate features, though this came at the cost of increased computational resources and training time.

- **Convolutional Neural Network (CNN):** The CNN excelled in image classification, achieving 98.8% accuracy on the MNIST dataset. Its performance surpassed both the single-layer network and DNN, largely due to its ability to capture spatial hierarchies within the image data. However, the CNN's training time was substantially longer, highlighting a trade-off between accuracy and computational expense.

In summary, the CNN proved to be the best-performing model for image classification, underscoring the importance of choosing the right architecture in neural networks. The findings indicate that complex models like CNNs are well-suited for tasks involving structured image data, whereas simpler networks may be preferable for less complex datasets or when prioritizing computational efficiency.