```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMat
from sklearn.preprocessing import label_binarize, StandardScaler
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import os
```

```
#loading MNIST dataset and applying normalization and standardization of the data
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data.to_numpy(), mnist.target.astype(int)

X = X / 255.0

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:1022: Fut
  warn(
```

```
#MLP Model definition
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(784, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

#Loading the saved MLP model
mlp_model = MLP()
```

```
drive_dir = '/content/drive/MyDrive/Colab Notebooks/harshith_neco'
model_filename = 'mlp_model.pth'
model_load_path = os.path.join(drive_dir, model_filename)
mlp_model.load_state_dict(torch.load(model_load_path))
```

```
<All keys matched successfully>
```

```
#Evaluating the MLP model
batch_size = 64
y_test_np = y_test.to_numpy()
test_dataset = TensorDataset(torch.tensor(X_test, dtype=torch.float32), torch.ter
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)


def evaluate_model(model, loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in loader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return correct / total



test_accuracy = evaluate_model(mlp_model, test_loader)
print(f"MLP Model Testing Accuracy: {test_accuracy:.4f}")
```

```
MLP Model Testing Accuracy: 0.9698
```

```
Start coding or generate with AI.
```

```
#CNN model definition
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.pool(x)
        x = x.view(-1, 64 * 7 * 7)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
#loading the cnn model
cnn_model = CNN()



cnn_model_filename = 'cnn_model.pth'
cnn_model_load_path = os.path.join(drive_dir, cnn_model_filename)

cnn_model.load_state_dict(torch.load(cnn_model_load_path))
```

```
<All keys matched successfully>
```

```
#evaluating the cnn model
test_dataset = TensorDataset(torch.tensor(X_test.reshape(-1, 1, 28, 28), dtype=tc
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

test_accuracy = evaluate_model(cnn_model, test_loader)
print(f"CNN model Testing Accuracy: {test_accuracy:.4f}")
```

```
CNN model Testing Accuracy: 0.9894
```

```
from sklearn.metrics import classification_report


def print_classification_report(model, X_test, y_test, model_name):
    model.eval()
    with torch.no_grad():
        outputs = model(torch.tensor(X_test, dtype=torch.float32))
        _, predicted = torch.max(outputs, 1)
        print(f"Classification Report for {model_name} model:")
        print(classification_report(y_test, predicted.numpy()))
```

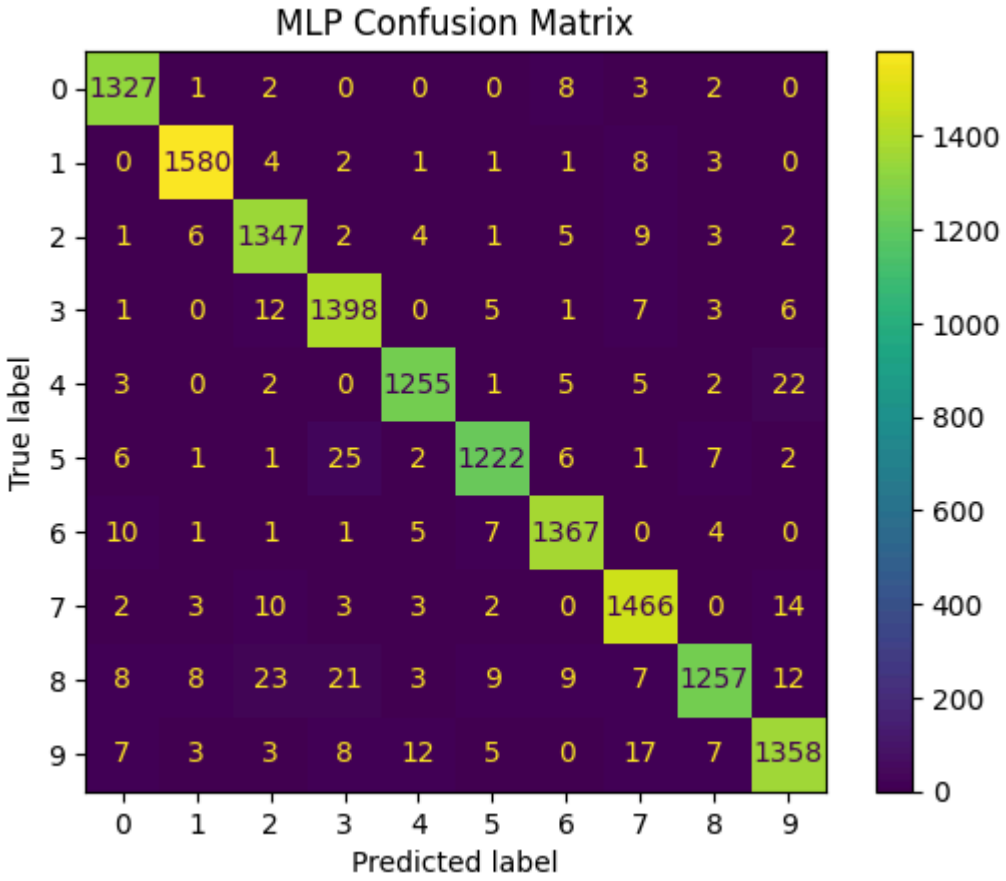
```
#classification report for MLP
print_classification_report(mlp_model, X_test, y_test_np, "MLP")

#classification report for CNN
print_classification_report(cnn_model, X_test.reshape(-1, 1, 28, 28), y_te
```

```
Classification Report for MLP model:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1343
           1       0.99      0.99      0.99      1600
           2       0.96      0.98      0.97      1380
           3       0.96      0.98      0.97      1433
           4       0.98      0.97      0.97      1295
           5       0.98      0.96      0.97      1273
           6       0.98      0.98      0.98      1396
           7       0.96      0.98      0.97      1503
           8       0.98      0.93      0.95      1357
           9       0.96      0.96      0.96      1420

    accuracy                           0.97     14000
   macro avg       0.97      0.97      0.97     14000
weighted avg       0.97      0.97      0.97     14000

Classification Report for CNN model:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1343
           1       0.99      1.00      0.99      1600
           2       0.99      0.99      0.99      1380
           3       0.99      0.98      0.99      1433
           4       1.00      0.98      0.99      1295
           5       0.99      0.99      0.99      1273
           6       1.00      0.99      0.99      1396
           7       0.99      0.99      0.99      1503
           8       0.98      0.98      0.98      1357
           9       0.98      0.99      0.99      1420

    accuracy                           0.99     14000
   macro avg       0.99      0.99      0.99     14000
weighted avg       0.99      0.99      0.99     14000
```

