Name : Shreya Kakade

Roll No : TEAD22541

Assignment No : 4

from sklearn.neural\_network import MLPClassifier

import numpy as np

# Training data

digits = {

0: [[1,1,1],[1,0,1],[1,0,1],[1,0,1],[1,1,1]],

1: [[0,1,0],[1,1,0],[0,1,0],[0,1,0],[1,1,1]],

2: [[1,1,1],[0,0,1],[1,1,1],[1,0,0],[1,1,1]],

3: [[1,1,1],[0,0,1],[0,1,1],[0,0,1],[1,1,1]],

4: [[1,0,1],[1,0,1],[1,1,1],[0,0,1],[0,0,1]],

5: [[1,1,1],[1,0,0],[1,1,1],[0,0,1],[1,1,1]],

6: [[1,1,1],[1,0,0],[1,1,1],[1,0,1],[1,1,1]],

7: [[1,1,1],[0,0,1],[0,1,0],[1,0,0],[1,0,0]],

8: [[1,1,1],[1,0,1],[1,1,1],[1,0,1],[1,1,1]],

9: [[1,1,1],[1,0,1],[1,1,1],[0,0,1],[1,1,1]]

}

X = [np.array(v).flatten() for v in digits.values()]

y = list(digits.keys())

# Use better solver and higher iteration

model = MLPClassifier(hidden\_layer\_sizes=(20,), solver='lbfgs', max\_iter=2000)

model.fit(X, y)

# Prediction function

def predict\_digit(matrix):

return model.predict([np.array(matrix).flatten()])[0]

# Test

test = [[1,1,1],[0,0,1],[1,1,1],[1,0,0],[1,1,1]] # Should predict 2

print("Predicted Digit:", predict\_digit(test))

Output :



Name : Shreya Kakade

Roll No : TEAD22541

Assignment No : 7

from sklearn.neural\_network import MLPClassifier

# XOR inputs and outputs

X = [[0, 0], [0, 1], [1, 0], [1, 1]]

y = [0, 1, 1, 0]

# Create and train the neural network

model = MLPClassifier(hidden\_layer\_sizes=(2,), activation='logistic', solver='lbfgs', max\_iter=1000)

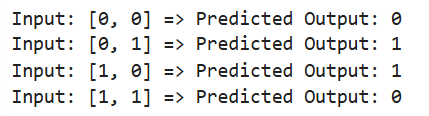
model.fit(X, y)

# Test the network

for i in X:

print(f"Input: {i} => Predicted Output: {round(model.predict([i])[0])}")

Output :



Name : Shreya Kakade

Roll No : TEAD22541

Assignment No : 8

import numpy as np

class ART\_1:

def \_\_init\_\_(self, input\_size, num\_clusters, vigilance=0.5):

self.input\_size = input\_size \* 2 # Complement coding doubles the input size

self.num\_clusters = num\_clusters

self.vigilance = vigilance

self.weights = np.ones((num\_clusters, self.input\_size)) # Adjust weight size

def complement\_code(self, input\_pattern):

return np.concatenate((input\_pattern, 1 - input\_pattern))

def match(self, input\_pattern, cluster):

return np.sum(np.minimum(input\_pattern, self.weights[cluster])) / np.sum(input\_pattern)

def update\_weights(self, cluster, input\_pattern):

self.weights[cluster] = self.weights[cluster] \* input\_pattern

def train(self, inputs):

inputs = np.array([self.complement\_code(inp) for inp in inputs]) # Apply complement coding

for input\_pattern in inputs:

assigned = False

for cluster in range(self.num\_clusters):

if self.match(input\_pattern, cluster) >= self.vigilance:

self.update\_weights(cluster, input\_pattern)

print(f"Processing input: {input\_pattern[:len(input\_pattern)//2]} → Assigned to cluster {cluster}")

assigned = True

break

if not assigned:

print(f"Processing input: {input\_pattern[:len(input\_pattern)//2]} → Assigned to cluster -1 (No match)")

def predict(self, input\_pattern):

input\_pattern = self.complement\_code(input\_pattern)

for cluster in range(self.num\_clusters):

if self.match(input\_pattern, cluster) >= self.vigilance:

return cluster

return -1 # No suitable cluster found

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

inputs = np.array([

[1, 0, 1, 0],

[1, 1, 0, 0],

[0, 1, 1, 0],

[0, 0, 1, 1]

