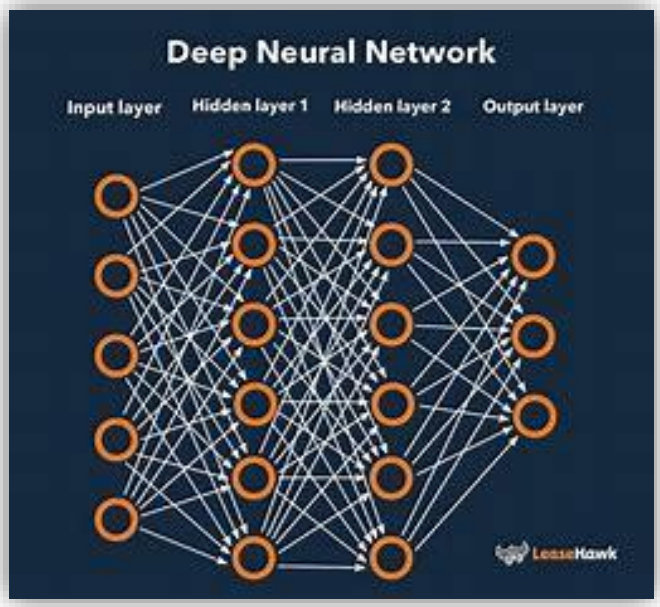
جامعة الإسكندرية
ALEXANDRIA UNIVERSITY
كلية الهندسة
Faculty of Engineering



Assignment 2

Machine Learning

THIS ASSIGNMENT IS ABOUT

Developing a robust handwritten digit recognition system using PyTorch and the MNIST dataset. It requires creating a custom neural network, optimizing training loops, analyzing performance metrics, and testing various hyperparameters to enhance model accuracy.

Mohamed Hussien El zohiry 7818
Alaa Mohamed Elkhouly 7721

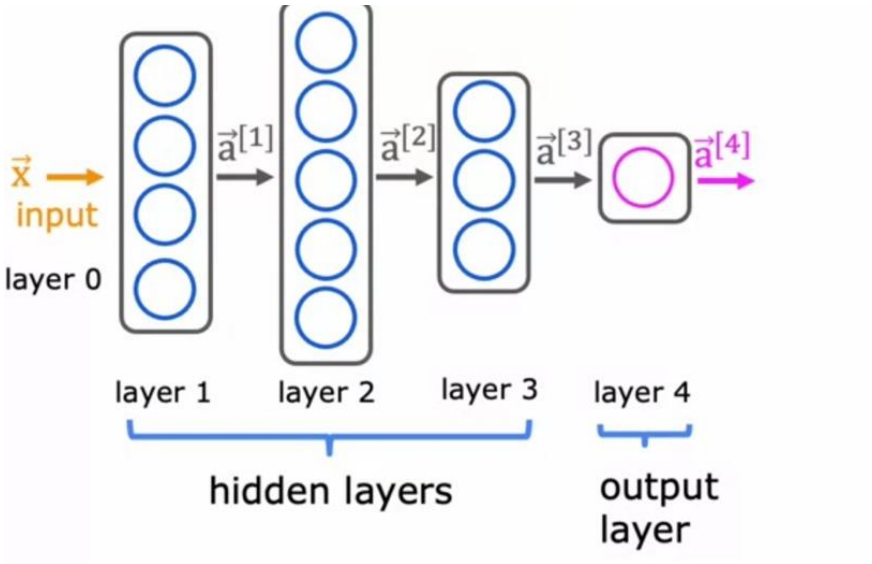
What is PyTorch?

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab. It provides tools for building and training neural networks. Key features include:

1. **Dynamic Computation Graphs**: Allows flexibility in building and modifying models during runtime.
2. **Tensor Operations**: Offers high-performance multidimensional arrays (tensors) similar to NumPy, with GPU acceleration.
3. **Deep Learning Models**: Simplifies the creation of neural networks using built-in modules like `torch.nn`.
4. **Automatic Differentiation**: Includes `autograd` for automatic calculation of gradients, essential for optimization tasks.
5. **Integration with Python**: Supports seamless integration with Python libraries and workflows.

PyTorch is widely used in research and production for tasks like computer vision, natural language processing, and reinforcement learning.

What is a neural network (NN)?



A **neural network** is a computational model inspired by the structure and function of the human brain. It is used in machine learning to solve complex problems by learning patterns from data. Neural networks consist of interconnected layers of nodes, called **neurons**, which process and transform inputs to produce outputs. Each neuron receives an activation signal from the preceding layer and computes some value that is propagated forward.

Key Components:

1. **Input Layer**: Receives the raw input data features (e.g., pixel values from an image).
2. **Hidden Layers**: Perform computations by applying weights, biases, and activation functions to transform the input into meaningful patterns.
3. **Output Layer**: Produces the final predictions or classifications (e.g., the digit class).

4. **Weights and Biases**: Parameters adjusted during training to improve model accuracy.
5. **Activation Functions**: Non-linear functions like ReLU, Sigmoid, or Tanh that determine the output of each neuron. Introducing non-linearity into the model, enables it to learn complex patterns.

Working Process:

1. **Forward Propagation**: Data flows through the network, and predictions are generated.
2. **Loss Calculation**: The difference between predictions and actual values is computed using a loss function.
3. **Backward Propagation**: Gradients of the loss are calculated and used to update weights through optimization techniques like gradient descent.

Neural networks excel in tasks like image recognition, natural language processing, and predictive analytics.

***Loss Function:** The loss function measures the error between the predicted outputs and the true labels. In this task, we'll use Cross Entropy Loss, which is commonly used for classification problems.

***The optimizer** (e.g., Stochastic Gradient Descent) updates the network's weights to minimize the loss function during training.

Equations for Neural Network (NN)

1. **Forward Propagation in a Fully Connected Layer**:

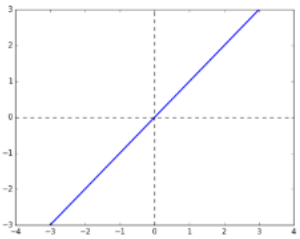
$$z = W \cdot x + b$$

   - z: Weighted sum (input to activation function).
   - W: Weight matrix.
   - x: Input vector.
   - b: Bias vector.
2. **Activation Function**:

$$a = f(z) = Ø(W \cdot x + b)$$

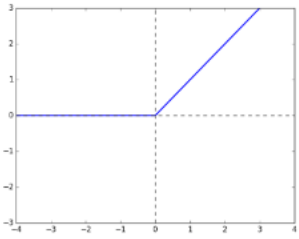
   - a: Output after activation.
   - f: Activation function (e.g., ReLU, Sigmoid).
   - Ø: is applied component-wise

Examples of activation functions:



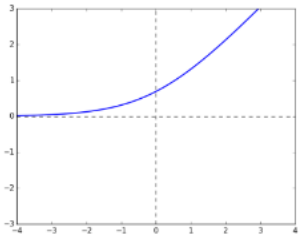
Identity

$$y = z$$



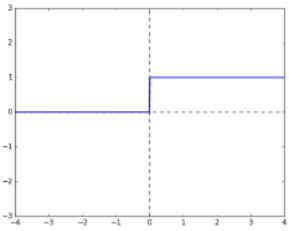
Rectified Linear Unit (ReLU)

$$y = \max(0, z)$$



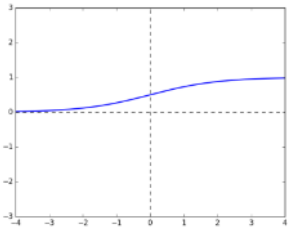
Soft ReLU

$$y = \log 1 + e^z$$



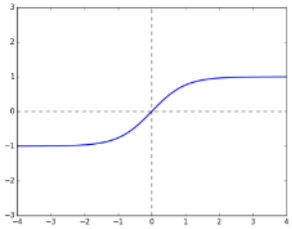
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Logistic

$$y = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent (tanh)

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

3. **Output Layer**:

$$\hat{y} = g(a)$$

   - $\hat{y}$: Model prediction.
   - g: Output activation (e.g., Softmax for classification).

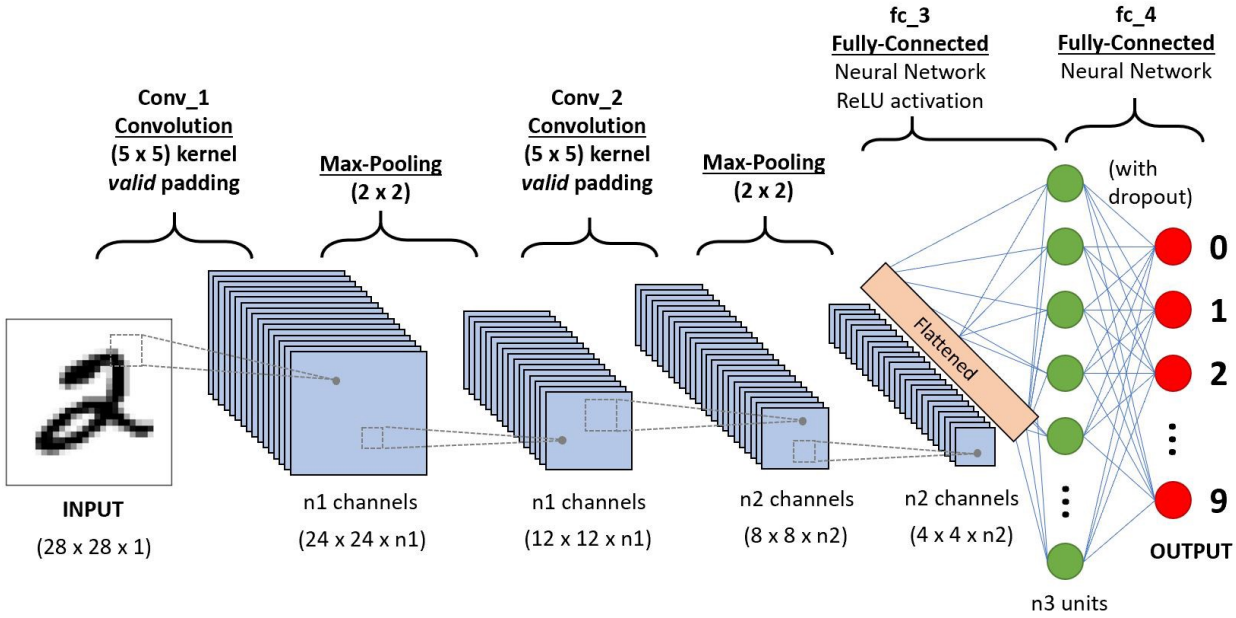
4. **Loss Function**: Example: Cross-Entropy Loss (for classification):

$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij}\log(\hat{y}_{ij})$$

   - y: True labels.
   - $\hat{y}$: Predicted probabilities.
   - n: Number of samples.
   - k: Number of classes.

