# Introduction

## what is a neural network?

A neural Network is a computational model inspired by the human brain. Each neuron recieves an activation signal from the preceeding layer and computes some value that is propoagated forward #### Key Concepts:

- **Input Layer**: Receives the raw input data (e.g., pixel values from an image).
- **Hidden Layers**: Perform computations to extract meaningful features.
- **Output Layer**: Produces the final predictions (e.g., the digit class).

#### Activation Functions: Activation functions (e.g., ReLU, sigmoid) introduce non-linearity into the model, enabling it to learn complex patterns.

#### 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.

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

#### Problem Statement: In this assignment, we will use the MNIST dataset to train a neural network to classify handwritten digits (0-9). We'll explore various hyperparameters, analyze their effects, and optionally use advanced techniques such as convolutional layers and dropout.

# 1 Data Processing

before we attempt to train our model we must make sure that our data conforms with required criteria:

- the samples are tensors
- 60 % training , 20 % validation , 20 % test
- keep a "straigtified" manner to avoid data skewing

```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np
import pandas as pd
from torchvision import datasets, transforms
from torch.utils.data import DataLoader,Subset,ConcatDataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
```

```
# Apply transformations (convert to tensor and normalize)
transform = transforms.Compose([
    transforms.ToTensor(), #convert vector values in range [0,1]
    transforms.Normalize((0.5,), (0.5,))  # Normalize to [-1, 1] range
(z socre)
])

# Load the MNIST dataset
train_dataset = datasets.MNIST(root='./data', train=True,
transform=transform) #this structure contains data and labels
test_dataset = datasets.MNIST(root='./data', train=False,
transform=transform)

#combine the datasets
combined_dataset=ConcatDataset([train_dataset,test_dataset])

#extract labels
train_labels=train_dataset.targets
test_labels=test_dataset.targets

# Combine labels from both train and test datasets
all_labels = torch.cat((train_labels, test_labels), dim=0)

#split into 60-40
train_idx, temp_idx = train_test_split(
    range(len(combined_dataset)), test_size=0.4, stratify=all_labels,
random_state=42) # stratify 3lshan not to get skewed data


#split again 20-20
# Split temp (40%) into validation (20%) and test (20%)
val_idx, test_idx = train_test_split(
    temp_idx, test_size=0.5, stratify=all_labels[temp_idx],
random_state=42) # returns indices


# Create Subset objects for train, validation, and test sets
train_data = Subset(combined_dataset, train_idx)
val_data = Subset(combined_dataset, val_idx)
test_data = Subset(combined_dataset, test_idx)


##recreate dataset w inheirt combatible structure with data loader
# Create DataLoader objects
batch_size = 32
train_loader = DataLoader(train_data, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_data, batch_size=batch_size,
shuffle=False)
test_loader = DataLoader(test_data, batch_size=batch_size,
```

```
shuffle=False)

# Print split sizes
print(f"Training set size: {len(train_data)}")
print(f"Validation set size: {len(val_data)}")
print(f"Test set size: {len(test_data)}")
print("Data preparation complete!")


Training set size: 42000
Validation set size: 14000
Test set size: 14000
Data preparation complete!
```

now we build our neural network architecture. we will start with a basic model containing a single hidden layer.

```
# 2 create neural network structure
class Neural_Network_1_hidden_layer(nn.Module):
        def __init__(self, input_size=28*28, hidden_size=10,
num_classes=10):
            super(Neural_Network_1_hidden_layer, self).__init__()
            self.fc1 = nn.Linear(input_size, hidden_size)
            self.fc2 = nn.Linear(hidden_size, num_classes)

        def forward(self, x):
            x = x.view(-1, 28*28)  # Flatten the input
            x = torch.relu(self.fc1(x))  # Apply ReLU activation after
first hidden layer
            x = self.fc2(x)  # Output layer
            return x
```

before we start the training process we need to make sure that we are using GPU accelerated training

```
# 3. Set device to GPU if available, else fallback to CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Instantiate the model and move it to the selected device
model = Neural_Network_1_hidden_layer().to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

we will now proceed to train our model for 10 epochs where each epoch consists of a training phase followed by a vildation phase.

```
# 4. Training Process
def train_model(model, train_loader, val_loader, criterion, optimizer,
num_epochs=10):
    train_losses, val_losses = [], []
    train_accuracies, val_accuracies = [], []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0 # this is the cumulative loss during epoch
        correct_train = 0
        total_train = 0

        # Train the model on the training set
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)  #
Move to GPU

            optimizer.zero_grad()
            outputs = model(images)  # Forward pass
            loss = criterion(outputs, labels)  # Compute loss
            loss.backward()  # Backward pass (compute gradients)
            optimizer.step()  # Update weights

            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)  # Get the predicted
class
            correct_train += (predicted == labels).sum().item()
            total_train += labels.size(0)

        avg_train_loss = running_loss / len(train_loader)
        train_accuracy = correct_train / total_train * 100
        train_losses.append(avg_train_loss)
        train_accuracies.append(train_accuracy)

        # Validation phase
        model.eval()
        running_loss = 0.0
        correct_val = 0
        total_val = 0

        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
# Move to GPU
                outputs = model(images)  # Forward pass
                loss = criterion(outputs, labels)

                running_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
```

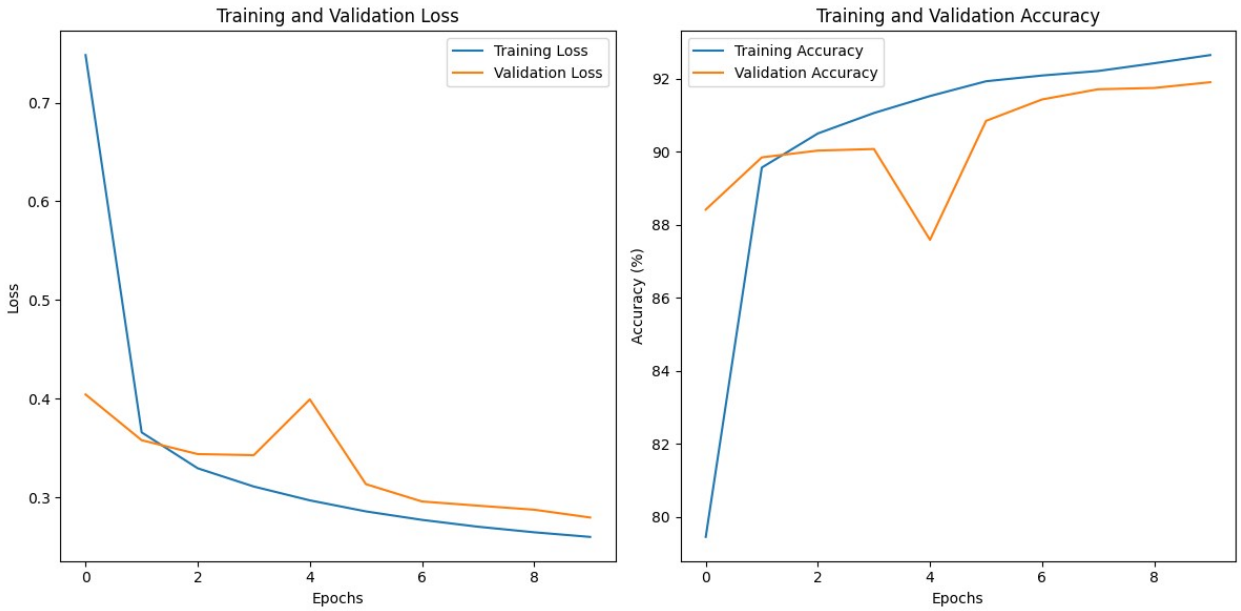
```
                correct_val += (predicted == labels).sum().item()
                total_val += labels.size(0)

        avg_val_loss = running_loss / len(val_loader)
        val_accuracy = correct_val / total_val * 100
        val_losses.append(avg_val_loss)
        val_accuracies.append(val_accuracy)

        print(f"Epoch [{epoch+1}/{num_epochs}], "
              f"Train Loss: {avg_train_loss:.4f}, Train Accuracy:
{train_accuracy:.2f}%, "
              f"Validation Loss: {avg_val_loss:.4f}, Validation
Accuracy: {val_accuracy:.2f}%")

    return train_losses, val_losses, train_accuracies, val_accuracies

# Train the model
train_losses, val_losses, train_accuracies, val_accuracies =
train_model(
    model, train_loader, val_loader, criterion, optimizer,
num_epochs=10
)
```

```
Epoch [1/10], Train Loss: 0.7482, Train Accuracy: 79.45%, Validation
Loss: 0.4044, Validation Accuracy: 88.41%
Epoch [2/10], Train Loss: 0.3660, Train Accuracy: 89.56%, Validation
Loss: 0.3581, Validation Accuracy: 89.84%
Epoch [3/10], Train Loss: 0.3296, Train Accuracy: 90.50%, Validation
Loss: 0.3440, Validation Accuracy: 90.03%
Epoch [4/10], Train Loss: 0.3111, Train Accuracy: 91.06%, Validation
Loss: 0.3429, Validation Accuracy: 90.07%
Epoch [5/10], Train Loss: 0.2971, Train Accuracy: 91.52%, Validation
Loss: 0.3994, Validation Accuracy: 87.59%
Epoch [6/10], Train Loss: 0.2860, Train Accuracy: 91.93%, Validation
Loss: 0.3136, Validation Accuracy: 90.84%
Epoch [7/10], Train Loss: 0.2774, Train Accuracy: 92.08%, Validation
Loss: 0.2960, Validation Accuracy: 91.43%
Epoch [8/10], Train Loss: 0.2704, Train Accuracy: 92.21%, Validation
Loss: 0.2918, Validation Accuracy: 91.71%
Epoch [9/10], Train Loss: 0.2648, Train Accuracy: 92.42%, Validation
Loss: 0.2877, Validation Accuracy: 91.74%
Epoch [10/10], Train Loss: 0.2601, Train Accuracy: 92.64%, Validation
Loss: 0.2798, Validation Accuracy: 91.90%
```

now we plot the performance of our simple model

```
# 5. Plot Training and Validation Loss
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Training Loss')
```

```
plt.plot(val_losses, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# 6. Plot Training and Validation Accuracy
plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Training Accuracy')
plt.plot(val_accuracies, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.legend()

plt.tight_layout()
plt.show()
```




# part 2

for this part we will attempt to tune the model parameters by tweaking a single paramter(the others remain constant) and retraining the model and then comparing results we will use the following paramters:

- number of hiddden layers =[1,3,5,7]
- number of neurons per hidden layer= [10,20,30,40]
- batch size for stochastic gradient descent= [32,64,128,256]
- learning rate =[0.10.01,0.001,0.0001]

in total, we train the model for a total of 16 times.

Total iterations: 6+6+6+6=24

we will create a function the iterates over the said parameters but fristly we define our parameter ranges in a dictionary.

```
# Hyperparameters for tuning
default_params = {
    "hidden_layers": 1,  # Number of hidden layers
    "hidden_size": 10,   # Number of neurons per hidden layer
    "batch_size": 32,    # Batch size for SGD
    "learning_rate": 0.01  # Learning rate
}

# Define the ranges for each hyperparameter
param_ranges = {
    "hidden_layers": [1, 3, 5, 7,9,11],
    "hidden_size": [10, 20, 30, 40, 50 , 60],
    "batch_size": [32, 64, 128, 256,500, 1000 ],
    "learning_rate": [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001]
}
```

then we proceed to create a neural network class with a variable layer size

```
def train_model(model, train_loader, val_loader, criterion, optimizer,
num_epochs=10):
    train_losses, val_losses = [], []
    train_accuracies, val_accuracies = [], []
    y_true, y_pred = [], []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct_train = 0
        total_train = 0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)  #
Move to GPU/CPU
            optimizer.zero_grad() # remove gradients to prevent
gradient accumulatioin
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() # add current batch loss
            _, predicted = torch.max(outputs, 1) # index of max
probability
            correct_train += (predicted == labels).sum().item()
            total_train += labels.size(0)# keep track of all samples
```

```
trained

        avg_train_loss = running_loss / len(train_loader)
        train_accuracy = correct_train / total_train * 100
        train_losses.append(avg_train_loss)
        train_accuracies.append(train_accuracy)

        # Validation phase
        model.eval()
        running_loss = 0.0
        correct_val = 0
        total_val = 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                running_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                correct_val += (predicted == labels).sum().item()
                total_val += labels.size(0)
                y_true.extend(labels.cpu().numpy()) ##move to cpu for
transforming into numpy array
                y_pred.extend(predicted.cpu().numpy())

        avg_val_loss = running_loss / len(val_loader)
        val_accuracy = correct_val / total_val * 100
        val_losses.append(avg_val_loss)
        val_accuracies.append(val_accuracy)

        print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss:
{avg_train_loss:.4f}, Train Accuracy: {train_accuracy:.2f}%, "
              f"Validation Loss: {avg_val_loss:.4f}, Validation
Accuracy: {val_accuracy:.2f}%")

    # Return all six variables
    return train_losses, val_losses, train_accuracies, val_accuracies,
y_true, y_pred

# Function to build a model with variable hidden layers
class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_layers, hidden_size,
num_classes):
        super(NeuralNetwork, self).__init__()
        layers = []
        for i in range(hidden_layers):
            if i == 0:
                layers.append(nn.Linear(input_size, hidden_size)) #
for inout layer
            else:
```

```
                layers.append(nn.Linear(hidden_size, hidden_size))
##every layer in between
            layers.append(nn.ReLU())
        layers.append(nn.Linear(hidden_size, num_classes))  # Final
output layer
        self.network = nn.Sequential(*layers) ##link layers

    def forward(self, x):
        x = x.view(-1, 28 * 28)  # Flatten input
        return self.network(x)
```

a function that iterares over parameter values

```
# Hyperparameter tuning function
def tune_hyperparameters(default_params, param_ranges, num_epochs=10):
    results = []

    # Iterate over each parameter
    for param, values in param_ranges.items():
        for value in values:
            # Update the parameter value
            params = default_params.copy()
            params[param] = value

            # Update DataLoader if batch_size changes
            batch_size = params["batch_size"]
            train_loader = DataLoader(train_data,
batch_size=batch_size, shuffle=True)
            val_loader = DataLoader(val_data, batch_size=batch_size,
shuffle=False)

            # Build the model
            model = NeuralNetwork(
                input_size=28 * 28,
                hidden_layers=params["hidden_layers"],
                hidden_size=params["hidden_size"],
                num_classes=10
            ).to(device)

            # Define optimizer with new learning rate
            optimizer = optim.SGD(model.parameters(),
lr=params["learning_rate"])
            criterion = nn.CrossEntropyLoss()

            # Train the model
            train_losses, val_losses, train_accuracies,
val_accuracies, y_true, y_pred = train_model(
                model, train_loader, val_loader, criterion, optimizer,
num_epochs=num_epochs
```

```
            )

            # Record results
            results.append({
                "param": param,
                "value": value,
                "train_accuracy": train_accuracies[-1],
                "val_accuracy": val_accuracies[-1],
                "train_loss": train_losses[-1],
                "val_loss": val_losses[-1],
                "y_true": y_true,
                "y_pred": y_pred
            })

            print(f"Completed training for {param} = {value}")

    return results

results = tune_hyperparameters(default_params, param_ranges)
```

```
Epoch [1/10], Train Loss: 0.7375, Train Accuracy: 78.66%, Validation
Loss: 0.4049, Validation Accuracy: 88.85%
Epoch [2/10], Train Loss: 0.3615, Train Accuracy: 89.62%, Validation
Loss: 0.3425, Validation Accuracy: 90.00%
Epoch [3/10], Train Loss: 0.3255, Train Accuracy: 90.47%, Validation
Loss: 0.3280, Validation Accuracy: 90.59%
Epoch [4/10], Train Loss: 0.3084, Train Accuracy: 91.12%, Validation
Loss: 0.3418, Validation Accuracy: 90.21%
Epoch [5/10], Train Loss: 0.2994, Train Accuracy: 91.34%, Validation
Loss: 0.3279, Validation Accuracy: 90.37%
Epoch [6/10], Train Loss: 0.2913, Train Accuracy: 91.56%, Validation
Loss: 0.3195, Validation Accuracy: 90.84%
Epoch [7/10], Train Loss: 0.2862, Train Accuracy: 91.77%, Validation
Loss: 0.3006, Validation Accuracy: 91.45%
Epoch [8/10], Train Loss: 0.2806, Train Accuracy: 92.00%, Validation
Loss: 0.3118, Validation Accuracy: 91.15%
Epoch [9/10], Train Loss: 0.2762, Train Accuracy: 92.06%, Validation
Loss: 0.2992, Validation Accuracy: 91.62%
Epoch [10/10], Train Loss: 0.2725, Train Accuracy: 92.13%, Validation
Loss: 0.3069, Validation Accuracy: 91.14%
Completed training for hidden_layers = 1
Epoch [1/10], Train Loss: 1.5809, Train Accuracy: 42.68%, Validation
Loss: 0.8274, Validation Accuracy: 71.73%
Epoch [2/10], Train Loss: 0.6531, Train Accuracy: 78.79%, Validation
Loss: 0.5995, Validation Accuracy: 80.88%
Epoch [3/10], Train Loss: 0.5063, Train Accuracy: 84.48%, Validation
Loss: 0.5137, Validation Accuracy: 83.95%
Epoch [4/10], Train Loss: 0.4301, Train Accuracy: 87.00%, Validation
Loss: 0.4086, Validation Accuracy: 87.99%
Epoch [5/10], Train Loss: 0.3751, Train Accuracy: 88.84%, Validation
```

```
Loss: 0.4090, Validation Accuracy: 87.93%
Epoch [6/10], Train Loss: 0.3418, Train Accuracy: 90.00%, Validation
Loss: 0.3557, Validation Accuracy: 89.86%
Epoch [7/10], Train Loss: 0.3157, Train Accuracy: 90.79%, Validation
Loss: 0.3233, Validation Accuracy: 90.57%
Epoch [8/10], Train Loss: 0.2968, Train Accuracy: 91.26%, Validation
Loss: 0.3579, Validation Accuracy: 89.38%
Epoch [9/10], Train Loss: 0.2832, Train Accuracy: 91.72%, Validation
Loss: 0.2979, Validation Accuracy: 91.59%
Epoch [10/10], Train Loss: 0.2712, Train Accuracy: 92.02%, Validation
Loss: 0.3433, Validation Accuracy: 89.72%
Completed training for hidden_layers = 3
Epoch [1/10], Train Loss: 2.2908, Train Accuracy: 12.75%, Validation
Loss: 2.2298, Validation Accuracy: 18.09%
Epoch [2/10], Train Loss: 1.9531, Train Accuracy: 22.17%, Validation
Loss: 1.7552, Validation Accuracy: 27.11%
Epoch [3/10], Train Loss: 1.5279, Train Accuracy: 40.68%, Validation
Loss: 1.3227, Validation Accuracy: 52.06%
Epoch [4/10], Train Loss: 1.1080, Train Accuracy: 61.25%, Validation
Loss: 0.7788, Validation Accuracy: 75.16%
Epoch [5/10], Train Loss: 0.6799, Train Accuracy: 79.25%, Validation
Loss: 0.5745, Validation Accuracy: 83.99%
Epoch [6/10], Train Loss: 0.5219, Train Accuracy: 84.90%, Validation
Loss: 0.4639, Validation Accuracy: 86.96%
Epoch [7/10], Train Loss: 0.4530, Train Accuracy: 86.78%, Validation
Loss: 0.4664, Validation Accuracy: 86.40%
Epoch [8/10], Train Loss: 0.4098, Train Accuracy: 88.05%, Validation
Loss: 0.4443, Validation Accuracy: 87.19%
Epoch [9/10], Train Loss: 0.3797, Train Accuracy: 88.95%, Validation
Loss: 0.4168, Validation Accuracy: 87.31%
Epoch [10/10], Train Loss: 0.3539, Train Accuracy: 89.63%, Validation
Loss: 0.3976, Validation Accuracy: 88.86%
Completed training for hidden_layers = 5
Epoch [1/10], Train Loss: 2.3053, Train Accuracy: 10.67%, Validation
Loss: 2.3017, Validation Accuracy: 11.26%
Epoch [2/10], Train Loss: 2.3014, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [3/10], Train Loss: 2.3012, Train Accuracy: 11.25%, Validation
Loss: 2.3010, Validation Accuracy: 11.26%
Epoch [4/10], Train Loss: 2.3012, Train Accuracy: 11.25%, Validation
Loss: 2.3009, Validation Accuracy: 11.26%
Epoch [5/10], Train Loss: 2.3011, Train Accuracy: 11.25%, Validation
Loss: 2.3008, Validation Accuracy: 11.26%
Epoch [6/10], Train Loss: 2.3008, Train Accuracy: 11.25%, Validation
Loss: 2.3005, Validation Accuracy: 11.26%
Epoch [7/10], Train Loss: 2.3001, Train Accuracy: 11.25%, Validation
Loss: 2.2989, Validation Accuracy: 11.26%
Epoch [8/10], Train Loss: 2.2944, Train Accuracy: 11.25%, Validation
Loss: 2.2821, Validation Accuracy: 11.26%
```

```
Epoch [9/10], Train Loss: 2.1158, Train Accuracy: 20.66%, Validation
Loss: 1.8593, Validation Accuracy: 19.96%
Epoch [10/10], Train Loss: 1.7073, Train Accuracy: 27.73%, Validation
Loss: 1.5794, Validation Accuracy: 34.14%
Completed training for hidden_layers = 7
Epoch [1/10], Train Loss: 2.3041, Train Accuracy: 10.35%, Validation
Loss: 2.3014, Validation Accuracy: 11.26%
Epoch [2/10], Train Loss: 2.3014, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [3/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [4/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [5/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [6/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [7/10], Train Loss: 2.3012, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [8/10], Train Loss: 2.3012, Train Accuracy: 11.25%, Validation
Loss: 2.3010, Validation Accuracy: 11.26%
Epoch [9/10], Train Loss: 2.3012, Train Accuracy: 11.25%, Validation
Loss: 2.3010, Validation Accuracy: 11.26%
Epoch [10/10], Train Loss: 2.3011, Train Accuracy: 11.25%, Validation
Loss: 2.3009, Validation Accuracy: 11.26%
Completed training for hidden_layers = 9
Epoch [1/10], Train Loss: 2.3072, Train Accuracy: 9.87%, Validation
Loss: 2.3017, Validation Accuracy: 11.26%
Epoch [2/10], Train Loss: 2.3015, Train Accuracy: 11.24%, Validation
Loss: 2.3012, Validation Accuracy: 11.26%
Epoch [3/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [4/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3012, Validation Accuracy: 11.26%
Epoch [5/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [6/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [7/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [8/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [9/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Epoch [10/10], Train Loss: 2.3013, Train Accuracy: 11.25%, Validation
Loss: 2.3011, Validation Accuracy: 11.26%
Completed training for hidden_layers = 11
Epoch [1/10], Train Loss: 0.7830, Train Accuracy: 78.27%, Validation
Loss: 0.4372, Validation Accuracy: 87.58%
```

```
Epoch [2/10], Train Loss: 0.3814, Train Accuracy: 88.90%, Validation
Loss: 0.3614, Validation Accuracy: 89.36%
Epoch [3/10], Train Loss: 0.3350, Train Accuracy: 90.36%, Validation
Loss: 0.3475, Validation Accuracy: 89.90%
Epoch [4/10], Train Loss: 0.3115, Train Accuracy: 91.08%, Validation
Loss: 0.3231, Validation Accuracy: 90.75%
Epoch [5/10], Train Loss: 0.2961, Train Accuracy: 91.46%, Validation
Loss: 0.3086, Validation Accuracy: 90.92%
Epoch [6/10], Train Loss: 0.2833, Train Accuracy: 91.88%, Validation
Loss: 0.2998, Validation Accuracy: 91.51%
Epoch [7/10], Train Loss: 0.2751, Train Accuracy: 92.00%, Validation
Loss: 0.3017, Validation Accuracy: 91.36%
Epoch [8/10], Train Loss: 0.2667, Train Accuracy: 92.17%, Validation
Loss: 0.2924, Validation Accuracy: 91.62%
Epoch [9/10], Train Loss: 0.2601, Train Accuracy: 92.45%, Validation
Loss: 0.2847, Validation Accuracy: 91.79%
Epoch [10/10], Train Loss: 0.2547, Train Accuracy: 92.53%, Validation
Loss: 0.3030, Validation Accuracy: 91.38%
Completed training for hidden_size = 10
Epoch [1/10], Train Loss: 0.7059, Train Accuracy: 80.71%, Validation
Loss: 0.3968, Validation Accuracy: 88.72%
Epoch [2/10], Train Loss: 0.3519, Train Accuracy: 89.83%, Validation
Loss: 0.3286, Validation Accuracy: 90.76%
Epoch [3/10], Train Loss: 0.3127, Train Accuracy: 91.00%, Validation
Loss: 0.3088, Validation Accuracy: 91.26%
Epoch [4/10], Train Loss: 0.2883, Train Accuracy: 91.61%, Validation
Loss: 0.2931, Validation Accuracy: 91.43%
Epoch [5/10], Train Loss: 0.2701, Train Accuracy: 92.24%, Validation
Loss: 0.2775, Validation Accuracy: 91.84%
Epoch [6/10], Train Loss: 0.2552, Train Accuracy: 92.68%, Validation
Loss: 0.2724, Validation Accuracy: 91.91%
Epoch [7/10], Train Loss: 0.2423, Train Accuracy: 92.93%, Validation
Loss: 0.2627, Validation Accuracy: 92.46%
Epoch [8/10], Train Loss: 0.2322, Train Accuracy: 93.30%, Validation
Loss: 0.2504, Validation Accuracy: 92.81%
Epoch [9/10], Train Loss: 0.2232, Train Accuracy: 93.59%, Validation
Loss: 0.2501, Validation Accuracy: 92.60%
Epoch [10/10], Train Loss: 0.2144, Train Accuracy: 93.77%, Validation
Loss: 0.2382, Validation Accuracy: 93.11%
Completed training for hidden_size = 20
Epoch [1/10], Train Loss: 0.6799, Train Accuracy: 82.15%, Validation
Loss: 0.3854, Validation Accuracy: 88.96%
Epoch [2/10], Train Loss: 0.3493, Train Accuracy: 90.03%, Validation
Loss: 0.3376, Validation Accuracy: 90.21%
Epoch [3/10], Train Loss: 0.3111, Train Accuracy: 91.03%, Validation
Loss: 0.3203, Validation Accuracy: 90.54%
Epoch [4/10], Train Loss: 0.2878, Train Accuracy: 91.74%, Validation
Loss: 0.2998, Validation Accuracy: 91.16%
Epoch [5/10], Train Loss: 0.2682, Train Accuracy: 92.25%, Validation
```

```
Loss: 0.2777, Validation Accuracy: 92.16%
Epoch [6/10], Train Loss: 0.2502, Train Accuracy: 92.82%, Validation
Loss: 0.2590, Validation Accuracy: 92.55%
Epoch [7/10], Train Loss: 0.2344, Train Accuracy: 93.33%, Validation
Loss: 0.2470, Validation Accuracy: 92.89%
Epoch [8/10], Train Loss: 0.2208, Train Accuracy: 93.64%, Validation
Loss: 0.2356, Validation Accuracy: 93.29%
Epoch [9/10], Train Loss: 0.2086, Train Accuracy: 94.15%, Validation
Loss: 0.2260, Validation Accuracy: 93.57%
Epoch [10/10], Train Loss: 0.1964, Train Accuracy: 94.40%, Validation
Loss: 0.2206, Validation Accuracy: 93.71%
Completed training for hidden_size = 30
Epoch [1/10], Train Loss: 0.6885, Train Accuracy: 82.46%, Validation
Loss: 0.3941, Validation Accuracy: 88.69%
Epoch [2/10], Train Loss: 0.3496, Train Accuracy: 89.99%, Validation
Loss: 0.3416, Validation Accuracy: 90.18%
Epoch [3/10], Train Loss: 0.3084, Train Accuracy: 91.11%, Validation
Loss: 0.3089, Validation Accuracy: 90.86%
Epoch [4/10], Train Loss: 0.2814, Train Accuracy: 91.87%, Validation
Loss: 0.2868, Validation Accuracy: 91.72%
Epoch [5/10], Train Loss: 0.2602, Train Accuracy: 92.56%, Validation
Loss: 0.2625, Validation Accuracy: 92.36%
Epoch [6/10], Train Loss: 0.2416, Train Accuracy: 93.03%, Validation
Loss: 0.2529, Validation Accuracy: 92.66%
Epoch [7/10], Train Loss: 0.2254, Train Accuracy: 93.51%, Validation
Loss: 0.2498, Validation Accuracy: 92.72%
Epoch [8/10], Train Loss: 0.2101, Train Accuracy: 94.01%, Validation
Loss: 0.2252, Validation Accuracy: 93.63%
Epoch [9/10], Train Loss: 0.1983, Train Accuracy: 94.37%, Validation
Loss: 0.2128, Validation Accuracy: 93.86%
Epoch [10/10], Train Loss: 0.1865, Train Accuracy: 94.65%, Validation
Loss: 0.2021, Validation Accuracy: 94.09%
Completed training for hidden_size = 40
Epoch [1/10], Train Loss: 0.6572, Train Accuracy: 82.99%, Validation
Loss: 0.3835, Validation Accuracy: 88.89%
Epoch [2/10], Train Loss: 0.3438, Train Accuracy: 90.05%, Validation
Loss: 0.3367, Validation Accuracy: 90.19%
Epoch [3/10], Train Loss: 0.3032, Train Accuracy: 91.23%, Validation
Loss: 0.3024, Validation Accuracy: 91.25%
Epoch [4/10], Train Loss: 0.2780, Train Accuracy: 91.93%, Validation
Loss: 0.2769, Validation Accuracy: 92.14%
Epoch [5/10], Train Loss: 0.2553, Train Accuracy: 92.55%, Validation
Loss: 0.2700, Validation Accuracy: 92.50%
Epoch [6/10], Train Loss: 0.2352, Train Accuracy: 93.25%, Validation
Loss: 0.2536, Validation Accuracy: 92.51%
Epoch [7/10], Train Loss: 0.2173, Train Accuracy: 93.80%, Validation
Loss: 0.2385, Validation Accuracy: 92.94%
Epoch [8/10], Train Loss: 0.2018, Train Accuracy: 94.24%, Validation
Loss: 0.2156, Validation Accuracy: 93.89%
```

```
Epoch [9/10], Train Loss: 0.1879, Train Accuracy: 94.61%, Validation
Loss: 0.2070, Validation Accuracy: 94.03%
Epoch [10/10], Train Loss: 0.1769, Train Accuracy: 95.01%, Validation
Loss: 0.1980, Validation Accuracy: 94.24%
Completed training for hidden_size = 50
Epoch [1/10], Train Loss: 0.6829, Train Accuracy: 82.25%, Validation
Loss: 0.3841, Validation Accuracy: 88.79%
Epoch [2/10], Train Loss: 0.3431, Train Accuracy: 90.19%, Validation
Loss: 0.3386, Validation Accuracy: 90.34%
Epoch [3/10], Train Loss: 0.3021, Train Accuracy: 91.25%, Validation
Loss: 0.3040, Validation Accuracy: 91.05%
Epoch [4/10], Train Loss: 0.2758, Train Accuracy: 92.11%, Validation
Loss: 0.2752, Validation Accuracy: 92.09%
Epoch [5/10], Train Loss: 0.2533, Train Accuracy: 92.73%, Validation
Loss: 0.2658, Validation Accuracy: 92.51%
Epoch [6/10], Train Loss: 0.2325, Train Accuracy: 93.41%, Validation
Loss: 0.2407, Validation Accuracy: 93.09%
Epoch [7/10], Train Loss: 0.2145, Train Accuracy: 94.00%, Validation
Loss: 0.2284, Validation Accuracy: 93.46%
Epoch [8/10], Train Loss: 0.1979, Train Accuracy: 94.40%, Validation
Loss: 0.2102, Validation Accuracy: 93.95%
Epoch [9/10], Train Loss: 0.1841, Train Accuracy: 94.73%, Validation
Loss: 0.1988, Validation Accuracy: 94.21%
Epoch [10/10], Train Loss: 0.1720, Train Accuracy: 95.11%, Validation
Loss: 0.2025, Validation Accuracy: 94.11%
Completed training for hidden_size = 60
Epoch [1/10], Train Loss: 0.7847, Train Accuracy: 76.92%, Validation
Loss: 0.4333, Validation Accuracy: 87.92%
Epoch [2/10], Train Loss: 0.3829, Train Accuracy: 88.84%, Validation
Loss: 0.4063, Validation Accuracy: 87.79%
Epoch [3/10], Train Loss: 0.3382, Train Accuracy: 90.24%, Validation
Loss: 0.3421, Validation Accuracy: 90.31%
Epoch [4/10], Train Loss: 0.3161, Train Accuracy: 90.88%, Validation
Loss: 0.3267, Validation Accuracy: 90.89%
Epoch [5/10], Train Loss: 0.3025, Train Accuracy: 91.31%, Validation
Loss: 0.3163, Validation Accuracy: 91.11%
Epoch [6/10], Train Loss: 0.2929, Train Accuracy: 91.50%, Validation
Loss: 0.3152, Validation Accuracy: 90.98%
Epoch [7/10], Train Loss: 0.2848, Train Accuracy: 91.84%, Validation
Loss: 0.3048, Validation Accuracy: 91.55%
Epoch [8/10], Train Loss: 0.2789, Train Accuracy: 91.95%, Validation
Loss: 0.3034, Validation Accuracy: 91.28%
Epoch [9/10], Train Loss: 0.2726, Train Accuracy: 92.27%, Validation
Loss: 0.2915, Validation Accuracy: 91.90%
Epoch [10/10], Train Loss: 0.2675, Train Accuracy: 92.31%, Validation
Loss: 0.2954, Validation Accuracy: 91.54%
Completed training for batch_size = 32
Epoch [1/10], Train Loss: 1.1709, Train Accuracy: 67.06%, Validation
Loss: 0.5740, Validation Accuracy: 84.07%
```