])

art = ART\_1(input\_size=4, num\_clusters=2, vigilance=0.5)

art.train(inputs)

test\_inputs = np.array([

[1, 0, 1, 0],

[1, 1, 0, 0],

[0, 0, 0, 1],

[1, 1, 1, 0]

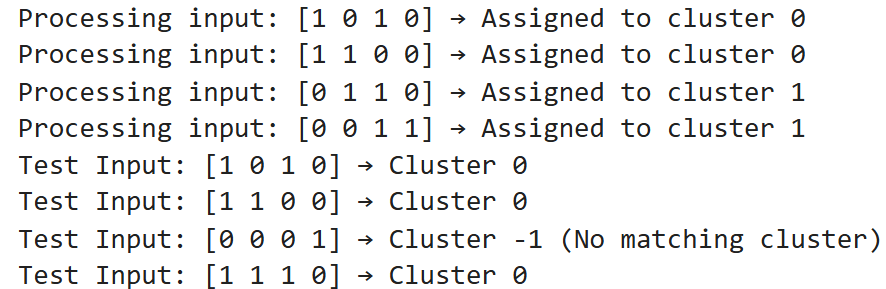
])

for test\_input in test\_inputs:

cluster = art.predict(test\_input)

print(f"Test Input: {test\_input} → Cluster {cluster if cluster != -1 else '-1 (No matching cluster)'}")

Output :



Name : Shreya Kakade

Roll No : TEAD22541

Assignment No : 9

class HopfieldNetwork:

def \_\_init\_\_(self, size):

self.size = size

self.weights = np.zeros((size, size))

def train(self, patterns):

for p in patterns:

p = p.reshape(-1, 1)

self.weights += np.dot(p, p.T)

np.fill\_diagonal(self.weights, 0) # No self-connection

def recall(self, pattern, steps=5):

for \_ in range(steps):

pattern = np.sign(np.dot(self.weights, pattern))

return pattern

# Define the stored patterns (4 vectors)

stored\_patterns = np.array([

[1, -1, 1, -1],

[-1, 1, -1, 1],

[1, 1, -1, -1],

[-1, -1, 1, 1]

])

# Initialize and train the Hopfield Network

hopfield\_net = HopfieldNetwork(size=4)

hopfield\_net.train(stored\_patterns)

# Test the network with a noisy input

noisy\_input = np.array([1, -1, -1, -1])

recalled\_pattern = hopfield\_net.recall(noisy\_input)

print("Noisy Input:", noisy\_input)

print("Recalled Pattern:", recalled\_pattern)

Output :



Name : Shreya Kakade

Roll No : TEAD22541

Assignment No : 10

import torch

import os

from PIL import Image, ImageDraw, ImageFont

import matplotlib.pyplot as plt

# Load pre-trained YOLOv5 model

model = torch.hub.load('ultralytics/yolov5', 'yolov5s') # Correct repository

model.conf = 0.25 # Set confidence threshold

def detect\_objects(image\_path):

"""Detect objects in an image using YOLOv5."""

if not os.path.exists(image\_path):

print(f"Error: Image '{image\_path}' not found.")

return None

image = Image.open(image\_path).convert("RGB") # Ensure correct format

results = model(image) # Perform inference

results.show() # Display the image with detections

detections = results.pandas().xyxy[0] # Get bounding boxes

if detections.empty:

print("No objects detected.")

return None

return detections

def plot\_detections(image\_path, detections):

"""Plot detected objects on the image."""

if detections is None:

print("No objects to plot.")

return

image = Image.open(image\_path).convert("RGB")

draw = ImageDraw.Draw(image)

font = ImageFont.load\_default()

for \_, row in detections.iterrows():

x1, y1, x2, y2 = int(row['xmin']), int(row['ymin']), int(row['xmax']), int(row['ymax'])

label = f"{row['name']} ({row['confidence']:.2f})"

draw.rectangle([x1, y1, x2, y2], outline="green", width=2)

draw.text((x1, y1 - 10), label, fill="green", font=font)

plt.imshow(image)

plt.axis("off")