NN Equations focus on matrix multiplication and activation for fully connected layers.

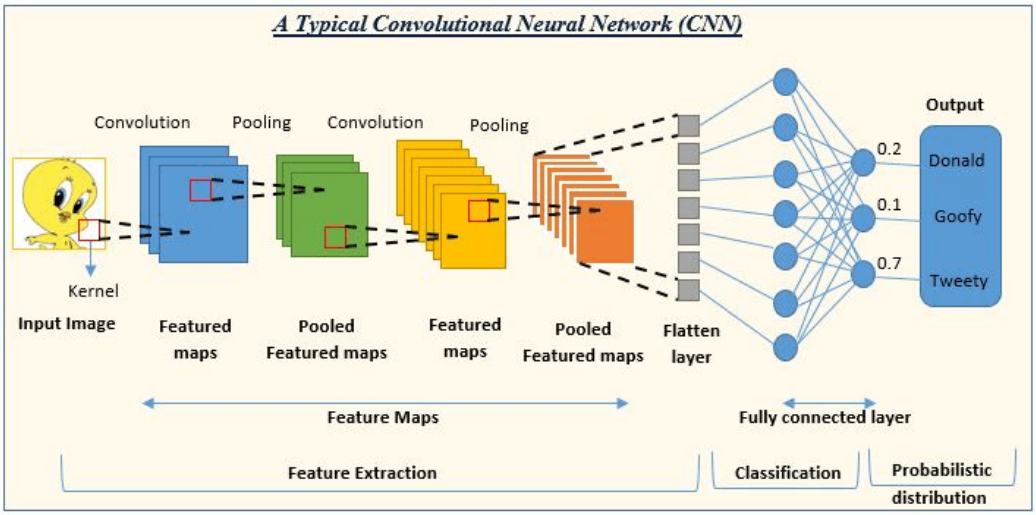
What is a Convolutional neural network (CNN)?



A **Convolutional Neural Network (CNN)** is a specialized type of neural network designed for processing structured grid data, such as images. It is widely used in tasks like image classification, object detection, and computer vision.

Key Features:

1. **Convolutional Layers**:
   - Perform convolution operations to detect patterns like edges, textures, or shapes in data.
   - Use filters (or kernels) that slide over the input data to extract features.
2. **Pooling Layers**:
   - Reduce the spatial dimensions of the data while retaining important information.
   - Common types include max pooling (takes the maximum value) and average pooling (takes the average value).
3. **Fully Connected Layers**:
   - Connect all neurons to the output, combining extracted features for final predictions.
4. **Activation Functions**:
   - Introduce non-linearity to model complex patterns (e.g., ReLU, Sigmoid).
5. **Dropout and Normalization**:
   - Dropout prevents overfitting by randomly deactivating neurons during training.
   - Normalization stabilizes and speeds up training by standardizing data.



Advantages:

- **Local Connectivity**: Focuses on small, localized regions of data to capture patterns.
- **Weight Sharing**: Reduces the number of parameters, making the model computationally efficient.
- **Hierarchical Feature Learning**: Builds simple features in earlier layers and combines them into complex features in later layers.

Common Applications:

- **Image and Video Recognition**: Classifying images or identifying objects in videos.
- **Medical Imaging**: Analyzing X-rays or MRIs for diagnostics.

Autonomous Vehicles: Detecting objects like pedestrians or traffic signs.

Equations for Convolutional Neural Network (CNN)

1. **Convolution Operation**:

$$z[i,j] = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} x[i+m, j+n] \cdot w[m,n] + b$$

   - z[i,j]: Convolution output at position (i, j).
   - x[i+m,j+n]: Input patch.
   - w[m,n]: Kernel/filter weights.
   - b: Bias.
   - k_h, k_w: Kernel height and width.
2. **Activation Function**:

$$a[i,j] = f(z[i,j])$$

   - a[i,j]: Activated output.
   - f: Activation function (e.g., ReLU).

3. **Pooling Operation** (e.g., Max Pooling):

$$p[i,j] = \max_{(m,n)\in R} a[i+m, j+n]$$

   - p[i, j]: Pooling output.
   - R: Region of pooling (e.g., 2×22 \times 2).
4. **Flattening and Fully Connected Layer**: After the convolutional and pooling layers:

$$z = W \cdot x + b$$

   (Similar to NN fully connected layer).

CNN Equations include convolution and pooling operations in addition to fully connected layers.

Preventing Overfitting

Overfitting happens when a model performs well on training data but poorly on unseen data.

1. **Regularization**:
   - Add a penalty to the loss function to reduce model complexity.
   - **L2 Regularization** (Ridge): Adds λ∑W2\lambda \sum W^2 to the loss.
   - **L1 Regularization** (Lasso): Adds λ∑|W|\lambda \sum |W| to the loss.
2. **Dropout**:
   - Randomly deactivate neurons during training to reduce dependency on specific neurons.
   - Example: Dropout rate of 0.5 deactivates 50% of neurons in a layer.
3. **Data Augmentation**:
   - Expand the dataset by applying transformations like rotations, scaling, or flipping (especially for images).
4. **Early Stopping**:
   - Stop training when the validation loss stops improving to avoid overtraining.
5. **Reduce Model Complexity**:
   - Simplify the architecture by using fewer layers or neurons.
6. **Increase Training Data**:
   - Add more samples to reduce the model's tendency to memorize patterns.
7. **Batch Normalization**:
   - Stabilizes and accelerates training, reducing reliance on specific feature values.

Preventing Underfitting

Underfitting occurs when the model fails to capture patterns in the data and performs poorly on both training and validation datasets.

1. **Increase Model Complexity**:
   - Use more layers, neurons, or advanced architectures to capture complex patterns.
2. **Train Longer**:
   - Allow the model to train for more epochs if the training loss has not converged.
3. **Use Appropriate Features**:
   - Ensure the input data has relevant features that reflect the problem domain.
4. **Optimize Hyperparameters**:
   - Adjust learning rate, batch size, or weight initialization methods for better learning.
5. **Reduce Regularization**:
   - Lower the strength of regularization techniques (e.g., smaller L2 or L1 coefficients).
6. **Use Better Activation Functions**:
   - Use non-linear activations like ReLU, Tanh, or Leaky ReLU to model complex patterns effectively.
7. **Ensure Sufficient Data Preprocessing**:
   - Scale or normalize inputs for better gradient flow and convergence.

Neural Networks (NNs) and Convolutional Neural Networks (CNNs) Comparison:

Feature	Neural Networks (NN)	Convolutional Neural Networks (CNN)
Architecture	Fully connected layers, where every neuron is linked to all neurons in adjacent layers.	Includes convolutional, pooling, and fully connected layers.
Input Type	Works with unstructured or structured data (e.g., tabular data).	Specialized for grid-like data such as images or time-series data.
Parameter Efficiency	Has a large number of parameters as every neuron connects to others.	Uses weight sharing in convolution layers, reducing parameters.
Feature Extraction	Relies on manually defined features or learned features in deeper layers.	Automatically extracts hierarchical features from data (e.g., edges, shapes).
Spatial Awareness	Does not account for spatial relationships in data.	Preserves spatial relationships, crucial for tasks like image recognition.
Performance on Images	Inefficient and less accurate for image-related tasks.	Highly effective for image and video processing.
Computational Complexity	Higher due to more parameters and connections.	Lower due to local connectivity and weight sharing.
Overfitting	More prone to overfitting with high-dimensional data.	Less prone to overfitting when using techniques like dropout and pooling.
Use Cases	Tabular data, time-series prediction, and non-image tasks.	Image recognition, object detection, and video analysis.