```
Epoch [2/10], Train Loss: 0.4599, Train Accuracy: 87.05%, Validation
Loss: 0.4144, Validation Accuracy: 88.34%
Epoch [3/10], Train Loss: 0.3769, Train Accuracy: 89.15%, Validation
Loss: 0.3688, Validation Accuracy: 89.34%
Epoch [4/10], Train Loss: 0.3460, Train Accuracy: 89.94%, Validation
Loss: 0.3608, Validation Accuracy: 89.37%
Epoch [5/10], Train Loss: 0.3289, Train Accuracy: 90.49%, Validation
Loss: 0.3336, Validation Accuracy: 90.36%
Epoch [6/10], Train Loss: 0.3163, Train Accuracy: 90.89%, Validation
Loss: 0.3378, Validation Accuracy: 89.97%
Epoch [7/10], Train Loss: 0.3064, Train Accuracy: 91.24%, Validation
Loss: 0.3227, Validation Accuracy: 90.85%
Epoch [8/10], Train Loss: 0.2994, Train Accuracy: 91.45%, Validation
Loss: 0.3245, Validation Accuracy: 90.83%
Epoch [9/10], Train Loss: 0.2937, Train Accuracy: 91.55%, Validation
Loss: 0.3337, Validation Accuracy: 90.08%
Epoch [10/10], Train Loss: 0.2875, Train Accuracy: 91.75%, Validation
Loss: 0.3057, Validation Accuracy: 91.14%
Completed training for batch_size = 64
Epoch [1/10], Train Loss: 1.4954, Train Accuracy: 58.78%, Validation
Loss: 0.8367, Validation Accuracy: 79.84%
Epoch [2/10], Train Loss: 0.6464, Train Accuracy: 83.33%, Validation
Loss: 0.5574, Validation Accuracy: 83.61%
Epoch [3/10], Train Loss: 0.4971, Train Accuracy: 86.19%, Validation
Loss: 0.4764, Validation Accuracy: 86.91%
Epoch [4/10], Train Loss: 0.4369, Train Accuracy: 87.69%, Validation
Loss: 0.4433, Validation Accuracy: 86.88%
Epoch [5/10], Train Loss: 0.4001, Train Accuracy: 88.70%, Validation
Loss: 0.4108, Validation Accuracy: 88.36%
Epoch [6/10], Train Loss: 0.3760, Train Accuracy: 89.44%, Validation
Loss: 0.3803, Validation Accuracy: 89.29%
Epoch [7/10], Train Loss: 0.3600, Train Accuracy: 89.80%, Validation
Loss: 0.3711, Validation Accuracy: 89.24%
Epoch [8/10], Train Loss: 0.3477, Train Accuracy: 90.14%, Validation
Loss: 0.3587, Validation Accuracy: 89.96%
Epoch [9/10], Train Loss: 0.3385, Train Accuracy: 90.42%, Validation
Loss: 0.3625, Validation Accuracy: 89.80%
Epoch [10/10], Train Loss: 0.3321, Train Accuracy: 90.58%, Validation
Loss: 0.3496, Validation Accuracy: 90.22%
Completed training for batch_size = 128
Epoch [1/10], Train Loss: 1.7533, Train Accuracy: 45.45%, Validation
Loss: 1.2389, Validation Accuracy: 67.31%
Epoch [2/10], Train Loss: 0.9887, Train Accuracy: 74.61%, Validation
Loss: 0.8346, Validation Accuracy: 78.26%
Epoch [3/10], Train Loss: 0.7366, Train Accuracy: 80.67%, Validation
Loss: 0.6726, Validation Accuracy: 81.87%
Epoch [4/10], Train Loss: 0.6124, Train Accuracy: 83.59%, Validation
Loss: 0.5793, Validation Accuracy: 84.37%
Epoch [5/10], Train Loss: 0.5442, Train Accuracy: 85.01%, Validation
```

```
Loss: 0.5370, Validation Accuracy: 85.05%
Epoch [6/10], Train Loss: 0.4978, Train Accuracy: 86.12%, Validation
Loss: 0.4957, Validation Accuracy: 86.00%
Epoch [7/10], Train Loss: 0.4649, Train Accuracy: 86.94%, Validation
Loss: 0.4705, Validation Accuracy: 87.06%
Epoch [8/10], Train Loss: 0.4405, Train Accuracy: 87.57%, Validation
Loss: 0.4435, Validation Accuracy: 87.66%
Epoch [9/10], Train Loss: 0.4230, Train Accuracy: 88.06%, Validation
Loss: 0.4312, Validation Accuracy: 87.79%
Epoch [10/10], Train Loss: 0.4100, Train Accuracy: 88.40%, Validation
Loss: 0.4349, Validation Accuracy: 87.54%
Completed training for batch_size = 256
Epoch [1/10], Train Loss: 2.0104, Train Accuracy: 36.34%, Validation
Loss: 1.7546, Validation Accuracy: 47.26%
Epoch [2/10], Train Loss: 1.5353, Train Accuracy: 57.02%, Validation
Loss: 1.3201, Validation Accuracy: 65.65%
Epoch [3/10], Train Loss: 1.1488, Train Accuracy: 71.29%, Validation
Loss: 1.0060, Validation Accuracy: 75.84%
Epoch [4/10], Train Loss: 0.8987, Train Accuracy: 78.14%, Validation
Loss: 0.8126, Validation Accuracy: 79.60%
Epoch [5/10], Train Loss: 0.7432, Train Accuracy: 81.49%, Validation
Loss: 0.6907, Validation Accuracy: 82.15%
Epoch [6/10], Train Loss: 0.6437, Train Accuracy: 83.61%, Validation
Loss: 0.6108, Validation Accuracy: 84.04%
Epoch [7/10], Train Loss: 0.5766, Train Accuracy: 85.04%, Validation
Loss: 0.5552, Validation Accuracy: 85.52%
Epoch [8/10], Train Loss: 0.5291, Train Accuracy: 85.99%, Validation
Loss: 0.5152, Validation Accuracy: 86.49%
Epoch [9/10], Train Loss: 0.4948, Train Accuracy: 86.76%, Validation
Loss: 0.4860, Validation Accuracy: 87.16%
Epoch [10/10], Train Loss: 0.4686, Train Accuracy: 87.32%, Validation
Loss: 0.4640, Validation Accuracy: 87.65%
Completed training for batch_size = 500
Epoch [1/10], Train Loss: 2.2332, Train Accuracy: 24.44%, Validation
Loss: 2.1368, Validation Accuracy: 31.89%
Epoch [2/10], Train Loss: 2.0512, Train Accuracy: 38.74%, Validation
Loss: 1.9639, Validation Accuracy: 44.25%
Epoch [3/10], Train Loss: 1.8703, Train Accuracy: 47.13%, Validation
Loss: 1.7719, Validation Accuracy: 49.64%
Epoch [4/10], Train Loss: 1.6690, Train Accuracy: 51.03%, Validation
Loss: 1.5720, Validation Accuracy: 52.77%
Epoch [5/10], Train Loss: 1.4785, Train Accuracy: 55.17%, Validation
Loss: 1.3959, Validation Accuracy: 58.09%
Epoch [6/10], Train Loss: 1.3185, Train Accuracy: 59.60%, Validation
Loss: 1.2530, Validation Accuracy: 61.51%
Epoch [7/10], Train Loss: 1.1886, Train Accuracy: 63.81%, Validation
Loss: 1.1346, Validation Accuracy: 65.31%
Epoch [8/10], Train Loss: 1.0761, Train Accuracy: 68.00%, Validation
Loss: 1.0273, Validation Accuracy: 70.03%
```

```
Epoch [9/10], Train Loss: 0.9769, Train Accuracy: 72.28%, Validation
Loss: 0.9377, Validation Accuracy: 74.06%
Epoch [10/10], Train Loss: 0.8952, Train Accuracy: 75.59%, Validation
Loss: 0.8636, Validation Accuracy: 76.41%
Completed training for batch_size = 1000
Epoch [1/10], Train Loss: 1.5825, Train Accuracy: 40.59%, Validation
Loss: 1.5127, Validation Accuracy: 43.09%
Epoch [2/10], Train Loss: 1.4450, Train Accuracy: 47.60%, Validation
Loss: 1.9999, Validation Accuracy: 30.75%
Epoch [3/10], Train Loss: 1.4132, Train Accuracy: 49.45%, Validation
Loss: 1.4774, Validation Accuracy: 45.89%
Epoch [4/10], Train Loss: 1.4174, Train Accuracy: 49.92%, Validation
Loss: 1.2822, Validation Accuracy: 52.94%
Epoch [5/10], Train Loss: 1.3994, Train Accuracy: 50.66%, Validation
Loss: 1.2698, Validation Accuracy: 55.84%
Epoch [6/10], Train Loss: 1.3902, Train Accuracy: 51.32%, Validation
Loss: 1.4865, Validation Accuracy: 46.70%
Epoch [7/10], Train Loss: 1.3996, Train Accuracy: 51.24%, Validation
Loss: 1.3105, Validation Accuracy: 52.69%
Epoch [8/10], Train Loss: 1.3798, Train Accuracy: 52.22%, Validation
Loss: 1.5280, Validation Accuracy: 42.46%
Epoch [9/10], Train Loss: 1.3959, Train Accuracy: 51.45%, Validation
Loss: 1.5854, Validation Accuracy: 48.56%
Epoch [10/10], Train Loss: 1.3954, Train Accuracy: 51.31%, Validation
Loss: 1.4562, Validation Accuracy: 52.24%
Completed training for learning_rate = 0.1
Epoch [1/10], Train Loss: 0.7632, Train Accuracy: 77.65%, Validation
Loss: 0.4697, Validation Accuracy: 87.01%
Epoch [2/10], Train Loss: 0.4062, Train Accuracy: 88.29%, Validation
Loss: 0.3892, Validation Accuracy: 89.02%
Epoch [3/10], Train Loss: 0.3493, Train Accuracy: 89.98%, Validation
Loss: 0.3756, Validation Accuracy: 88.93%
Epoch [4/10], Train Loss: 0.3238, Train Accuracy: 90.80%, Validation
Loss: 0.3406, Validation Accuracy: 90.23%
Epoch [5/10], Train Loss: 0.3091, Train Accuracy: 91.28%, Validation
Loss: 0.3290, Validation Accuracy: 90.48%
Epoch [6/10], Train Loss: 0.2981, Train Accuracy: 91.58%, Validation
Loss: 0.3862, Validation Accuracy: 88.66%
Epoch [7/10], Train Loss: 0.2910, Train Accuracy: 91.76%, Validation
Loss: 0.3117, Validation Accuracy: 91.05%
Epoch [8/10], Train Loss: 0.2847, Train Accuracy: 91.97%, Validation
Loss: 0.3117, Validation Accuracy: 91.19%
Epoch [9/10], Train Loss: 0.2781, Train Accuracy: 92.16%, Validation
Loss: 0.3039, Validation Accuracy: 91.27%
Epoch [10/10], Train Loss: 0.2725, Train Accuracy: 92.25%, Validation
Loss: 0.2952, Validation Accuracy: 91.54%
Completed training for learning_rate = 0.01
Epoch [1/10], Train Loss: 1.9715, Train Accuracy: 34.96%, Validation
Loss: 1.6808, Validation Accuracy: 49.49%
```

```
Epoch [2/10], Train Loss: 1.4039, Train Accuracy: 63.21%, Validation
Loss: 1.1364, Validation Accuracy: 73.24%
Epoch [3/10], Train Loss: 0.9567, Train Accuracy: 76.05%, Validation
Loss: 0.8264, Validation Accuracy: 78.46%
Epoch [4/10], Train Loss: 0.7429, Train Accuracy: 79.79%, Validation
Loss: 0.6832, Validation Accuracy: 81.44%
Epoch [5/10], Train Loss: 0.6359, Train Accuracy: 82.38%, Validation
Loss: 0.6021, Validation Accuracy: 83.33%
Epoch [6/10], Train Loss: 0.5704, Train Accuracy: 84.10%, Validation
Loss: 0.5497, Validation Accuracy: 84.59%
Epoch [7/10], Train Loss: 0.5250, Train Accuracy: 85.22%, Validation
Loss: 0.5108, Validation Accuracy: 85.81%
Epoch [8/10], Train Loss: 0.4912, Train Accuracy: 86.13%, Validation
Loss: 0.4818, Validation Accuracy: 86.36%
Epoch [9/10], Train Loss: 0.4646, Train Accuracy: 86.78%, Validation
Loss: 0.4590, Validation Accuracy: 87.15%
Epoch [10/10], Train Loss: 0.4436, Train Accuracy: 87.44%, Validation
Loss: 0.4409, Validation Accuracy: 87.44%
Completed training for learning_rate = 0.001
Epoch [1/10], Train Loss: 2.2552, Train Accuracy: 18.57%, Validation
Loss: 2.1967, Validation Accuracy: 27.56%
Epoch [2/10], Train Loss: 2.1459, Train Accuracy: 34.03%, Validation
Loss: 2.0996, Validation Accuracy: 38.24%
Epoch [3/10], Train Loss: 2.0518, Train Accuracy: 41.58%, Validation
Loss: 2.0044, Validation Accuracy: 45.31%
Epoch [4/10], Train Loss: 1.9575, Train Accuracy: 48.05%, Validation
Loss: 1.9144, Validation Accuracy: 50.22%
Epoch [5/10], Train Loss: 1.8720, Train Accuracy: 51.59%, Validation
Loss: 1.8331, Validation Accuracy: 52.94%
Epoch [6/10], Train Loss: 1.7941, Train Accuracy: 53.77%, Validation
Loss: 1.7588, Validation Accuracy: 55.06%
Epoch [7/10], Train Loss: 1.7226, Train Accuracy: 55.56%, Validation
Loss: 1.6901, Validation Accuracy: 56.54%
Epoch [8/10], Train Loss: 1.6560, Train Accuracy: 57.13%, Validation
Loss: 1.6256, Validation Accuracy: 57.64%
Epoch [9/10], Train Loss: 1.5934, Train Accuracy: 58.37%, Validation
Loss: 1.5648, Validation Accuracy: 59.46%
Epoch [10/10], Train Loss: 1.5339, Train Accuracy: 59.72%, Validation
Loss: 1.5063, Validation Accuracy: 60.79%
Completed training for learning_rate = 0.0001
Epoch [1/10], Train Loss: 2.3281, Train Accuracy: 11.28%, Validation
Loss: 2.3236, Validation Accuracy: 11.39%
Epoch [2/10], Train Loss: 2.3212, Train Accuracy: 11.64%, Validation
Loss: 2.3187, Validation Accuracy: 11.56%
Epoch [3/10], Train Loss: 2.3173, Train Accuracy: 11.78%, Validation
Loss: 2.3158, Validation Accuracy: 11.59%
Epoch [4/10], Train Loss: 2.3150, Train Accuracy: 11.83%, Validation
Loss: 2.3139, Validation Accuracy: 11.59%
Epoch [5/10], Train Loss: 2.3133, Train Accuracy: 11.85%, Validation
```

```
Loss: 2.3125, Validation Accuracy: 11.63%
Epoch [6/10], Train Loss: 2.3119, Train Accuracy: 11.86%, Validation
Loss: 2.3113, Validation Accuracy: 11.68%
Epoch [7/10], Train Loss: 2.3107, Train Accuracy: 11.94%, Validation
Loss: 2.3101, Validation Accuracy: 11.70%
Epoch [8/10], Train Loss: 2.3095, Train Accuracy: 11.95%, Validation
Loss: 2.3090, Validation Accuracy: 11.76%
Epoch [9/10], Train Loss: 2.3084, Train Accuracy: 11.97%, Validation
Loss: 2.3079, Validation Accuracy: 11.76%
Epoch [10/10], Train Loss: 2.3072, Train Accuracy: 12.06%, Validation
Loss: 2.3067, Validation Accuracy: 11.84%
Completed training for learning_rate = 1e-05
Epoch [1/10], Train Loss: 2.3287, Train Accuracy: 10.54%, Validation
Loss: 2.3273, Validation Accuracy: 10.68%
Epoch [2/10], Train Loss: 2.3270, Train Accuracy: 10.52%, Validation
Loss: 2.3256, Validation Accuracy: 10.72%
Epoch [3/10], Train Loss: 2.3253, Train Accuracy: 10.48%, Validation
Loss: 2.3239, Validation Accuracy: 10.61%
Epoch [4/10], Train Loss: 2.3237, Train Accuracy: 10.46%, Validation
Loss: 2.3224, Validation Accuracy: 10.56%
Epoch [5/10], Train Loss: 2.3221, Train Accuracy: 10.39%, Validation
Loss: 2.3208, Validation Accuracy: 10.51%
Epoch [6/10], Train Loss: 2.3206, Train Accuracy: 10.35%, Validation
Loss: 2.3194, Validation Accuracy: 10.49%
Epoch [7/10], Train Loss: 2.3191, Train Accuracy: 10.30%, Validation
Loss: 2.3179, Validation Accuracy: 10.44%
Epoch [8/10], Train Loss: 2.3177, Train Accuracy: 10.23%, Validation
Loss: 2.3165, Validation Accuracy: 10.41%
Epoch [9/10], Train Loss: 2.3163, Train Accuracy: 10.20%, Validation
Loss: 2.3152, Validation Accuracy: 10.36%
Epoch [10/10], Train Loss: 2.3150, Train Accuracy: 10.18%, Validation
Loss: 2.3138, Validation Accuracy: 10.32%
Completed training for learning_rate = 1e-06
```

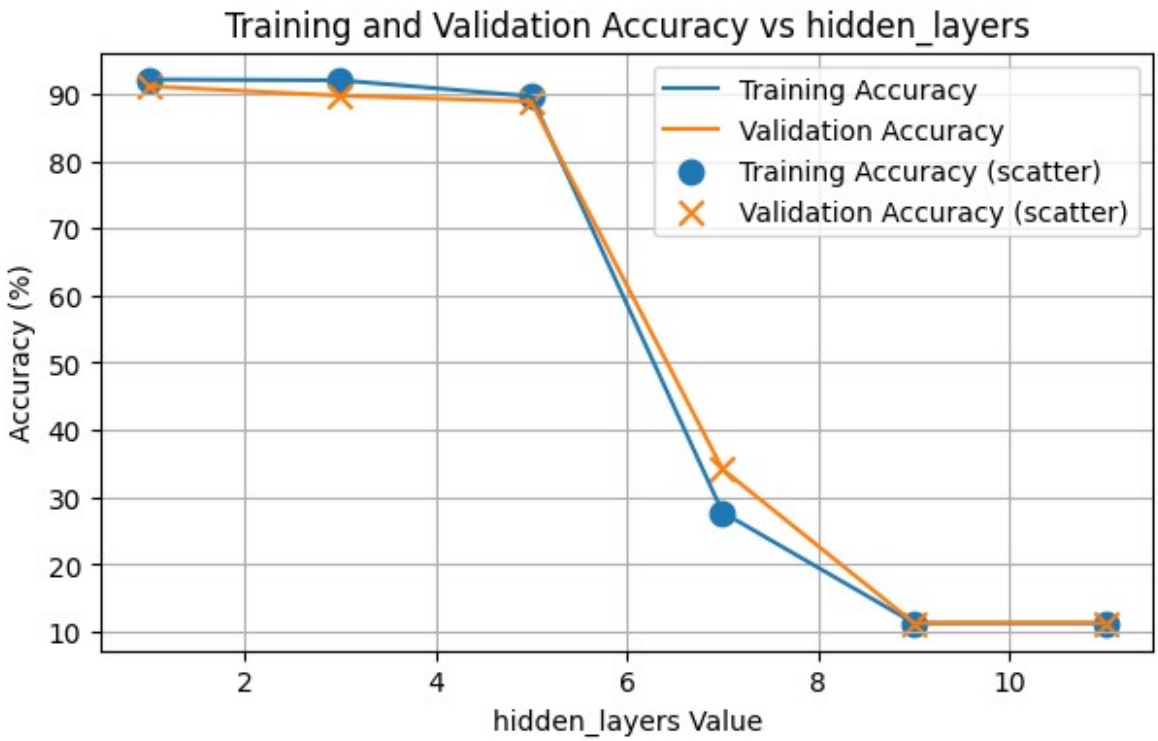
```
import pandas as pd

# Convert the results list to a DataFrame
df = pd.DataFrame(results)

# Inspect the columns to ensure they match expected names
print("Columns in DataFrame:", df.columns)

# Verify sample rows to confirm data integrity
print(df.head())
```

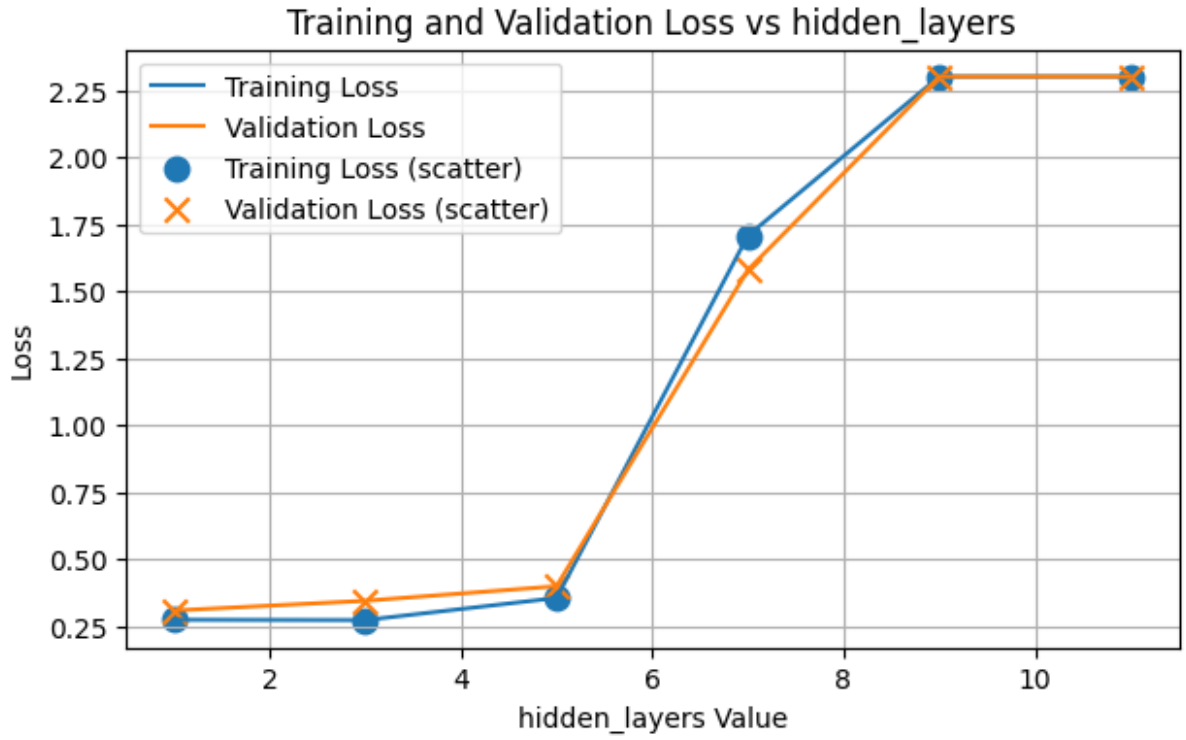
```
Columns in DataFrame: Index(['param', 'value', 'train_accuracy',
'val_accuracy', 'train_loss',
       'val_loss', 'y_true', 'y_pred'],
      dtype='object')
          param  value  train_accuracy  val_accuracy  train_loss
```

```
val_loss  \
0  hidden_layers    1.0       92.130952     91.135714    0.272497
0.306914
1  hidden_layers    3.0       92.021429     89.721429    0.271163
0.343308
2  hidden_layers    5.0       89.633333     88.857143    0.353852
0.397629
3  hidden_layers    7.0       27.730952     34.135714    1.707313
1.579375
4  hidden_layers    9.0       11.252381     11.257143    2.301109
2.300940

                                                  y_true  \
0  [3, 6, 1, 0, 3, 3, 6, 4, 0, 4, 3, 9, 1, 8, 1, ...
1  [3, 6, 1, 0, 3, 3, 6, 4, 0, 4, 3, 9, 1, 8, 1, ...
2  [3, 6, 1, 0, 3, 3, 6, 4, 0, 4, 3, 9, 1, 8, 1, ...
3  [3, 6, 1, 0, 3, 3, 6, 4, 0, 4, 3, 9, 1, 8, 1, ...
4  [3, 6, 1, 0, 3, 3, 6, 4, 0, 4, 3, 9, 1, 8, 1, ...

                                                  y_pred
0  [3, 6, 1, 0, 8, 3, 6, 4, 0, 4, 2, 9, 1, 8, 1, ...
1  [5, 6, 1, 0, 7, 5, 6, 4, 0, 7, 8, 9, 1, 8, 1, ...
2  [8, 2, 1, 6, 1, 8, 8, 1, 8, 1, 1, 1, 1, 2, 1, ...
3  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
4  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
```

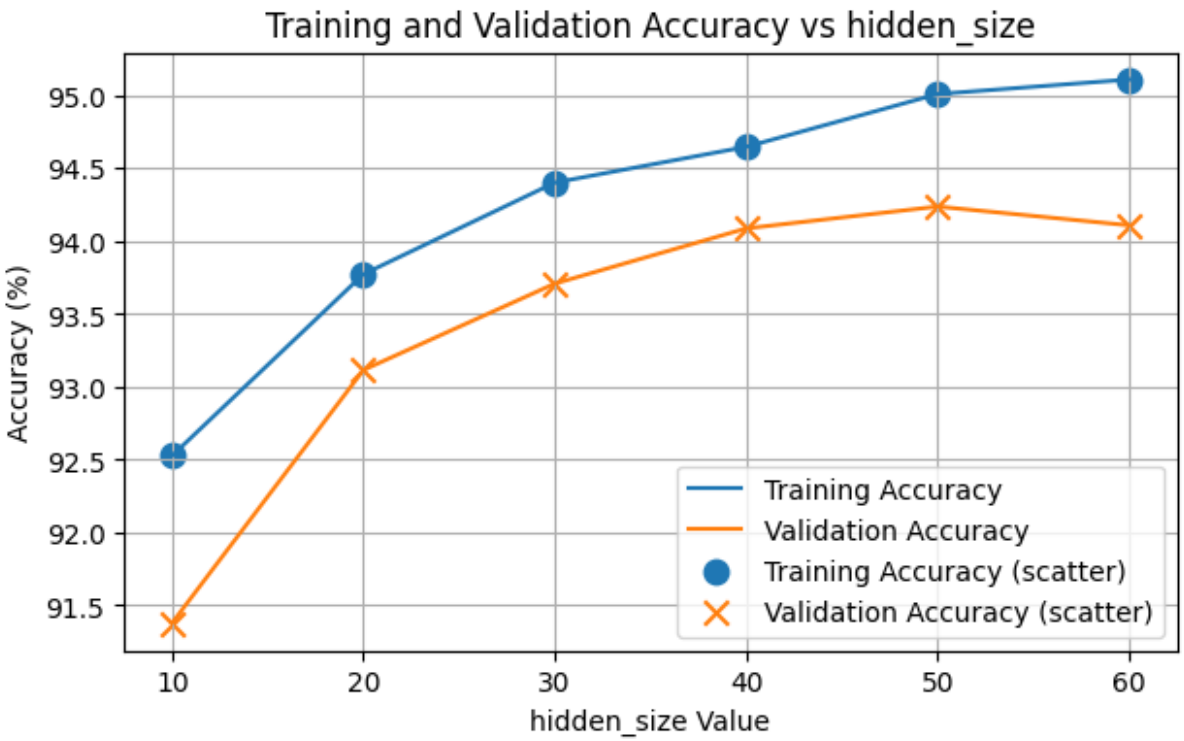
```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Plot Training & Validation Accuracy and Loss for each parameter
separately
unique_params = df["param"].unique()

for param in unique_params:
    # Filter the DataFrame for the current parameter
    subset = df[df["param"] == param]

    print(f"---------------------------------------------- Accuracy and
Loss for {param} ---------------------------------------------- ")

    # Plot Training and Validation Accuracy on the same graph
    plt.figure(figsize=(7, 4))
    plt.plot(subset["value"], subset["train_accuracy"], linestyle='-',
label="Training Accuracy")
    plt.plot(subset["value"], subset["val_accuracy"], linestyle='-',
label="Validation Accuracy")
    plt.scatter(subset["value"], subset["train_accuracy"], marker='o',
s=80, label="Training Accuracy (scatter)")
    plt.scatter(subset["value"], subset["val_accuracy"], marker='x',
s=80, label="Validation Accuracy (scatter)")
```

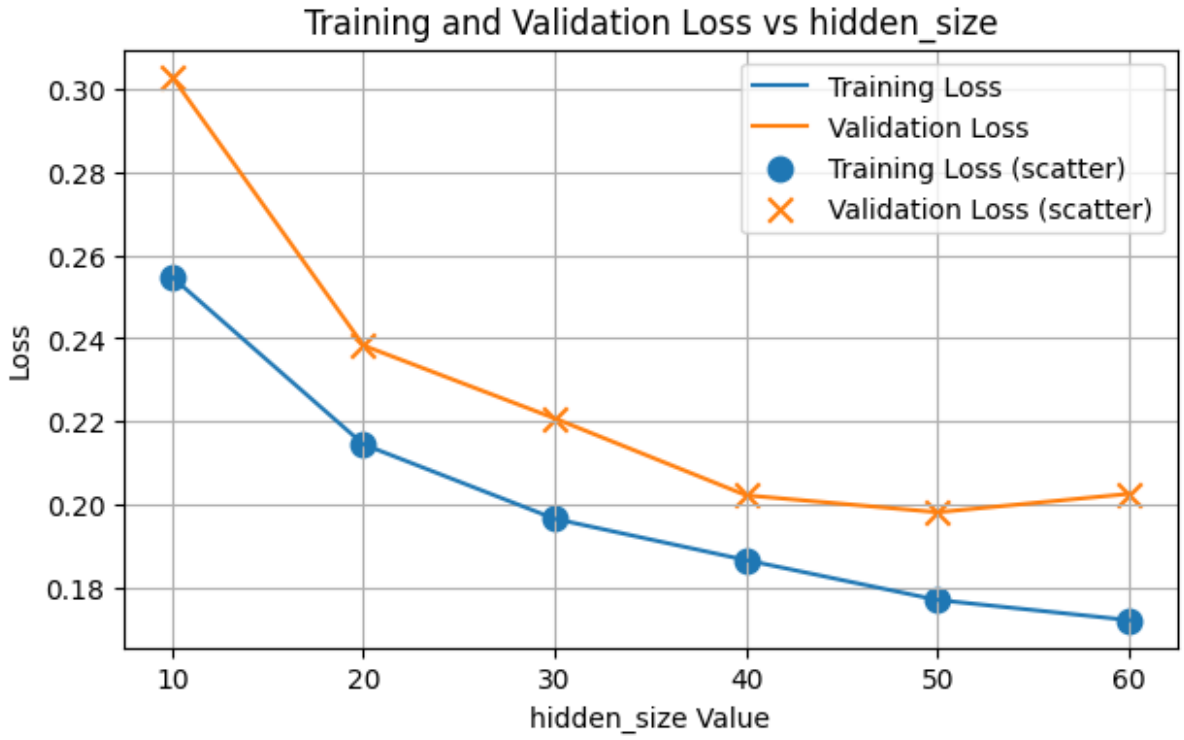
```
    plt.title(f"Training and Validation Accuracy vs {param}")
    plt.xlabel(f"{param} Value")
    plt.ylabel("Accuracy (%)")
    plt.grid()
    plt.legend()
    plt.show()

