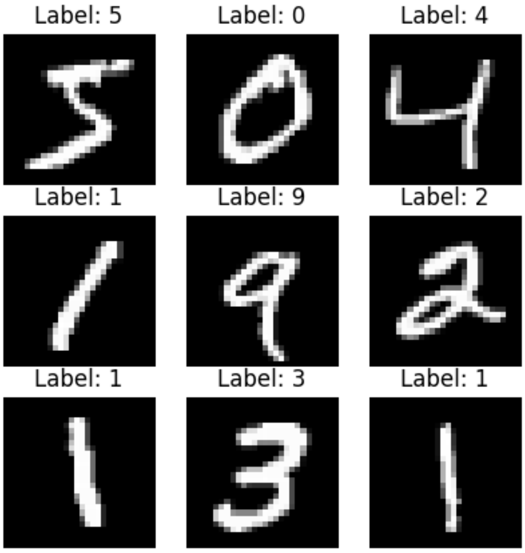
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
```

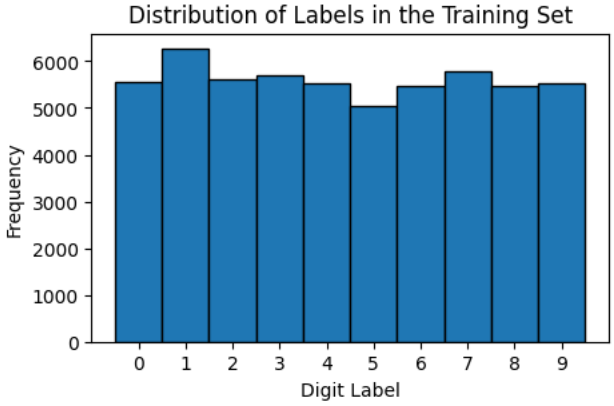
```
#load the MNIST data set
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data.to_numpy(), mnist.target.astype(int)

print(f"Shape of y: {y.shape}")

plt.figure(figsize=(5, 5))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(X[i].reshape(28, 28), cmap='gray')
    plt.title(f"Label: {y[i]}")
    plt.axis('off')
plt.show()
```

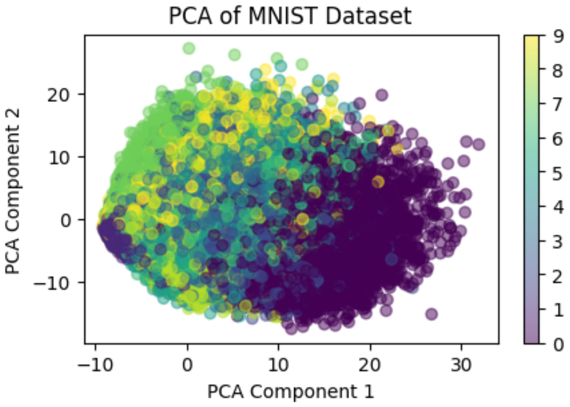
```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:1022: Fut
  warn(
Shape of y: (70000,)
```



```
#NApplying normalisation and standardization on the dataset
X = X / 255.0

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

```
plt.figure(figsize=(5, 3))
plt.hist(y_train, bins=np.arange(11) - 0.5, edgecolor='black')
plt.xticks(np.arange(10))
plt.xlabel('Digit Label')
plt.ylabel('Frequency')
plt.title('Distribution of Labels in the Training Set')
plt.show()

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)

plt.figure(figsize=(5,3))
scatter = plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap='viridis', alpha=0.5)
plt.colorbar(scatter, ticks=np.arange(10))
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.title('PCA of MNIST Dataset')
plt.show()
```





```
#defining the MLP structure
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

batch_size = 64
y_train_np = y_train.to_numpy()
train_dataset = TensorDataset(torch.tensor(X_train, dtype=torch.float32), torch.tensor(y_train_np, dtype=torch.long))
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

mlp_model = MLP()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(mlp_model.parameters(), lr=0.001)
```

```
#trainging MLP model
num_epochs = 10
mlp_train_losses = []
for epoch in range(num_epochs):
    mlp_model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = mlp_model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    mlp_train_losses.append(running_loss / len(train_loader))
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}")

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

train_accuracy = evaluate_model(mlp_model, train_loader)

print(f"training accuracy of the MLP model is: {train_accuracy:.4f}")
```

```
Epoch [1/10], Loss: 0.2380
Epoch [2/10], Loss: 0.1000
Epoch [3/10], Loss: 0.0644
Epoch [4/10], Loss: 0.0461
Epoch [5/10], Loss: 0.0319
Epoch [6/10], Loss: 0.0236
Epoch [7/10], Loss: 0.0178
Epoch [8/10], Loss: 0.0155
Epoch [9/10], Loss: 0.0152
Epoch [10/10], Loss: 0.0130
training accuracy of the MLP model is: 0.9964
```

```
#saving the MLP model
import os
drive_dir = '/content/drive/MyDrive/Colab Notebooks/harshith_neco'
model_filename = 'mlp_model.pth'
model_save_path = os.path.join(drive_dir, model_filename)

torch.save(mlp_model.state_dict(), model_save_path)
print(f"Model saved to {model_save_path}")
```

```
Model saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/mlp_model.pth
```

```
#defining the cnn model
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
```

```
#dataloaders for CNN
train_dataset = TensorDataset(torch.tensor(X_train.reshape(-1, 1, 28, 28), dtype=torch.float32), torch.tensor(y_train_np, dt