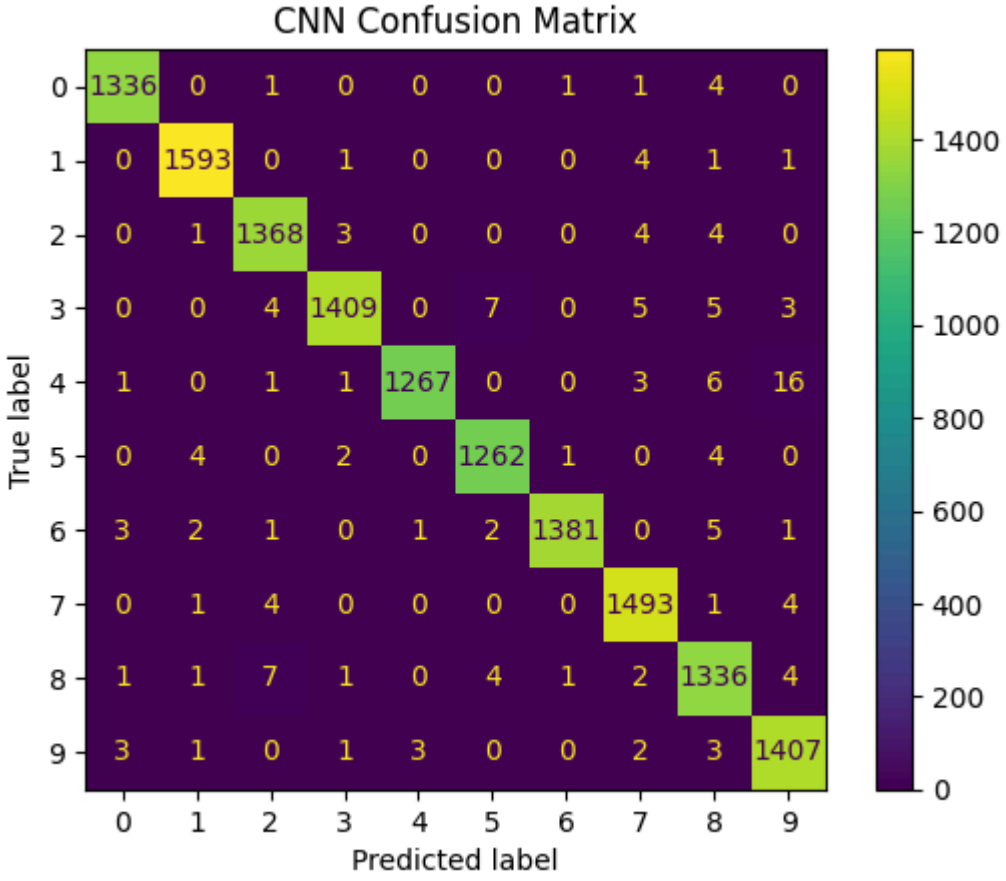
```
#confusionmatrix for MLP and CNN model
def plot_confusion_matrix(model, X_test, y_test, model_name):
    model.eval()
    with torch.no_grad():
        outputs = model(torch.tensor(X_test, dtype=torch.float32))
        _, predicted = torch.max(outputs, 1)
        cm = confusion_matrix(y_test, predicted.numpy())
        disp = ConfusionMatrixDisplay(confusion_matrix=cm)
        disp.plot()
        plt.title(f'{model_name} Confusion Matrix')
        plt.show()

# Plot confusion matrices
```

```
print("confusionmatrix for MLP model")
plot_confusion_matrix(mlp_model, X_test, y_test, 'MLP')
print("confusionmatrix for CNN model")
plot_confusion_matrix(cnn_model, X_test.reshape(-1, 1, 28, 28), y_test, 'C
```

```
confusionmatrix for MLP model
```