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

# Updated path to the image

image\_path = r"C:\Users\kakad\OneDrive\Desktop\th (2).jpeg" # Updated image path

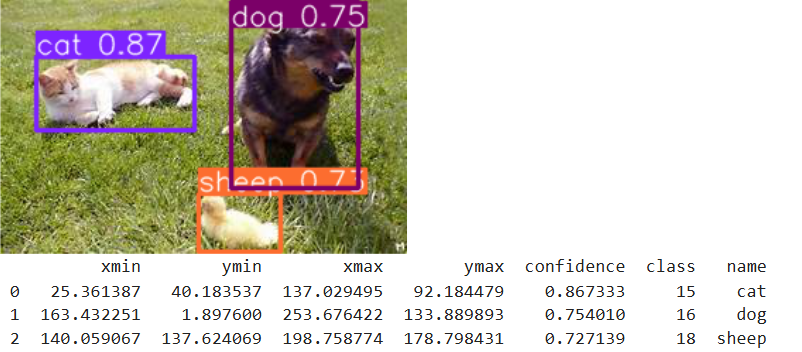
detections = detect\_objects(image\_path)

if detections is not None:

print(detections)

plot\_detections(image\_path, detections)

Output :



Name : Shreya Kakade

Roll No : TEAD22541

Assignment No : 11

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

import numpy as np

# Load CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Normalize pixel values

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Flatten labels (no one-hot needed for sparse categorical loss)

y\_train, y\_test = y\_train.flatten(), y\_test.flatten()

# Model creation function

def create\_model(dropout\_rate=0.3):

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dropout(dropout\_rate),

layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

return model

# Data augmentation

datagen = ImageDataGenerator(

rotation\_range=15,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

horizontal\_flip=True

)

datagen.fit(x\_train)

# Create and train the model

model = create\_model(dropout\_rate=0.3)

history = model.fit(datagen.flow(x\_train, y\_train, batch\_size=64),

epochs=20,

validation\_data=(x\_test, y\_test))

# Plotting function

def plot\_metrics(history, title\_suffix=''):

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

# Accuracy

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Val Accuracy')

plt.title('Accuracy' + title\_suffix)

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

# Loss

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Val Loss')

plt.title('Loss' + title\_suffix)

plt.xlabel('Epoch')

plt.ylabel('Loss')

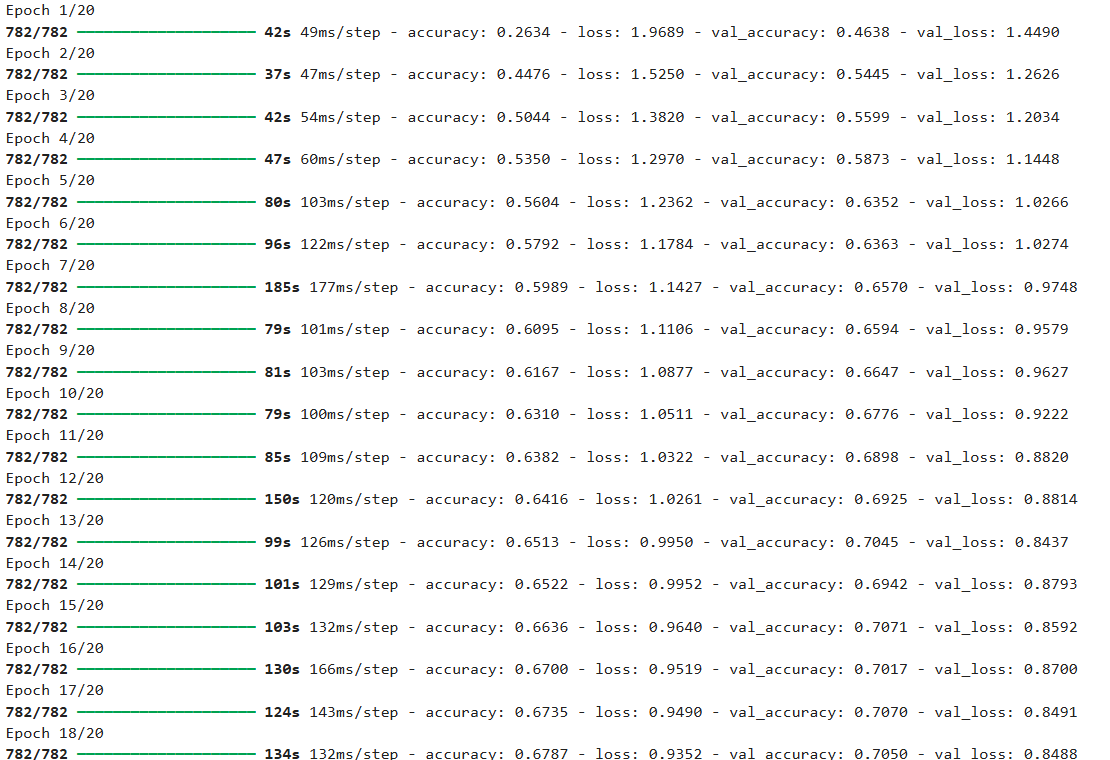
plt.legend()

