Name: Atharva Admile

Batch no: 06

Run all cells in notebook

1. Image Classification Project

• Objective:

Build an image classification model using traditional ML and deep learning techniques.

• Dataset:

MNIST Digit Dataset

• Tasks:

Perform image preprocessing (resize, normalization)

Split dataset into train and test sets

Train at least 2 models (e.g., CNN and Logistic Regression)

Compare model performance

• Evaluation Metrics:

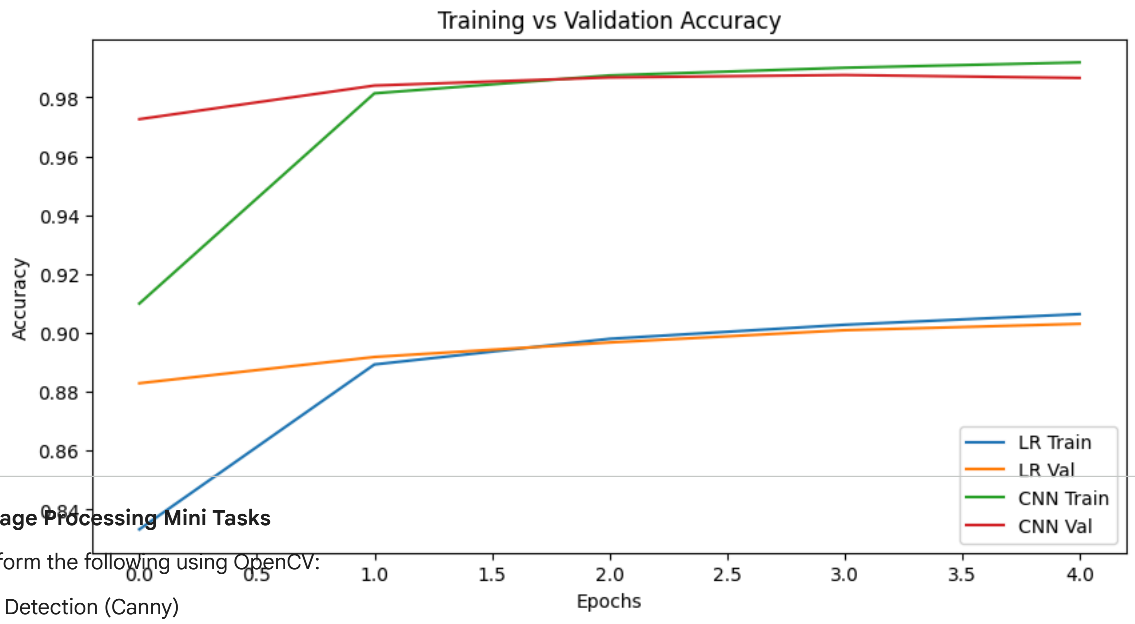
Accuracy

Confusion Matrix

Training vs Validation Accuracy Graph

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader, random_split
from sklearn.metrics import confusion_matrix, accuracy_score

# Load and preprocess MNIST
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)

# Split train into train/val
train_size = int(0.8 * len(trainset))
val_size = len(trainset) - train_size
train_dataset, val_dataset = random_split(trainset, [train_size, val_size])

trainloader = DataLoader(train_dataset, batch_size=64, shuffle=True)
valloader = DataLoader(val_dataset, batch_size=64, shuffle=False)
testloader = DataLoader(testset, batch_size=64, shuffle=False)

# Logistic Regression model
class LR(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(28*28, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        return self.linear(x)

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

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

# Training function
def train_model(model, optimizer, criterion, epochs=5):
    train_accs = []
    val_accs = []
    for epoch in rang Run all cells in notebook
        model.train()
        correct, total = 0, 0
        for inputs, labels in trainloader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        train_accs.append(correct / total)

        model.eval()
        correct, total = 0, 0
        with torch.no_grad():
            for inputs, labels in valloader:
                outputs = model(inputs)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        val_accs.append(correct / total)
        print(f'Epoch {epoch+1}: Train Acc {train_accs[-1]:.4f}, Val Acc {val_accs[-1]:.4f}')
    return train_accs, val_accs

# Train LR
lr_model = LR()
optimizer_lr = optim.SGD(lr_model.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()
print("Training LR:")
lr_train, lr_val = train_model(lr_model, optimizer_lr, criterion)

# Train CNN
cnn_model = CNN()
optimizer_cnn = optim.SGD(cnn_model.parameters(), lr=0.01, momentum=0.9)
print("\nTraining CNN:")
cnn_train, cnn_val = train_model(cnn_model, optimizer_cnn, criterion)

# Evaluate on test
def evaluate(model, name):
    model.eval()
    y_true, y_pred = [], []
    with torch.no_grad():
        for inputs, labels in testloader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            y_true.extend(labels.numpy())
            y_pred.extend(predicted.numpy())
    acc = accuracy_score(y_true, y_pred)
    cm = confusion_matrix(y_true, y_pred)
    print(f"{name} Test Accuracy: {acc:.4f}")
    print(f"{name} Confusion Matrix:\n{cm}")
    return acc, cm

lr_acc, lr_cm = evaluate(lr_model, "LR")
cnn_acc, cnn_cm = evaluate(cnn_model, "CNN")