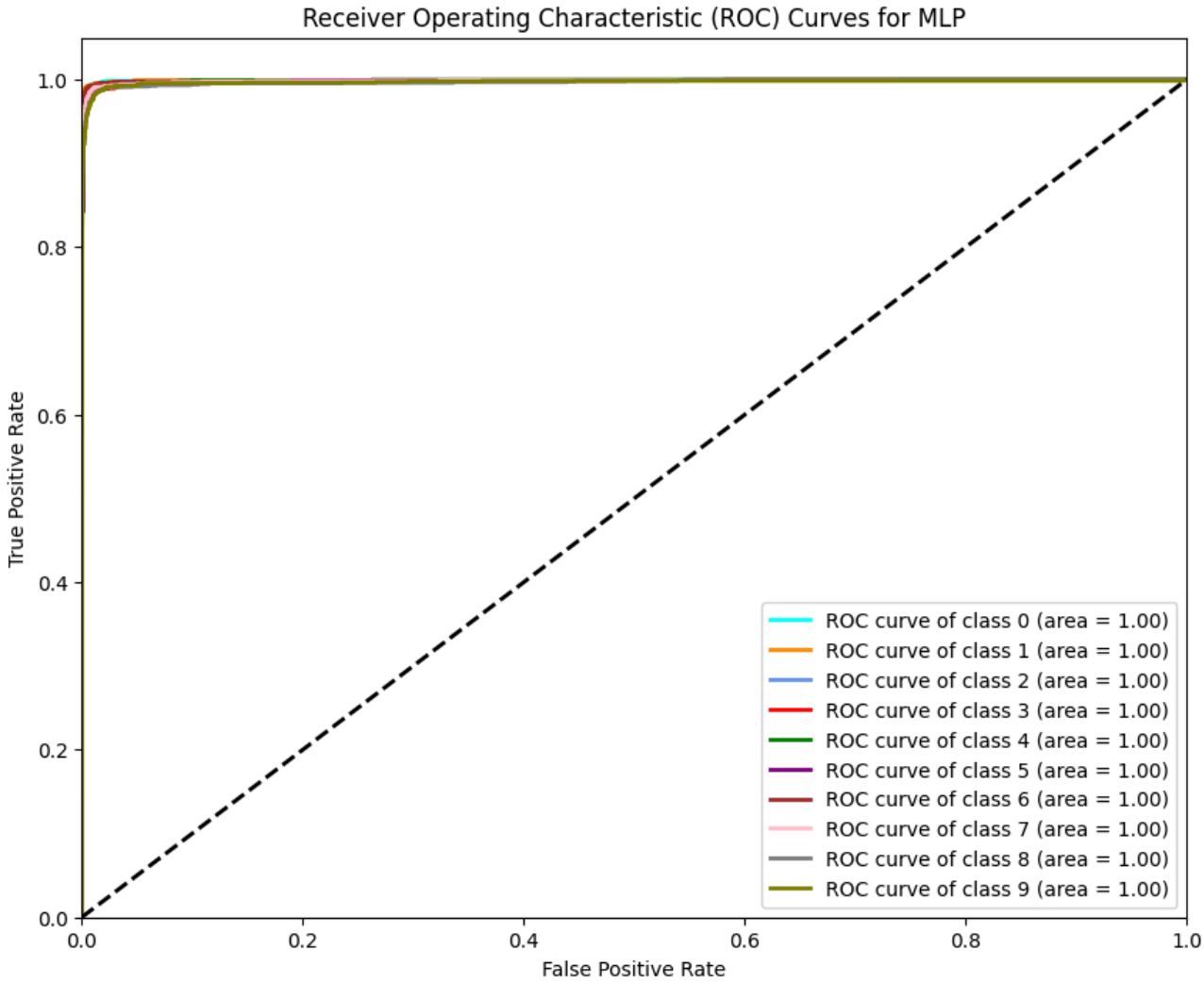



```
confusionmatrix for CNN model
```