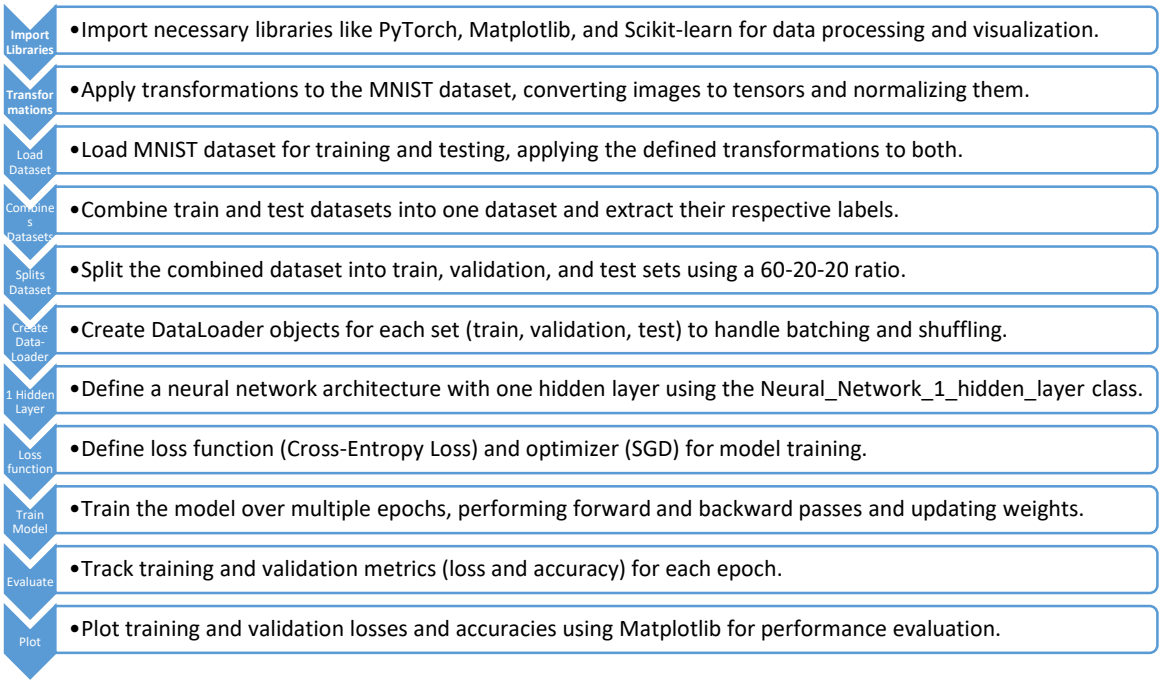
While NNs are more general-purpose and can be applied to various types of data, CNNs are specifically designed for spatially organized data like images. CNNs outperform traditional NNs in tasks requiring feature extraction from visual data.

What does the code do?

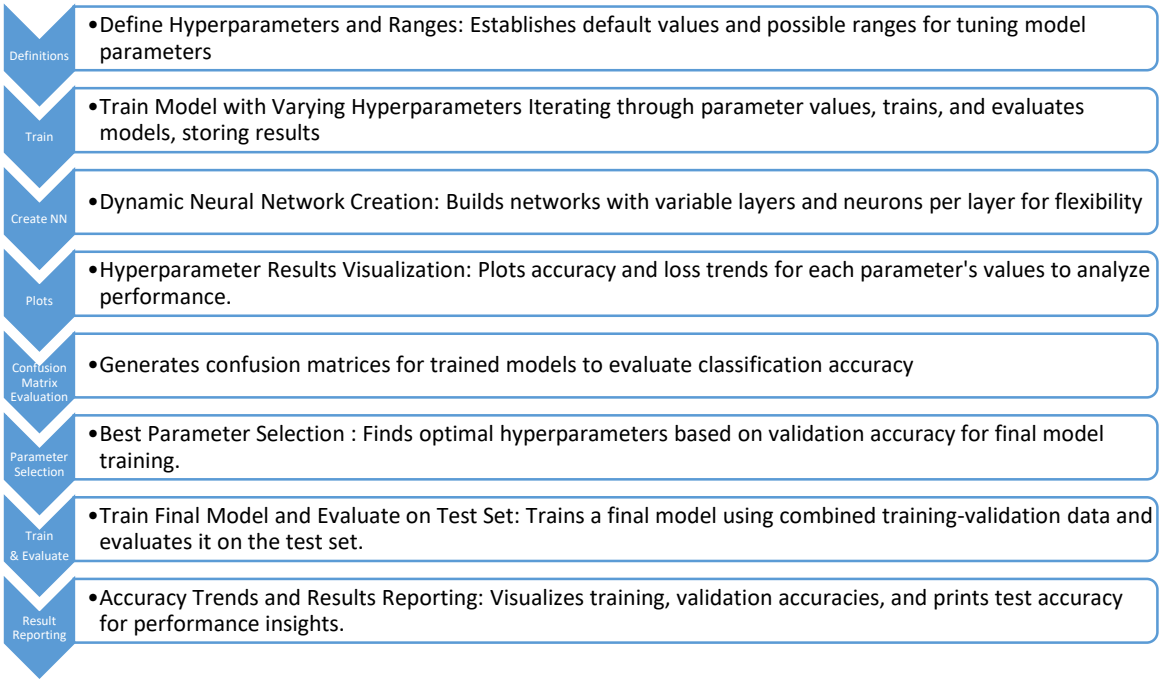
Part 1:

trains a neural network on the MNIST dataset and evaluates its performance by plotting loss and accuracy curves.

Part 1 flow code:

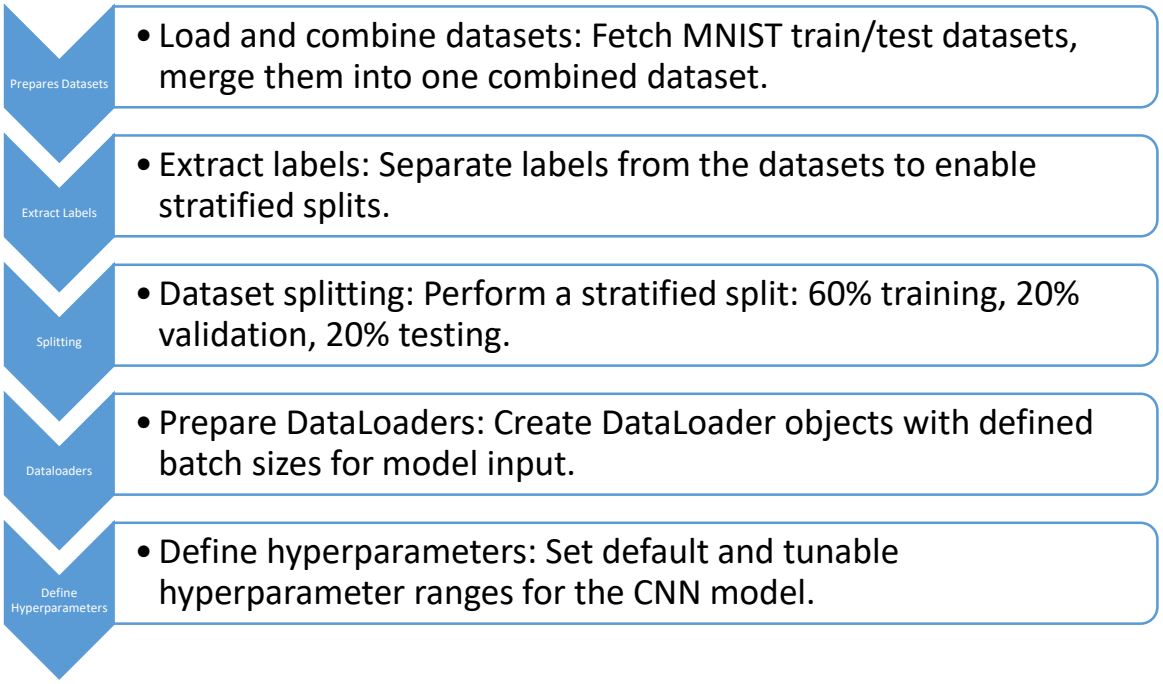


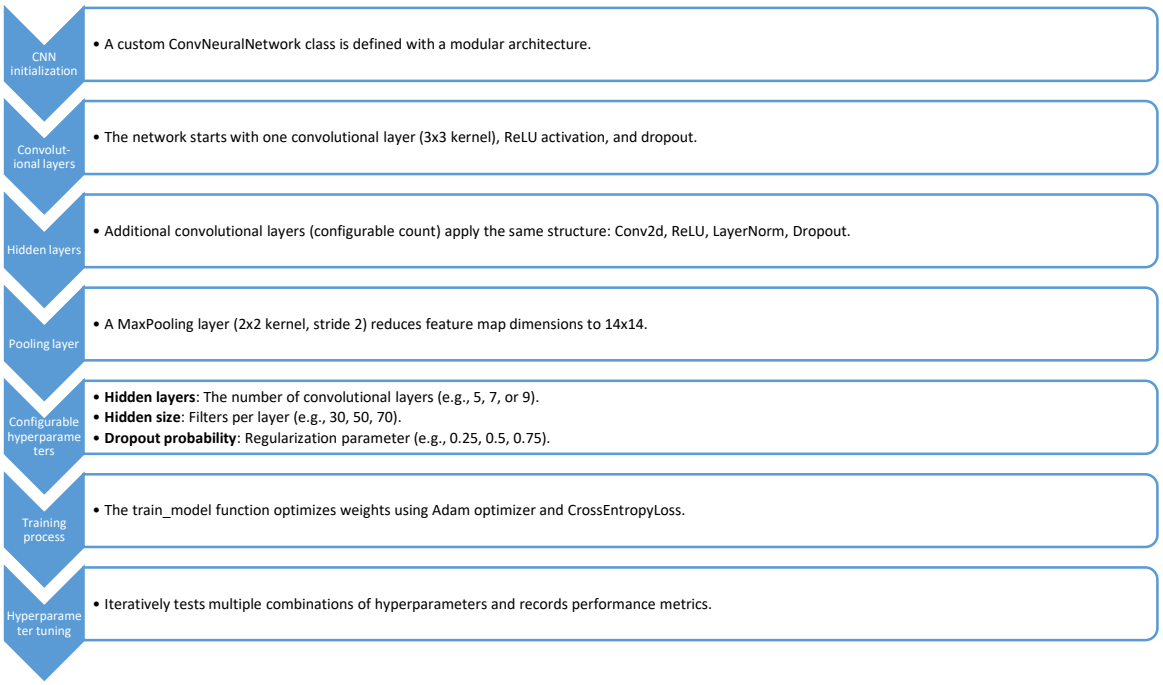
Code Continuation:



Bonus part of code: (CNN)

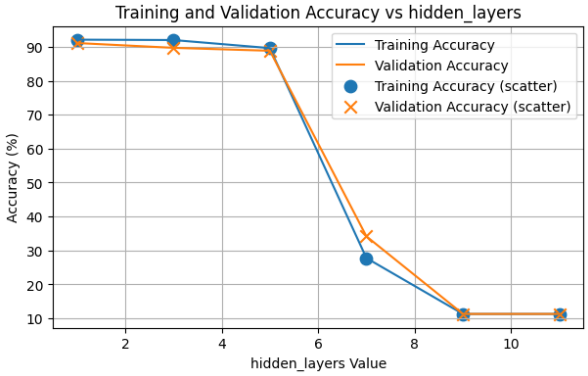
Here unlike Neural Networks (NN) there is no need to flatten the matrix first

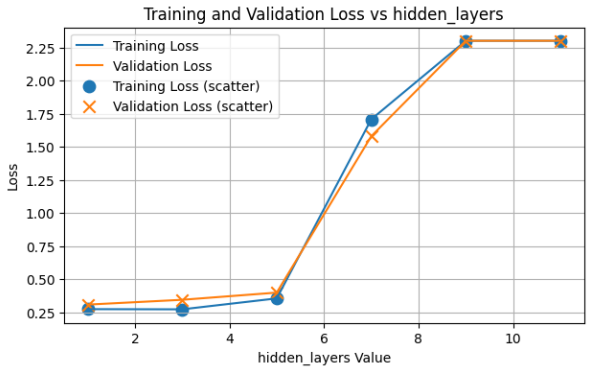




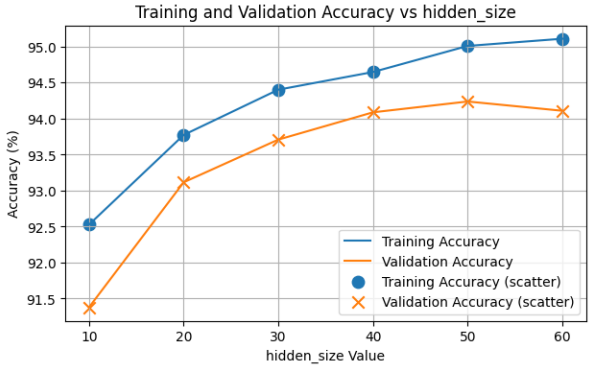
Graphs and their explanation:

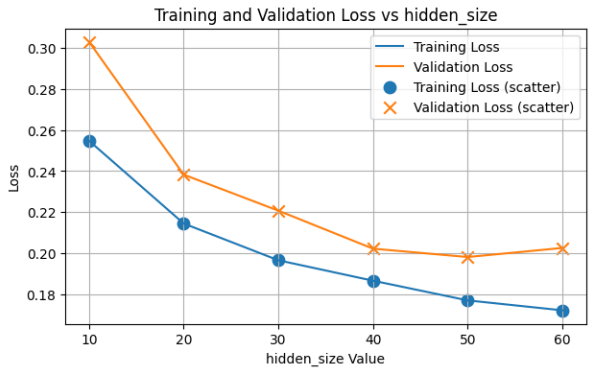
This was before tuning any parameters (base case)



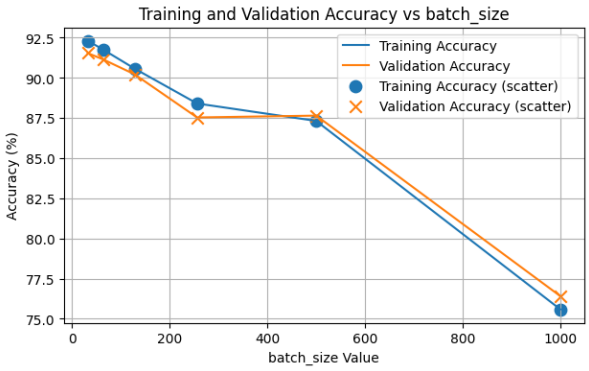


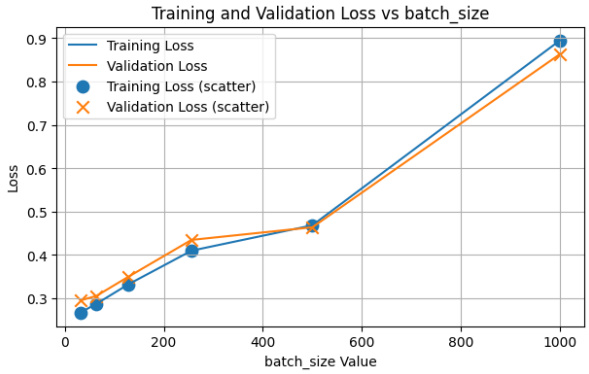
Accuracy and loss graphs above show low variance and high bias indicating underfit. Optimal range from 1 to 5 for number of hidden layers



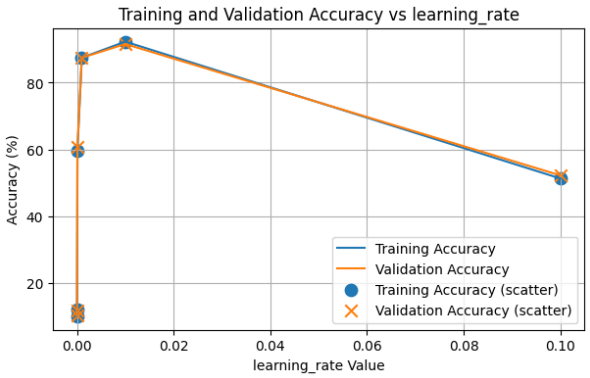


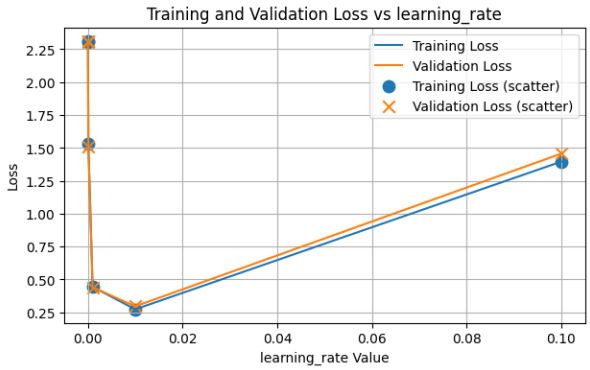
Accuracy and loss graphs above show high variance and low bias indicating overfit, optimal range from 30 to 50 for number of neurons





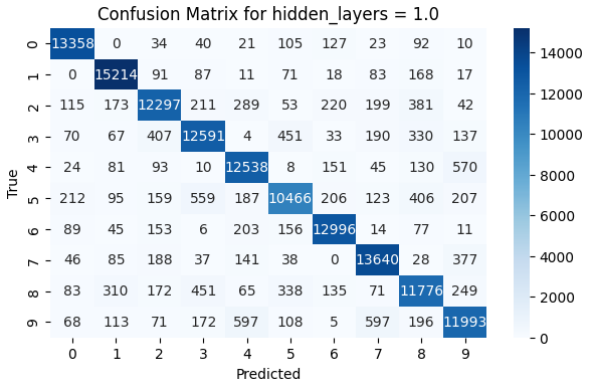
Accuracy and loss graphs above show low variance and high bias indicating underfit ,optimal range here from 250 to 500 batch size.

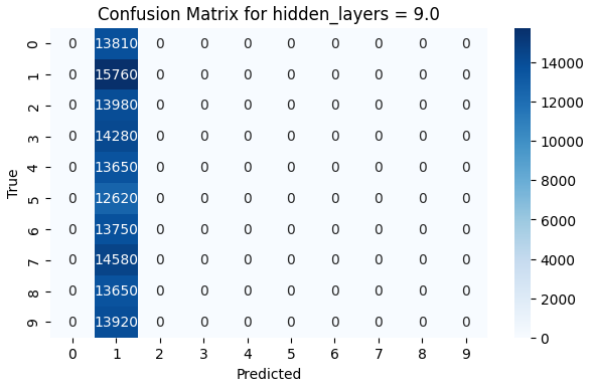


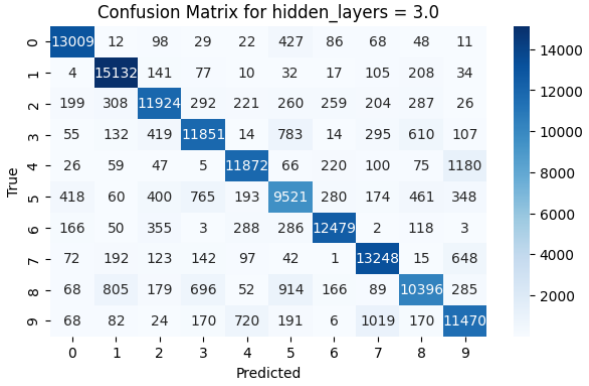


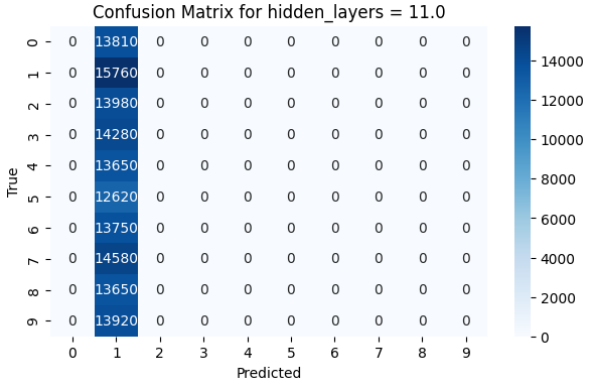
Accuracy and loss graphs above show low variance and high bias indicating underfit ,optimal range here from 0.00001 to 0.01 for learning rate