    # Plot Training and Validation Loss on the same graph
    plt.figure(figsize=(7, 4))
    plt.plot(subset["value"], subset["train_loss"], linestyle='-',
label="Training Loss")
    plt.plot(subset["value"], subset["val_loss"], linestyle='-',
label="Validation Loss")
    plt.scatter(subset["value"], subset["train_loss"], marker='o',
s=80, label="Training Loss (scatter)")
    plt.scatter(subset["value"], subset["val_loss"], marker='x', s=80,
label="Validation Loss (scatter)")
    plt.title(f"Training and Validation Loss vs {param}")
    plt.xlabel(f"{param} Value")
    plt.ylabel("Loss")
    plt.grid()
    plt.legend()
    plt.show()

print(f"------------------------------------------- Confusion Matrix
for {param} ------------------------------------------- ")

# Plot confusion matrix for each trained model
def plot_confusion_matrix(y_true, y_pred, param, value):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(7, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=range(10), yticklabels=range(10))
    plt.title(f'Confusion Matrix for {param} = {value}')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Generate confusion matrices for each parameter and value
for idx, row in df.iterrows():
    plot_confusion_matrix(row["y_true"], row["y_pred"], row["param"],
row["value"])

------------------------------------------- Accuracy and Loss for
hidden_layers -------------------------------------------
```



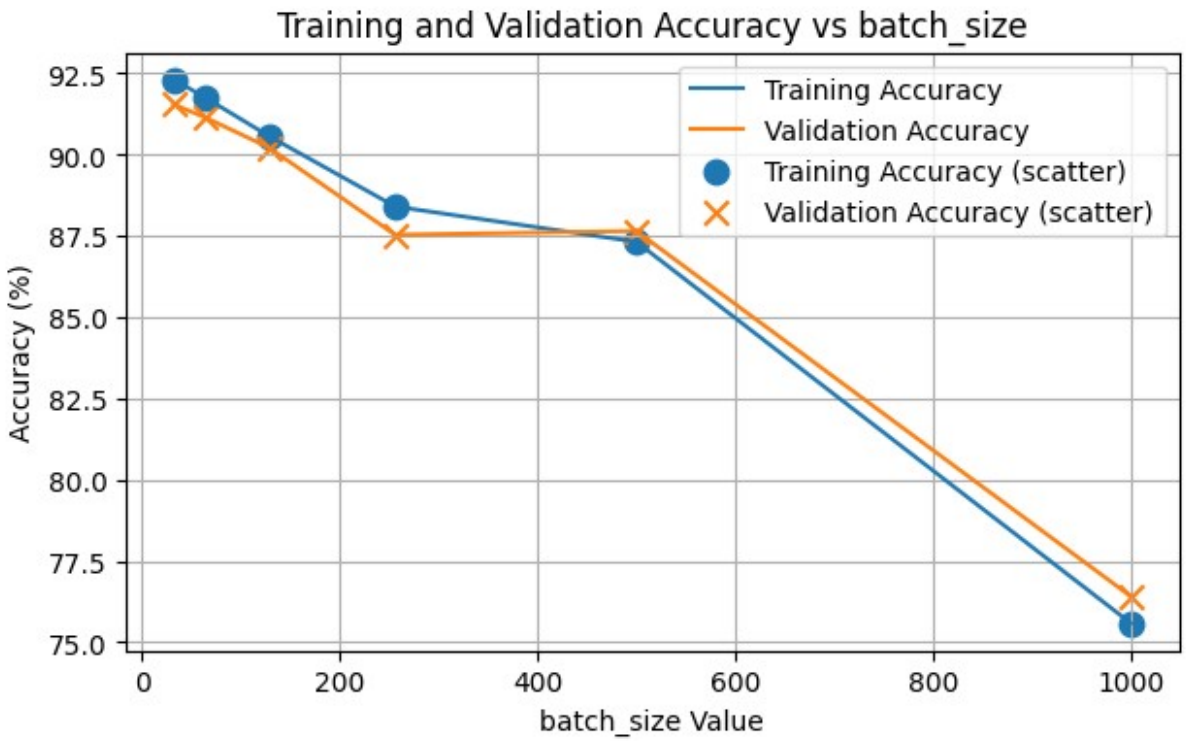



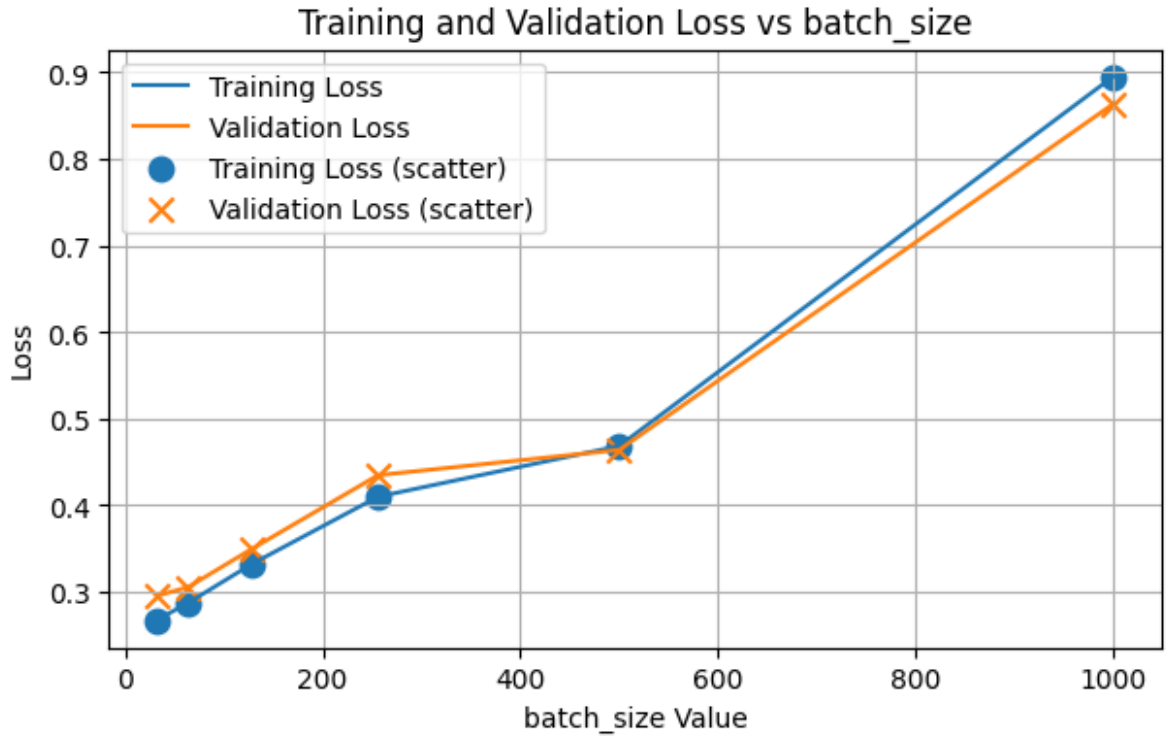

```
---------------------------------------------- Accuracy and Loss for hidden_size ----------------------------------------------
```



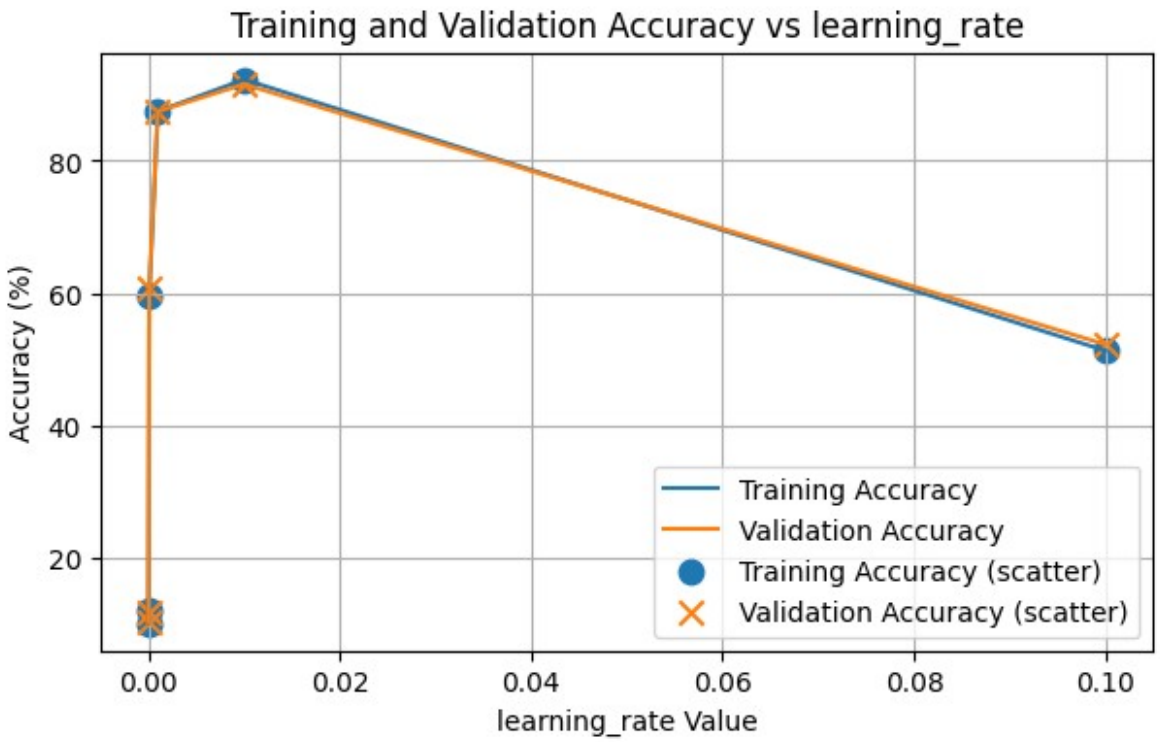



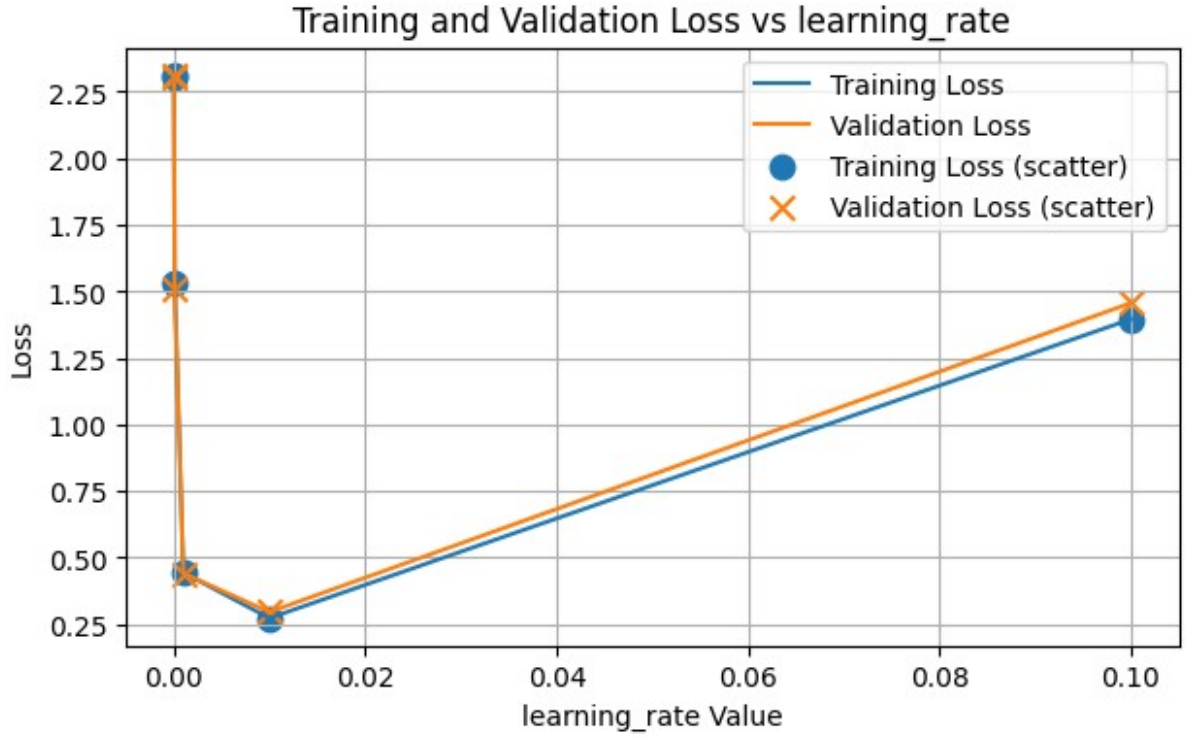

```
---------------------------------------------- Accuracy and Loss for batch_size ----------------------------------------------
```



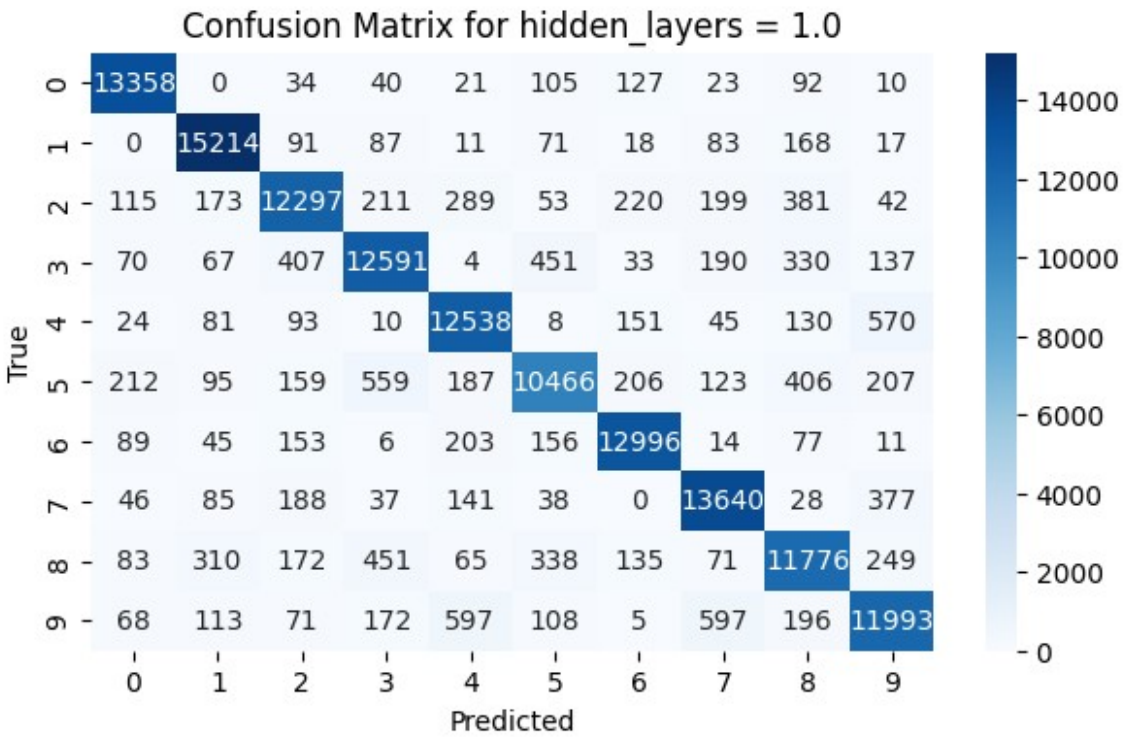



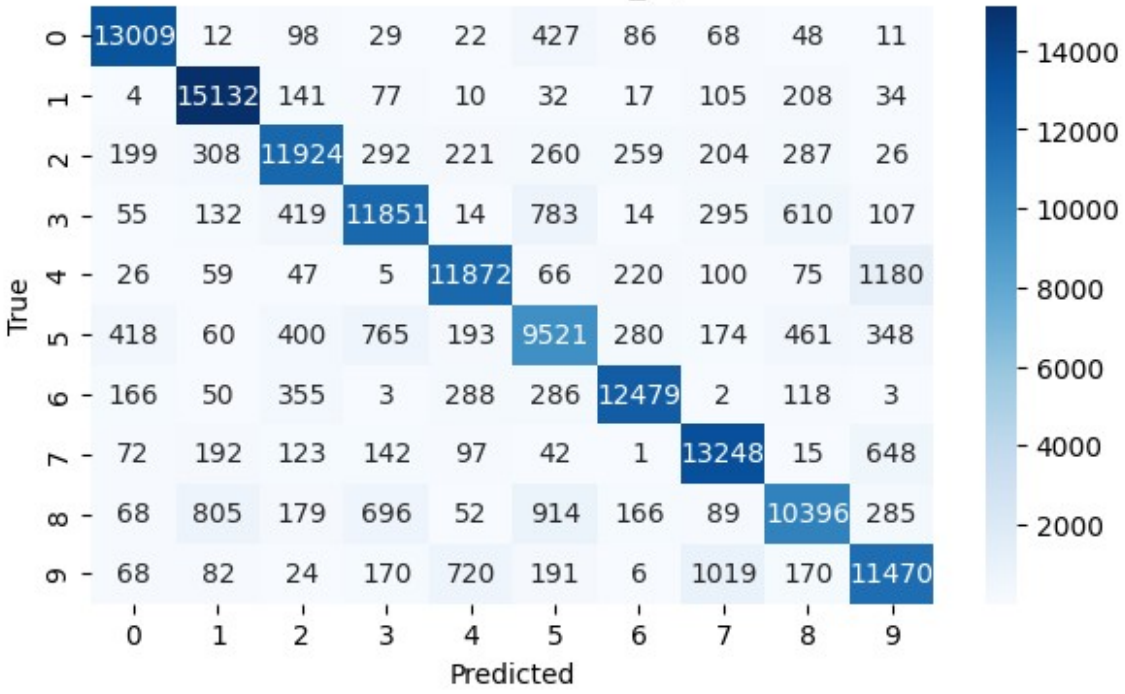

```
---------------------------------------------- Accuracy and Loss for learning_rate ----------------------------------------------
```

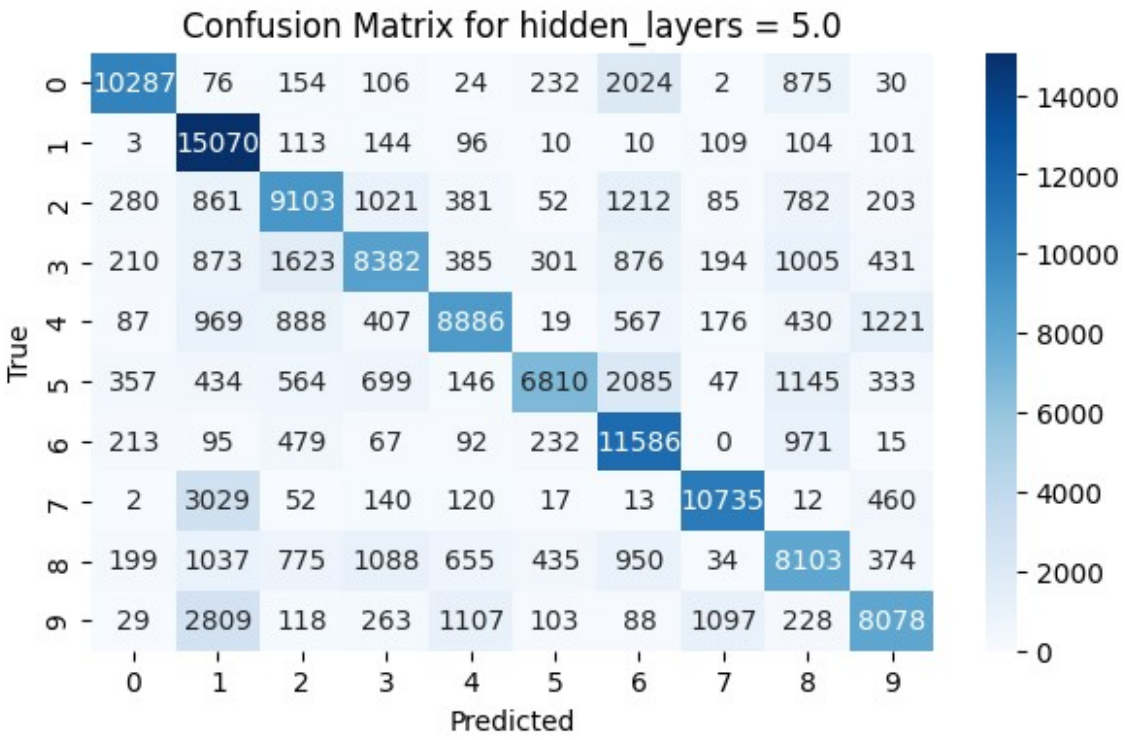


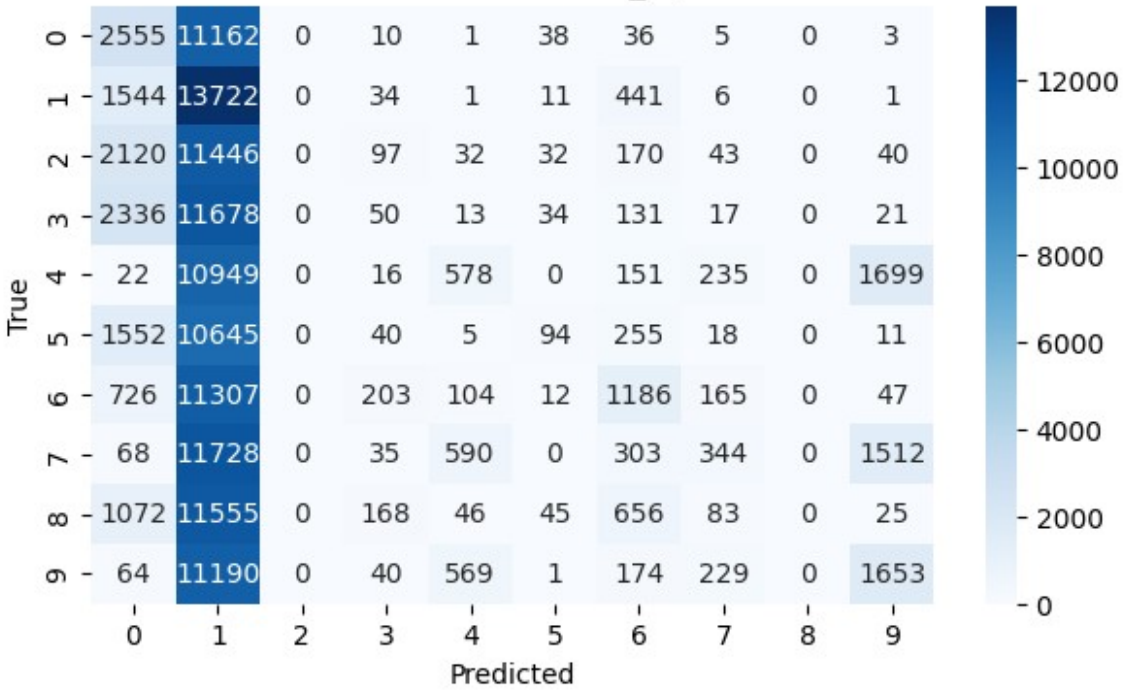



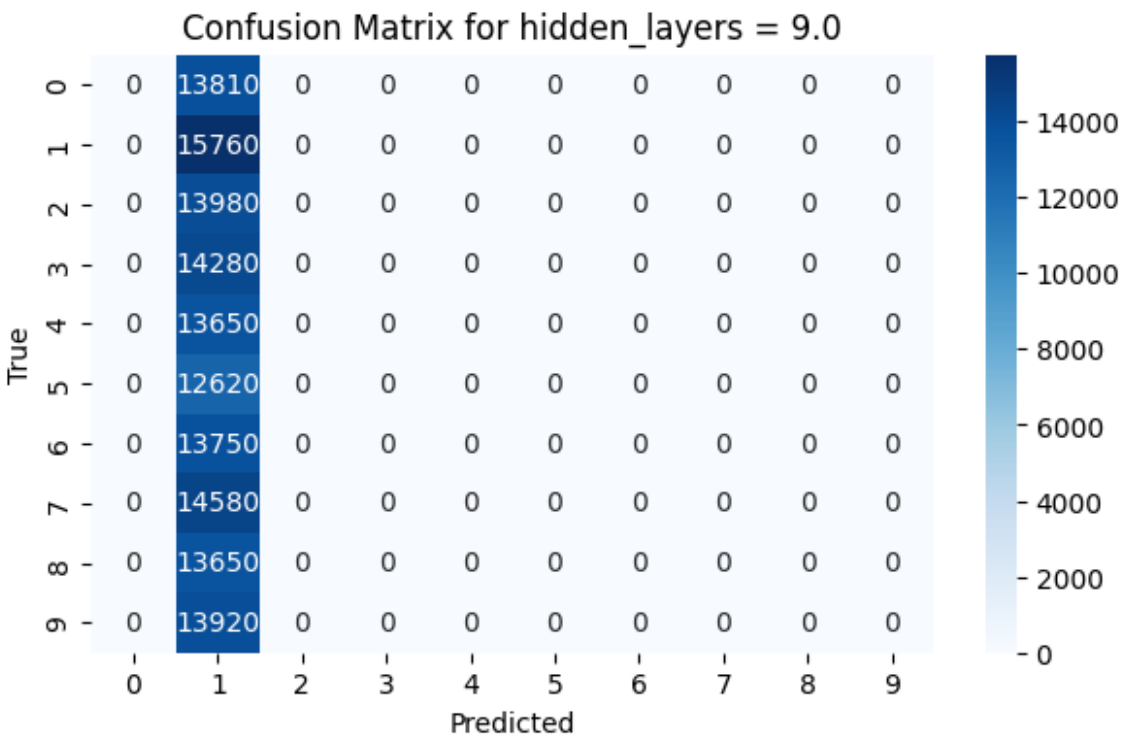