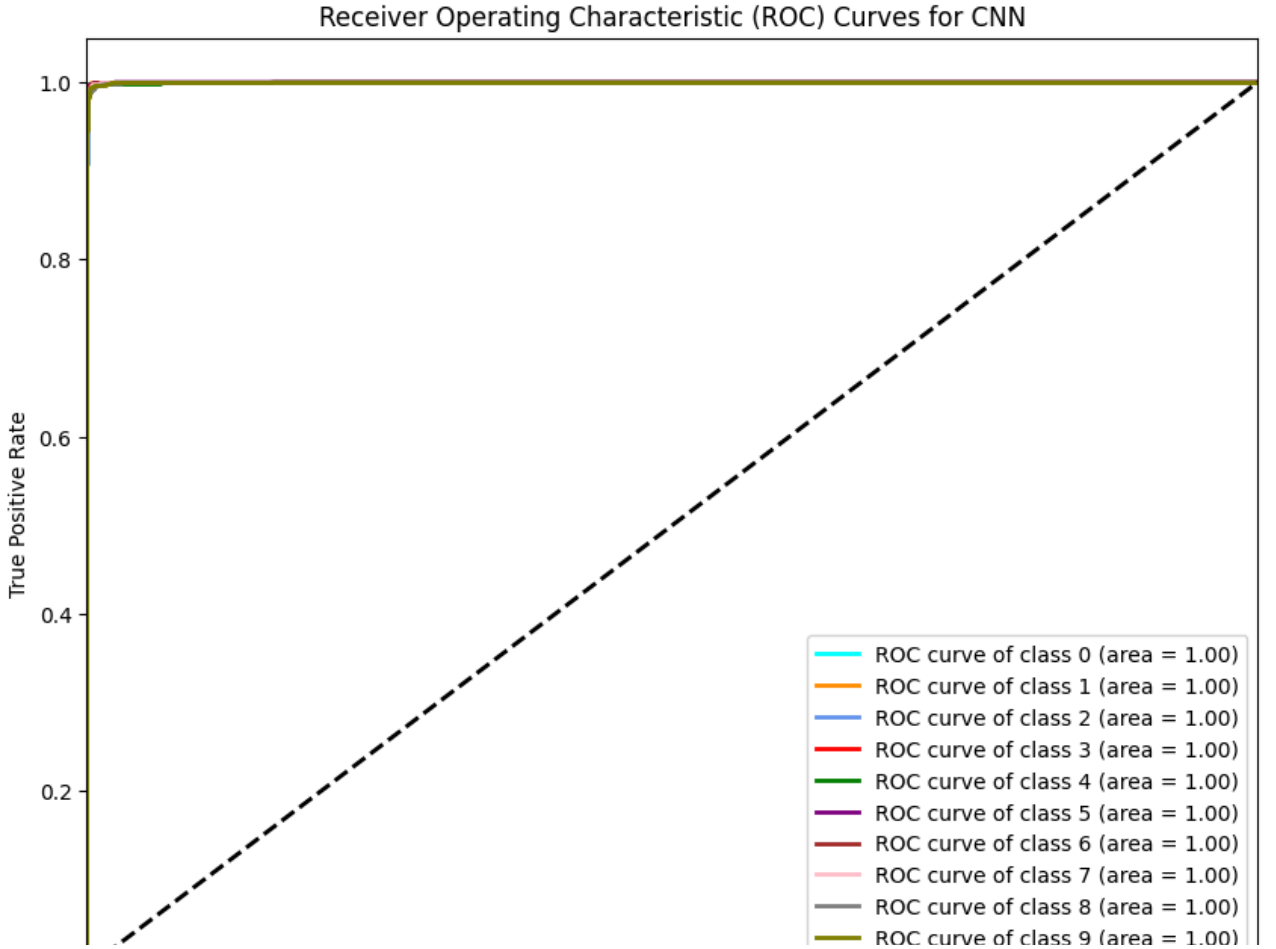


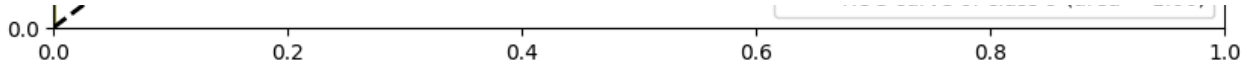
```
#ROC AUC curve for MLP and CNN models
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
from itertools import cycle

def plot_roc_auc(model, X_test, y_test, model_name):
    model.eval()
    with torch.no_grad():
        outputs = model(torch.tensor(X_test, dtype=torch.float32))
        probabilities = torch.softmax(outputs, dim=1).numpy()

    y_test_binarized = label_binarize(y_test, classes=np.arange(10))

    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(10):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], probabilitie
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure(figsize=(10, 8))
    colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'red', 'green'
    for i, color in zip(range(10), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label=f'ROC curve of class {i} (area = {roc_auc[i]:0.2f})

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver Operating Characteristic (ROC) Curves for {model_
    plt.legend(loc='lower right')
    plt.show()

print("ROC AUC curve for MLP")
plot_roc_auc(mlp_model, X_test, y_test_np, "MLP")

print("ROC AUC curve for CNN")
plot_roc_auc(cnn_model, X_test.reshape(-1, 1, 28, 28), y_test_np, "CNN")
```

ROC AUC curve for MLP



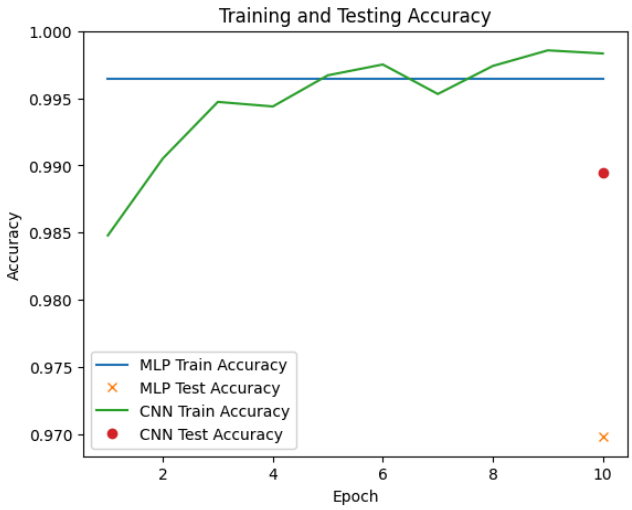
ROC AUC curve for CNN

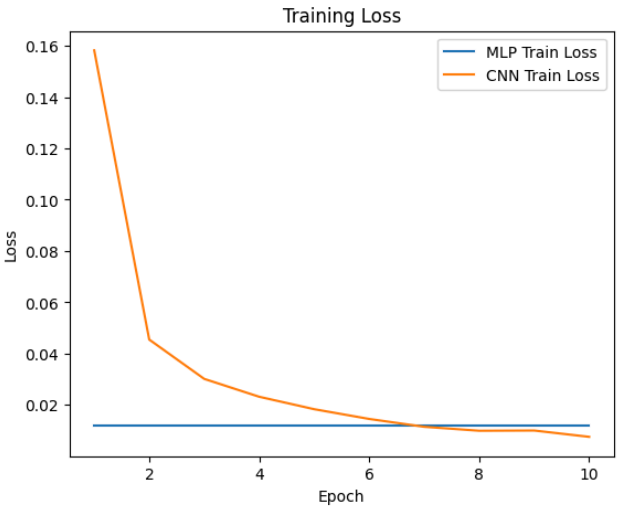




Start coding or generate with AI.

```
#Training and Test Accuracy and Loss comparison