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

cnn_model = CNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(cnn_model.parameters(), lr=0.001)


#training the cnn model
cnn_train_losses = []
for epoch in range(num_epochs):
    cnn_model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = cnn_model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    cnn_train_losses.append(running_loss / len(train_loader))
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}")

train_accuracy = evaluate_model(cnn_model, train_loader)
print(f"CNN Training Accuracy: {train_accuracy:.4f}")
```

```
Epoch [1/10], Loss: 0.1513
Epoch [2/10], Loss: 0.0450
Epoch [3/10], Loss: 0.0306
Epoch [4/10], Loss: 0.0221
Epoch [5/10], Loss: 0.0200
Epoch [6/10], Loss: 0.0168
Epoch [7/10], Loss: 0.0126
Epoch [8/10], Loss: 0.0101
Epoch [9/10], Loss: 0.0077
Epoch [10/10], Loss: 0.0083
CNN Training Accuracy: 0.9990
```

```
#saving the cnn model
cnn_model_filename = 'cnn_model.pth'
cnn_model_save_path = os.path.join(drive_dir, cnn_model_filename)
torch.save(cnn_model.state_dict(), cnn_model_save_path)
print(f"CNN Model saved to {cnn_model_save_path}")
```

```
CNN Model saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/cnn_model.pth
```

```
#calculating and storing the accuracy and loss parameters of the trained models
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay


# Prepare MLP data loader (input is flat, 1D)
mlp_train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Prepare CNN data loader (input is 2D, with shape [batch_size, 1, 28, 28])
cnn_train_dataset = TensorDataset(torch.tensor(X_train.reshape(-1, 1, 28, 28), dtype=torch.float32), torch.tensor(y_train_np
cnn_train_loader = DataLoader(cnn_train_dataset, batch_size=batch_size, shuffle=True)

# Instantiate the CNN model, define the loss function and the optimizer
cnn_model = CNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(cnn_model.parameters(), lr=0.001)

# Modified CNN training loop to store accuracy and loss
cnn_train_losses = []
cnn_train_accuracies = []

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

mlp_train_losses = []
mlp_train_accuracies = []

num_epochs = 10
for epoch in range(num_epochs):
    mlp_model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = mlp_model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    mlp_train_losses.append(running_loss / len(train_loader))
    mlp_train_accuracies.append(evaluate_model(mlp_model, train_loader))


cnn_train_losses = []
cnn_train_accuracies = []

for epoch in range(num_epochs):
    cnn_model.train()
    running_loss = 0.0
    for inputs, labels in cnn_train_loader:  # Use cnn_train_loader here
        optimizer.zero_grad()
        outputs = cnn_model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    cnn_train_losses.append(running_loss / len(cnn_train_loader))
    cnn_train_accuracies.append(evaluate_model(cnn_model, cnn_train_loader))
```

```
mlp_losses_path = os.path.join(drive_dir, 'mlp_train_losses.npy')
mlp_accuracies_path = os.path.join(drive_dir, 'mlp_train_accuracies.npy')
cnn_losses_path = os.path.join(drive_dir, 'cnn_train_losses.npy')