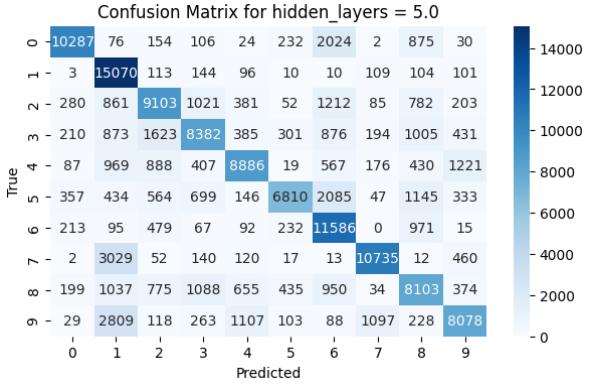
In the confusion matrices below we are hoping the negative diagonal have the greatest values possible (darkest shades of blue)

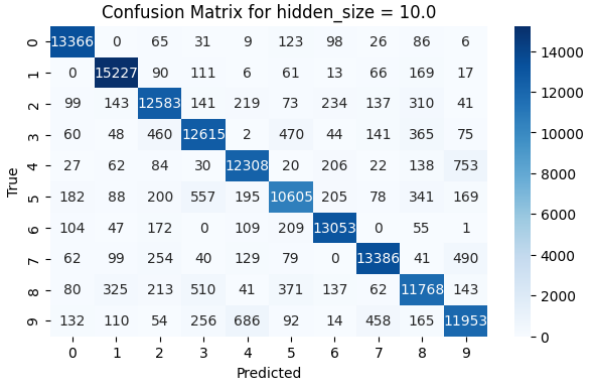


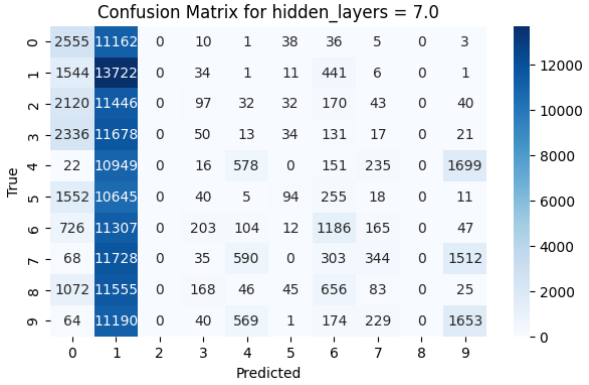


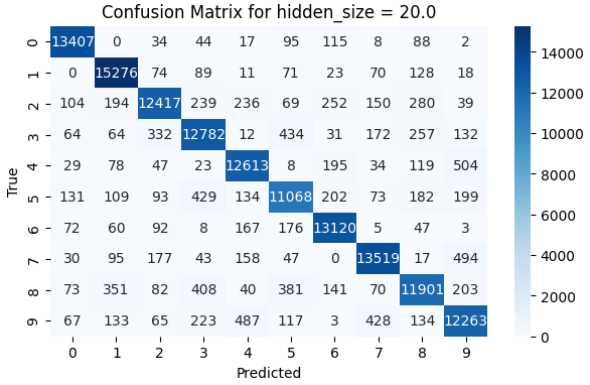


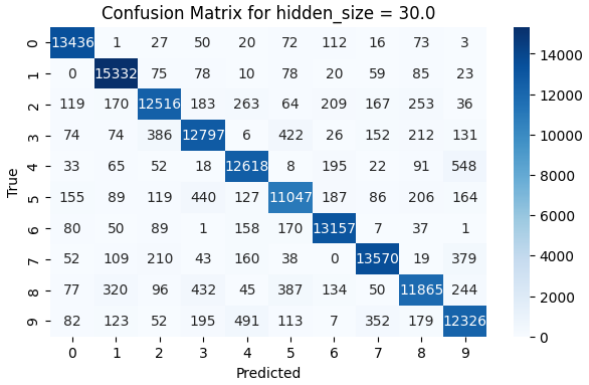


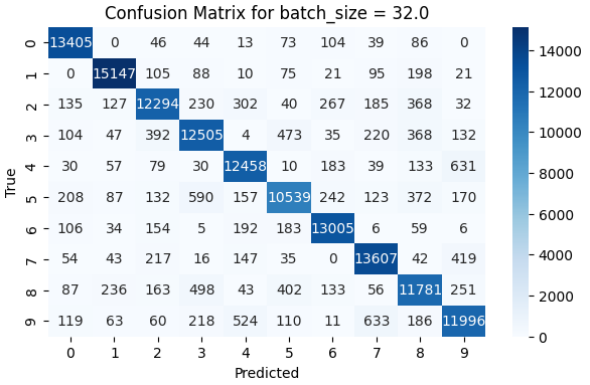


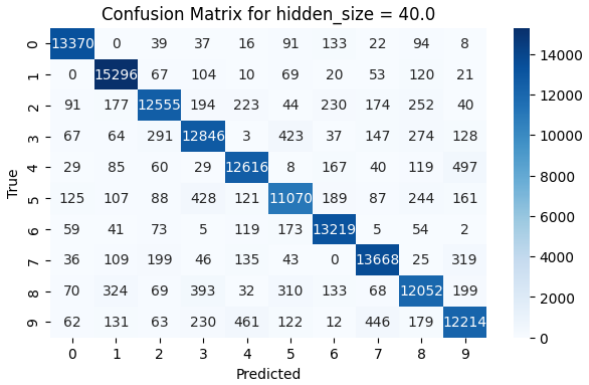


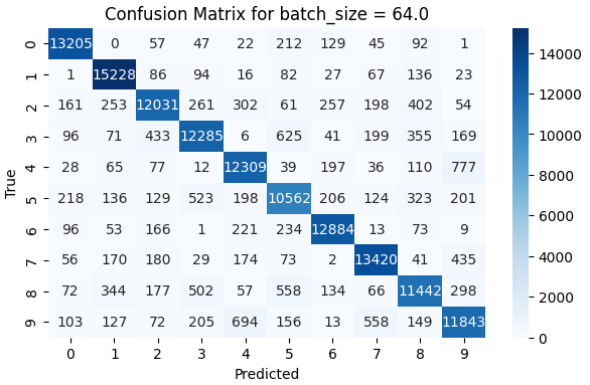


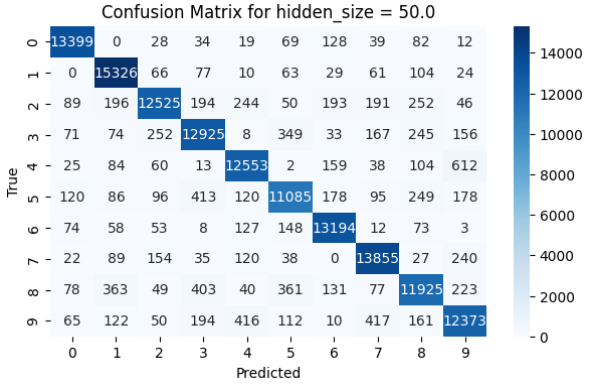


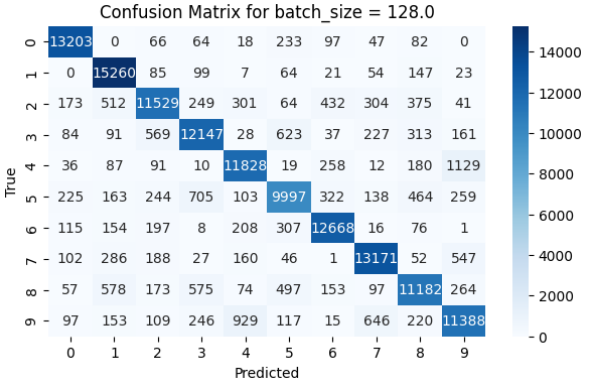

Confusion Matrix for hidden_size = 30.0
True
Predicted

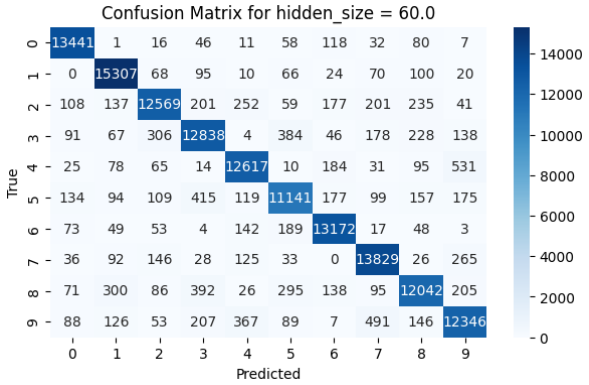

Confusion Matrix for batch_size = 32.0
True
Predicted

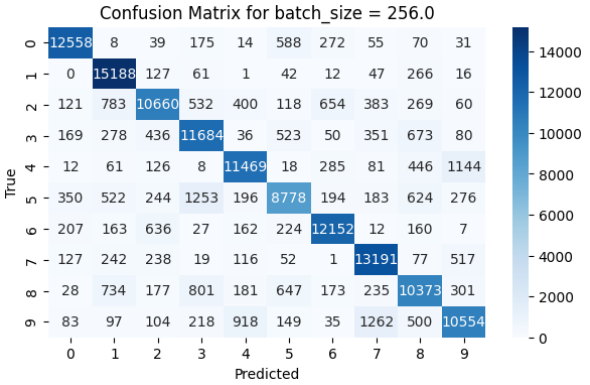

Confusion Matrix for hidden_size = 40.0
True
Predicted

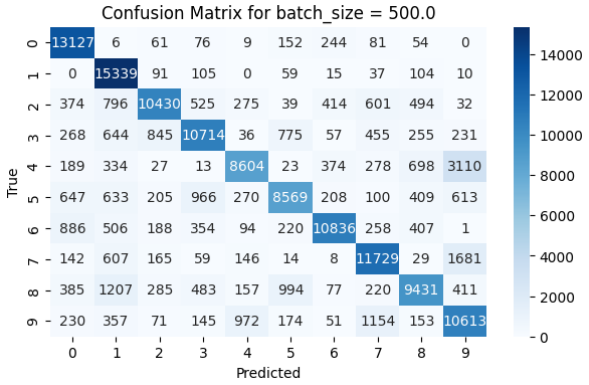

Confusion Matrix for batch_size = 64.0
True
Predicted

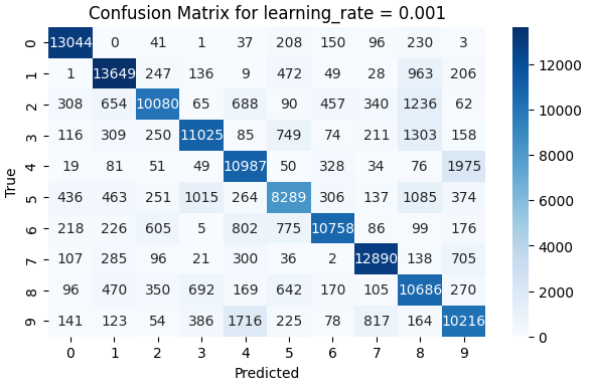

Confusion Matrix for hidden_size = 50.0
True
Predicted

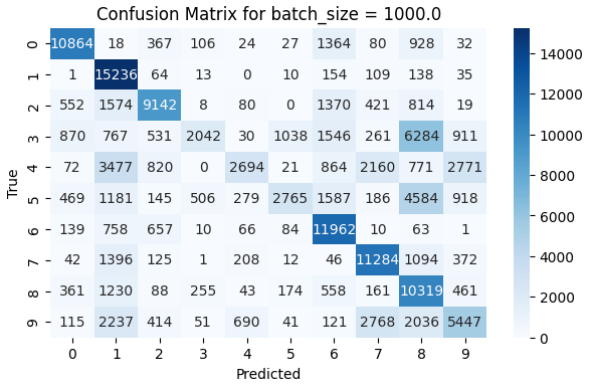

Confusion Matrix for batch_size = 128.0