mlp_train_losses_filename = 'mlp_train_losses.npy'
mlp_train_accuracies_filename = 'mlp_train_accuracies.npy'
cnn_train_losses_filename = 'cnn_train_losses.npy'
cnn_train_accuracies_filename = 'cnn_train_accuracies.npy'

mlp_train_losses_path = os.path.join(drive_dir, mlp_train_losses_filename)
mlp_train_accuracies_path = os.path.join(drive_dir, mlp_train_accuracies_f
cnn_train_losses_path = os.path.join(drive_dir, cnn_train_losses_filename)
cnn_train_accuracies_path = os.path.join(drive_dir, cnn_train_accuracies_f

mlp_train_losses = np.load(mlp_train_losses_path)
mlp_train_accuracies = np.load(mlp_train_accuracies_path)

cnn_train_losses = np.load(cnn_train_losses_path)
cnn_train_accuracies = np.load(cnn_train_accuracies_path)

mlp_test_accuracies = evaluate_model(mlp_model, test_loader)
cnn_test_accuracies = evaluate_model(cnn_model, test_loader)

plt.figure(figsize=(14, 5))

num_epochs = 10

plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs + 1), mlp_train_accuracies, label='MLP Train
plt.plot([num_epochs], [mlp_test_accuracies], 'x', label='MLP Test Accurac
plt.plot(range(1, num_epochs + 1), cnn_train_accuracies, label='CNN Train
plt.plot([num_epochs], [cnn_test_accuracies], 'o', label='CNN Test Accurac
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs + 1), mlp_train_losses, label='MLP Train Loss
plt.plot(range(1, num_epochs + 1), cnn_train_losses, label='CNN Train Loss
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.legend()

plt.show()
```