cnn_accuracies_path = os.path.join(drive_dir, 'cnn_train_accuracies.npy')

np.save(mlp_losses_path, np.array(mlp_train_losses))
np.save(mlp_accuracies_path, np.array(mlp_train_accuracies))
np.save(cnn_losses_path, np.array(cnn_train_losses))
np.save(cnn_accuracies_path, np.array(cnn_train_accuracies))

print(f"MLP losses saved to {mlp_losses_path}")
print(f"MLP accuracies saved to {mlp_accuracies_path}")
print(f"CNN losses saved to {cnn_losses_path}")
print(f"CNN accuracies saved to {cnn_accuracies_path}")
```

```
MLP losses saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/mlp_train_losses.npy
MLP accuracies saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/mlp_train_accuracies.npy
CNN losses saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/cnn_train_losses.npy
CNN accuracies saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/cnn_train_accuracies.npy
```

```
#Training simple mlp and cnn with augmented data
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import numpy as np

#this will add random noise to the dataset
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


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_state=42)
y_train_np = y_train.to_numpy()
y_test_np = y_test.to_numpy()

noise_factor = 0.2
X_train_noisy = add_noise(X_train, noise_factor)

X_train_augmented = np.concatenate((X_train, X_train_noisy), axis=0)
y_train_augmented = np.concatenate((y_train_np, y_train_np), axis=0)

batch_size = 64
train_dataset_augmented = TensorDataset(torch.tensor(X_train_augmented, dtype=torch.float32), torch.tensor(y_train_augmentec
train_loader_augmented = DataLoader(train_dataset_augmented, batch_size=batch_size, shuffle=True)

#Redefining the MLP and CNN models
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()

criterion = nn.CrossEntropyLoss()
mlp_optimizer = optim.Adam(mlp_model_augmented.parameters(), lr=0.001)
cnn_optimizer = optim.Adam(cnn_model_augmented.parameters(), lr=0.001)

#training the mlp model
def train_model_mlp(model, optimizer, train_loader, num_epochs):
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}")

#training the cnn model
def train_model_cnn(model, optimizer, train_loader, num_epochs):
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            inputs = inputs.view(-1, 1, 28, 28)  # Reshape inputs for CNN
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}")

#mlp model training with augmented data
print("mlp model training with augmented data...")
train_model_mlp(mlp_model_augmented, mlp_optimizer, train_loader_augmented, num_epochs=10)

#CNN model training with augmented data
print("CNN model training with augmented data...")
train_model_cnn(cnn_model_augmented, cnn_optimizer, train_loader_augmented, num_epochs=10)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:1022: FutureWarning: The default value of `parser` w
  warn(
mlp model training with augmented data...
Epoch [1/10], Loss: 0.2910
Epoch [2/10], Loss: 0.1165
Epoch [3/10], Loss: 0.0715
Epoch [4/10], Loss: 0.0476
Epoch [5/10], Loss: 0.0335
Epoch [6/10], Loss: 0.0242
Epoch [7/10], Loss: 0.0177
Epoch [8/10], Loss: 0.0141
Epoch [9/10], Loss: 0.0111
Epoch [10/10], Loss: 0.0089
CNN model training with augmented data...