True
Predicted

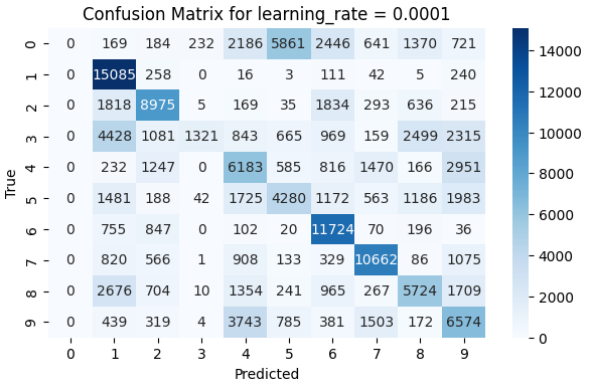

Confusion Matrix for hidden_size = 60.0
True
Predicted

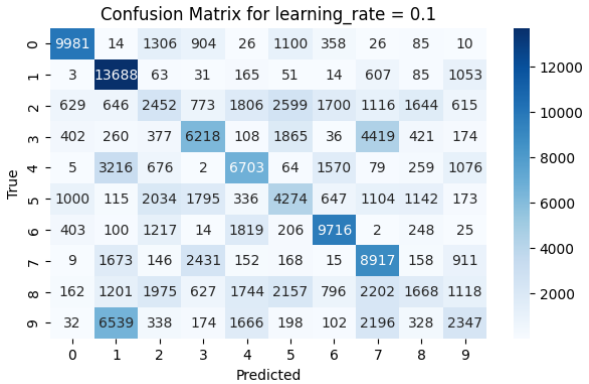

Confusion Matrix for batch_size = 256.0
True
Predicted

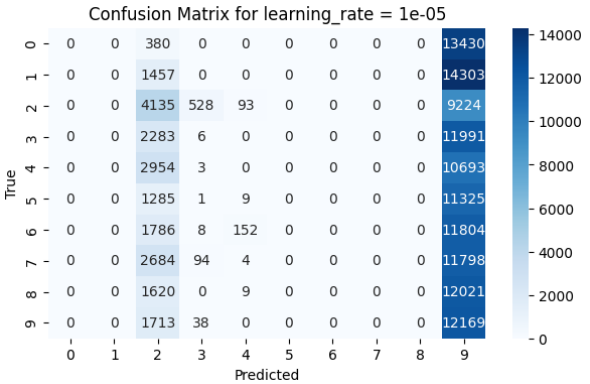

Confusion Matrix for batch_size = 500.0
True
Predicted

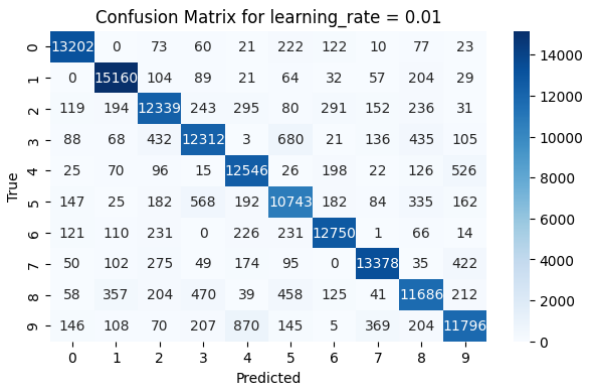

Confusion Matrix for learning_rate = 0.001
True
Predicted

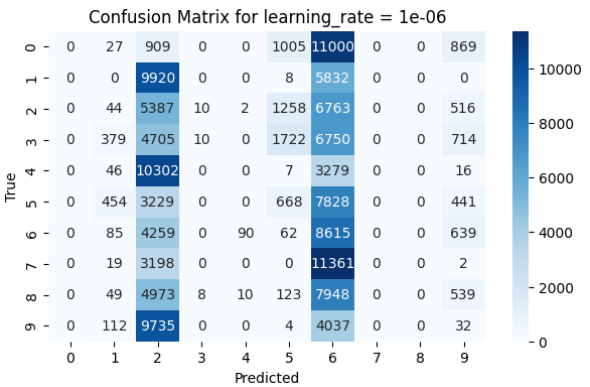

Confusion Matrix for batch_size = 1000.0
True
Predicted

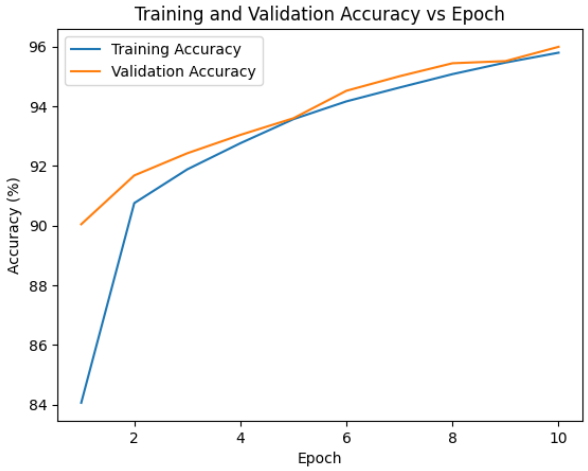

Confusion Matrix for learning_rate = 0.0001
True
Predicted

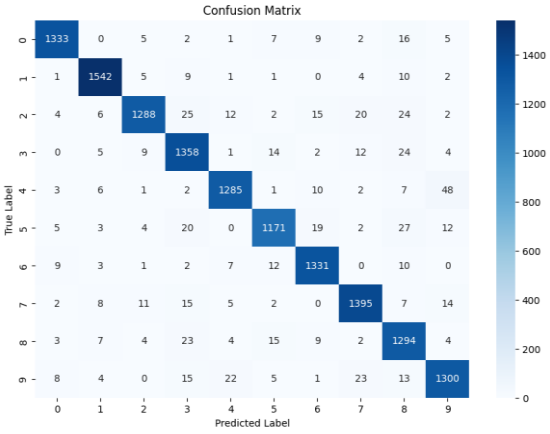

Confusion Matrix for learning_rate = 0.1
True
Predicted


Confusion Matrix for learning_rate = 1e-05
True
Predicted

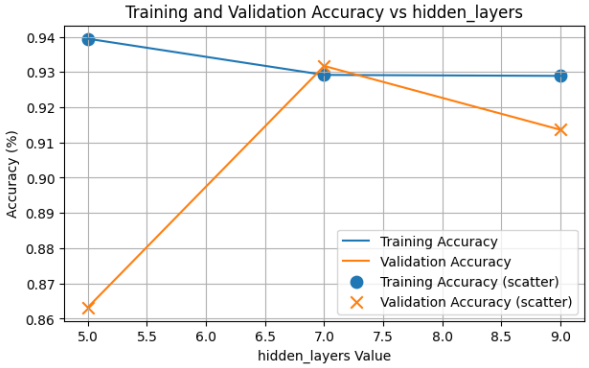

Confusion Matrix for learning_rate = 0.01
True
Predicted

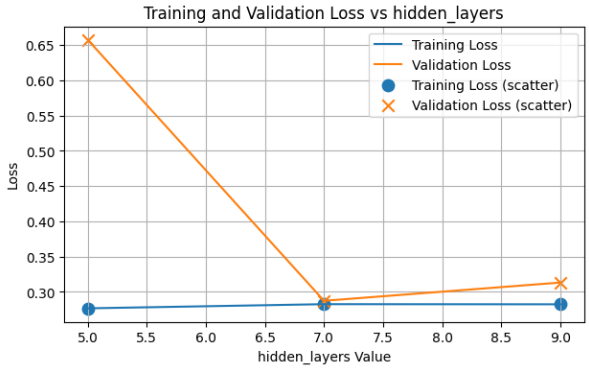

Confusion Matrix for learning_rate = 1e-06
True
Predicted



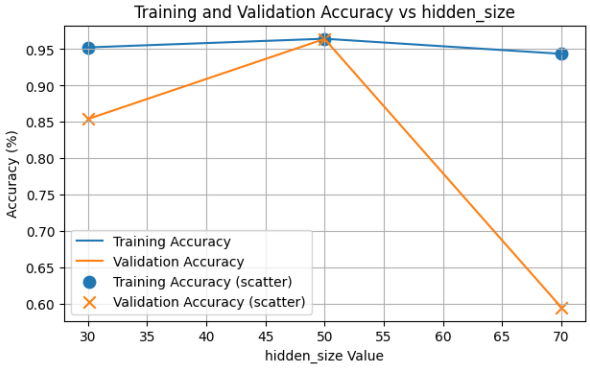


Graphs and their explanation for Bonus (CNN) part:

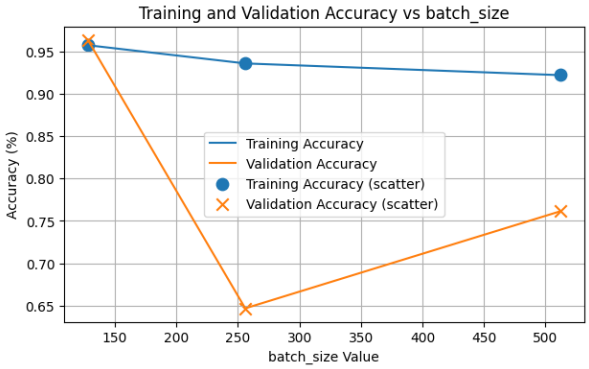


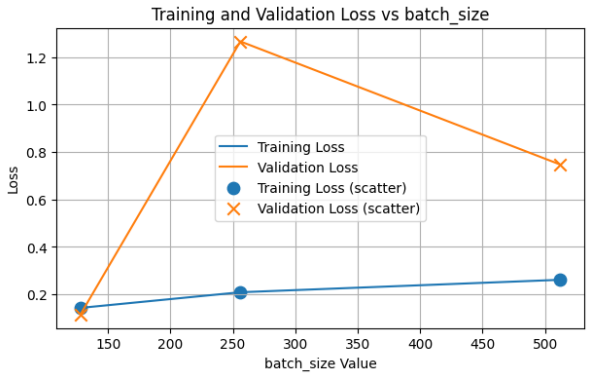


Accuracy and loss graphs above show increasing variance and low bias indicating overfit ,optimal range here from 7 to 8 hidden layers.

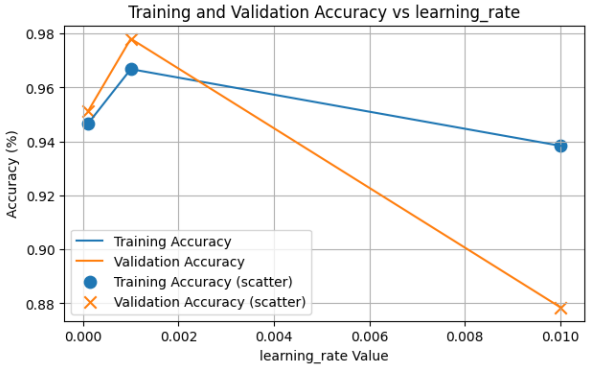


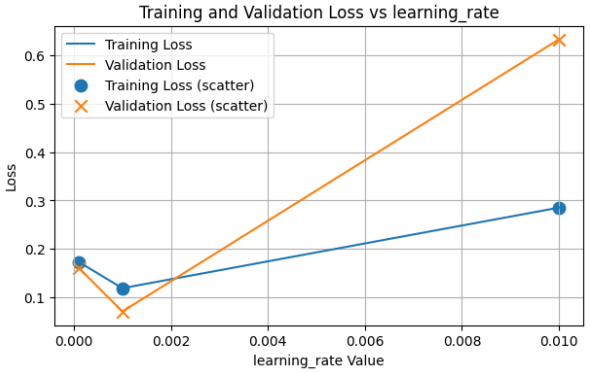
The previous accuracy and loss graphs show increasing variance and low bias indicating overfit ,optimal range here from 30 to 50 for number of neuron networks.



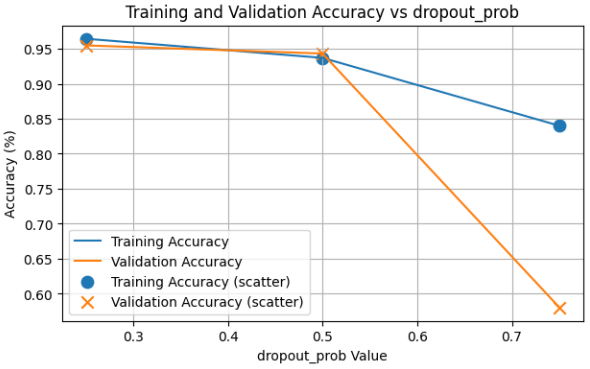


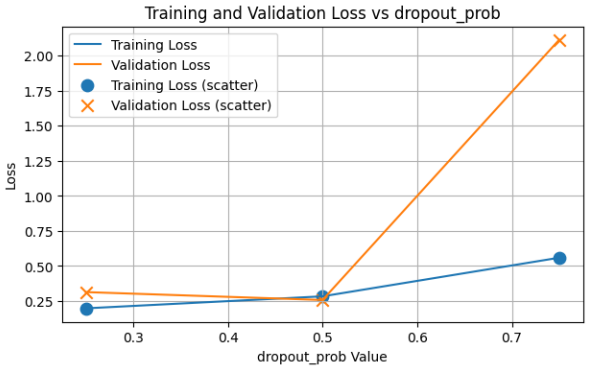
Accuracy and loss graphs above show decreasing variance and increasing bias indicating underfit.





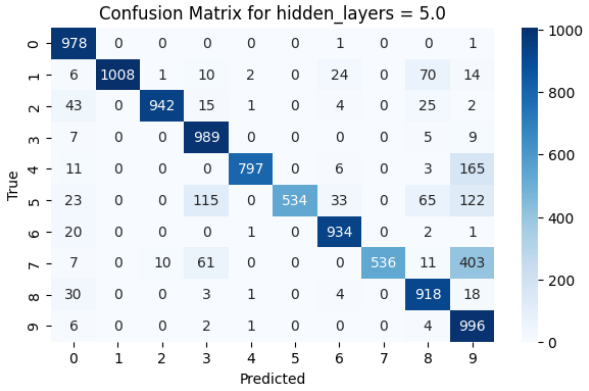
Accuracy and loss graphs above show increasing variance and increasing bias.

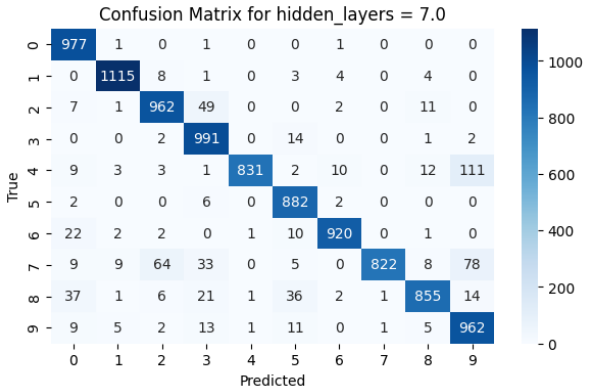


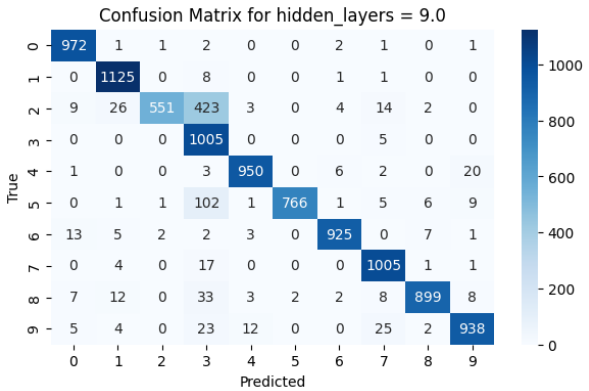


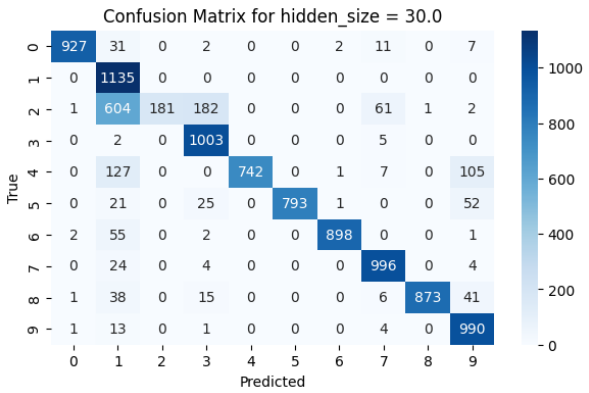
Accuracy and loss graphs above show decreasing variance and decreasing bias indicating in the optimal range 0.1 to 0.5 for drop out probability.