# Plot Training vs Validation Accuracy
plt.figure(figsize=(10, 5))
plt.plot(lr_train, label='LR Train')
plt.plot(lr_val, label='LR Val')
plt.plot(cnn_train, label='CNN Train')
plt.plot(cnn_val, label='CNN Val')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training vs Validation Accuracy')
plt.show()
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 23.4MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 863kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 5.98MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 3.25MB/s]
Training LR:
Epoch 1: Train Acc 0.8333, Val Acc 0.8829
Epoch 2: Train Acc 0. Run all cells in notebook 918
Epoch 3: Train Acc 0.8980, Val Acc 0.8968
Epoch 4: Train Acc 0.9028, Val Acc 0.9009
Epoch 5: Train Acc 0.9064, Val Acc 0.9031

Training CNN:
Epoch 1: Train Acc 0.9100, Val Acc 0.9726
Epoch 2: Train Acc 0.9814, Val Acc 0.9840
Epoch 3: Train Acc 0.9874, Val Acc 0.9868
Epoch 4: Train Acc 0.9900, Val Acc 0.9876
Epoch 5: Train Acc 0.9919, Val Acc 0.9866
LR Test Accuracy: 0.9129
LR Confusion Matrix:
[[ 953    0    2    2    0    8   11    1    3    0]
 [   0 1100    2    3    0    3    4    1   22    0]
 [  11    8  897   18   14    1   15   19   42    7]
 [   4    1   16  909    1   33    2   16   19    9]
 [   1    5    5    2  906    1   11    1   11   39]
 [   8    3    3   37   10  763   16   10   35    7]
 [  10    3    4    3   10   15  905    4    4    0]
 [   2   13   22    7    8    0    0  941    3   32]
 [   6   10    7   21    7   27   11   13  861   11]
 [  11    6    3   11   37   10    0   27   10  894]]
CNN Test Accuracy: 0.9890
CNN Confusion Matrix:
[[ 977    0    0    0    0    0    0    2    1    0]
 [   0 1134    0    0    0    0    1    0    0    0]
 [   3    2 1021    0    2    0    0    4    0    0]
 [   1    0    5  988    0    7    0    4    3    2]
 [   0    0    0    0  981    0    0    0    0    1]
 [   2    0    0    3    0  883    2    0    0    2]
 [   8    2    0    0    4    5  939    0    0    0]
 [   0    4    4    0    0    0    0 1018    1    1]
 [   4    0    4    0    2    1    0    0  960    3]
 [   0    1    0    0    8    2    0    8    1  989]]
```



2. Image Processing Mini Tasks

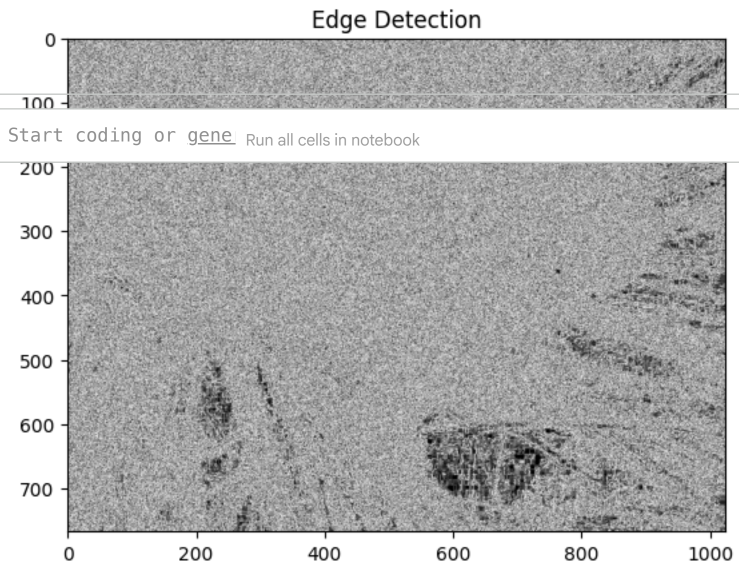
• Perform the following using OpenCV:

Edge Detection (Canny)

Image Thresholding

Image Augmentation (flip, rotate, brightness adjustment)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage
from scipy.datasets import face

# Load sample image (RGB) and convert to grayscale
rgb_image = face()
image = np.mean(rgb_image, axis=-1).astype(np.uint8)

# Edge Detection (approx Canny with Sobel)
edges_x = ndimage.sobel(image, axis=0)
edges_y = ndimage.sobel(image, axis=1)
edges = np.sqrt(edges_x**2 + edges_y**2)
plt.figure()
plt.imshow(edges, cmap='gray')
plt.title('Edge Detection')
plt.show()

# Image Thresholding (binary)
threshold = 128
thresh_image = np.where(image > threshold, 255, 0).astype(np.uint8)
plt.figure()
plt.imshow(thresh_image, cmap='gray')
plt.title('Thresholding')
plt.show()

# Image Augmentation
# Flip (horizontal)
flipped = np.fliplr(image)

# Rotate (90 degrees)
rotated = ndimage.rotate(image, 90)

# Brightness adjustment (+50)
bright = np.clip(image.astype(int) + 50, 0, 255).astype(np.uint8)

fig, axs = plt.subplots(1, 3)
axs[0].imshow(flipped, cmap='gray'); axs[0].set_title('Flip')
axs[1].imshow(rotated, cmap='gray'); axs[1].set_title('Rotate')
axs[2].imshow(bright, cmap='gray'); axs[2].set_title('Brighten')
plt.show()
```



Start coding or gene Run all cells in notebook