```
------------------------------------------- Confusion Matrix for learning_rate -------------------------------------------
```

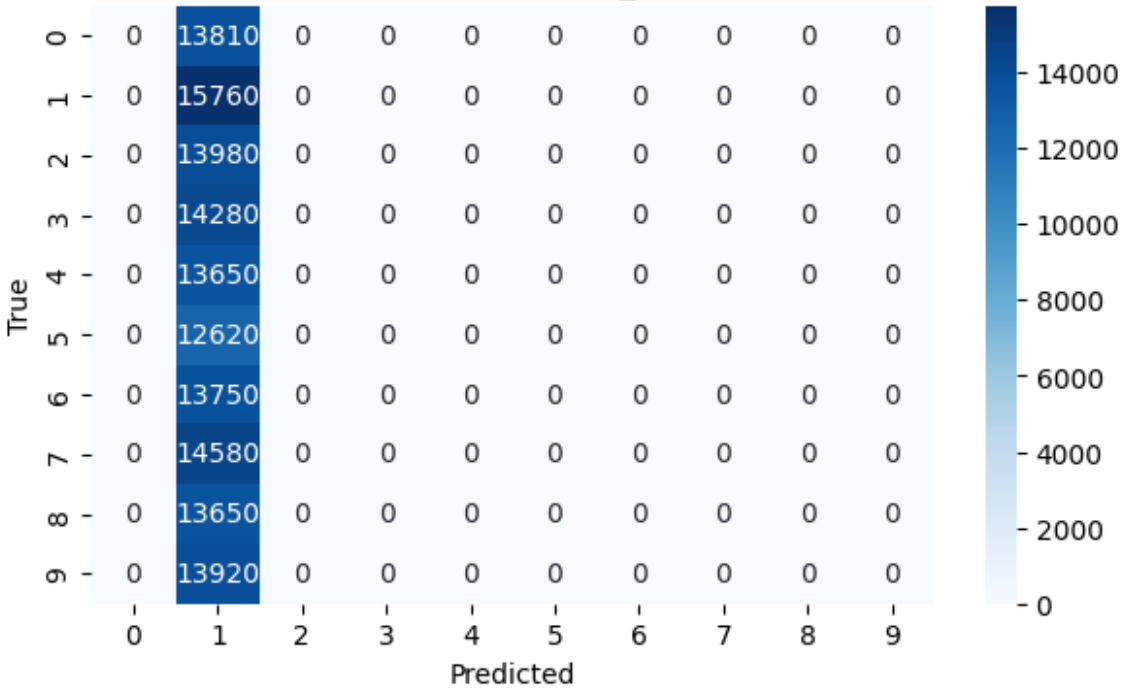

Confusion Matrix for hidden_layers = 1.0
True
Predicted
0 13358 0 34 40 21 105 127 23 92 10
1 0 15214 91 87 11 71 18 83 168 17
2 115 173 12297 211 289 53 220 199 381 42
3 70 67 407 12591 4 451 33 190 330 137
4 24 81 93 10 12538 8 151 45 130 570
5 212 95 159 559 187 10466 206 123 406 207
6 89 45 153 6 203 156 12996 14 77 11
7 46 85 188 37 141 38 0 13640 28 377
8 83 310 172 451 65 338 135 71 11776 249
9 68 113 71 172 597 108 5 597 196 11993
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

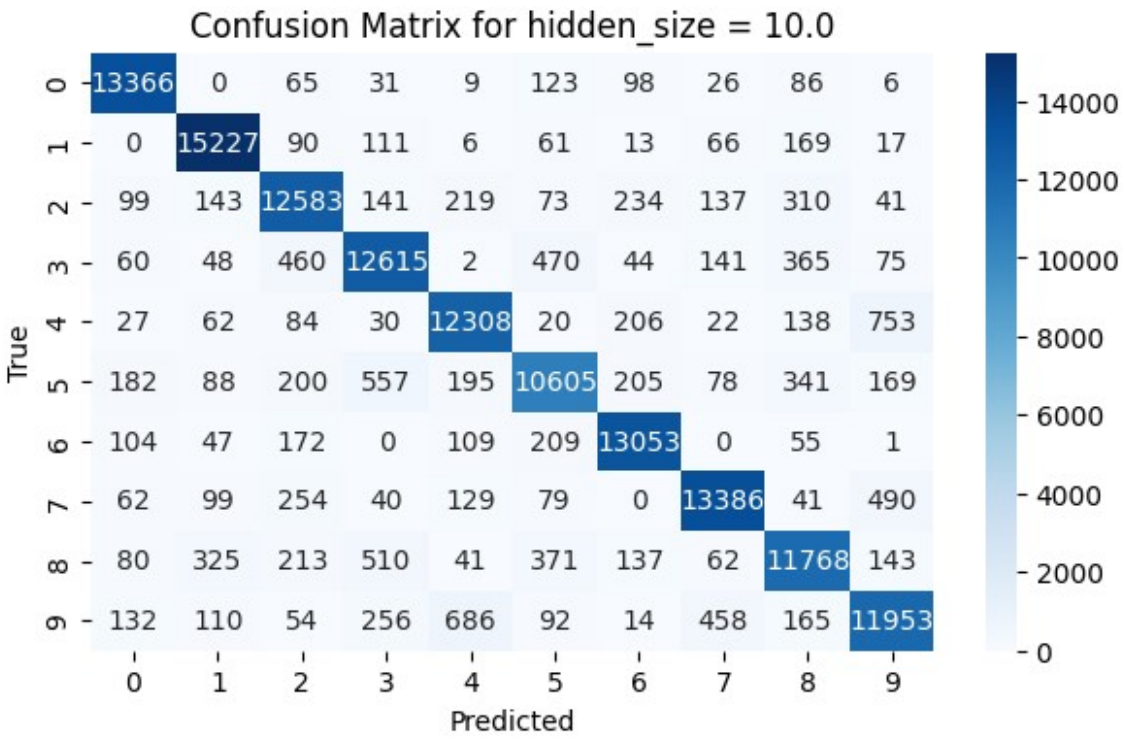

Confusion Matrix for hidden_layers = 3.0
True
Predicted
0 13009 12 98 29 22 427 86 68 48 11
1 4 15132 141 77 10 32 17 105 208 34
2 199 308 11924 292 221 260 259 204 287 26
3 55 132 419 11851 14 783 14 295 610 107
4 26 59 47 5 11872 66 220 100 75 1180
5 418 60 400 765 193 9521 280 174 461 348
6 166 50 355 3 288 286 12479 2 118 3
7 72 192 123 142 97 42 1 13248 15 648
8 68 805 179 696 52 914 166 89 10396 285
9 68 82 24 170 720 191 6 1019 170 11470
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000

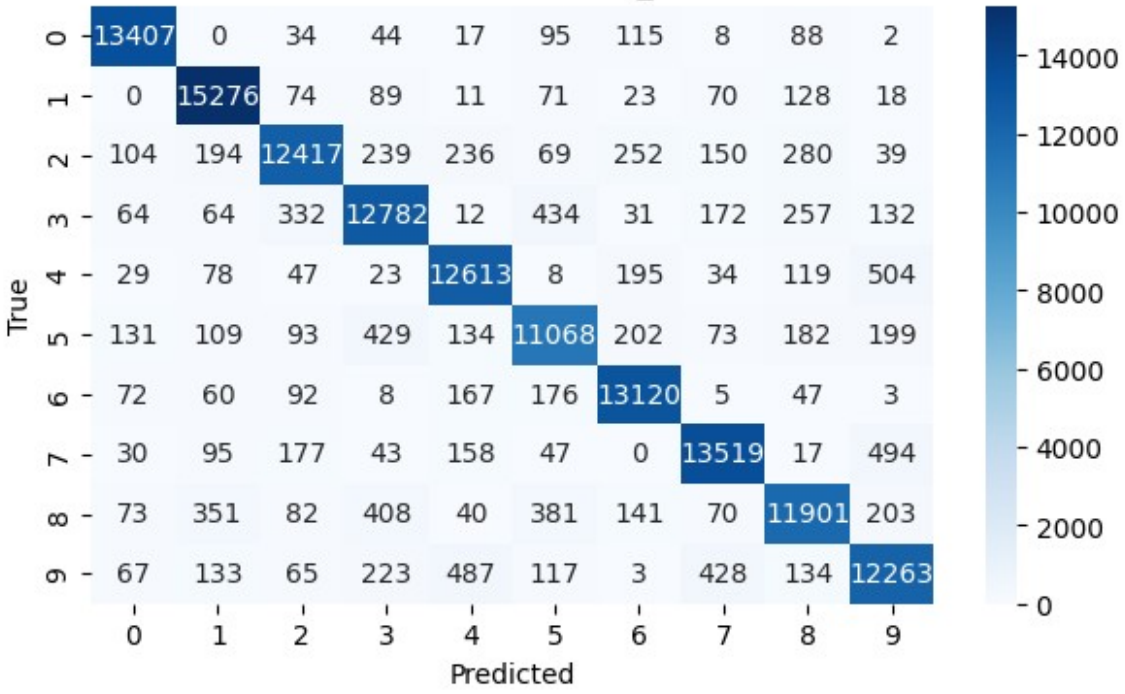

Confusion Matrix for hidden_layers = 5.0
True
Predicted
0 10287 76 154 106 24 232 2024 2 875 30
1 3 15070 113 144 96 10 10 109 104 101
2 280 861 9103 1021 381 52 1212 85 782 203
3 210 873 1623 8382 385 301 876 194 1005 431
4 87 969 888 407 8886 19 567 176 430 1221
5 357 434 564 699 146 6810 2085 47 1145 333
6 213 95 479 67 92 232 11586 0 971 15
7 2 3029 52 140 120 17 13 10735 12 460
8 199 1037 775 1088 655 435 950 34 8103 374
9 29 2809 118 263 1107 103 88 1097 228 8078
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

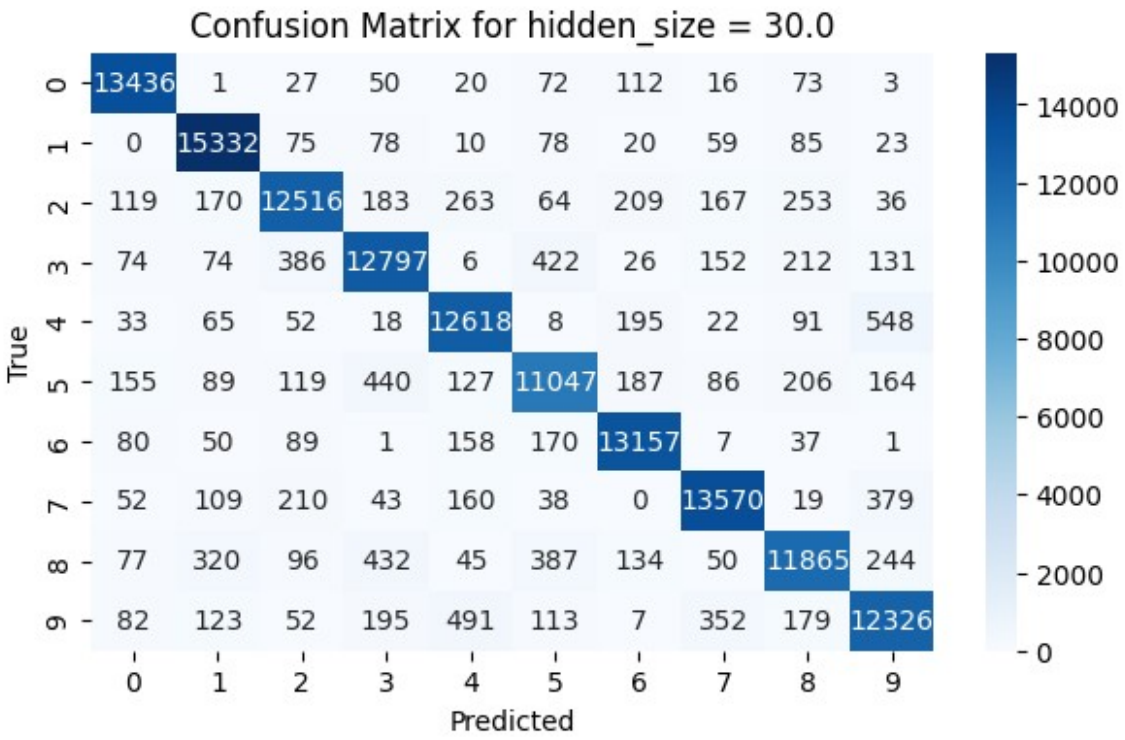

Confusion Matrix for hidden_layers = 7.0
True
Predicted
0 2555 11162 0 10 1 38 36 5 0 3
1 1544 13722 0 34 1 11 441 6 0 1
2 2120 11446 0 97 32 32 170 43 0 40
3 2336 11678 0 50 13 34 131 17 0 21
4 22 10949 0 16 578 0 151 235 0 1699
5 1552 10645 0 40 5 94 255 18 0 11
6 726 11307 0 203 104 12 1186 165 0 47
7 68 11728 0 35 590 0 303 344 0 1512
8 1072 11555 0 168 46 45 656 83 0 25
9 64 11190 0 40 569 1 174 229 0 1653
0 1 2 3 4 5 6 7 8 9
12000
10000
8000
6000
4000
2000
0

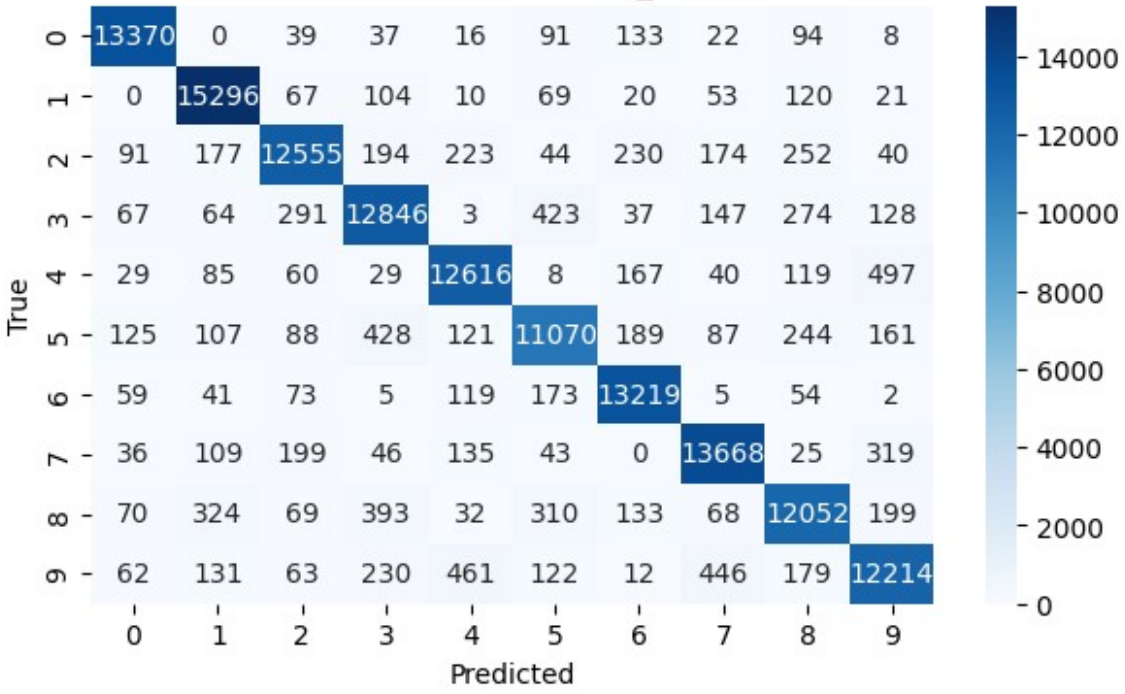

Confusion Matrix for hidden_layers = 9.0
True
Predicted
0 13810 0 0 0 0 0 0 0 0
0 15760 0 0 0 0 0 0 0 0
0 13980 0 0 0 0 0 0 0 0
0 14280 0 0 0 0 0 0 0 0
0 13650 0 0 0 0 0 0 0 0
0 12620 0 0 0 0 0 0 0 0
0 13750 0 0 0 0 0 0 0 0
0 14580 0 0 0 0 0 0 0 0
0 13650 0 0 0 0 0 0 0 0
0 13920 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

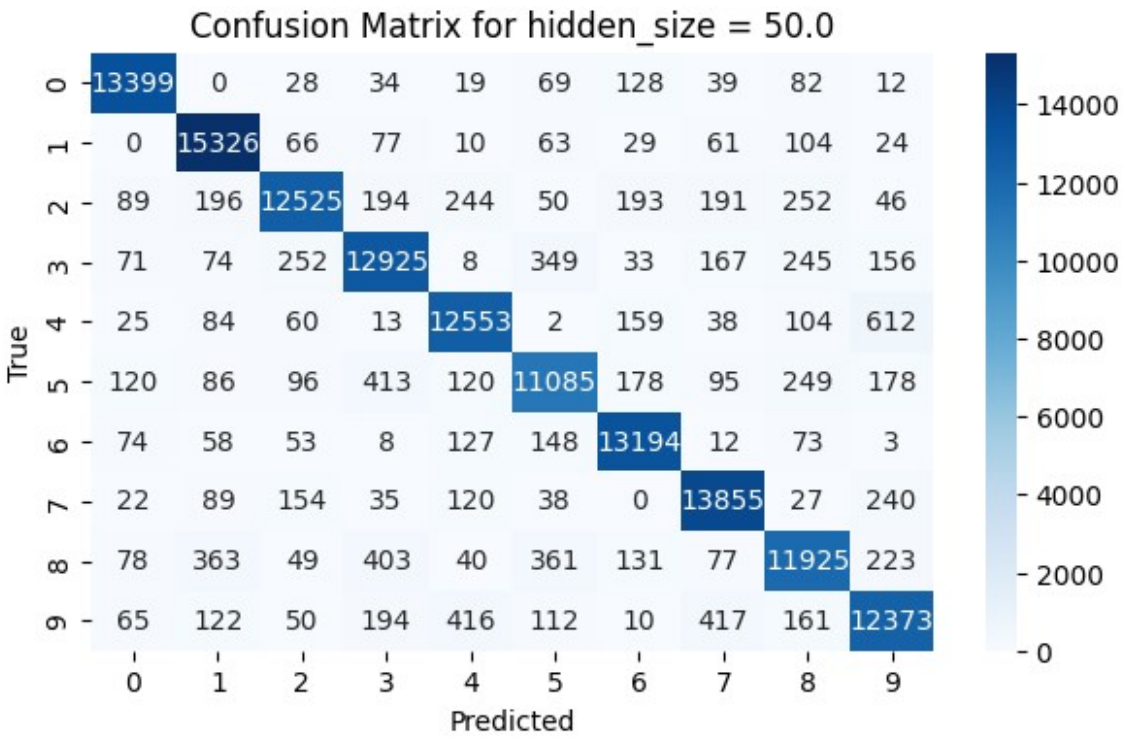

Confusion Matrix for hidden_layers = 11.0
True
Predicted
0 13810 0 0 0 0 0 0 0 0
0 15760 0 0 0 0 0 0 0 0
0 13980 0 0 0 0 0 0 0 0
0 14280 0 0 0 0 0 0 0 0
0 13650 0 0 0 0 0 0 0 0
0 12620 0 0 0 0 0 0 0 0
0 13750 0 0 0 0 0 0 0 0
0 14580 0 0 0 0 0 0 0 0
0 13650 0 0 0 0 0 0 0 0
0 13920 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

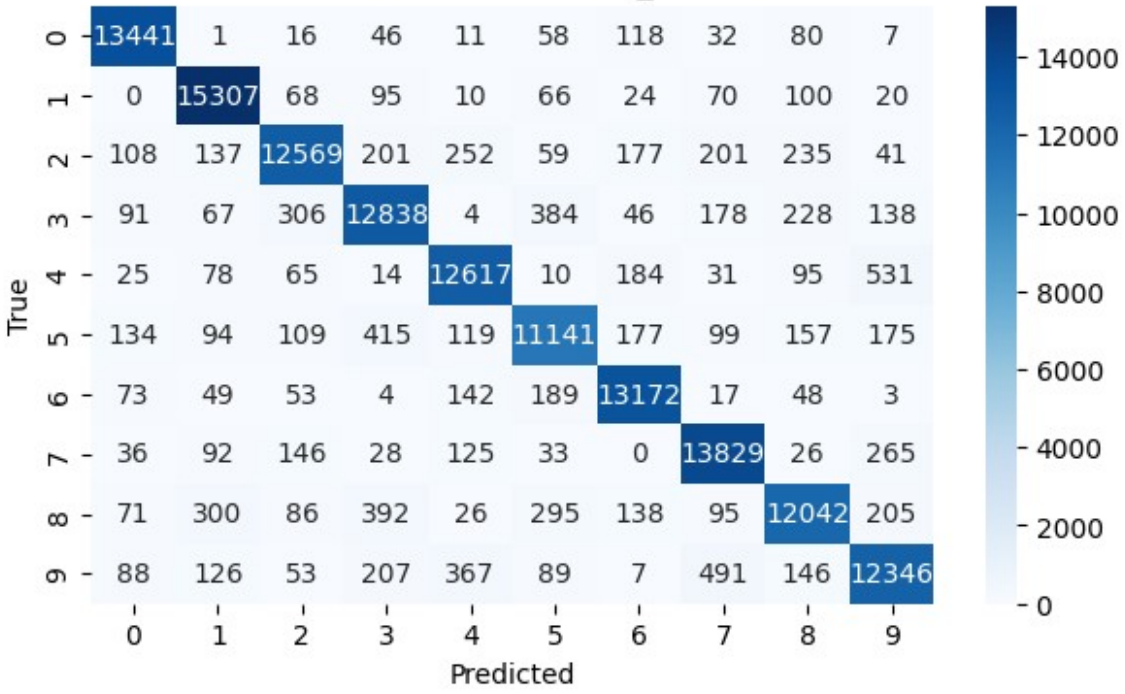

Confusion Matrix for hidden_size = 10.0
True
Predicted
0 13366 0 65 31 9 123 98 26 86 6
1 0 15227 90 111 6 61 13 66 169 17
2 99 143 12583 141 219 73 234 137 310 41
3 60 48 460 12615 2 470 44 141 365 75
4 27 62 84 30 12308 20 206 22 138 753
5 182 88 200 557 195 10605 205 78 341 169
6 104 47 172 0 109 209 13053 0 55 1
7 62 99 254 40 129 79 0 13386 41 490
8 80 325 213 510 41 371 137 62 11768 143
9 132 110 54 256 686 92 14 458 165 11953
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

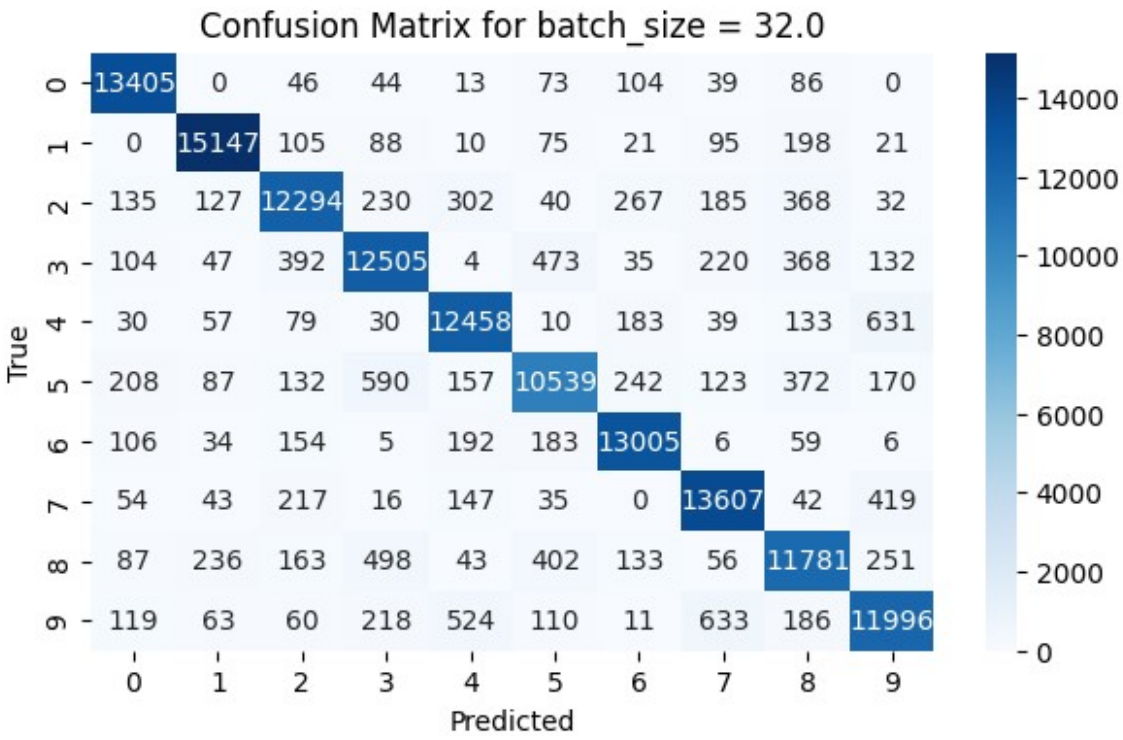

Confusion Matrix for hidden_size = 20.0
True
Predicted
0 13407 0 34 44 17 95 115 8 88 2
1 0 15276 74 89 11 71 23 70 128 18
2 104 194 12417 239 236 69 252 150 280 39
3 64 64 332 12782 12 434 31 172 257 132
4 29 78 47 23 12613 8 195 34 119 504
5 131 109 93 429 134 11068 202 73 182 199
6 72 60 92 8 167 176 13120 5 47 3
7 30 95 177 43 158 47 0 13519 17 494
8 73 351 82 408 40 381 141 70 11901 203
9 67 133 65 223 487 117 3 428 134 12263
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

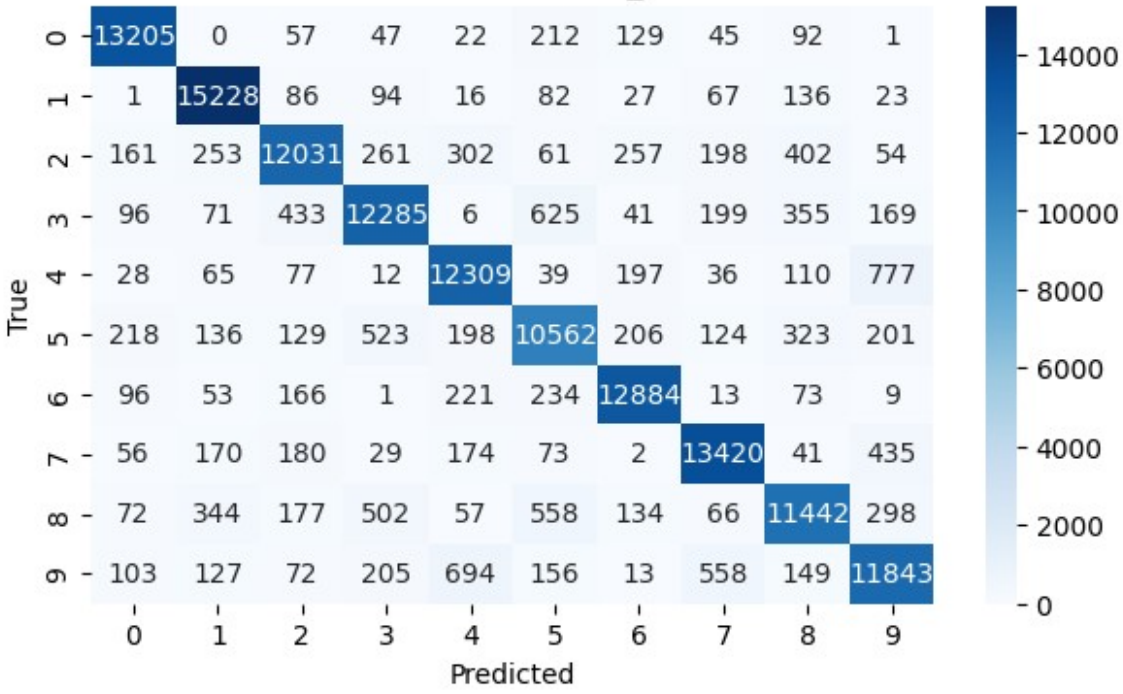

Confusion Matrix for hidden_size = 30.0
True
Predicted
0 13436 1 27 50 20 72 112 16 73 3
1 0 15332 75 78 10 78 20 59 85 23
2 119 170 12516 183 263 64 209 167 253 36
3 74 74 386 12797 6 422 26 152 212 131
4 33 65 52 18 12618 8 195 22 91 548
5 155 89 119 440 127 11047 187 86 206 164
6 80 50 89 1 158 170 13157 7 37 1
7 52 109 210 43 160 38 0 13570 19 379
8 77 320 96 432 45 387 134 50 11865 244
9 82 123 52 195 491 113 7 352 179 12326
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

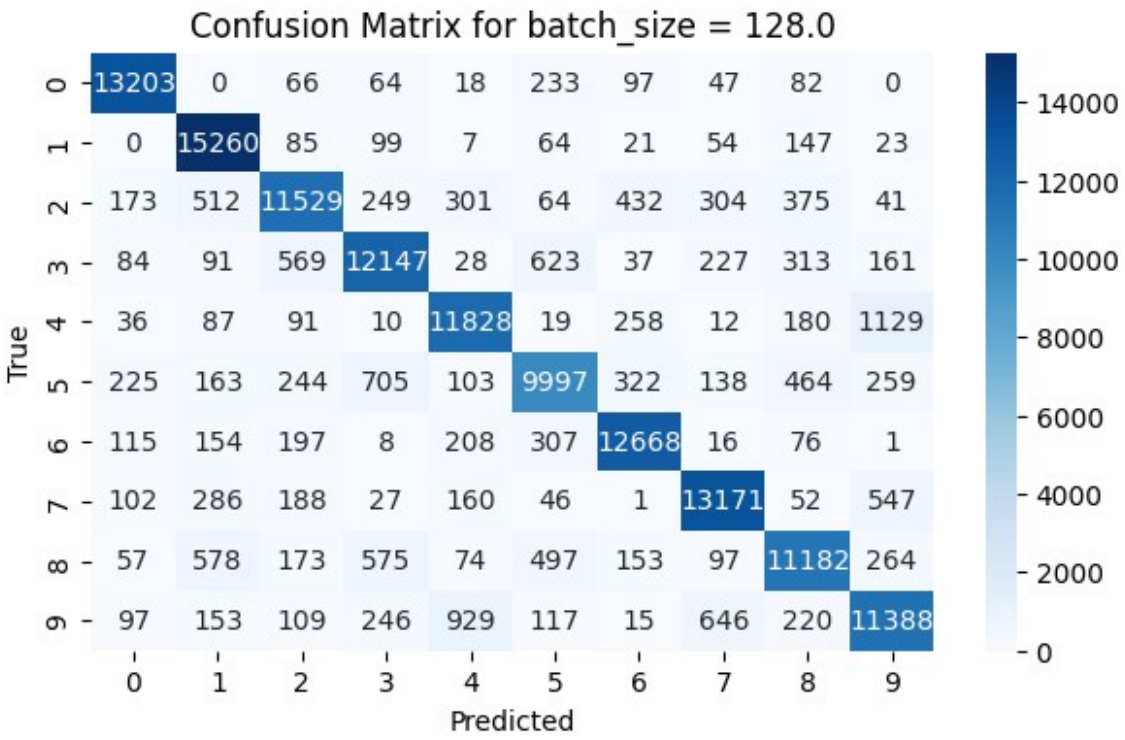

Confusion Matrix for hidden_size = 40.0
True
Predicted
0 13370 0 39 37 16 91 133 22 94 8
1 0 15296 67 104 10 69 20 53 120 21
2 91 177 12555 194 223 44 230 174 252 40
3 67 64 291 12846 3 423 37 147 274 128
4 29 85 60 29 12616 8 167 40 119 497
5 125 107 88 428 121 11070 189 87 244 161
6 59 41 73 5 119 173 13219 5 54 2
7 36 109 199 46 135 43 0 13668 25 319
8 70 324 69 393 32 310 133 68 12052 199
9 62 131 63 230 461 122 12 446 179 12214
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

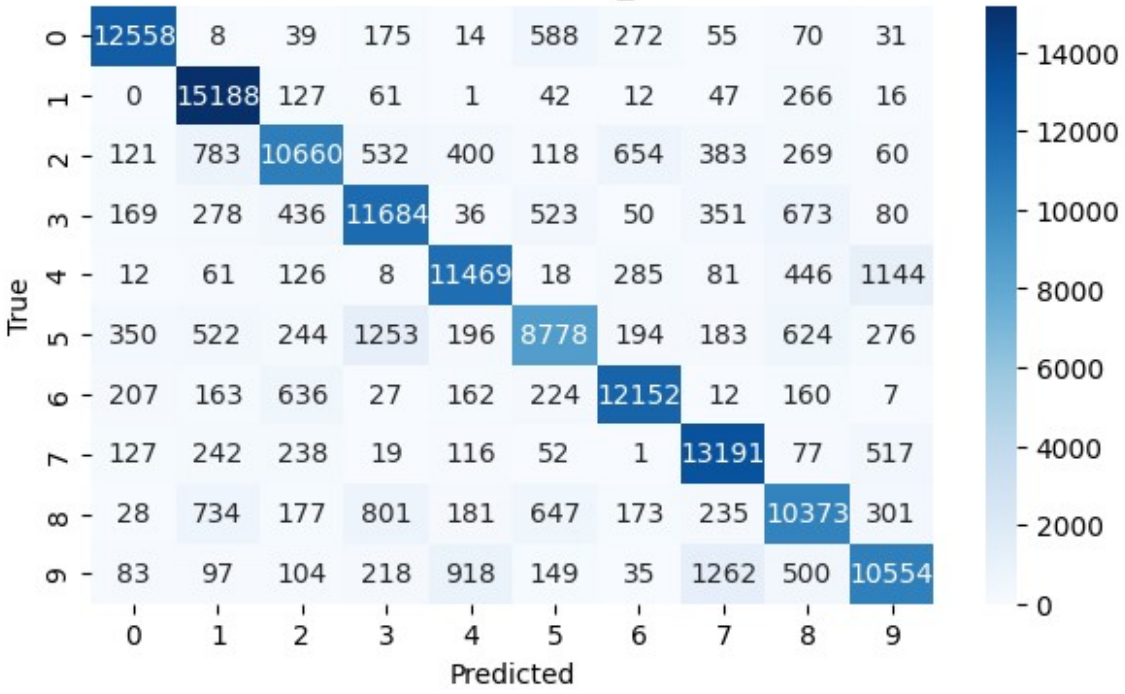

Confusion Matrix for hidden_size = 50.0
True
Predicted
0 13399 0 28 34 19 69 128 39 82 12
1 0 15326 66 77 10 63 29 61 104 24
2 89 196 12525 194 244 50 193 191 252 46
3 71 74 252 12925 8 349 33 167 245 156
4 25 84 60 13 12553 2 159 38 104 612
5 120 86 96 413 120 11085 178 95 249 178
6 74 58 53 8 127 148 13194 12 73 3
7 22 89 154 35 120 38 0 13855 27 240
8 78 363 49 403 40 361 131 77 11925 223
9 65 122 50 194 416 112 10 417 161 12373
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

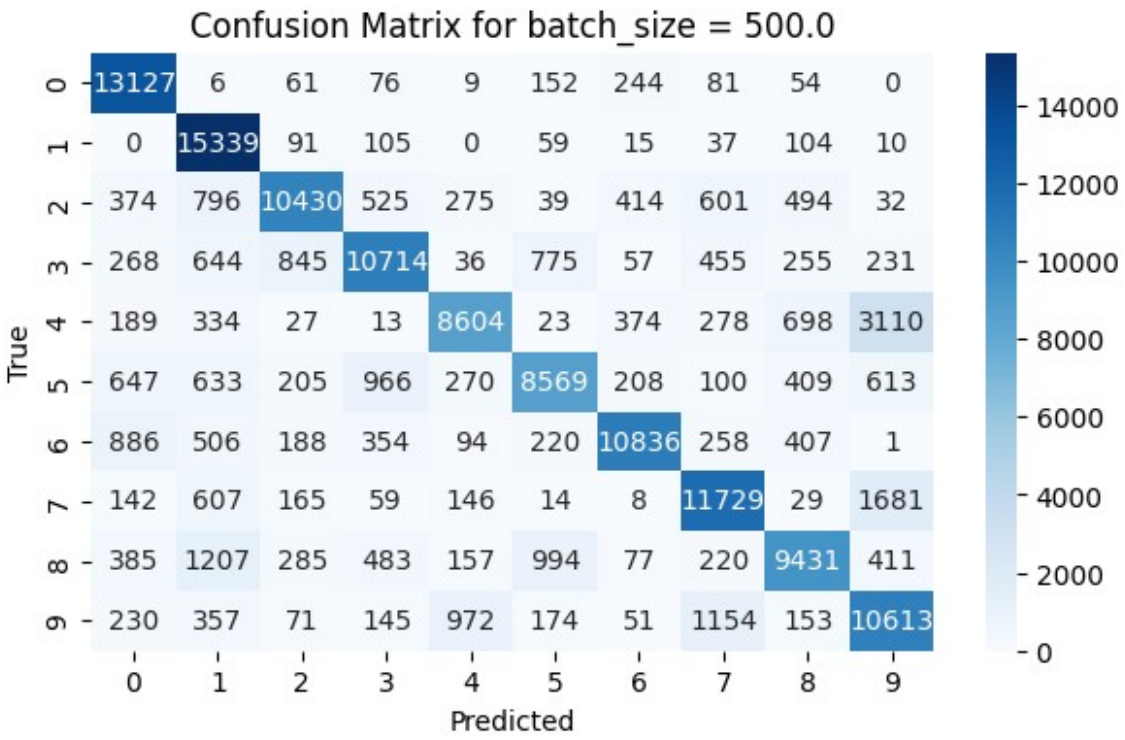

Confusion Matrix for hidden_size = 60.0
True
Predicted
0 13441 1 16 46 11 58 118 32 80 7
1 0 15307 68 95 10 66 24 70 100 20
2 108 137 12569 201 252 59 177 201 235 41
3 91 67 306 12838 4 384 46 178 228 138
4 25 78 65 14 12617 10 184 31 95 531
5 134 94 109 415 119 11141 177 99 157 175
6 73 49 53 4 142 189 13172 17 48 3
7 36 92 146 28 125 33 0 13829 26 265
8 71 300 86 392 26 295 138 95 12042 205
9 88 126 53 207 367 89 7 491 146 12346
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

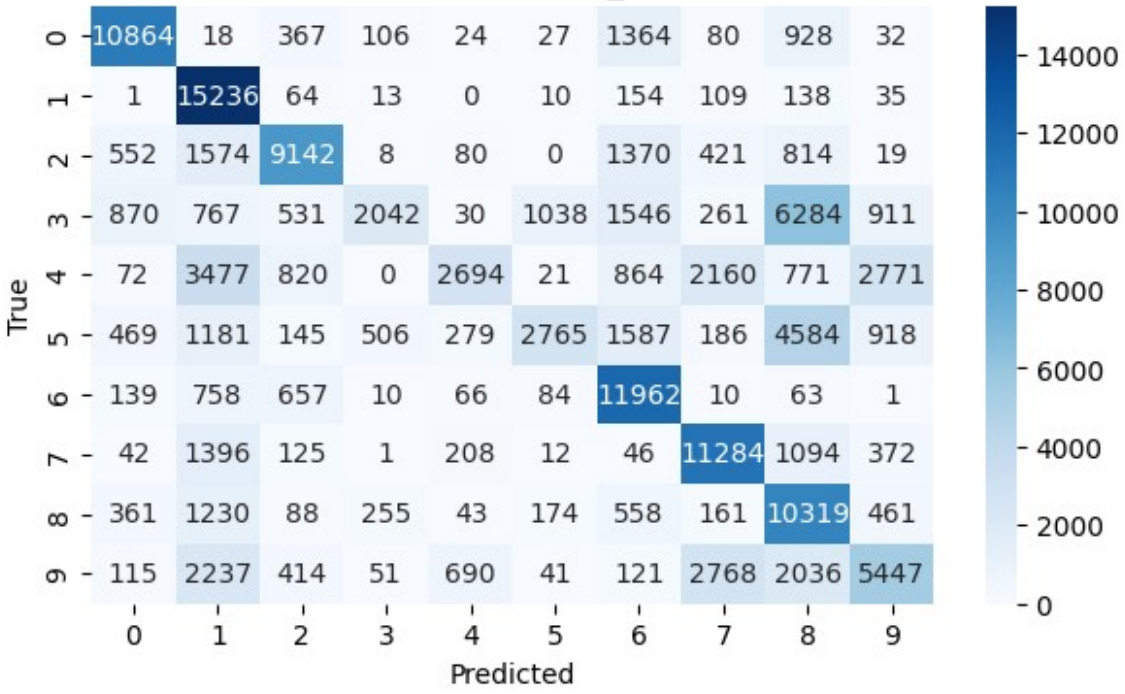

Confusion Matrix for batch_size = 32.0
True
Predicted
0 13405 0 46 44 13 73 104 39 86 0
1 0 15147 105 88 10 75 21 95 198 21
2 135 127 12294 230 302 40 267 185 368 32
3 104 47 392 12505 4 473 35 220 368 132
4 30 57 79 30 12458 10 183 39 133 631
5 208 87 132 590 157 10539 242 123 372 170
6 106 34 154 5 192 183 13005 6 59 6
7 54 43 217 16 147 35 0 13607 42 419
8 87 236 163 498 43 402 133 56 11781 251
9 119 63 60 218 524 110 11 633 186 11996
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

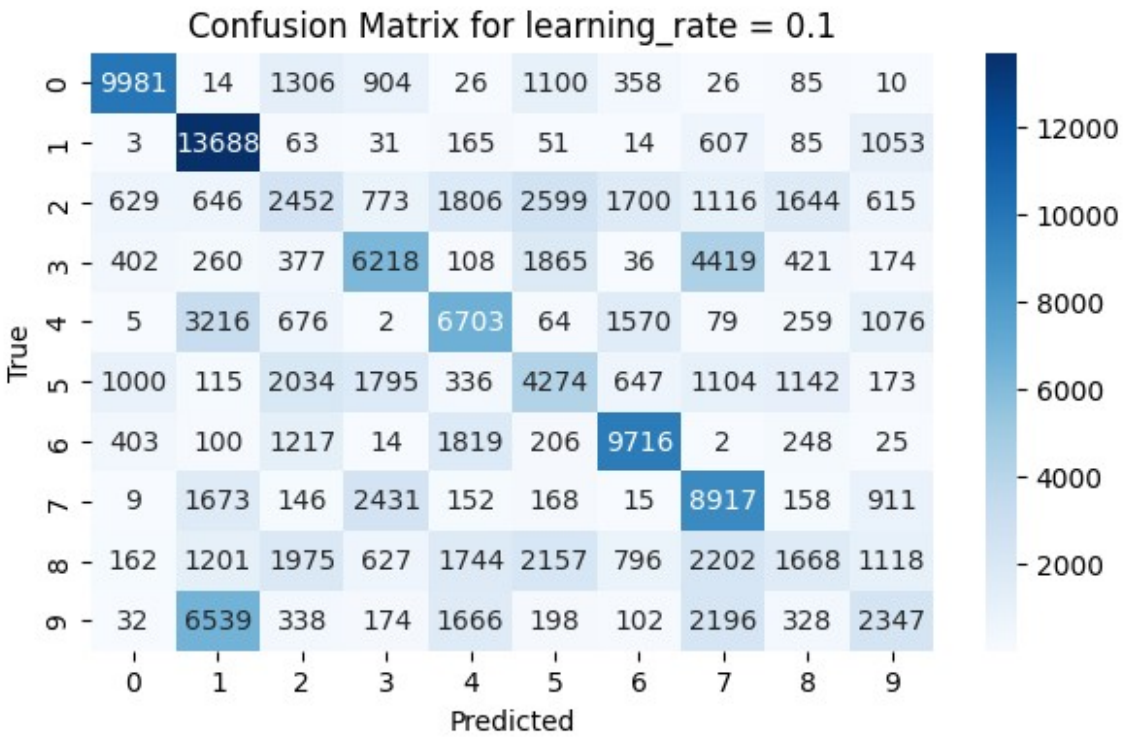

Confusion Matrix for batch_size = 64.0
True
Predicted
0 13205 0 57 47 22 212 129 45 92 1
1 1 15228 86 94 16 82 27 67 136 23
2 161 253 12031 261 302 61 257 198 402 54
3 96 71 433 12285 6 625 41 199 355 169
4 28 65 77 12 12309 39 197 36 110 777
5 218 136 129 523 198 10562 206 124 323 201
6 96 53 166 1 221 234 12884 13 73 9
7 56 170 180 29 174 73 2 13420 41 435
8 72 344 177 502 57 558 134 66 11442 298
9 103 127 72 205 694 156 13 558 149 11843
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

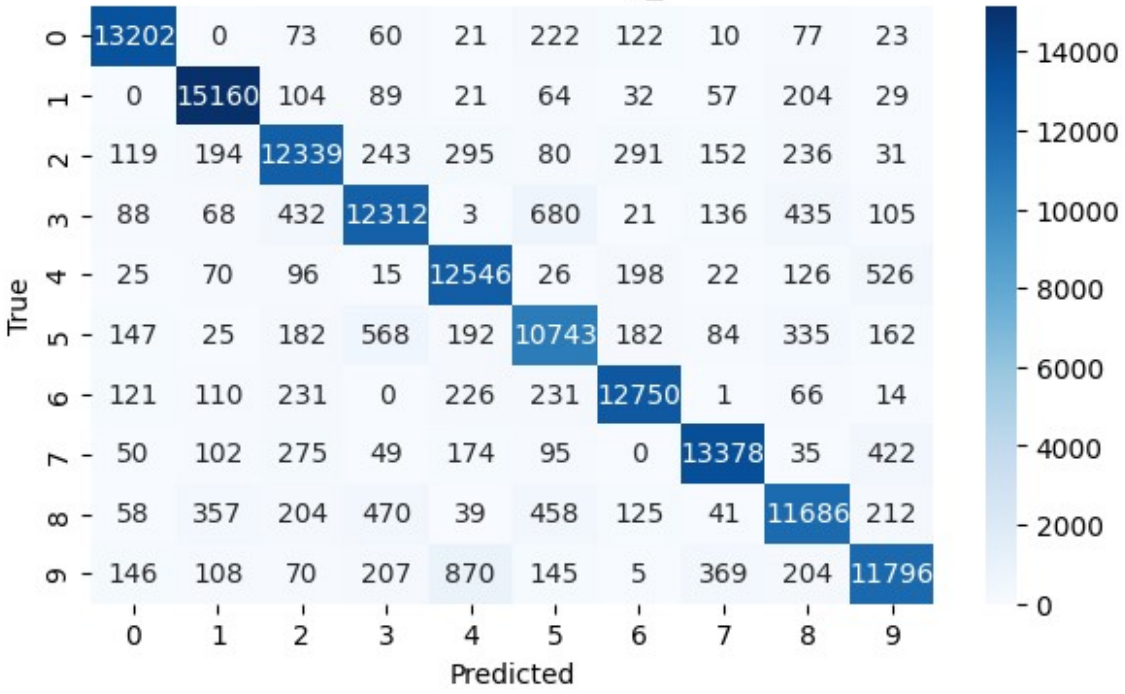

Confusion Matrix for batch_size = 128.0
True
Predicted
0 13203 0 66 64 18 233 97 47 82 0
1 0 15260 85 99 7 64 21 54 147 23
2 173 512 11529 249 301 64 432 304 375 41
3 84 91 569 12147 28 623 37 227 313 161
4 36 87 91 10 11828 19 258 12 180 1129
5 225 163 244 705 103 9997 322 138 464 259
6 115 154 197 8 208 307 12668 16 76 1
7 102 286 188 27 160 46 1 13171 52 547
8 57 578 173 575 74 497 153 97 11182 264
9 97 153 109 246 929 117 15 646 220 11388
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

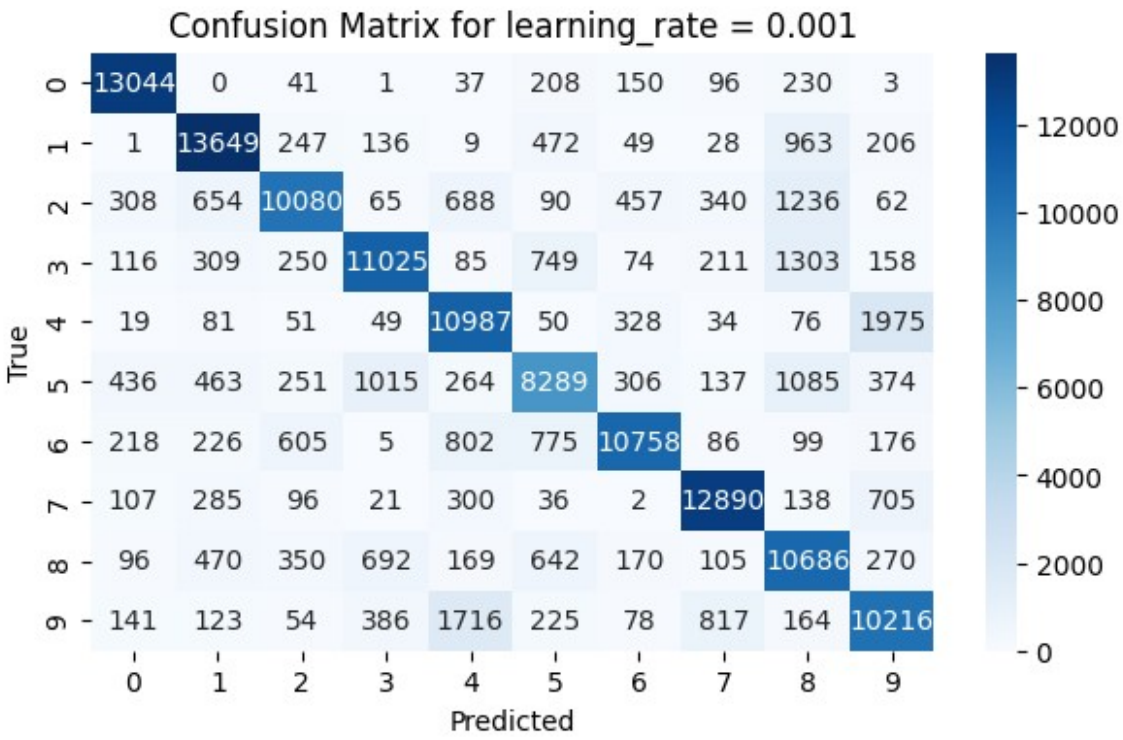

Confusion Matrix for batch_size = 256.0
True
Predicted
0 12558 8 39 175 14 588 272 55 70 31
1 0 15188 127 61 1 42 12 47 266 16
2 121 783 10660 532 400 118 654 383 269 60
3 169 278 436 11684 36 523 50 351 673 80
4 12 61 126 8 11469 18 285 81 446 1144
5 350 522 244 1253 196 8778 194 183 624 276
6 207 163 636 27 162 224 12152 12 160 7
7 127 242 238 19 116 52 1 13191 77 517
8 28 734 177 801 181 647 173 235 10373 301
9 83 97 104 218 918 149 35 1262 500 10554
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

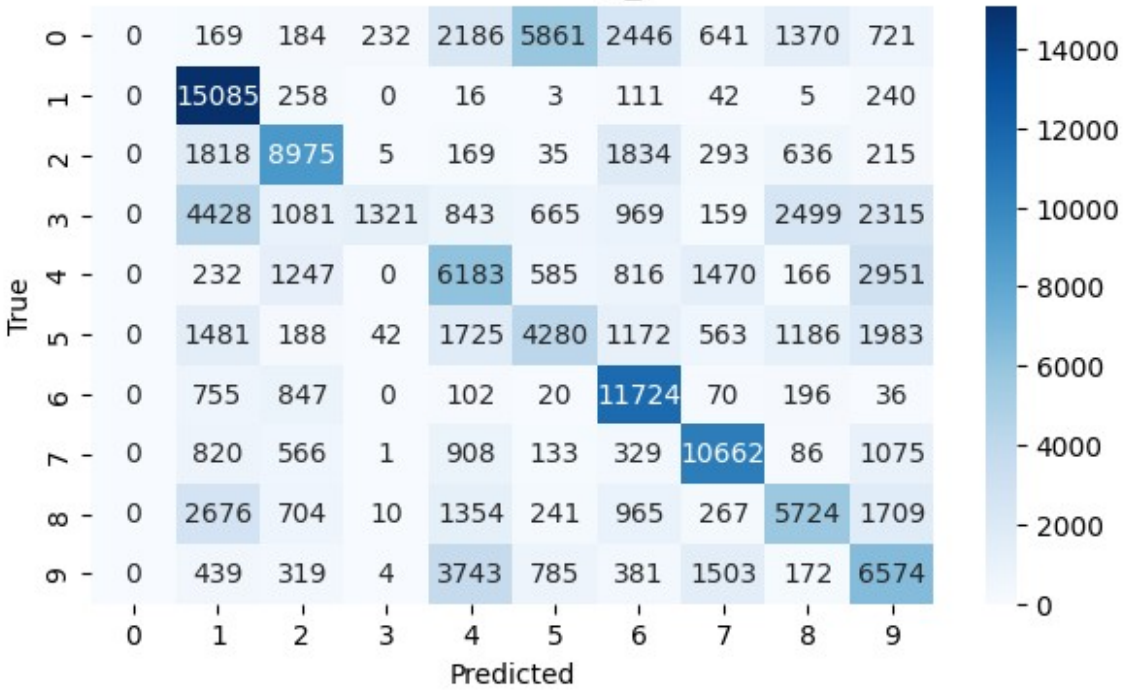

Confusion Matrix for batch_size = 500.0
True
Predicted
0 13127 6 61 76 9 152 244 81 54 0
1 0 15339 91 105 0 59 15 37 104 10
2 374 796 10430 525 275 39 414 601 494 32
3 268 644 845 10714 36 775 57 455 255 231
4 189 334 27 13 8604 23 374 278 698 3110
5 647 633 205 966 270 8569 208 100 409 613
6 886 506 188 354 94 220 10836 258 407 1
7 142 607 165 59 146 14 8 11729 29 1681
8 385 1207 285 483 157 994 77 220 9431 411
9 230 357 71 145 972 174 51 1154 153 10613
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

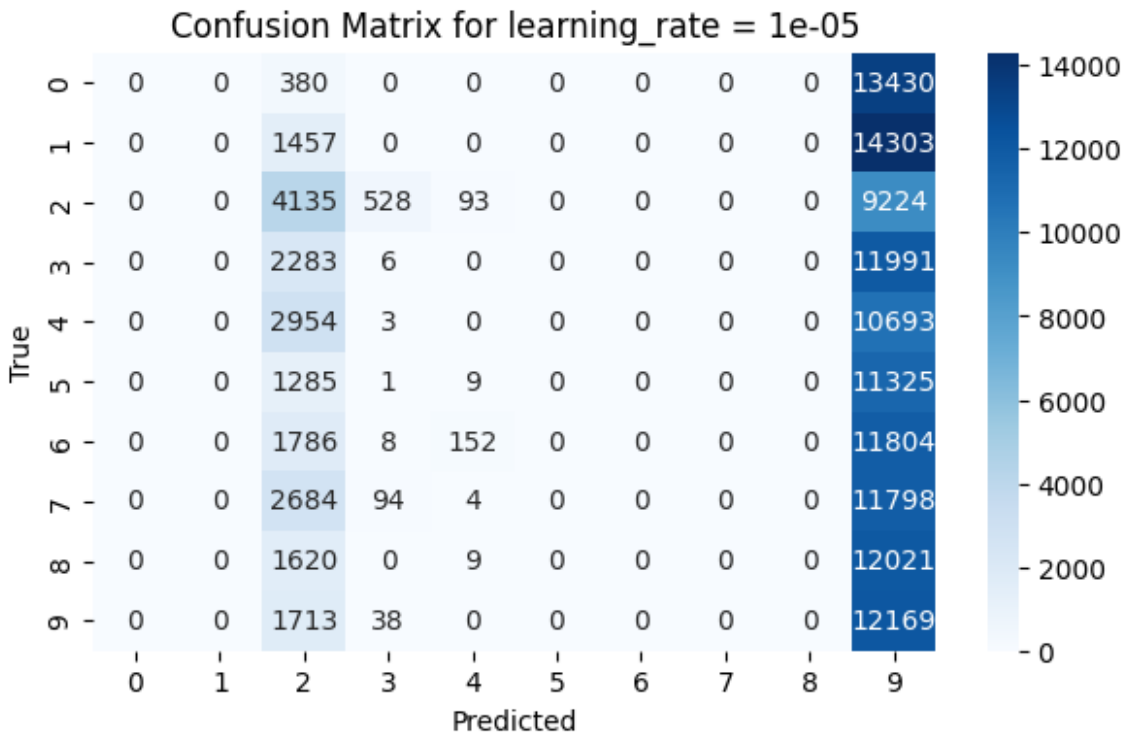

Confusion Matrix for batch_size = 1000.0
True
Predicted
0 10864 18 367 106 24 27 1364 80 928 32
1 1 15236 64 13 0 10 154 109 138 35
2 552 1574 9142 8 80 0 1370 421 814 19
3 870 767 531 2042 30 1038 1546 261 6284 911
4 72 3477 820 0 2694 21 864 2160 771 2771
5 469 1181 145 506 279 2765 1587 186 4584 918
6 139 758 657 10 66 84 11962 10 63 1
7 42 1396 125 1 208 12 46 11284 1094 372
8 361 1230 88 255 43 174 558 161 10319 461
9 115 2237 414 51 690 41 121 2768 2036 5447
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0

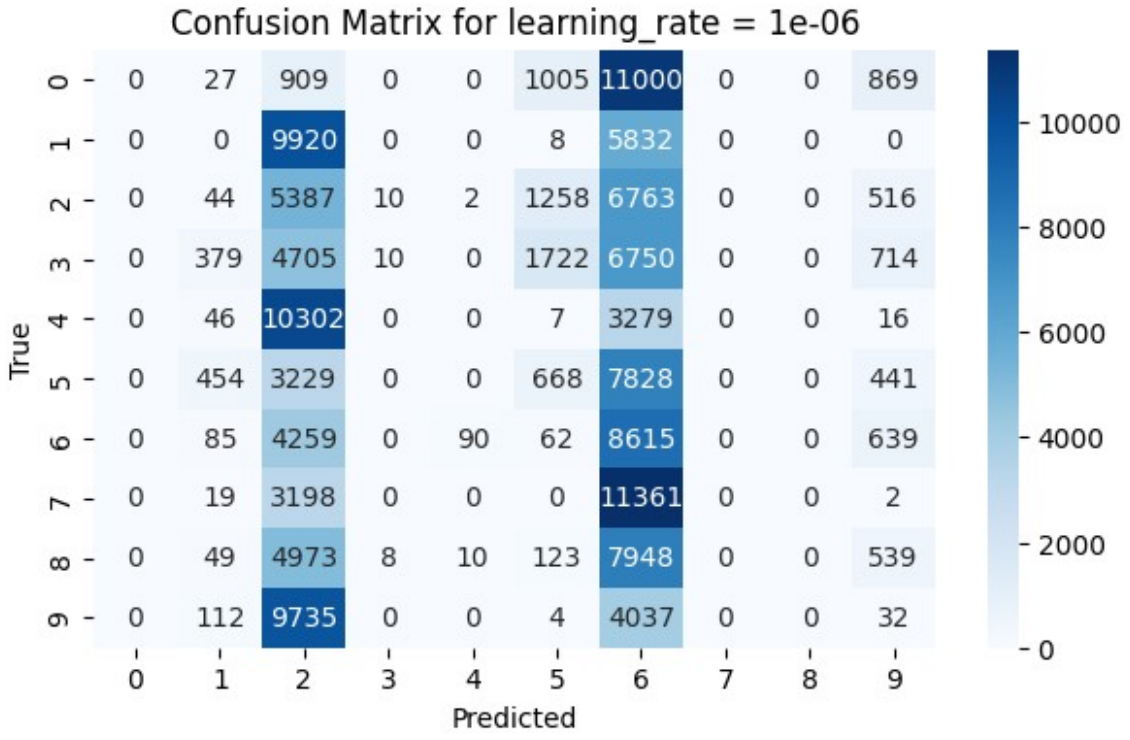

Confusion Matrix for learning_rate = 0.1
True
Predicted
0 9981 14 1306 904 26 1100 358 26 85 10
1 3 13688 63 31 165 51 14 607 85 1053
2 629 646 2452 773 1806 2599 1700 1116 1644 615
3 402 260 377 6218 108 1865 36 4419 421 174
4 5 3216 676 2 6703 64 1570 79 259 1076
5 1000 115 2034 1795 336 4274 647 1104 1142 173
6 403 100 1217 14 1819 206 9716 2 248 25
7 9 1673 146 2431 152 168 15 8917 158 911
8 162 1201 1975 627 1744 2157 796 2202 1668 1118
9 32 6539 338 174 1666 198 102 2196 328 2347
0 1 2 3 4 5 6 7 8 9
12000
10000
8000
6000
4000
2000


Confusion Matrix for learning_rate = 0.01
True
Predicted
0 13202 0 73 60 21 222 122 10 77 23
1 0 15160 104 89 21 64 32 57 204 29
2 119 194 12339 243 295 80 291 152 236 31
3 88 68 432 12312 3 680 21 136 435 105
4 25 70 96 15 12546 26 198 22 126 526
5 147 25 182 568 192 10743 182 84 335 162
6 121 110 231 0 226 231 12750 1 66 14
7 50 102 275 49 174 95 0 13378 35 422
8 58 357 204 470 39 458 125 41 11686 212
9 146 108 70 207 870 145 5 369 204 11796
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0


Confusion Matrix for learning_rate = 0.001
True
Predicted
0 13044 0 41 1 37 208 150 96 230 3
1 1 13649 247 136 9 472 49 28 963 206
2 308 654 10080 65 688 90 457 340 1236 62
3 116 309 250 11025 85 749 74 211 1303 158
4 19 81 51 49 10987 50 328 34 76 1975
5 436 463 251 1015 264 8289 306 137 1085 374
6 218 226 605 5 802 775 10758 86 99 176
7 107 285 96 21 300 36 2 12890 138 705
8 96 470 350 692 169 642 170 105 10686 270
9 141 123 54 386 1716 225 78 817 164 10216
0 1 2 3 4 5 6 7 8 9
12000
10000
8000
6000
4000
2000
0


Confusion Matrix for learning_rate = 0.0001
True
Predicted
0 0 169 184 232 2186 5861 2446 641 1370 721
1 0 15085 258 0 16 3 111 42 5 240
2 0 1818 8975 5 169 35 1834 293 636 215
3 0 4428 1081 1321 843 665 969 159 2499 2315
4 0 232 1247 0 6183 585 816 1470 166 2951
5 0 1481 188 42 1725 4280 1172 563 1186 1983
6 0 755 847 0 102 20 11724 70 196 36
7 0 820 566 1 908 133 329 10662 86 1075
8 0 2676 704 10 1354 241 965 267 5724 1709
9 0 439 319 4 3743 785 381 1503 172 6574
0 1 2 3 4 5 6 7 8 9
14000
12000
10000
8000
6000
4000
2000
0




now we inspect the pool for the combination of parameters with the best accuracy, retrain the model using these parameters and finally verify on the test set.

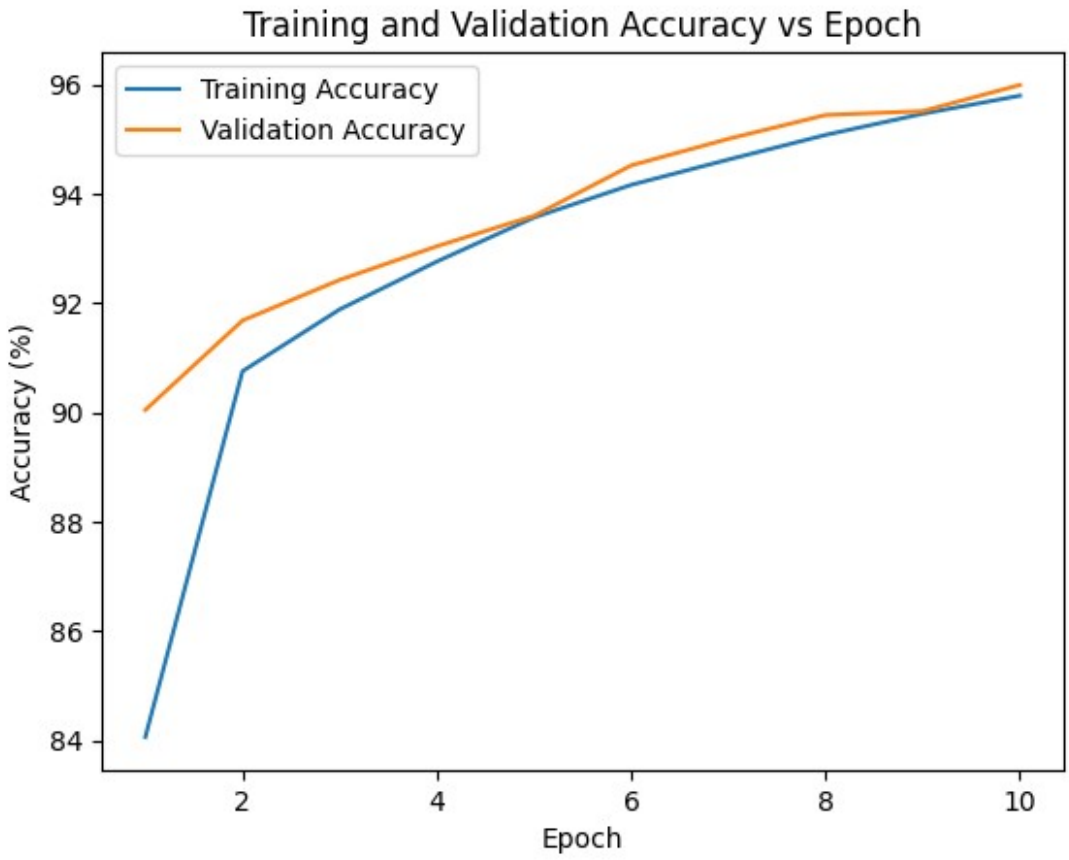
```python
# 1. Select the best parameters based on validation accuracy
def select_best_params(results):
    best_result = max(results, key=lambda x: x["val_accuracy"])  # Find the best validation accuracy
    best_params = best_result["param"], best_result["value"]  # Extract the parameter name and value
    print(f"Best Parameter: {best_params[0]} = {best_params[1]}")
    return best_params, best_result