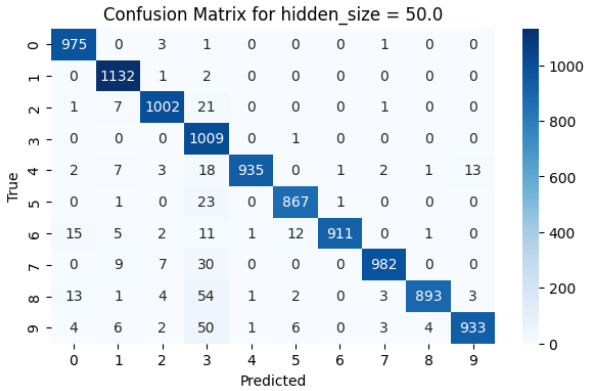
In the confusion matrices below we are hoping the negative diagonal have the greatest values possible (darkest shades of blue)

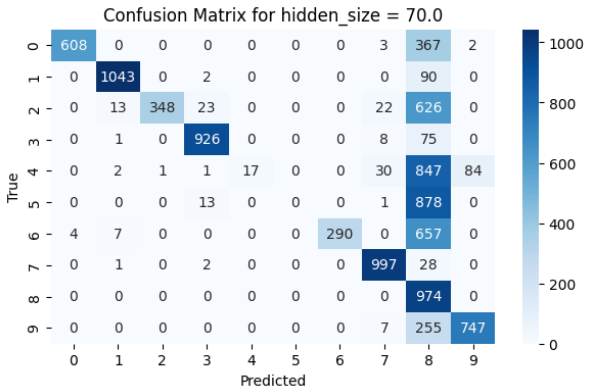


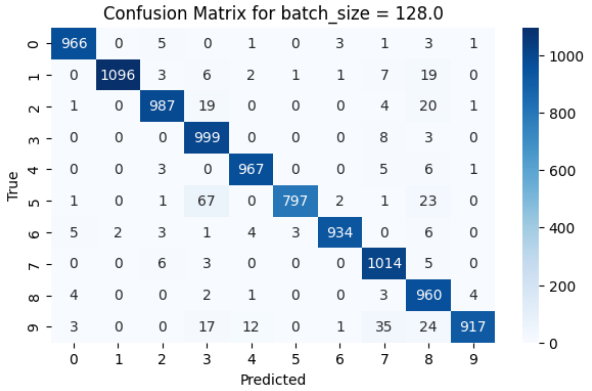


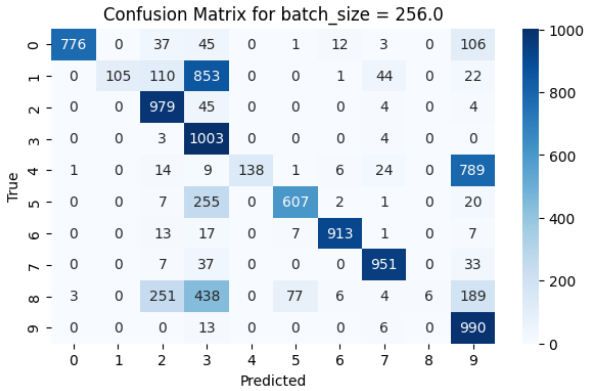


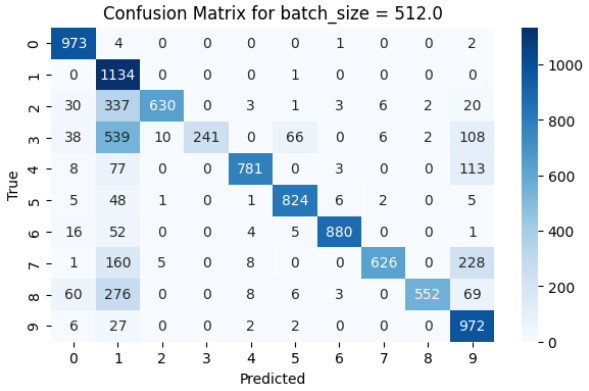


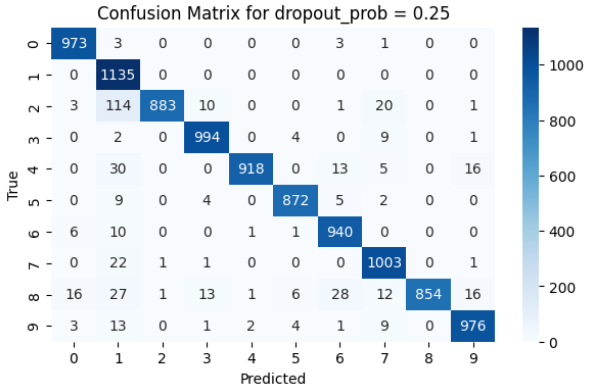


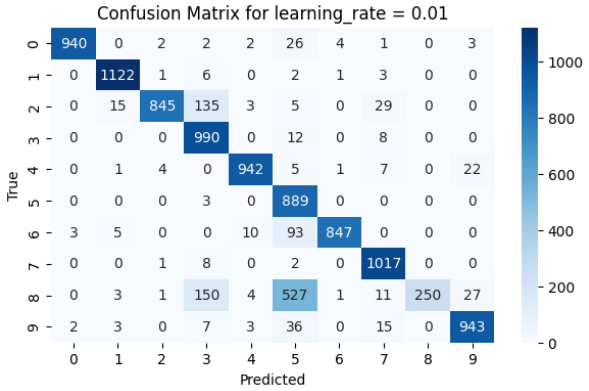


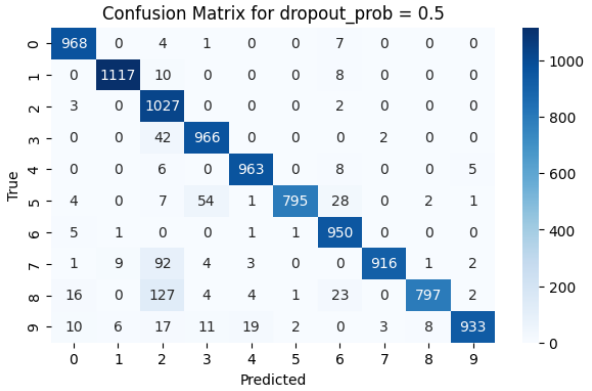


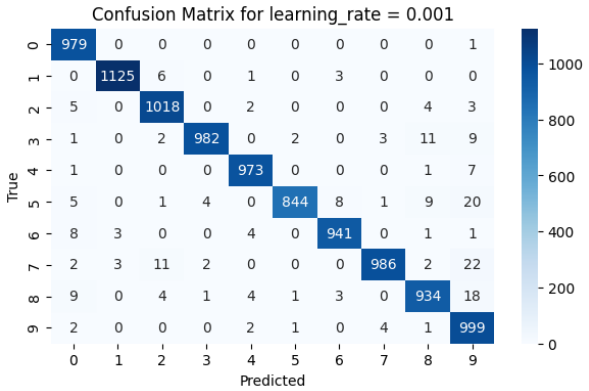


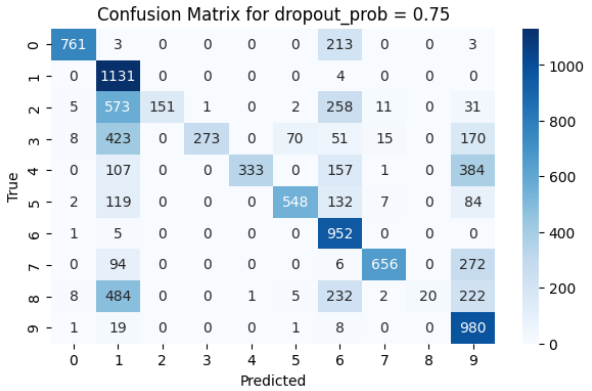

Confusion Matrix for batch_size = 512.0
True
Predicted

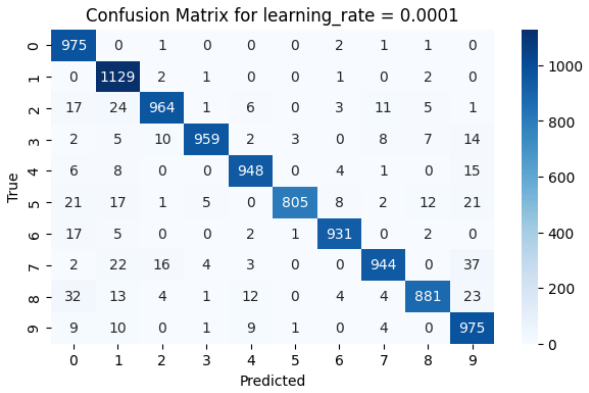

Confusion Matrix for dropout_prob = 0.25
True
Predicted


Confusion Matrix for learning_rate = 0.01
True
Predicted


Confusion Matrix for dropout_prob = 0.5
True
Predicted


Confusion Matrix for learning_rate = 0.001
True
Predicted


Confusion Matrix for dropout_prob = 0.75
True
Predicted


Confusion Matrix for learning_rate = 0.0001
True
Predicted