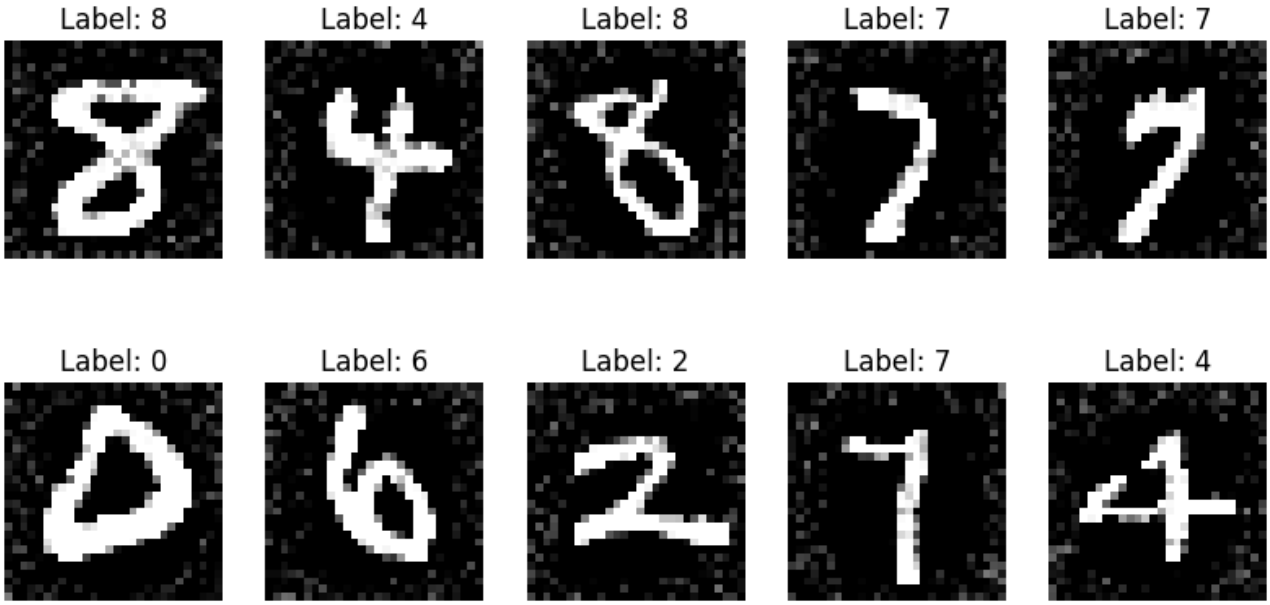




Start coding or generate with AI.

```
#Adding random noise to images and evaluating the trained models on noisy
import numpy as np

def add_noise(images, noise_factor=0.2):
    noisy_images = images + noise_factor * np.random.randn(*images.shape)
    noisy_images = np.clip(noisy_images, 0., 1.)
    return noisy_images

X_test_noisy = add_noise(X_test)
X_test_noisy_reshaped = X_test_noisy.reshape(-1, 1, 28, 28)

def evaluate_noisy_model(model, X_test_noisy, y_test, model_name):
    model.eval()
    with torch.no_grad():
        outputs = model(torch.tensor(X_test_noisy, dtype=torch.float32))
        _, predicted = torch.max(outputs, 1)
        accuracy = (predicted.numpy() == y_test).mean()
    print(f"{model_name} Accuracy on Noisy Test Images: {accuracy:.4f}")

evaluate_noisy_model(mlp_model, X_test_noisy, y_test_np, "MLP")

evaluate_noisy_model(cnn_model, X_test_noisy_reshaped, y_test_np, "CNN")

plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test_noisy[i].reshape(28, 28), cmap='gray')
    plt.title(f"Label: {y_test_np[i]}")
    plt.axis('off')
plt.show()
```

```
MLP Accuracy on Noisy Test Images: 0.9204
CNN Accuracy on Noisy Test Images: 0.8739
```