best_params, best_result = select_best_params(results)
print(best_params)

# 2. Train the final model using the best parameters on combined train + validation data
combined_train_data = torch.utils.data.ConcatDataset([train_data, val_data])
combined_train_loader = DataLoader(combined_train_data, batch_size=default_params["batch_size"], shuffle=True)

# Reinitialize the model with the best hyperparameters
if best_params[0] == "hidden_layers":
    default_params["hidden_layers"] = best_params[1]
elif best_params[0] == "hidden_size":
    default_params["hidden_size"] = best_params[1]  # Use the value, not the name
elif best_params[0] == "batch_size":
    default_params["batch_size"] = best_params[1]  # Correct this to use the value
elif best_params[0] == "learning_rate":
    default_params["learning_rate"] = best_params[1]  # Correct this to use the value

# Create and train the model
final_model = Neural_Network_1_hidden_layer(
    input_size=28*28,
    hidden_size=default_params["hidden_size"],
    num_classes=10
).to(device)

final_optimizer = optim.SGD(final_model.parameters(), lr=default_params["learning_rate"])
final_criterion = nn.CrossEntropyLoss()

# Train the model on combined data
_, _, train_accs,val_accs,_ ,_ = train_model(
    final_model, combined_train_loader, val_loader, final_criterion, final_optimizer, num_epochs=10
)
```

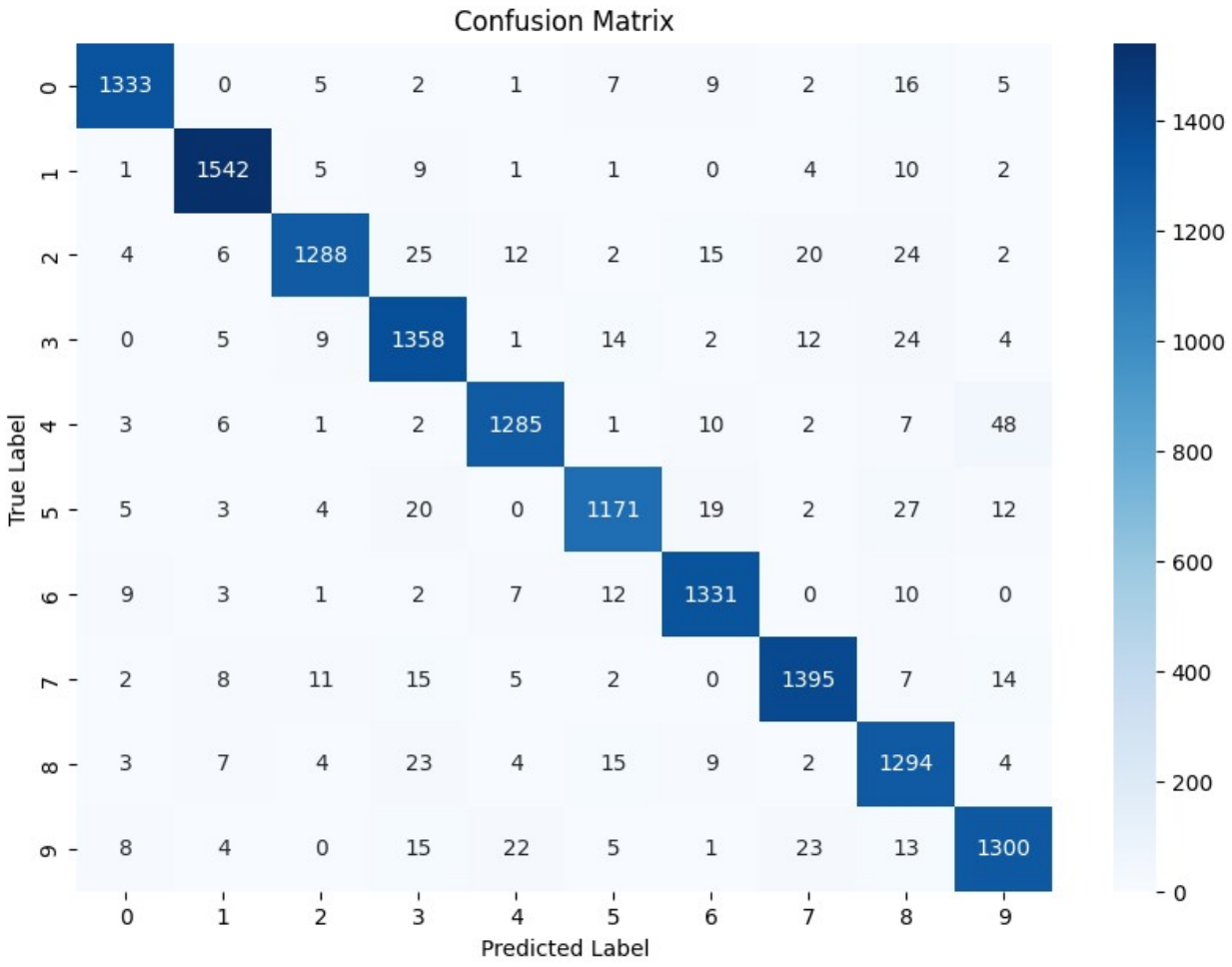
```
Best Parameter: hidden_size = 50
('hidden_size', 50)
Epoch [1/10], Train Loss: 0.6035, Train Accuracy: 84.06%, Validation
Loss: 0.3581, Validation Accuracy: 90.04%
Epoch [2/10], Train Loss: 0.3235, Train Accuracy: 90.75%, Validation
Loss: 0.2958, Validation Accuracy: 91.68%
Epoch [3/10], Train Loss: 0.2814, Train Accuracy: 91.88%, Validation
Loss: 0.2654, Validation Accuracy: 92.42%
Epoch [4/10], Train Loss: 0.2513, Train Accuracy: 92.76%, Validation
Loss: 0.2376, Validation Accuracy: 93.04%
Epoch [5/10], Train Loss: 0.2249, Train Accuracy: 93.56%, Validation
Loss: 0.2168, Validation Accuracy: 93.59%
Epoch [6/10], Train Loss: 0.2036, Train Accuracy: 94.16%, Validation
Loss: 0.1931, Validation Accuracy: 94.51%
Epoch [7/10], Train Loss: 0.1862, Train Accuracy: 94.62%, Validation
Loss: 0.1788, Validation Accuracy: 95.00%
Epoch [8/10], Train Loss: 0.1713, Train Accuracy: 95.07%, Validation
Loss: 0.1622, Validation Accuracy: 95.44%
Epoch [9/10], Train Loss: 0.1590, Train Accuracy: 95.46%, Validation
Loss: 0.1598, Validation Accuracy: 95.51%
Epoch [10/10], Train Loss: 0.1488, Train Accuracy: 95.79%, Validation
Loss: 0.1437, Validation Accuracy: 95.99%
```

```python
def evaluate_on_test_set(model, test_loader):
    model.eval()  # Set model to evaluation mode
    correct_test = 0
    total_test = 0
    y_true = []
    y_pred = []

    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total_test += labels.size(0)
            correct_test += (predicted == labels).sum().item()

            y_true.extend(labels.cpu().numpy())
            y_pred.extend(predicted.cpu().numpy())

    test_acc = 100 * correct_test / total_test
    print(f'Test Accuracy: {test_acc:.2f}%')

    return y_true, y_pred, test_acc

# Evaluate on the test set
y_true, y_pred, test_acc = evaluate_on_test_set(final_model,
test_loader)
```

```
Test Accuracy: 94.98%
```

```python
# Plot the accuracies for training and validation
epochs = list(range(1, 11))
plt.plot(epochs,train_accs, label="Training Accuracy")
plt.plot(epochs,val_accs, label="Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.title("Training and Validation Accuracy vs Epoch")
plt.legend()
plt.show()

# Plot confusion matrix
plot_confusion_matrix(y_true, y_pred)

# Optionally, print the test accuracy
print(f"Test Accuracy: {test_acc:.2f}%")
```






```
Test Accuracy: 94.98%
```

# Bonus (CNN)

we will adjust the previous code and repeat the process for convoltional Neural Networks and study their features closely.