Epoch [1/10], Loss: 0.1284
Epoch [2/10], Loss: 0.0343
Epoch [3/10], Loss: 0.0210
Epoch [4/10], Loss: 0.0147
Epoch [5/10], Loss: 0.0097
Epoch [6/10], Loss: 0.0076
Epoch [7/10], Loss: 0.0059
Epoch [8/10], Loss: 0.0053
Epoch [9/10], Loss: 0.0049
Epoch [10/10], Loss: 0.0034
```

```
mlp_augmented_save_path = os.path.join(drive_dir, 'mlp_model_augmented.pth')
cnn_augmented_save_path = os.path.join(drive_dir, 'cnn_model_augmented.pth')

torch.save(mlp_model_augmented.state_dict(), mlp_augmented_save_path)
torch.save(cnn_model_augmented.state_dict(), cnn_augmented_save_path)

print(f"MLP Augmented Model saved to {mlp_augmented_save_path}")
print(f"CNN Augmented Model saved to {cnn_augmented_save_path}")
```

```
MLP Augmented Model saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/mlp_model_augmented.pth
CNN Augmented Model saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/cnn_model_augmented.pth
```

Start coding or generate with AI.

```
#Training hyperparameter mlp and cnn without augmented data


import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
import numpy as np
from sklearn.model_selection import ParameterGrid
import json

#this will add random noise to images with a specified noise factor
```

```
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

#the noise factor
noise_factor = 0.2
X_train_noisy = add_noise(X_train, noise_factor)

X_train_augmented = np.concatenate((X_train, X_train_noisy), axis=0)
y_train_augmented = np.concatenate((y_train_np, y_train_np), axis=0)

batch_size = 64
train_dataset_augmented = TensorDataset(torch.tensor(X_train_augmented, dtype=torch.float32), torch.tensor(y_train_augmented,

train_size = int(0.8 * len(train_dataset_augmented))
val_size = len(train_dataset_augmented) - train_size
train_dataset, val_dataset = random_split(train_dataset_augmented, [train_size, val_size])
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
train_loader_augmented = DataLoader(train_dataset_augmented, batch_size=batch_size, shuffle=True)

#redefining the ml and cnn with ReLU activation
class MLPHyperparam(nn.Module):
    def __init__(self, input_size=784, hidden_size=128, output_size=10):
        super(MLPHyperparam, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.activation = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

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

class CNNHyperparam(nn.Module):
    def __init__(self):
        super(CNNHyperparam, 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.activation = nn.ReLU()

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

#hyperparameters
mlp_param_grid = {
    'hidden_size': [64, 128, 256]
}

cnn_param_grid = {}

learning_rates = [0.001, 0.01, 0.1]

#training the models
def train_model_mlp_hyperparam(model, optimizer, train_loader, num_epochs=10):
    model.train()
    criterion = nn.CrossEntropyLoss()
    for epoch in range(num_epochs):
        running_loss = 0.0
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
    return model
    return model

def train_model_cnn_hyperparam(model, optimizer, train_loader, num_epochs=10):
    model.train()
    criterion = nn.CrossEntropyLoss()
    for epoch in range(num_epochs):
        running_loss = 0.0
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            inputs = inputs.view(-1, 1, 28, 28)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
    return model

#evaluating the accuracies
def evaluate_model_hyperparam(model, data_loader, model_name):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in data_loader:
            if model_name.startswith("CNN"):
                inputs = inputs.view(-1, 1, 28, 28)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = correct / total
    return accuracy

#this will perform hyperparameter tuning
def hyperparameter_tuning(model_class, param_grid, learning_rates, train_loader, val_loader, num_epochs=10):
    best_params = None
    best_score = 0
    for param_comb in ParameterGrid(param_grid):
        for lr in learning_rates:
            model = model_class(**param_comb)
            optimizer = optim.Adam(model.parameters(), lr=lr)

            if model_class == MLPHyperparam:
                model = train_model_mlp_hyperparam(model, optimizer, train_loader, num_epochs)
            else:
                model = train_model_cnn_hyperparam(model, optimizer, train_loader, num_epochs)

            val_accuracy = evaluate_model_hyperparam(model, val_loader, model_class.__name__)
            print(f"Parameters: {param_comb}, lr: {lr}, Validation Accuracy: {val_accuracy:.4f}")

            if val_accuracy > best_score:
                best_score = val_accuracy
                best_params = {**param_comb, 'lr': lr}

    return best_params, best_score

print("Tuning MLP...")