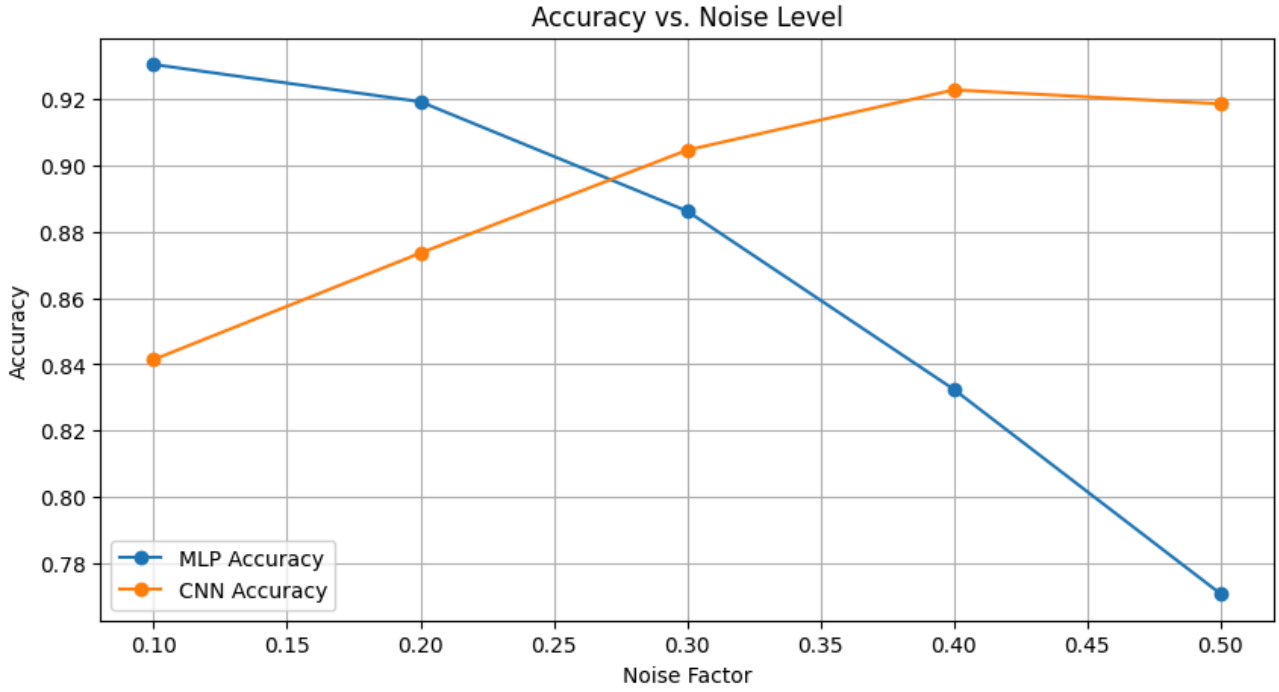


Start coding or generate with AI.

```
#adding noise to images with a specific noise factor and evaluating the tr
import torch
import matplotlib.pyplot as plt
import numpy as np

def add_noise(images, noise_factor):
    noisy_images = images + noise_factor * np.random.randn(*images.shape)
    noisy_images = np.clip(noisy_images, 0., 1.)
    return noisy_images

noise_levels = [0.1, 0.2, 0.3, 0.4, 0.5]

mlp_accuracies = []
cnn_accuracies = []

def evaluate_noisy_model(model, X_test_noisy, y_test, model_name):
    model.eval()
    with torch.no_grad():
        outputs = model(torch.tensor(X_test_noisy, dtype=torch.float32))
        _, predicted = torch.max(outputs, 1)
        accuracy = (predicted.numpy() == y_test).mean()
    print(f"{model_name} Accuracy on Noisy Test Images: {accuracy:.4f}")
    return accuracy

for noise_factor in noise_levels:
    X_test_noisy = add_noise(X_test, noise_factor)
    X_test_noisy_reshaped = X_test_noisy.reshape(-1, 1, 28, 28)

    mlp_accuracy = evaluate_noisy_model(mlp_model, X_test_noisy, y_test_np
    mlp_accuracies.append(mlp_accuracy)

    cnn_accuracy = evaluate_noisy_model(cnn_model, X_test_noisy_reshaped,
    cnn_accuracies.append(cnn_accuracy)
```

```
    cnn_accuracies.append(cnn_accuracy)

plt.figure(figsize=(10, 5))
plt.plot(noise_levels, mlp_accuracies, label='MLP Accuracy', marker='o')
plt.plot(noise_levels, cnn_accuracies, label='CNN Accuracy', marker='o')
plt.xlabel('Noise Factor')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Noise Level')
plt.legend()
plt.grid(True)
plt.show()
```

```
MLP (Noise 0.1) Accuracy on Noisy Test Images: 0.9303
CNN (Noise 0.1) Accuracy on Noisy Test Images: 0.8414
MLP (Noise 0.2) Accuracy on Noisy Test Images: 0.9191
CNN (Noise 0.2) Accuracy on Noisy Test Images: 0.8735
MLP (Noise 0.3) Accuracy on Noisy Test Images: 0.8861
CNN (Noise 0.3) Accuracy on Noisy Test Images: 0.9045
MLP (Noise 0.4) Accuracy on Noisy Test Images: 0.8324
CNN (Noise 0.4) Accuracy on Noisy Test Images: 0.9226
MLP (Noise 0.5) Accuracy on Noisy Test Images: 0.7709
CNN (Noise 0.5) Accuracy on Noisy Test Images: 0.9184
```




```
Start coding or generate with AI.
```

```
#Testing simple mlp and cnn with augmented data and noisy images
```

```
import torch
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import torch.nn as nn

y_train_np = y_train.to_numpy()
y_test_np = y_test.to_numpy()
```

```
def add_noise(images, noise_factor):
    noisy_images = images + noise_factor * np.random.randn(*images.shape)
    noisy_images = np.clip(noisy_images, 0., 1.)
    return noisy_images

noise_factor = 0.2

class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(784, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.pool(x)
        x = x.view(-1, 64 * 7 * 7)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

mlp_model_augmented = MLP()
cnn_model_augmented = CNN()

mlp_model_augmented_filename = 'mlp_model_augmented.pth'
cnn_model_augmented_filename = 'cnn_model_augmented.pth'