```
# Load the MNIST dataset
train_dataset = datasets.MNIST(root='./data', train=True) #this
structure contains data and labels
test_dataset = datasets.MNIST(root='./data', train=False)

#combine the datasets
combined_dataset=ConcatDataset([train_dataset,test_dataset])

#extract labels
train_labels=train_dataset.targets
test_labels=test_dataset.targets
```

```python
# Combine labels from both train and test datasets
all_labels = torch.cat((train_labels, test_labels), dim=0)

#split into 60-40
train_idx, temp_idx = train_test_split(
    range(len(combined_dataset)), test_size=0.4, stratify=all_labels,
random_state=42) # stratify 3lshan not to get skewed data


#split again 20-20
# Split temp (40%) into validation (20%) and test (20%)
val_idx, test_idx = train_test_split(
    temp_idx, test_size=0.5, stratify=all_labels[temp_idx],
random_state=42) # returns indices


# Create Subset objects for train, validation, and test sets
train_data = Subset(combined_dataset, train_idx)
val_data = Subset(combined_dataset, val_idx)
test_data = Subset(combined_dataset, test_idx)


##recreate dataset w inheirt combatible structure with data loader
# Create DataLoader objects
batch_size = 32
train_loader = DataLoader(train_data, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_data, batch_size=batch_size,
shuffle=False)
test_loader = DataLoader(test_data, batch_size=batch_size,
shuffle=False)

# Print split sizes
print(f"Training set size: {len(train_data)}")
print(f"Validation set size: {len(val_data)}")
print(f"Test set size: {len(test_data)}")
print("Data preparation complete!")
```

```
Training set size: 42000
Validation set size: 14000
Test set size: 14000
Data preparation complete!
```

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
```

```
# Default hyperparameters
default_params_cnn = {
    "hidden_layers": 5,  # Number of convolutional layers
    "hidden_size": 10,   # Number of filters per layer
    "batch_size": 32,    # Batch size
    "learning_rate": 0.01,
    "dropout_prob": 0.5  # Dropout probability
}

# Hyperparameter ranges for tuning
param_ranges_cnn = {
    "hidden_layers": [5, 7, 9],
    "hidden_size": [30, 50, 70],
    "batch_size": [128, 256, 512],
    "learning_rate": [0.01, 0.001, 0.0001],
    "dropout_prob": [0.25, 0.5, 0.75]
}

# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Data preparation
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_data = datasets.MNIST(root="data", train=True, download=True,
transform=transform)
val_data = datasets.MNIST(root="data", train=False, download=True,
transform=transform)


class ConvNeuralNetwork(nn.Module):
    def __init__(self, input_channels, hidden_layers, hidden_size,
num_classes, dropout_prob):
        super(ConvNeuralNetwork, self).__init__()

        layers = []

        # First convolutional layer
        layers.append(nn.Conv2d(input_channels, hidden_size,
kernel_size=3, padding=1))  # 3x3 kernel
        layers.append(nn.ReLU())
        layers.append(nn.LayerNorm([hidden_size, 28, 28]))
        layers.append(nn.Dropout(dropout_prob))

        # Additional convolutional layers
        for i in range(1, hidden_layers):
            layers.append(nn.Conv2d(hidden_size, hidden_size,
```

```
kernel_size=3, padding=1))
            layers.append(nn.ReLU())
            layers.append(nn.LayerNorm([hidden_size, 28, 28]))
            layers.append(nn.Dropout(dropout_prob))

        # Pooling layer
        layers.append(nn.MaxPool2d(kernel_size=2, stride=2))  #
Reduces dimensions to 14x14

        # Flatten layer for the fully connected layer
        self.flatten = nn.Flatten()

        # Fully connected layer
        fc_input_size = hidden_size * 14 * 14  # After pooling
        self.fc = nn.Linear(fc_input_size, num_classes)

        self.network = nn.Sequential(*layers)

    def forward(self, x):
        x = self.network(x)
        x = self.flatten(x)
        x = self.fc(x)
        return x


def train_model(model, train_loader, val_loader, criterion, optimizer,
num_epochs=10):
    train_losses, val_losses = [], []
    train_accuracies, val_accuracies = [], []

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        correct, total, train_loss = 0, 0, 0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

        train_losses.append(train_loss / len(train_loader))
        train_accuracies.append(correct / total)
```

```
        # Validation phase
        model.eval()
        correct, total, val_loss = 0, 0, 0
        y_true, y_pred = [], []
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                correct += (predicted == labels).sum().item()
                total += labels.size(0)
                y_true.extend(labels.cpu().numpy())
                y_pred.extend(predicted.cpu().numpy())

        val_losses.append(val_loss / len(val_loader))
        val_accuracies.append(correct / total)

        print(f"Epoch {epoch + 1}/{num_epochs} - "
              f"Train Loss: {train_losses[-1]:.4f}, Train Accuracy:
{train_accuracies[-1]:.4f} - "
              f"Val Loss: {val_losses[-1]:.4f}, Val Accuracy:
{val_accuracies[-1]:.4f}")

    return train_losses, val_losses, train_accuracies, val_accuracies,
y_true, y_pred


def tune_hyperparameters(default_params, param_ranges, num_epochs=10):
    results = []

    # Iterate over each parameter
    for param, values in param_ranges.items():
        for value in values:
            # Update the parameter value
            params = default_params.copy()
            params[param] = value

            # Update DataLoader if batch_size changes
            batch_size = params["batch_size"]
            train_loader = DataLoader(train_data,
batch_size=batch_size, shuffle=True)
            val_loader = DataLoader(val_data, batch_size=batch_size,
shuffle=False)

            # Build the model
            model = ConvNeuralNetwork(
                input_channels=1,  # MNIST has 1 channel
```

```
                hidden_layers=params["hidden_layers"],
                hidden_size=params["hidden_size"],
                num_classes=10,
                dropout_prob=params["dropout_prob"]
            ).to(device)

            # Define optimizer with new learning rate
            optimizer = optim.Adam(model.parameters(),
lr=params["learning_rate"])
            criterion = nn.CrossEntropyLoss()

            # Train the model
            train_losses, val_losses, train_accuracies,
val_accuracies, y_true, y_pred = train_model(
                model, train_loader, val_loader, criterion, optimizer,
num_epochs=num_epochs
            )

            # Record results
            results.append({
                "param": param,
                "value": value,
                "train_accuracy": train_accuracies[-1],
                "val_accuracy": val_accuracies[-1],
                "train_loss": train_losses[-1],
                "val_loss": val_losses[-1],
                "y_true": y_true,
                "y_pred": y_pred
            })

            print(f"Completed training for {param} = {value}")

    return results


# Example usage
results = tune_hyperparameters(default_params_cnn, param_ranges_cnn,
num_epochs=5)
```

```
Epoch 1/5 - Train Loss: 0.6286, Train Accuracy: 0.8488 - Val Loss:
0.3645, Val Accuracy: 0.8917
Epoch 2/5 - Train Loss: 0.2694, Train Accuracy: 0.9290 - Val Loss:
0.2912, Val Accuracy: 0.9140
Epoch 3/5 - Train Loss: 0.2581, Train Accuracy: 0.9356 - Val Loss:
1.0703, Val Accuracy: 0.7722
Epoch 4/5 - Train Loss: 0.2719, Train Accuracy: 0.9366 - Val Loss:
0.6632, Val Accuracy: 0.8681
Epoch 5/5 - Train Loss: 0.2767, Train Accuracy: 0.9394 - Val Loss:
0.6576, Val Accuracy: 0.8632
Completed training for hidden_layers = 5
```

```
Epoch 1/5 - Train Loss: 0.5572, Train Accuracy: 0.8637 - Val Loss:
1.8260, Val Accuracy: 0.4912
Epoch 2/5 - Train Loss: 0.2922, Train Accuracy: 0.9168 - Val Loss:
1.0553, Val Accuracy: 0.7176
Epoch 3/5 - Train Loss: 0.2916, Train Accuracy: 0.9214 - Val Loss:
0.4345, Val Accuracy: 0.8793
Epoch 4/5 - Train Loss: 0.2892, Train Accuracy: 0.9253 - Val Loss:
0.3959, Val Accuracy: 0.9011
Epoch 5/5 - Train Loss: 0.2826, Train Accuracy: 0.9292 - Val Loss:
0.2874, Val Accuracy: 0.9317
Completed training for hidden_layers = 7
Epoch 1/5 - Train Loss: 0.6660, Train Accuracy: 0.8247 - Val Loss:
0.4822, Val Accuracy: 0.8478
Epoch 2/5 - Train Loss: 0.3209, Train Accuracy: 0.9074 - Val Loss:
0.3325, Val Accuracy: 0.9049
Epoch 3/5 - Train Loss: 0.2915, Train Accuracy: 0.9190 - Val Loss:
0.3287, Val Accuracy: 0.9201
Epoch 4/5 - Train Loss: 0.2958, Train Accuracy: 0.9228 - Val Loss:
0.3433, Val Accuracy: 0.8984
Epoch 5/5 - Train Loss: 0.2824, Train Accuracy: 0.9289 - Val Loss:
0.3132, Val Accuracy: 0.9136
Completed training for hidden_layers = 9
Epoch 1/5 - Train Loss: 0.7969, Train Accuracy: 0.8651 - Val Loss:
4.9081, Val Accuracy: 0.2901
Epoch 2/5 - Train Loss: 0.3784, Train Accuracy: 0.9261 - Val Loss:
3.0263, Val Accuracy: 0.7134
Epoch 3/5 - Train Loss: 0.4684, Train Accuracy: 0.9377 - Val Loss:
0.8222, Val Accuracy: 0.8899
Epoch 4/5 - Train Loss: 0.5289, Train Accuracy: 0.9456 - Val Loss:
0.6941, Val Accuracy: 0.9365
Epoch 5/5 - Train Loss: 0.6020, Train Accuracy: 0.9523 - Val Loss:
2.3588, Val Accuracy: 0.8538
Completed training for hidden_size = 30
Epoch 1/5 - Train Loss: 1.3207, Train Accuracy: 0.8887 - Val Loss:
2.1529, Val Accuracy: 0.7586
Epoch 2/5 - Train Loss: 0.6117, Train Accuracy: 0.9394 - Val Loss:
0.9601, Val Accuracy: 0.9368
Epoch 3/5 - Train Loss: 0.7775, Train Accuracy: 0.9505 - Val Loss:
1.7333, Val Accuracy: 0.8878
Epoch 4/5 - Train Loss: 0.8765, Train Accuracy: 0.9589 - Val Loss:
2.3877, Val Accuracy: 0.9195
Epoch 5/5 - Train Loss: 1.0356, Train Accuracy: 0.9642 - Val Loss:
1.0668, Val Accuracy: 0.9639
Completed training for hidden_size = 50
Epoch 1/5 - Train Loss: 1.5462, Train Accuracy: 0.8262 - Val Loss:
30.3408, Val Accuracy: 0.1994
Epoch 2/5 - Train Loss: 0.8332, Train Accuracy: 0.8967 - Val Loss:
19.1903, Val Accuracy: 0.3361
Epoch 3/5 - Train Loss: 1.0777, Train Accuracy: 0.9205 - Val Loss:
```

```
11.8448, Val Accuracy: 0.5724
Epoch 4/5 - Train Loss: 1.3010, Train Accuracy: 0.9346 - Val Loss:
19.3208, Val Accuracy: 0.5599
Epoch 5/5 - Train Loss: 1.4493, Train Accuracy: 0.9435 - Val Loss:
23.3425, Val Accuracy: 0.5950
Completed training for hidden_size = 70
Epoch 1/5 - Train Loss: 0.7810, Train Accuracy: 0.8201 - Val Loss:
1.8387, Val Accuracy: 0.4948
Epoch 2/5 - Train Loss: 0.2548, Train Accuracy: 0.9252 - Val Loss:
0.4694, Val Accuracy: 0.8480
Epoch 3/5 - Train Loss: 0.1823, Train Accuracy: 0.9465 - Val Loss:
0.2784, Val Accuracy: 0.9070
Epoch 4/5 - Train Loss: 0.1542, Train Accuracy: 0.9556 - Val Loss:
0.1537, Val Accuracy: 0.9507
Epoch 5/5 - Train Loss: 0.1430, Train Accuracy: 0.9573 - Val Loss:
0.1159, Val Accuracy: 0.9637
Completed training for batch_size = 128
Epoch 1/5 - Train Loss: 1.4737, Train Accuracy: 0.7449 - Val Loss:
5.6710, Val Accuracy: 0.3106
Epoch 2/5 - Train Loss: 0.4352, Train Accuracy: 0.8661 - Val Loss:
2.4213, Val Accuracy: 0.4797
Epoch 3/5 - Train Loss: 0.3141, Train Accuracy: 0.9019 - Val Loss:
1.8878, Val Accuracy: 0.5518
Epoch 4/5 - Train Loss: 0.2500, Train Accuracy: 0.9230 - Val Loss:
2.1881, Val Accuracy: 0.5091
Epoch 5/5 - Train Loss: 0.2089, Train Accuracy: 0.9357 - Val Loss:
1.2663, Val Accuracy: 0.6468
Completed training for batch_size = 256
Epoch 1/5 - Train Loss: 2.0295, Train Accuracy: 0.6843 - Val Loss:
3.6860, Val Accuracy: 0.4168
Epoch 2/5 - Train Loss: 0.5258, Train Accuracy: 0.8383 - Val Loss:
2.7441, Val Accuracy: 0.4734
Epoch 3/5 - Train Loss: 0.4646, Train Accuracy: 0.8554 - Val Loss:
2.0511, Val Accuracy: 0.5145
Epoch 4/5 - Train Loss: 0.3481, Train Accuracy: 0.8916 - Val Loss:
1.1292, Val Accuracy: 0.6891
Epoch 5/5 - Train Loss: 0.2614, Train Accuracy: 0.9219 - Val Loss:
0.7475, Val Accuracy: 0.7613
Completed training for batch_size = 512
Epoch 1/5 - Train Loss: 0.5938, Train Accuracy: 0.8592 - Val Loss:
0.4149, Val Accuracy: 0.8692
Epoch 2/5 - Train Loss: 0.2716, Train Accuracy: 0.9252 - Val Loss:
0.1999, Val Accuracy: 0.9441
Epoch 3/5 - Train Loss: 0.2781, Train Accuracy: 0.9304 - Val Loss:
0.5721, Val Accuracy: 0.8686
Epoch 4/5 - Train Loss: 0.2720, Train Accuracy: 0.9355 - Val Loss:
0.7362, Val Accuracy: 0.8507
Epoch 5/5 - Train Loss: 0.2852, Train Accuracy: 0.9384 - Val Loss:
0.6331, Val Accuracy: 0.8785
```

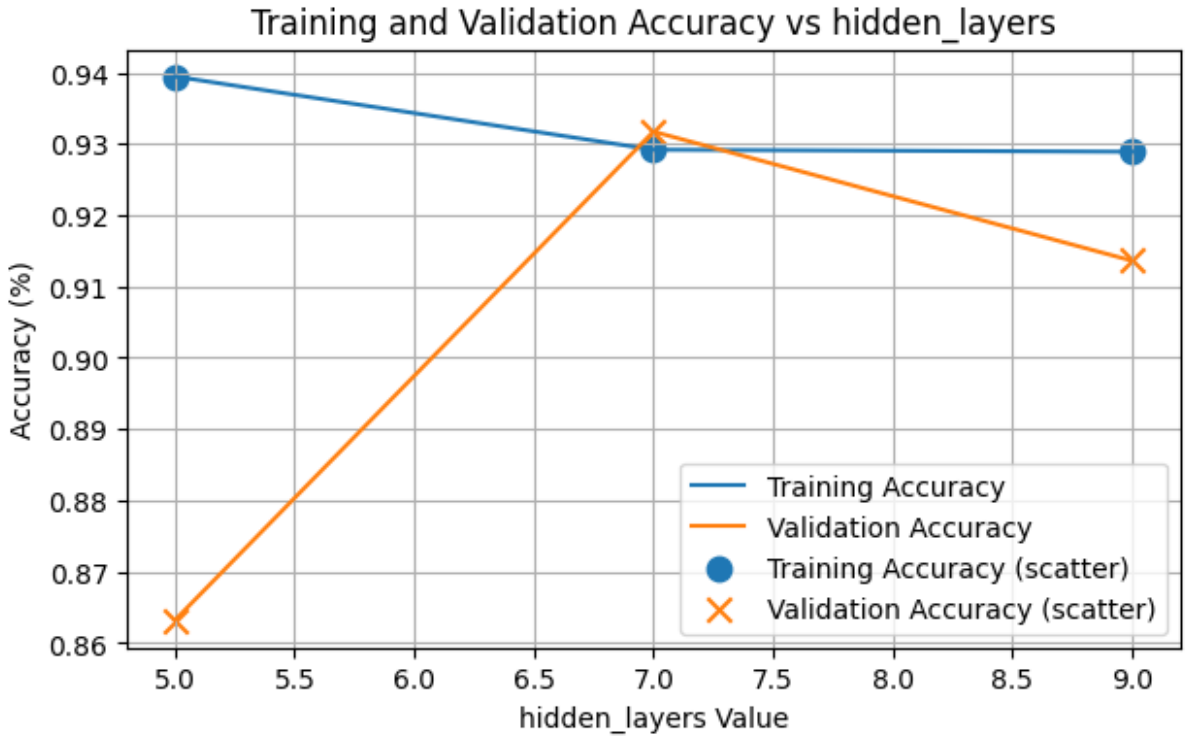
```
Completed training for learning_rate = 0.01
Epoch 1/5 - Train Loss: 0.4270, Train Accuracy: 0.8684 - Val Loss:
0.1632, Val Accuracy: 0.9499
Epoch 2/5 - Train Loss: 0.1981, Train Accuracy: 0.9429 - Val Loss:
0.1018, Val Accuracy: 0.9703
Epoch 3/5 - Train Loss: 0.1560, Train Accuracy: 0.9572 - Val Loss:
0.0898, Val Accuracy: 0.9734
Epoch 4/5 - Train Loss: 0.1376, Train Accuracy: 0.9621 - Val Loss:
0.0885, Val Accuracy: 0.9753
Epoch 5/5 - Train Loss: 0.1187, Train Accuracy: 0.9668 - Val Loss:
0.0706, Val Accuracy: 0.9781
Completed training for learning_rate = 0.001
Epoch 1/5 - Train Loss: 1.0787, Train Accuracy: 0.6462 - Val Loss:
0.4337, Val Accuracy: 0.8765
Epoch 2/5 - Train Loss: 0.3629, Train Accuracy: 0.8865 - Val Loss:
0.2982, Val Accuracy: 0.9088
Epoch 3/5 - Train Loss: 0.2604, Train Accuracy: 0.9189 - Val Loss:
0.2076, Val Accuracy: 0.9361
Epoch 4/5 - Train Loss: 0.2086, Train Accuracy: 0.9355 - Val Loss:
0.1822, Val Accuracy: 0.9462
Epoch 5/5 - Train Loss: 0.1730, Train Accuracy: 0.9466 - Val Loss:
0.1608, Val Accuracy: 0.9511
Completed training for learning_rate = 0.0001
Epoch 1/5 - Train Loss: 0.4303, Train Accuracy: 0.9040 - Val Loss:
0.1624, Val Accuracy: 0.9519
Epoch 2/5 - Train Loss: 0.1589, Train Accuracy: 0.9575 - Val Loss:
0.1885, Val Accuracy: 0.9544
Epoch 3/5 - Train Loss: 0.1715, Train Accuracy: 0.9598 - Val Loss:
0.2638, Val Accuracy: 0.9492
Epoch 4/5 - Train Loss: 0.1849, Train Accuracy: 0.9624 - Val Loss:
0.1579, Val Accuracy: 0.9669
Epoch 5/5 - Train Loss: 0.1979, Train Accuracy: 0.9644 - Val Loss:
0.3136, Val Accuracy: 0.9548
Completed training for dropout_prob = 0.25
Epoch 1/5 - Train Loss: 0.6011, Train Accuracy: 0.8551 - Val Loss:
0.8950, Val Accuracy: 0.6911
Epoch 2/5 - Train Loss: 0.2787, Train Accuracy: 0.9250 - Val Loss:
0.9171, Val Accuracy: 0.7662
Epoch 3/5 - Train Loss: 0.2750, Train Accuracy: 0.9308 - Val Loss:
0.3244, Val Accuracy: 0.9098
Epoch 4/5 - Train Loss: 0.2721, Train Accuracy: 0.9338 - Val Loss:
0.3477, Val Accuracy: 0.9098
Epoch 5/5 - Train Loss: 0.2834, Train Accuracy: 0.9370 - Val Loss:
0.2573, Val Accuracy: 0.9432
Completed training for dropout_prob = 0.5
Epoch 1/5 - Train Loss: 1.2692, Train Accuracy: 0.7437 - Val Loss:
0.7023, Val Accuracy: 0.7478
Epoch 2/5 - Train Loss: 0.5890, Train Accuracy: 0.8254 - Val Loss:
1.0721, Val Accuracy: 0.7369
```

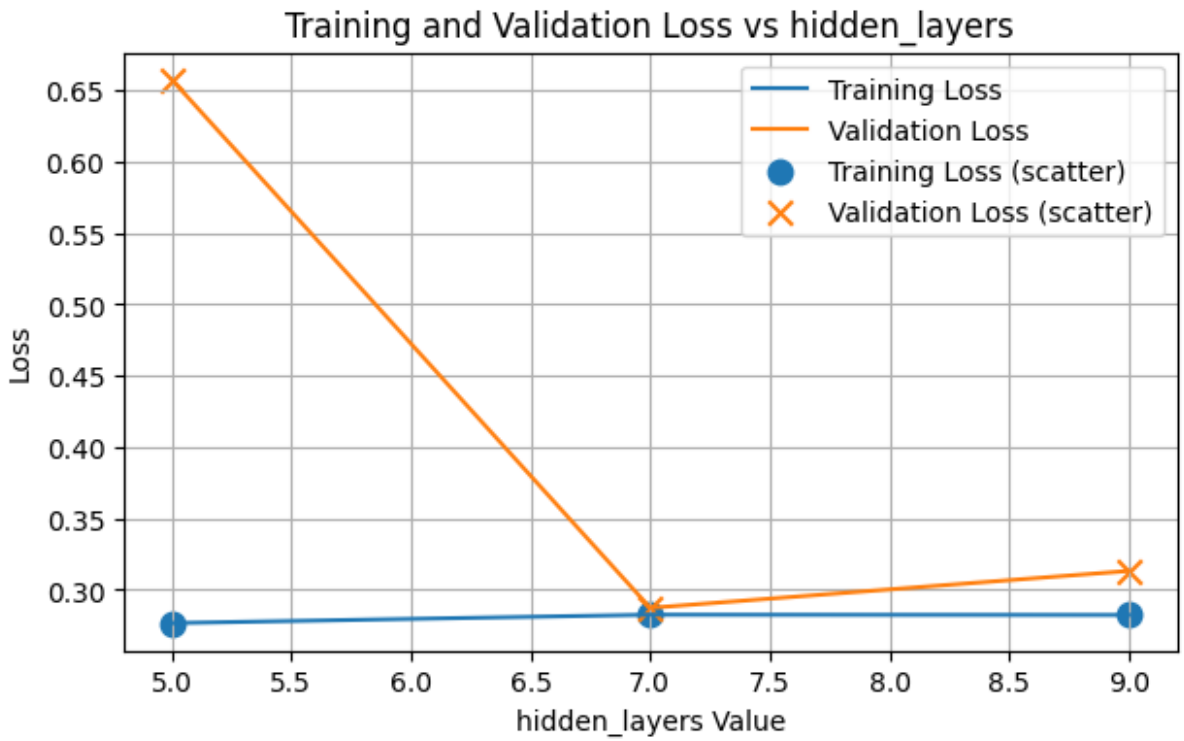
```
Epoch 3/5 - Train Loss: 0.5701, Train Accuracy: 0.8323 - Val Loss:
0.8930, Val Accuracy: 0.7472
Epoch 4/5 - Train Loss: 0.5742, Train Accuracy: 0.8328 - Val Loss:
2.2996, Val Accuracy: 0.5820
Epoch 5/5 - Train Loss: 0.5587, Train Accuracy: 0.8400 - Val Loss:
2.1126, Val Accuracy: 0.5805
Completed training for dropout_prob = 0.75


# Convert the results list to a DataFrame
df = pd.DataFrame(results)

# Inspect the columns to ensure they match expected names
print("Columns in DataFrame:", df.columns)

# Verify sample rows to confirm data integrity
print(df.head())

Columns in DataFrame: Index(['param', 'value', 'train_accuracy',
'val_accuracy', 'train_loss',
       'val_loss', 'y_true', 'y_pred'],
      dtype='object')
          param  value  train_accuracy  val_accuracy  train_loss
val_loss  \
0  hidden_layers    5.0        0.939433        0.8632    0.276677
0.657580
1  hidden_layers    7.0        0.929167        0.9317    0.282560
0.287354
2  hidden_layers    9.0        0.928883        0.9136    0.282382
0.313214
3    hidden_size   30.0        0.952283        0.8538    0.601990
2.358824
4    hidden_size   50.0        0.964200        0.9639    1.035592
1.066752

                                                 y_true  \
0  [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, ...
1  [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, ...
2  [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, ...
3  [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, ...
4  [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, ...

                                                 y_pred
0  [7, 2, 1, 0, 4, 1, 4, 9, 6, 9, 0, 6, 9, 0, 1, ...
1  [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, ...
2  [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, ...
3  [7, 1, 1, 0, 4, 1, 1, 9, 5, 9, 0, 6, 9, 0, 1, ...
4  [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, ...
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Plot Training & Validation Accuracy and Loss for each parameter
separately
unique_params = df["param"].unique()

for param in unique_params:
    # Filter the DataFrame for the current parameter
    subset = df[df["param"] == param]

    print(f"---------------------------------------------- Accuracy and
Loss for {param} ---------------------------------------------- ")

    # Plot Training and Validation Accuracy on the same graph
    plt.figure(figsize=(7, 4))
    plt.plot(subset["value"], subset["train_accuracy"], linestyle='-',
label="Training Accuracy")
    plt.plot(subset["value"], subset["val_accuracy"], linestyle='-',
label="Validation Accuracy")
    plt.scatter(subset["value"], subset["train_accuracy"], marker='o',
s=80, label="Training Accuracy (scatter)")
    plt.scatter(subset["value"], subset["val_accuracy"], marker='x',
s=80, label="Validation Accuracy (scatter)")
    plt.title(f"Training and Validation Accuracy vs {param}")
    plt.xlabel(f"{param} Value")
    plt.ylabel("Accuracy (%)")
    plt.grid()
    plt.legend()
    plt.show()

    # Plot Training and Validation Loss on the same graph
    plt.figure(figsize=(7, 4))
    plt.plot(subset["value"], subset["train_loss"], linestyle='-',
label="Training Loss")
    plt.plot(subset["value"], subset["val_loss"], linestyle='-',
label="Validation Loss")
    plt.scatter(subset["value"], subset["train_loss"], marker='o',
s=80, label="Training Loss (scatter)")
    plt.scatter(subset["value"], subset["val_loss"], marker='x', s=80,
label="Validation Loss (scatter)")
    plt.title(f"Training and Validation Loss vs {param}")
    plt.xlabel(f"{param} Value")
    plt.ylabel("Loss")
    plt.grid()
    plt.legend()
    plt.show()

print(f"---------------------------------------------- Confusion Matrix
for {param} ---------------------------------------------- ")
```

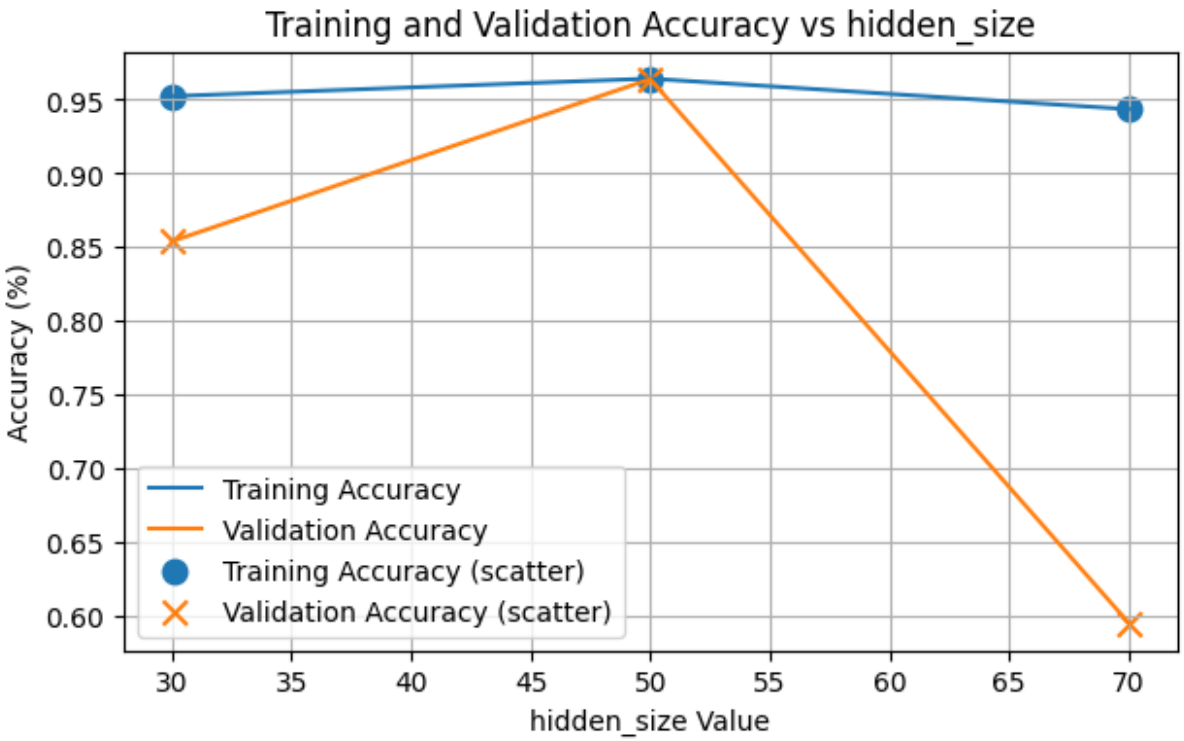
```python
# Plot confusion matrix for each trained model
def plot_confusion_matrix(y_true, y_pred, param, value):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(7, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=range(10), yticklabels=range(10))
    plt.title(f'Confusion Matrix for {param} = {value}')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Generate confusion matrices for each parameter and value
for idx, row in df.iterrows():
    plot_confusion_matrix(row["y_true"], row["y_pred"], row["param"],
row["value"])
```

```
------------------------------------------- Accuracy and Loss for
hidden_layers -------------------------------------------
```