best_mlp_params, best_mlp_val_score = hyperparameter_tuning(MLPHyperparam, mlp_param_grid, learning_rates, train_loader, val_
print(f"Best MLP params: {best_mlp_params}, Validation Accuracy: {best_mlp_val_score:.4f}")

print("Tuning CNN...")
best_cnn_params, best_cnn_val_score = hyperparameter_tuning(CNNHyperparam, cnn_param_grid, learning_rates, train_loader, val_
print(f"Best CNN params: {best_cnn_params}, Validation Accuracy: {best_cnn_val_score:.4f}")

#saving the best hyperparameters
mlp_params_file = os.path.join(drive_dir, 'best_mlp_params.json')
cnn_params_file = os.path.join(drive_dir, 'best_cnn_params.json')

with open(mlp_params_file, 'w') as f:
    json.dump(best_mlp_params, f)
print(f"Best MLP hyperparameters saved to {mlp_params_file}")

with open(cnn_params_file, 'w') as f:
    json.dump(best_cnn_params, f)
print(f"Best CNN hyperparameters saved to {cnn_params_file}")

print("best cnn hyperparameters saved")


best_mlp_model = MLPHyperparam(**{k: v for k, v in best_mlp_params.items() if k != 'lr'})
best_mlp_optimizer = optim.Adam(best_mlp_model.parameters(), lr=best_mlp_params['lr'])
best_mlp_model = train_model_mlp_hyperparam(best_mlp_model, best_mlp_optimizer, train_loader_augmented)
torch.save(best_mlp_model.state_dict(), 'best_mlp_model.pth')
```

```
best_cnn_model = CNNHyperparam()
best_cnn_optimizer = optim.Adam(best_cnn_model.parameters(), lr=best_cnn_params['lr'])
best_cnn_model = train_model_cnn_hyperparam(best_cnn_model, best_cnn_optimizer, train_loader_augmented)
torch.save(best_cnn_model.state_dict(), 'best_cnn_model.pth')
```

```
Tuning MLP...
Parameters: {'hidden_size': 64}, lr: 0.001, Validation Accuracy: 0.9730
Parameters: {'hidden_size': 64}, lr: 0.01, Validation Accuracy: 0.9650
Parameters: {'hidden_size': 64}, lr: 0.1, Validation Accuracy: 0.4897
Parameters: {'hidden_size': 128}, lr: 0.001, Validation Accuracy: 0.9783
Parameters: {'hidden_size': 128}, lr: 0.01, Validation Accuracy: 0.9674
Parameters: {'hidden_size': 128}, lr: 0.1, Validation Accuracy: 0.5768
Parameters: {'hidden_size': 256}, lr: 0.001, Validation Accuracy: 0.9803
Parameters: {'hidden_size': 256}, lr: 0.01, Validation Accuracy: 0.9674
Parameters: {'hidden_size': 256}, lr: 0.1, Validation Accuracy: 0.5751
Best MLP params: {'hidden_size': 256, 'lr': 0.001}, Validation Accuracy: 0.9803
Tuning CNN...
Parameters: {}, lr: 0.001, Validation Accuracy: 0.9936
Parameters: {}, lr: 0.01, Validation Accuracy: 0.9826
Parameters: {}, lr: 0.1, Validation Accuracy: 0.1020
Best CNN params: {'lr': 0.001}, Validation Accuracy: 0.9936
Best MLP hyperparameters saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/best_mlp_params.json
Best CNN hyperparameters saved to /content/drive/MyDrive/Colab Notebooks/harshith_neco/best_cnn_params.json
best cnn hyperparameters saved
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

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.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.