mlp_model_augmented_path = os.path.join(drive_dir, mlp_model_augmented_fil
cnn_model_augmented_path = os.path.join(drive_dir, cnn_model_augmented_fil

mlp_model_augmented.load_state_dict(torch.load(mlp_model_augmented_path))
```

```
cnn_model_augmented.load_state_dict(torch.load(cnn_model_augmented_path))

def evaluate_noisy_model(model, X_test_noisy, y_test, model_name):
    model.eval()
    with torch.no_grad():
        inputs = torch.tensor(X_test_noisy, dtype=torch.float32)
        if model_name.startswith("CNN"):
            inputs = inputs.view(-1, 1, 28, 28)  # Reshape inputs for CNN
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        accuracy = (predicted.numpy() == y_test).mean()
    print(f"{model_name} Accuracy on Noisy Test Images: {accuracy:.4f}")
    return accuracy

print("Evaluating MLP trained on original train set...")
mlp_accuracy_original = evaluate_noisy_model(mlp_model_augmented, X_test,

print("Evaluating MLP trained on noisy train set...")
mlp_accuracy_noisy = evaluate_noisy_model(mlp_model_augmented, add_noise(X

print("Evaluating CNN trained on original train set...")
cnn_accuracy_original = evaluate_noisy_model(cnn_model_augmented, X_test.r

print("Evaluating CNN trained on noisy train set...")
cnn_accuracy_noisy = evaluate_noisy_model(cnn_model_augmented, add_noise(X
```

```
Evaluating MLP trained on original train set...
MLP  Accuracy on Noisy Test Images: 0.9138
Evaluating MLP trained on noisy train set...
MLP  Accuracy on Noisy Test Images: 0.9692
Evaluating CNN trained on original train set...
CNN  Accuracy on Noisy Test Images: 0.9334
Evaluating CNN trained on noisy train set...
CNN  Accuracy on Noisy Test Images: 0.9879
```

```
#Testing hyperparameter mlp and cnn with augmented data
```

```
Loader, TensorDataset
```

```
 images with a specified noise factor
r):
_factor * np.random.randn(*images.shape)
images, 0., 1.)
```

```
es with ReLU activation
```

```
=784, hidden_size=128, output_size=10):
).__init__()
t_size, hidden_size)
()
len_size, output_size)
```

```
).__init__()
 32, kernel_size=3, padding=1)
, 64, kernel_size=3, padding=1)
kernel_size=2, stride=2)
 7 * 7, 128)
 10)
()
```

```
')
```

```
in(drive_dir, 'best_mlp_params.json')
in(drive_dir, 'best_cnn_params.json')

r') as f:
)

r') as f:
)

ve_dir, 'best_mlp_model.pth')
ve_dir, 'best_cnn_model.pth')

{k: v for k, v in best_mlp_params.items() if k != 'lr'})
rch.load(mlp_model_path))
```

```
rch.load(cnn_model_path))

 and return accuracy
lel, data_loader, model_name):



a_loader:
with("CNN"):
iew(-1, 1, 28, 28)  # Reshape inputs for CNN
s)
 max(outputs, 1)
 0)
 == labels).sum().item()



l_hyperparam(best_mlp_model, DataLoader(TensorDataset(torch.tensor(X_test
mlp_test_accuracy:.4f}")

l_hyperparam(best_cnn_model, DataLoader(TensorDataset(torch.tensor(X_test
cnn_test_accuracy:.4f}")
```

```
Best MLP Test Accuracy: 0.9174
Best CNN Test Accuracy: 0.9324
```

+ Code + Text

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.