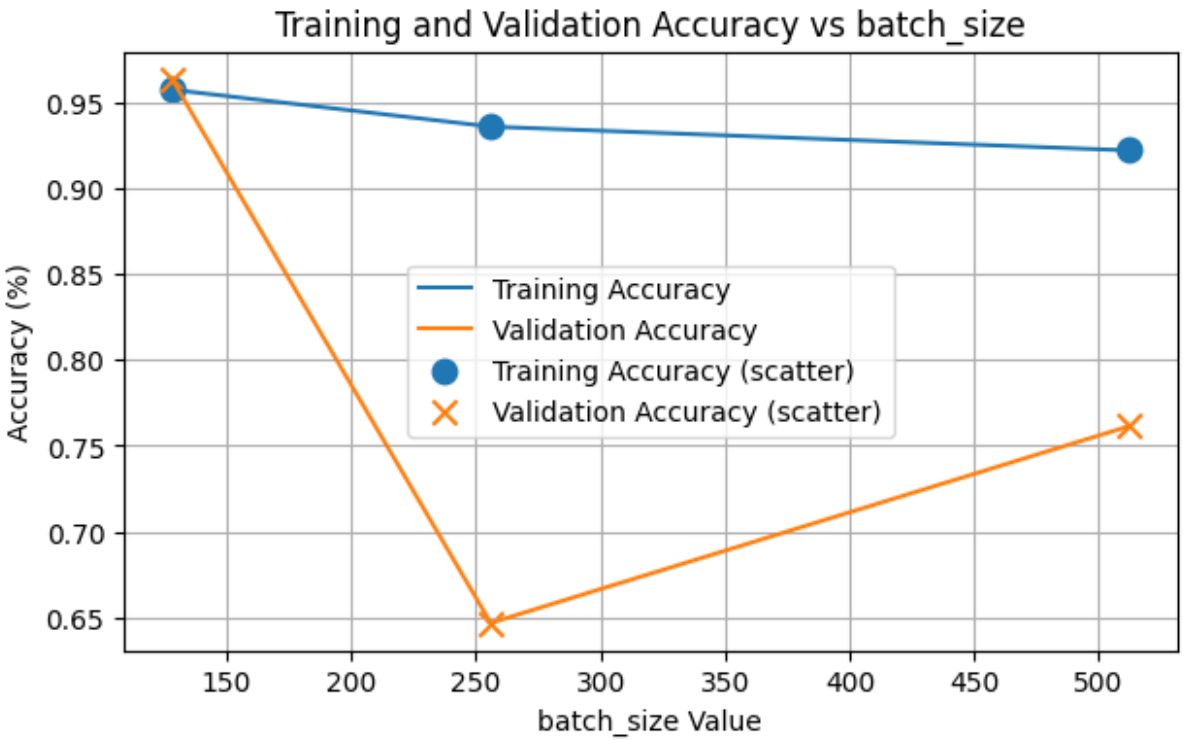


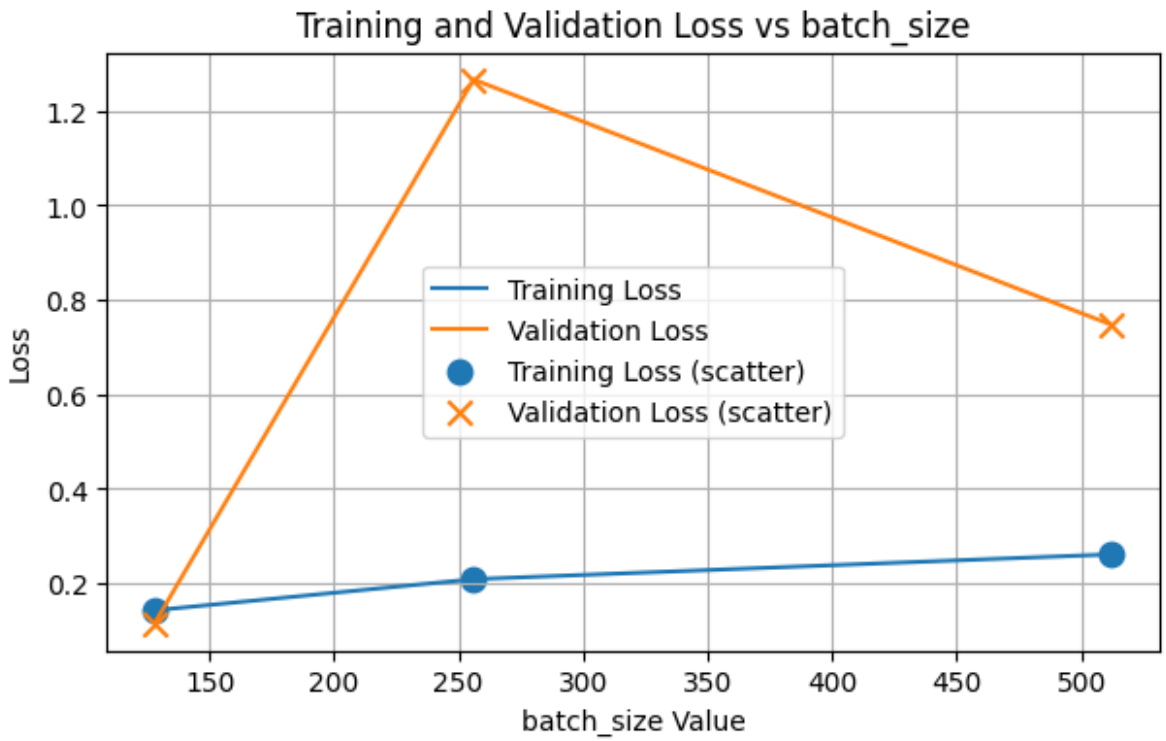



```
---------------------------------------------- Accuracy and Loss for hidden_size ----------------------------------------------
```

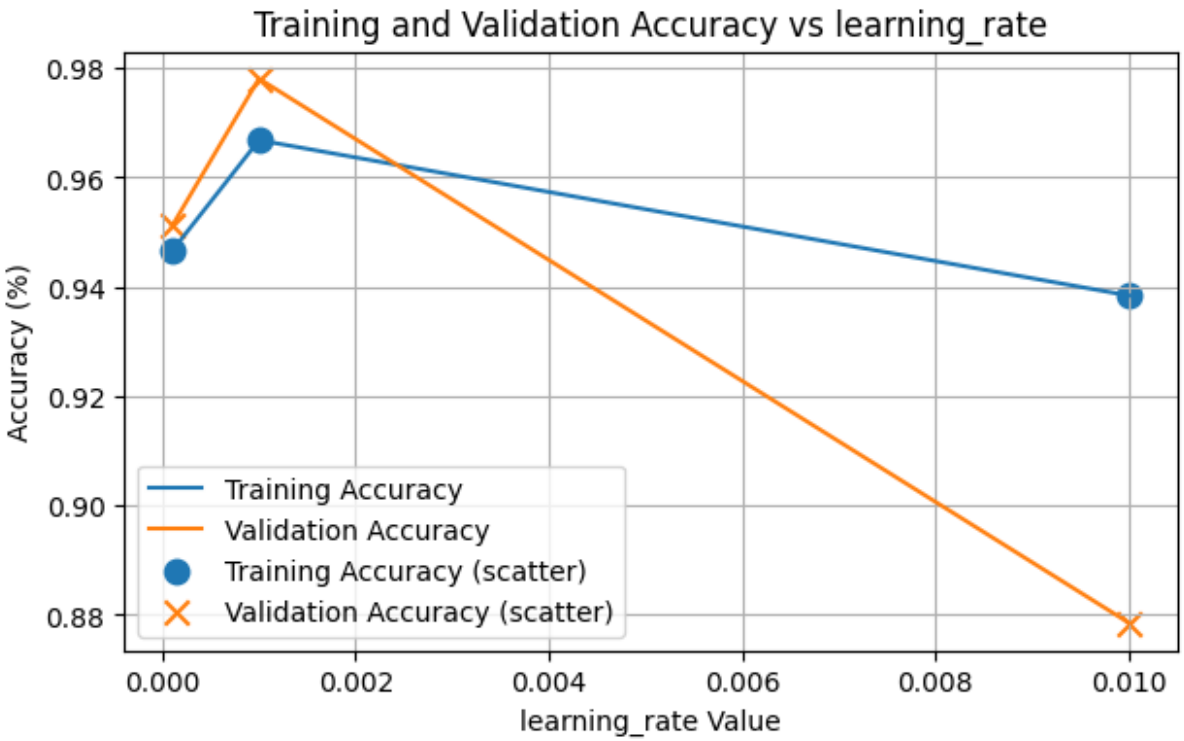


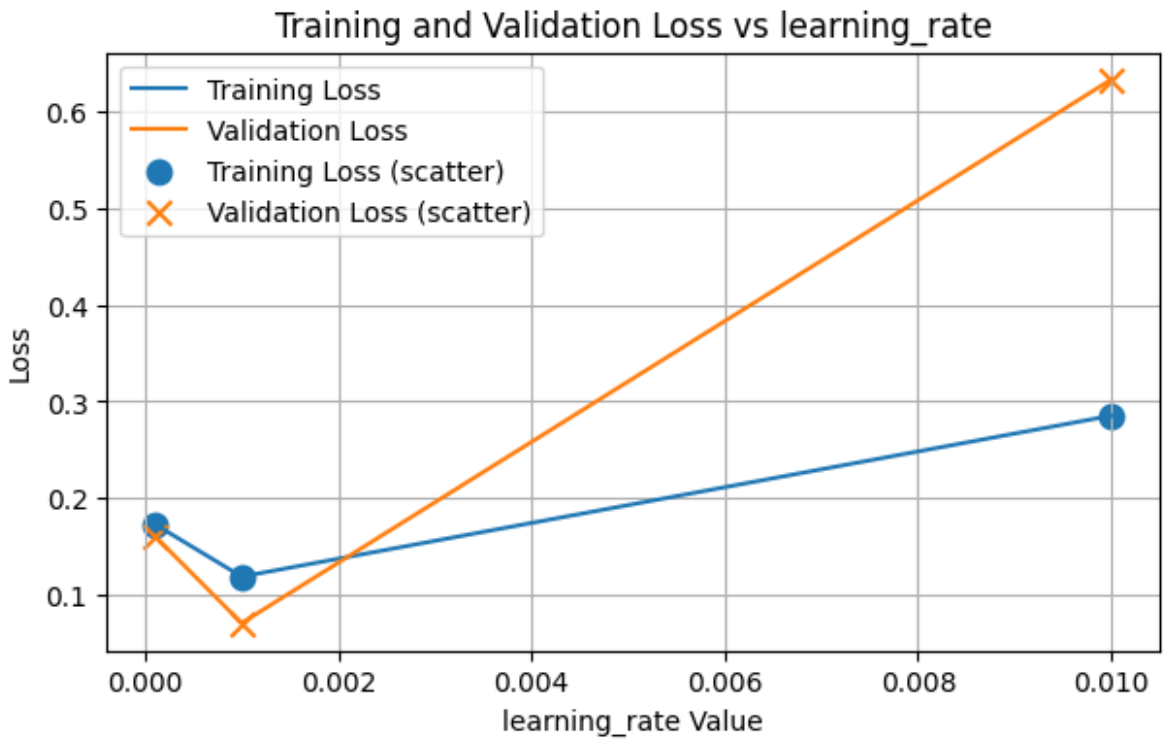
```
---------------------------------------------- Accuracy and Loss for batch_size ----------------------------------------------
```



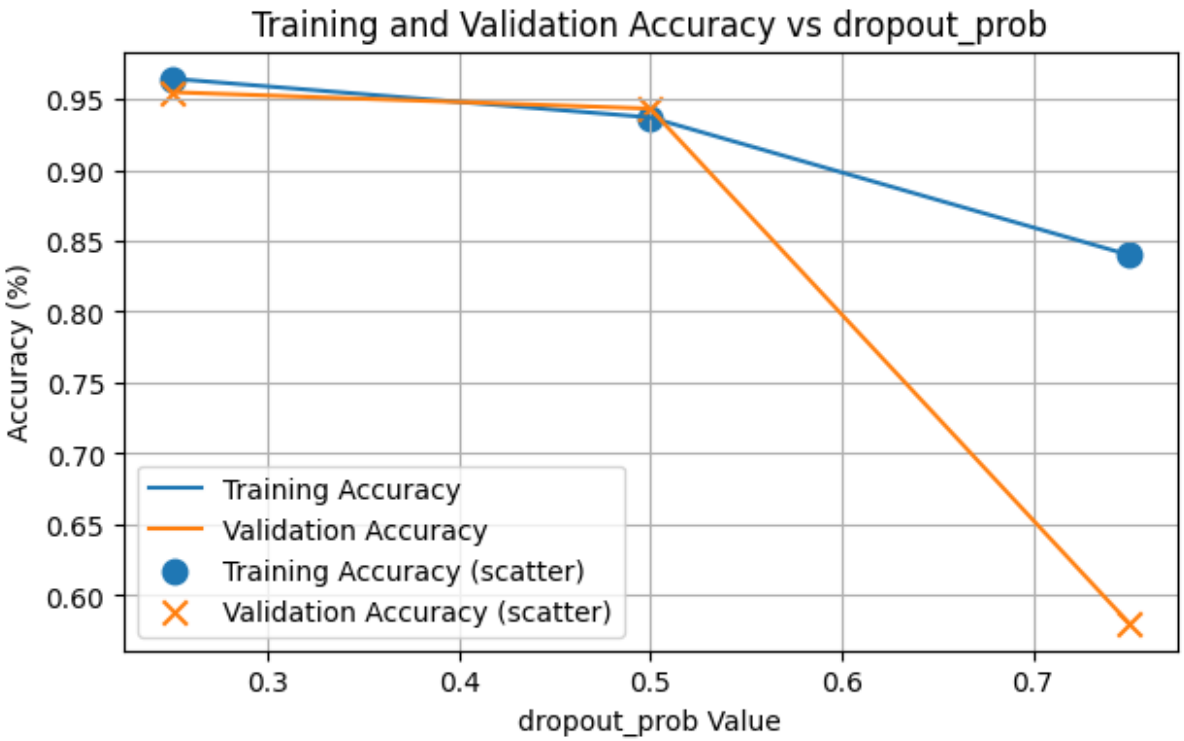


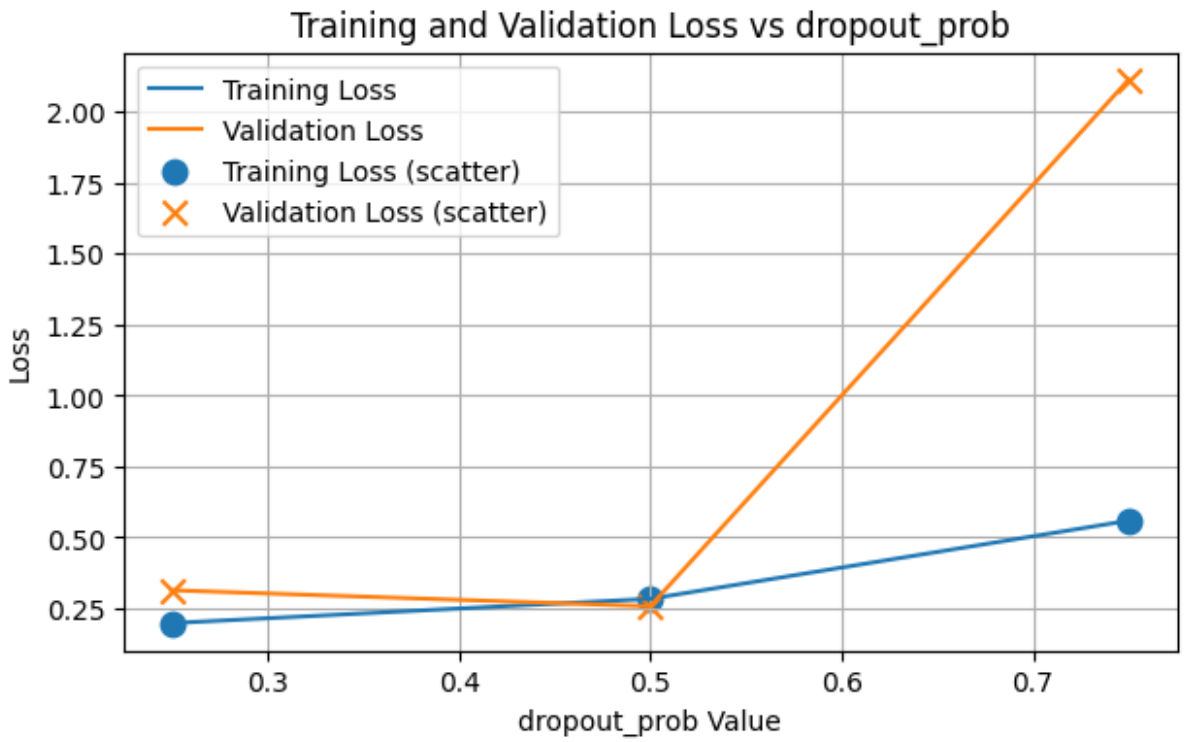

```
---------------------------------------------- Accuracy and Loss for learning_rate ----------------------------------------------
```



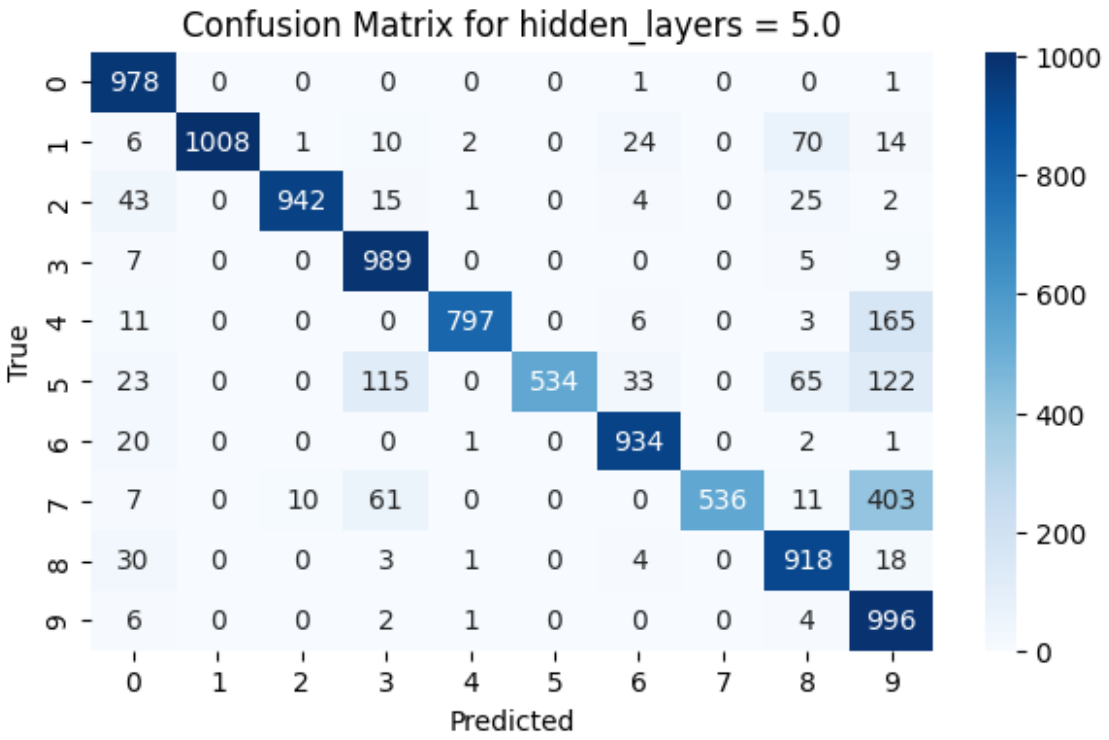


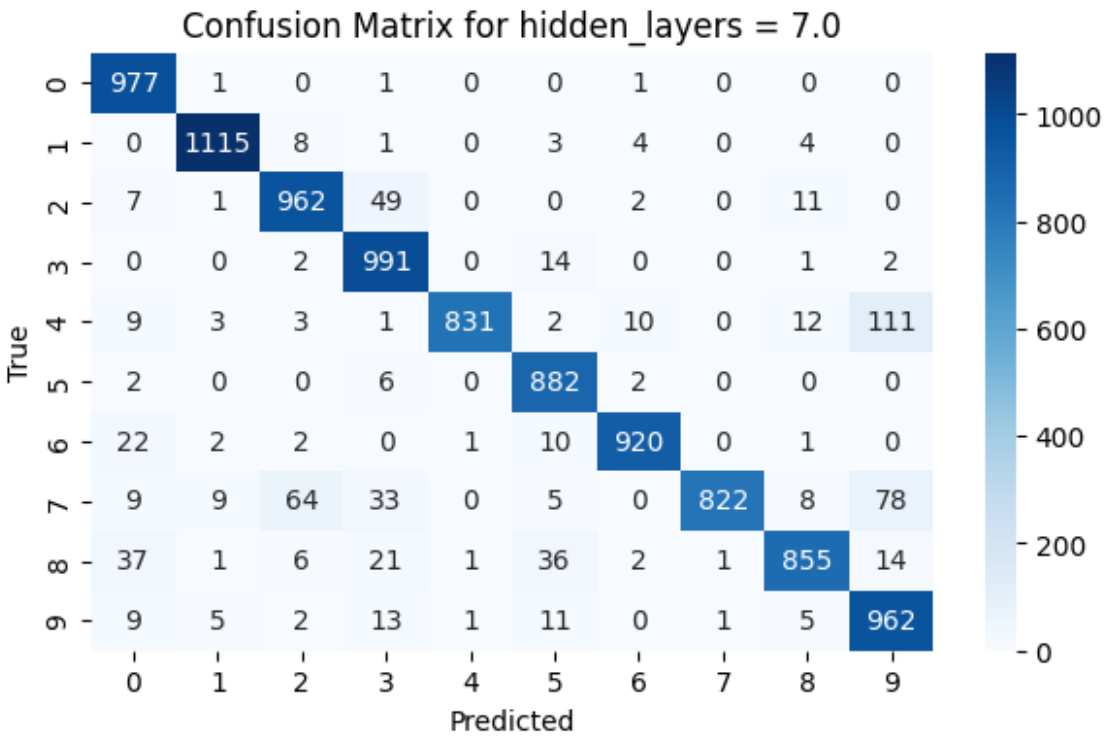

```
---------------------------------------------- Accuracy and Loss for dropout_prob ----------------------------------------------
```

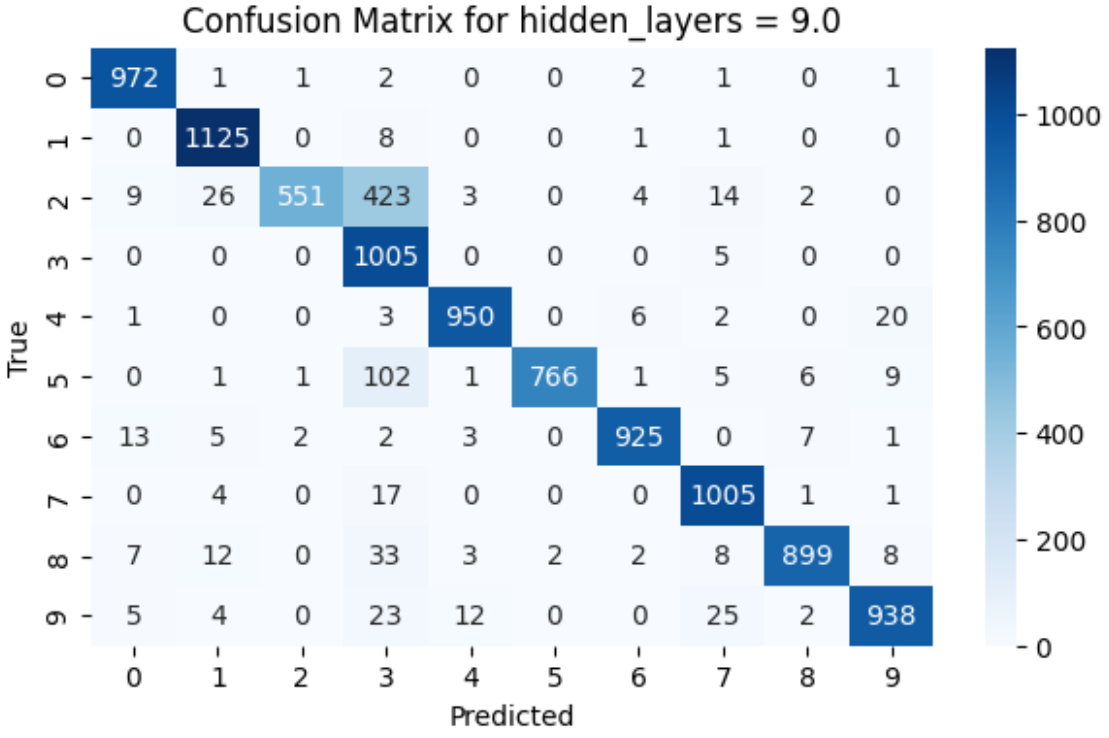


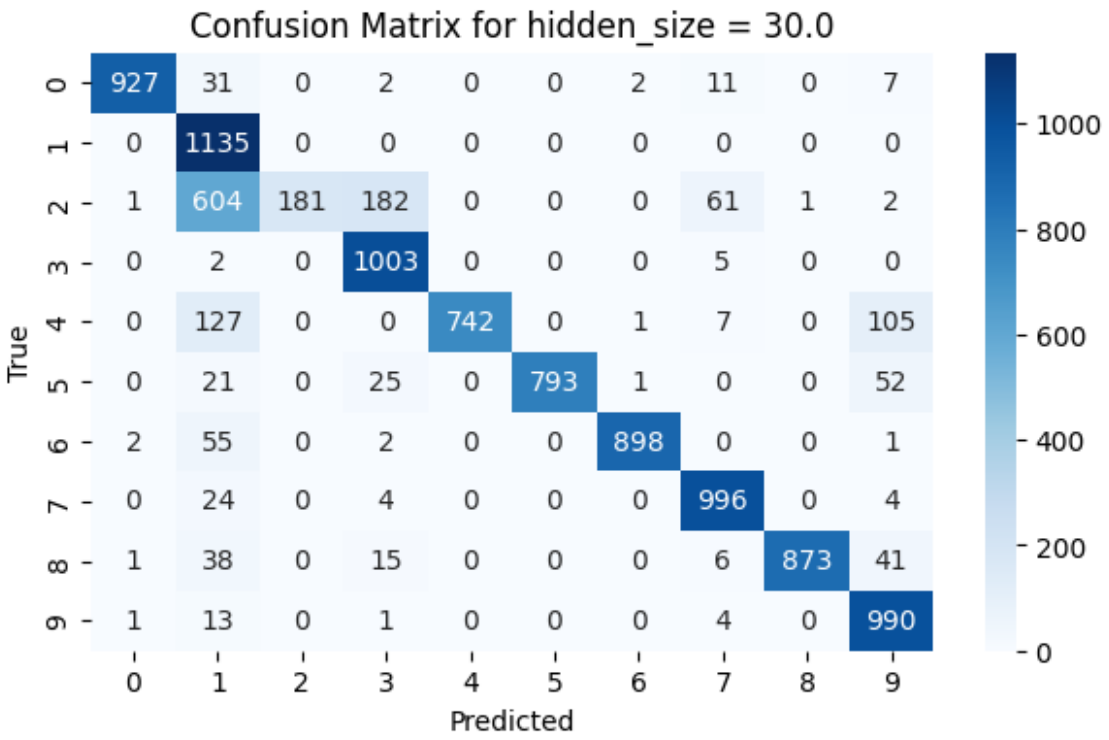


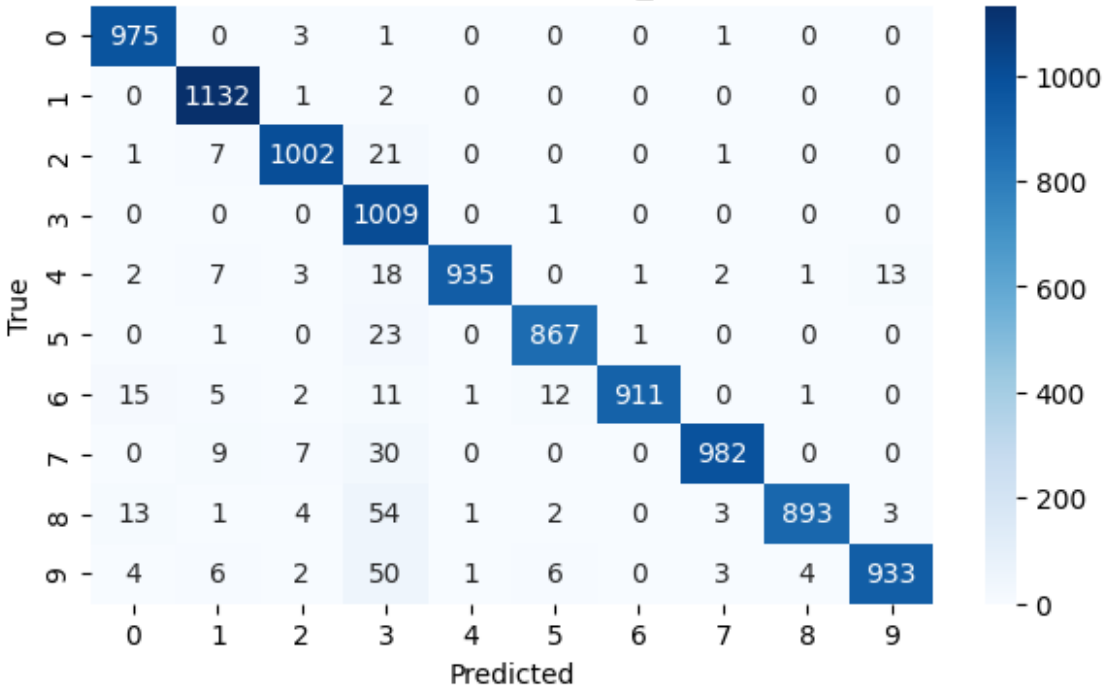

```
---------------------------------------------- Confusion Matrix for dropout_prob ----------------------------------------------
```

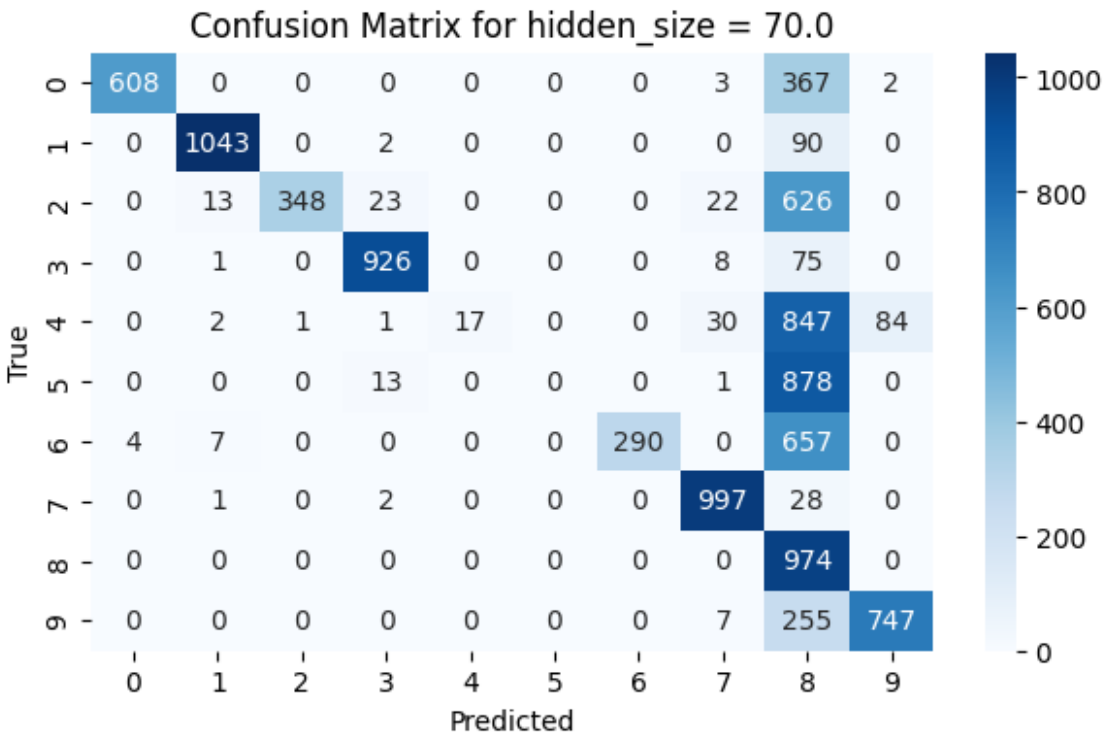


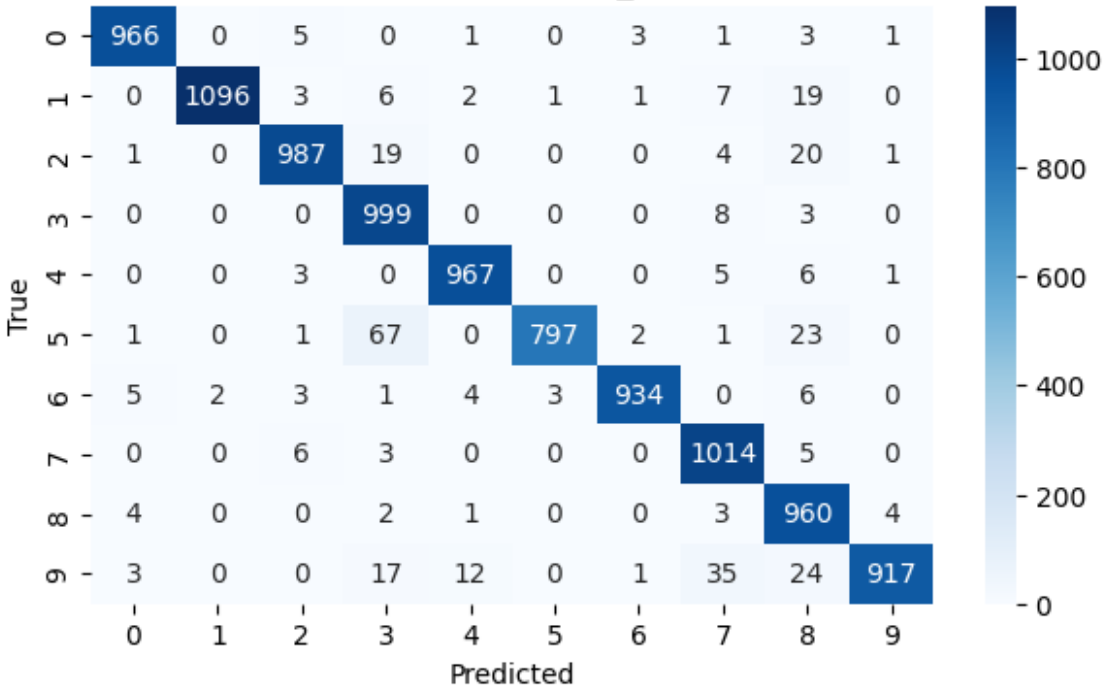

Confusion Matrix for hidden_layers = 7.0
True
Predicted
0 977 1 0 1 0 0 1 0 0 0
1 0 1115 8 1 0 3 4 0 4 0
2 7 1 962 49 0 0 2 0 11 0
3 0 0 2 991 0 14 0 0 1 2
4 9 3 3 1 831 2 10 0 12 111
5 2 0 0 6 0 882 2 0 0 0
6 22 2 2 0 1 10 920 0 1 0
7 9 9 64 33 0 5 0 822 8 78
8 37 1 6 21 1 36 2 1 855 14
9 9 5 2 13 1 11 0 1 5 962
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

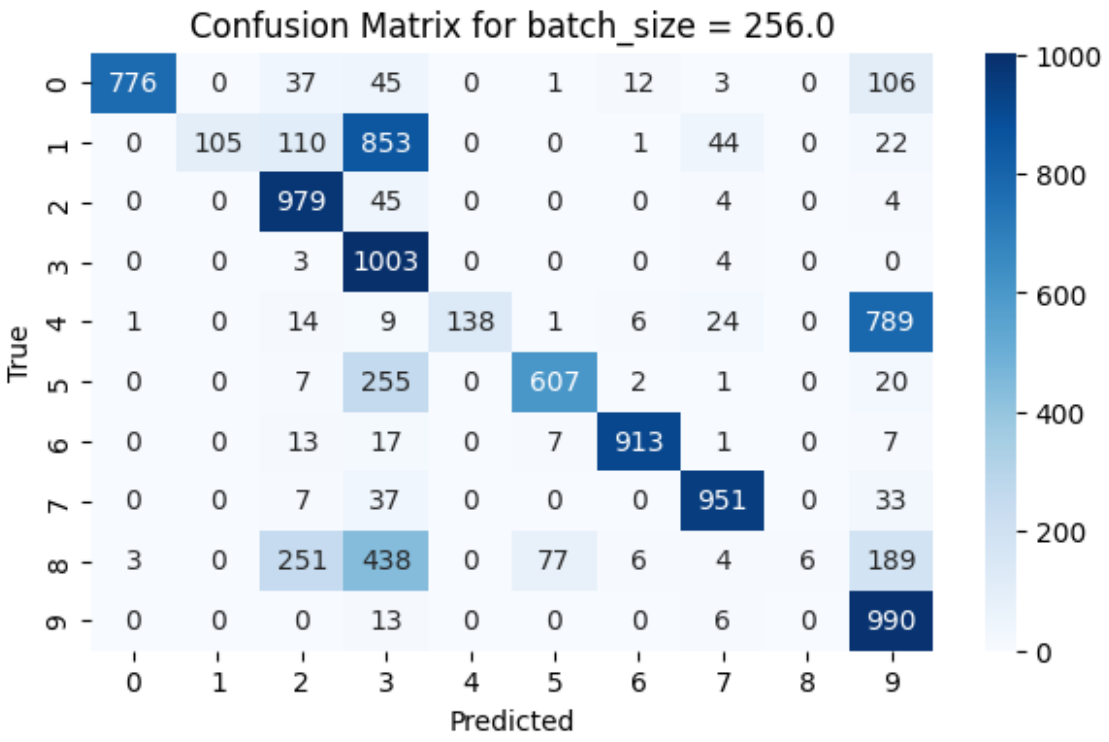

Confusion Matrix for hidden_layers = 9.0
True
Predicted
0 972 1 1 2 0 0 2 1 0 1
1 0 1125 0 8 0 0 1 1 0 0
2 9 26 551 423 3 0 4 14 2 0
3 0 0 0 1005 0 0 0 5 0 0
4 1 0 0 3 950 0 6 2 0 20
5 0 1 1 102 1 766 1 5 6 9
6 13 5 2 2 3 0 925 0 7 1
7 0 4 0 17 0 0 0 1005 1 1
8 7 12 0 33 3 2 2 8 899 8
9 5 4 0 23 12 0 0 25 2 938
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

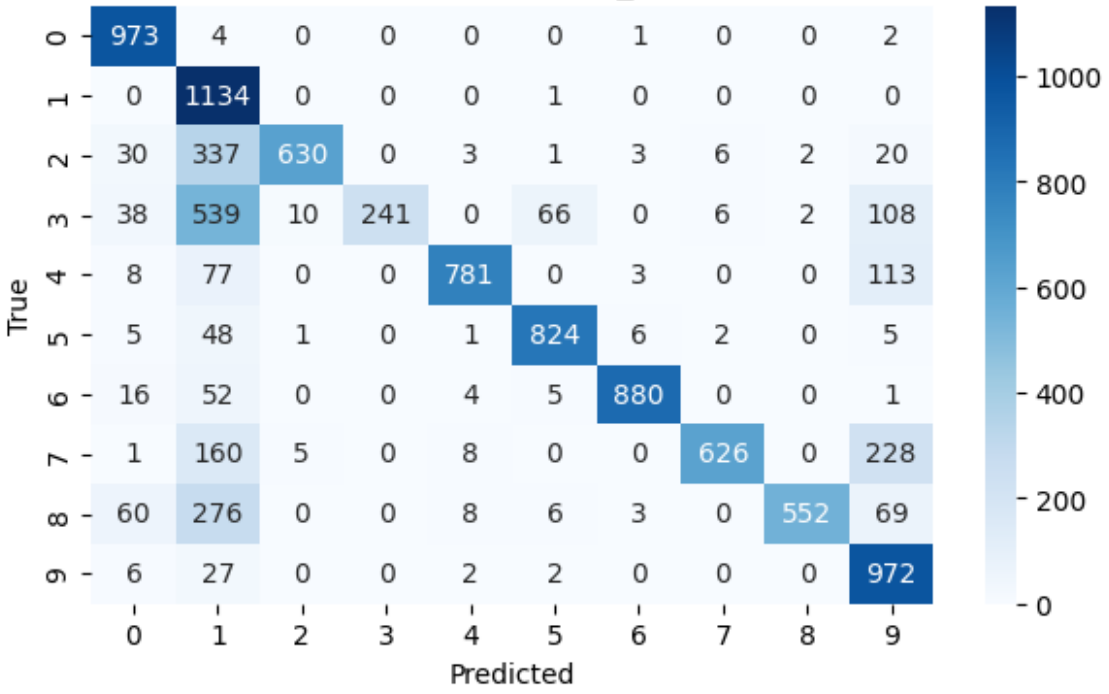

Confusion Matrix for hidden_size = 30.0
True
Predicted
0 927 31 0 2 0 0 2 11 0 7
1 0 1135 0 0 0 0 0 0 0 0
2 1 604 181 182 0 0 0 61 1 2
3 0 2 0 1003 0 0 0 5 0 0
4 0 127 0 0 742 0 1 7 0 105
5 0 21 0 25 0 793 1 0 0 52
6 2 55 0 2 0 0 898 0 0 1
7 0 24 0 4 0 0 0 996 0 4
8 1 38 0 15 0 0 0 6 873 41
9 1 13 0 1 0 0 0 4 0 990
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

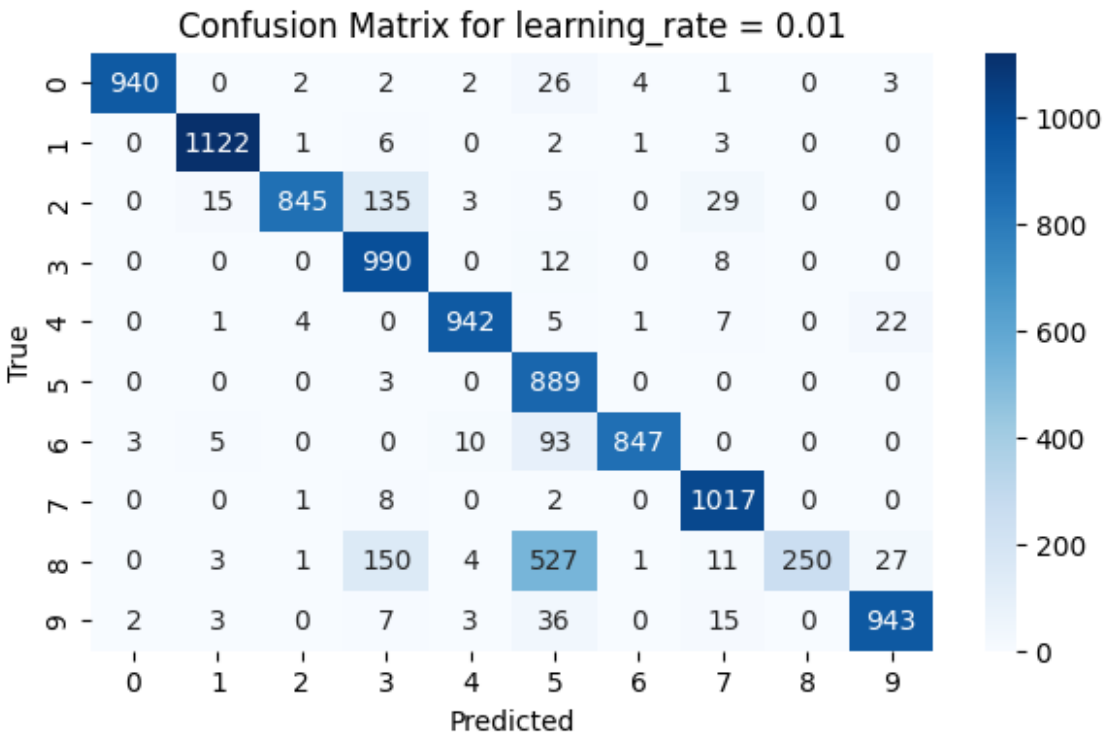

Confusion Matrix for hidden_size = 50.0
True
Predicted
0 975 0 3 1 0 0 0 1 0 0
1 0 1132 1 2 0 0 0 0 0 0
2 1 7 1002 21 0 0 0 1 0 0
3 0 0 0 1009 0 1 0 0 0 0
4 2 7 3 18 935 0 1 2 1 13
5 0 1 0 23 0 867 1 0 0 0
6 15 5 2 11 1 12 911 0 1 0
7 0 9 7 30 0 0 0 982 0 0
8 13 1 4 54 1 2 0 3 893 3
9 4 6 2 50 1 6 0 3 4 933
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

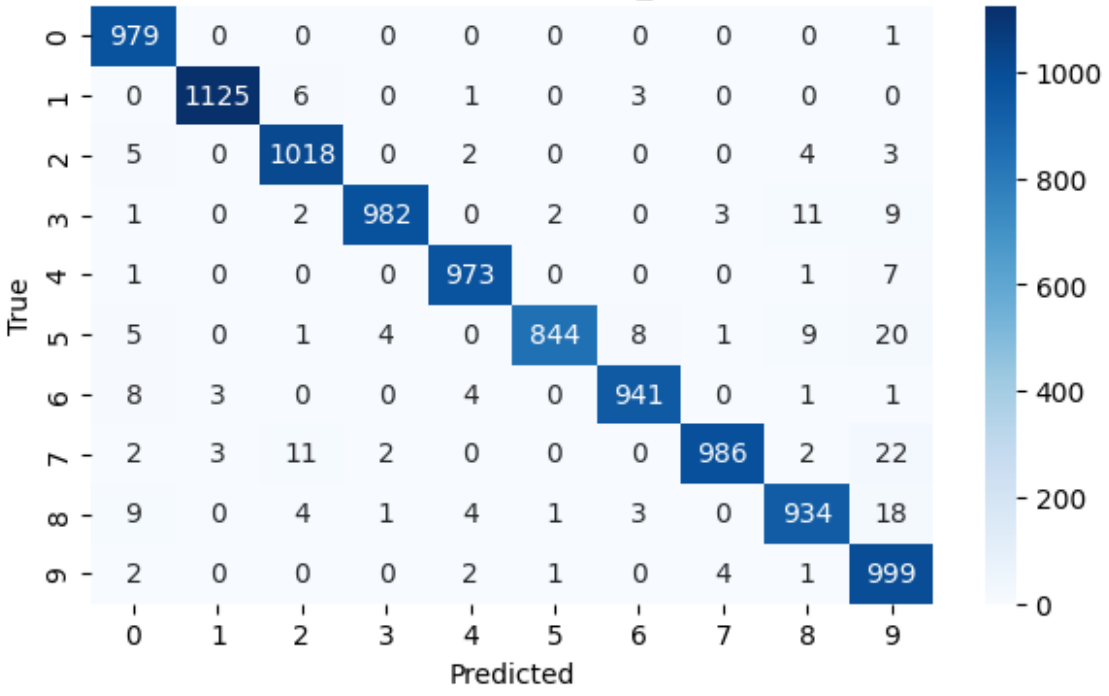

Confusion Matrix for hidden_size = 70.0
True
Predicted
0 608 0 0 0 0 0 0 3 367 2
1 0 1043 0 2 0 0 0 0 90 0
2 0 13 348 23 0 0 0 22 626 0
3 0 1 0 926 0 0 0 8 75 0
4 0 2 1 1 17 0 0 30 847 84
5 0 0 0 13 0 0 0 1 878 0
6 4 7 0 0 0 0 290 0 657 0
7 0 1 0 2 0 0 0 997 28 0
8 0 0 0 0 0 0 0 0 974 0
9 0 0 0 0 0 0 0 7 255 747
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

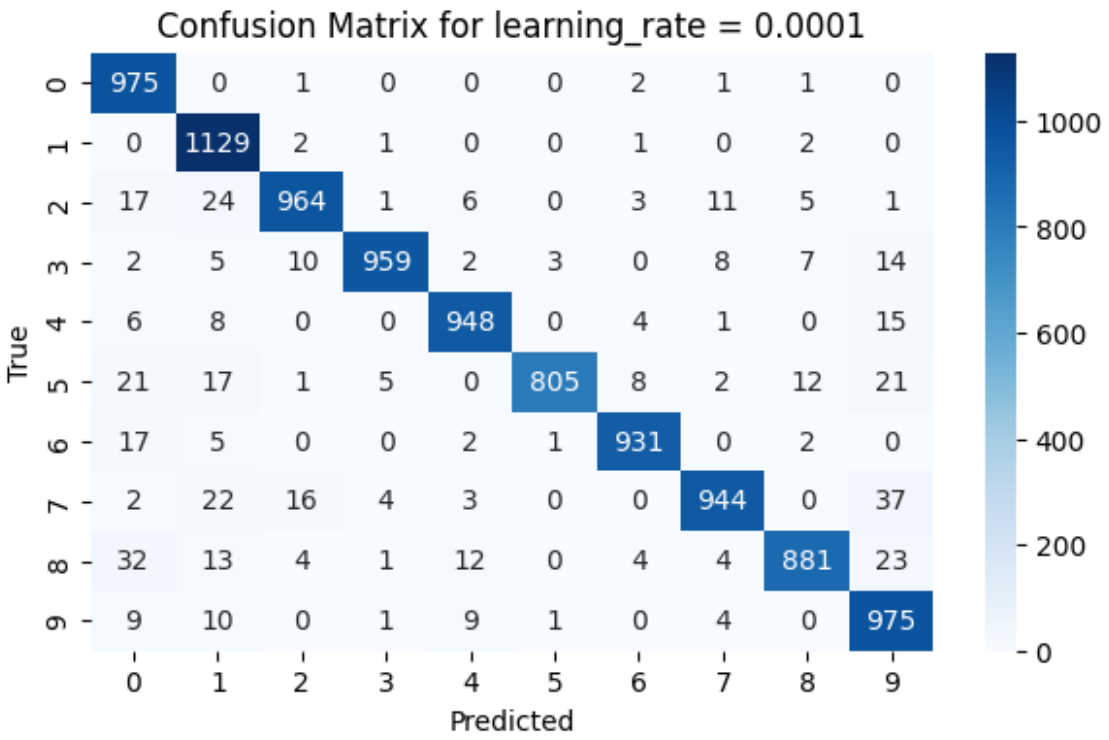

Confusion Matrix for batch_size = 128.0
True
Predicted
0 966 0 5 0 1 0 3 1 3 1
1 0 1096 3 6 2 1 1 7 19 0
2 1 0 987 19 0 0 0 4 20 1
3 0 0 0 999 0 0 0 8 3 0
4 0 0 3 0 967 0 0 5 6 1
5 1 0 1 67 0 797 2 1 23 0
6 5 2 3 1 4 3 934 0 6 0
7 0 0 6 3 0 0 0 1014 5 0
8 4 0 0 2 1 0 0 3 960 4
9 3 0 0 17 12 0 1 35 24 917
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

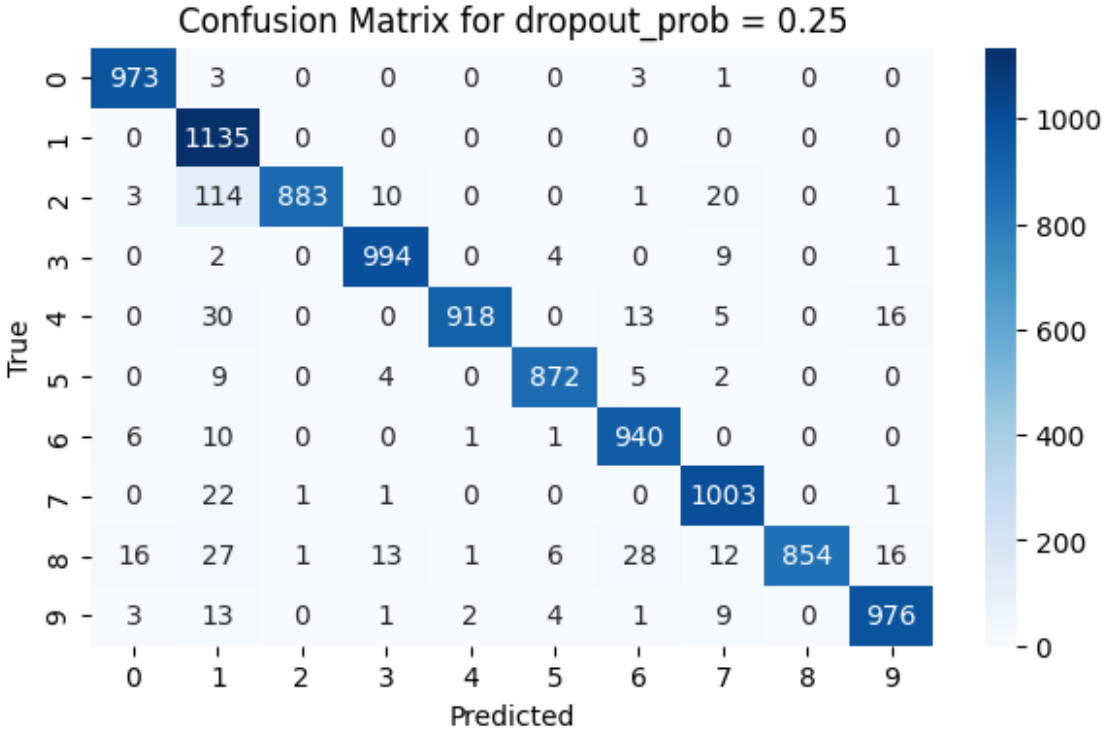

Confusion Matrix for batch_size = 256.0
True
Predicted
0 776 0 37 45 0 1 12 3 0 106
1 0 105 110 853 0 0 1 44 0 22
2 0 0 979 45 0 0 0 4 0 4
3 0 0 3 1003 0 0 0 4 0 0
4 1 0 14 9 138 1 6 24 0 789
5 0 0 7 255 0 607 2 1 0 20
6 0 0 13 17 0 7 913 1 0 7
7 0 0 7 37 0 0 0 951 0 33
8 3 0 251 438 0 77 6 4 6 189
9 0 0 0 13 0 0 0 6 0 990
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

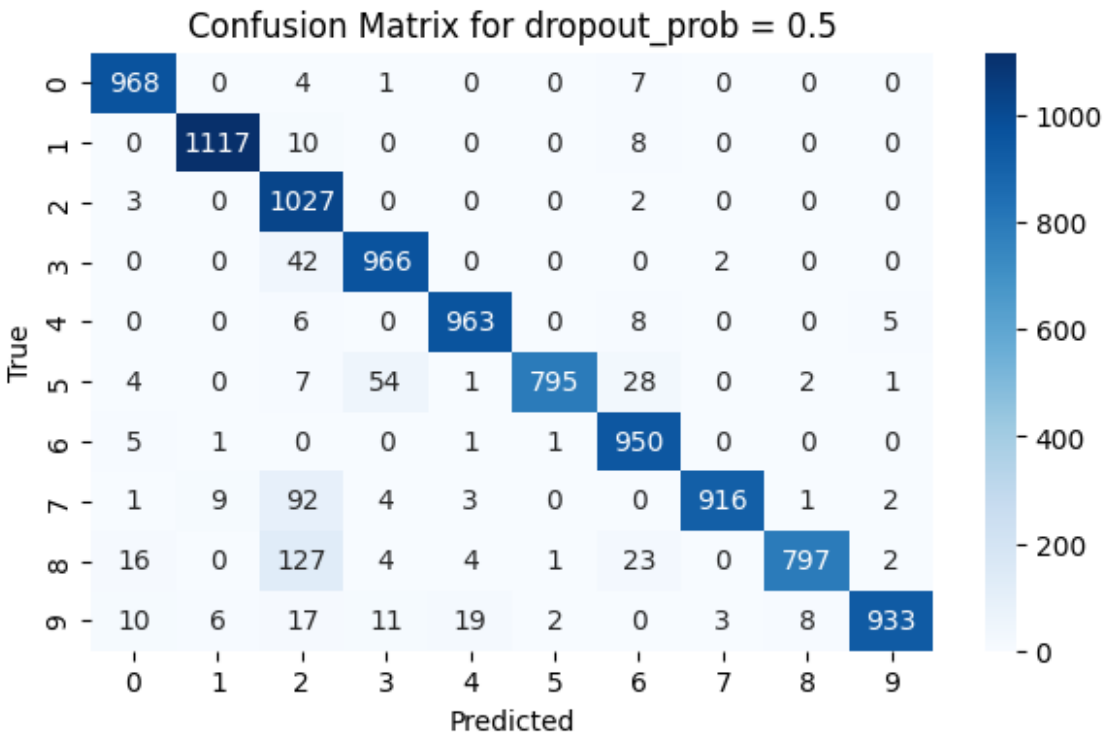

Confusion Matrix for batch_size = 512.0
True
Predicted
0 973 4 0 0 0 0 1 0 0 2
1 0 1134 0 0 0 1 0 0 0 0
2 30 337 630 0 3 1 3 6 2 20
3 38 539 10 241 0 66 0 6 2 108
4 8 77 0 0 781 0 3 0 0 113
5 5 48 1 0 1 824 6 2 0 5
6 16 52 0 0 4 5 880 0 0 1
7 1 160 5 0 8 0 0 626 0 228
8 60 276 0 0 8 6 3 0 552 69
9 6 27 0 0 2 2 0 0 0 972
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

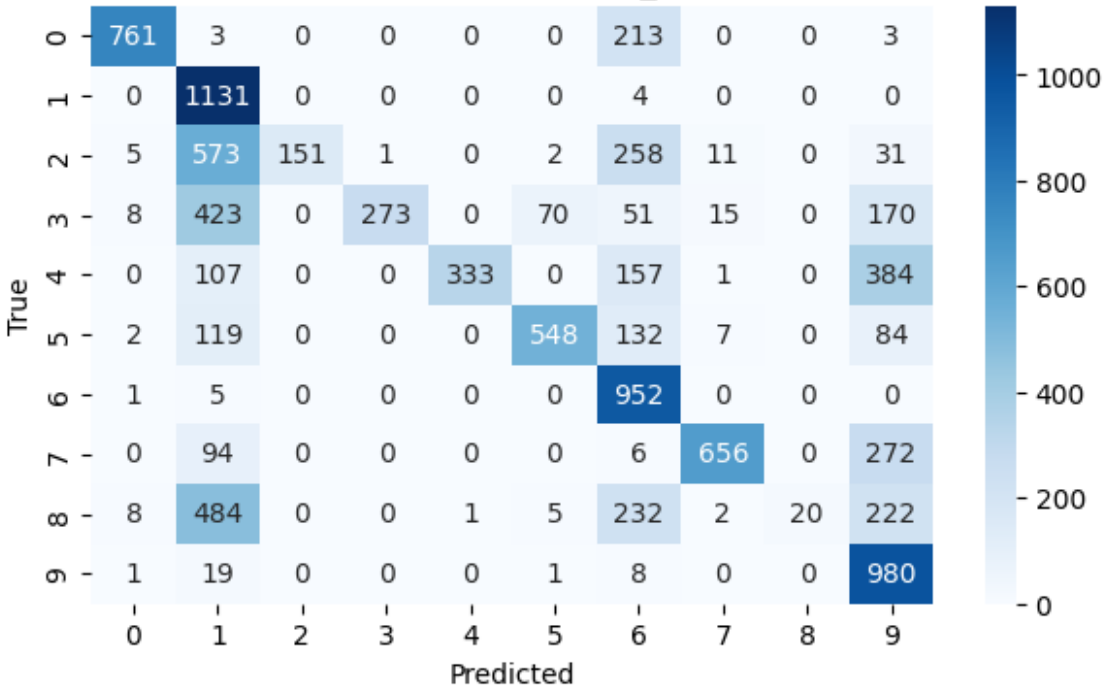

Confusion Matrix for learning_rate = 0.01
True
Predicted
0 940 0 2 2 2 26 4 1 0 3
1 0 1122 1 6 0 2 1 3 0 0
2 0 15 845 135 3 5 0 29 0 0
3 0 0 0 990 0 12 0 8 0 0
4 0 1 4 0 942 5 1 7 0 22
5 0 0 0 3 0 889 0 0 0 0
6 3 5 0 0 10 93 847 0 0 0
7 0 0 1 8 0 2 0 1017 0 0
8 0 3 1 150 4 527 1 11 250 27
9 2 3 0 7 3 36 0 15 0 943
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0


Confusion Matrix for learning_rate = 0.001
True
Predicted
0 979 0 0 0 0 0 0 0 0 1
1 0 1125 6 0 1 0 3 0 0 0
2 5 0 1018 0 2 0 0 0 4 3
3 1 0 2 982 0 2 0 3 11 9
4 1 0 0 0 973 0 0 0 1 7
5 5 0 1 4 0 844 8 1 9 20
6 8 3 0 0 4 0 941 0 1 1
7 2 3 11 2 0 0 0 986 2 22
8 9 0 4 1 4 1 3 0 934 18
9 2 0 0 0 2 1 0 4 1 999
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0


Confusion Matrix for learning_rate = 0.0001
True
Predicted
0 975 0 1 0 0 0 2 1 1 0
1 0 1129 2 1 0 0 1 0 2 0
2 17 24 964 1 6 0 3 11 5 1
3 2 5 10 959 2 3 0 8 7 14
4 6 8 0 0 948 0 4 1 0 15
5 21 17 1 5 0 805 8 2 12 21
6 17 5 0 0 2 1 931 0 2 0
7 2 22 16 4 3 0 0 944 0 37
8 32 13 4 1 12 0 4 4 881 23
9 9 10 0 1 9 1 0 4 0 975
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0


Confusion Matrix for dropout_prob = 0.25
True
Predicted
0 973 3 0 0 0 0 3 1 0 0
1 0 1135 0 0 0 0 0 0 0 0
2 3 114 883 10 0 0 1 20 0 1
3 0 2 0 994 0 4 0 9 0 1
4 0 30 0 0 918 0 13 5 0 16
5 0 9 0 4 0 872 5 2 0 0
6 6 10 0 0 1 1 940 0 0 0
7 0 22 1 1 0 0 0 1003 0 1
8 16 27 1 13 1 6 28 12 854 16
9 3 13 0 1 2 4 1 9 0 976
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0


Confusion Matrix for dropout_prob = 0.5
True
Predicted
0 968 0 4 1 0 0 7 0 0 0
1 0 1117 10 0 0 0 8 0 0 0
2 3 0 1027 0 0 0 2 0 0 0
3 0 0 42 966 0 0 0 2 0 0
4 0 0 6 0 963 0 8 0 0 5
5 4 0 7 54 1 795 28 0 2 1
6 5 1 0 0 1 1 950 0 0 0
7 1 9 92 4 3 0 0 916 1 2
8 16 0 127 4 4 1 23 0 797 2
9 10 6 17 11 19 2 0 3 8 933
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0


Confusion Matrix for dropout_prob = 0.75
True
Predicted
0 761 3 0 0 0 0 213 0 0 3
1 0 1131 0 0 0 0 4 0 0 0
2 5 573 151 1 0 2 258 11 0 31
3 8 423 0 273 0 70 51 15 0 170
4 0 107 0 0 333 0 157 1 0 384
5 2 119 0 0 0 548 132 7 0 84
6 1 5 0 0 0 0 952 0 0 0
7 0 94 0 0 0 0 6 656 0 272
8 8 484 0 0 1 5 232 2 20 222
9 1 19 0 0 0 1 8 0 0 980
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0