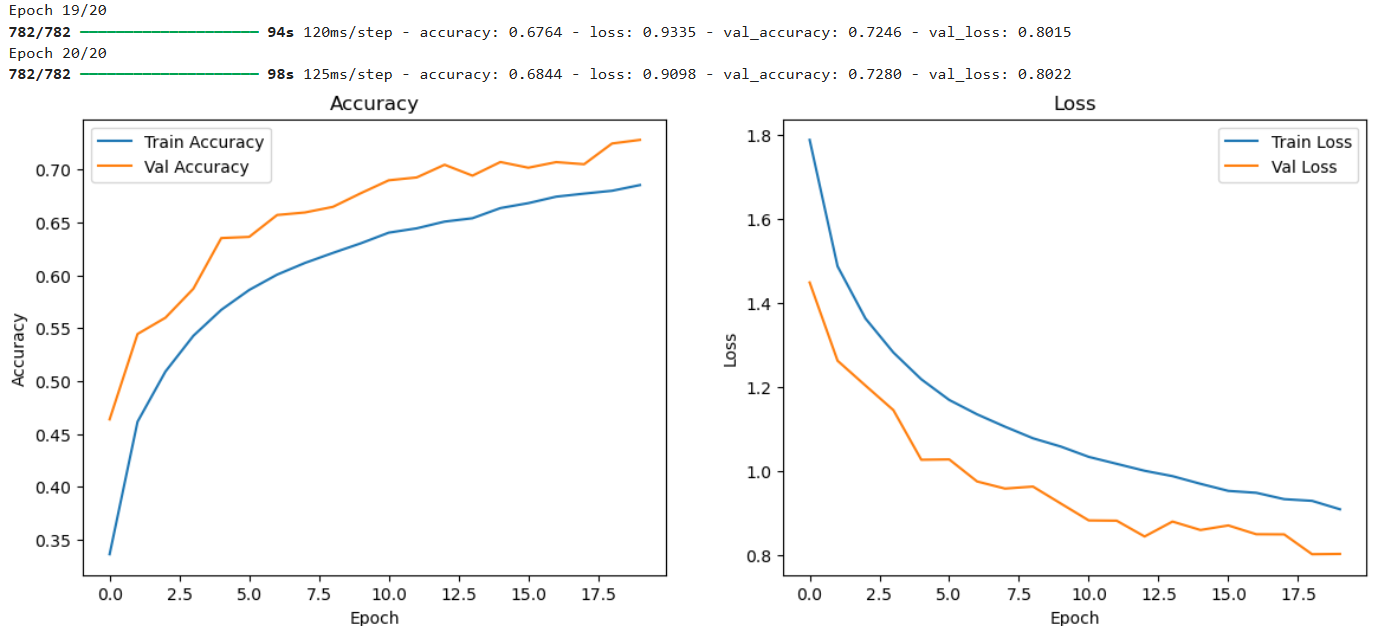
plt.show()

# Call plot function

plot\_metrics(history)

Output :





Name : Shreya Kakade

Roll No : TEAD22541

Assignment No : 12

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import seaborn as sns

import numpy as np

from PIL import Image

# Data transformations

transform\_train = transforms.Compose([

transforms.RandomHorizontalFlip(),

transforms.RandomCrop(32, padding=4),

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

transform\_test = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

# Load CIFAR-10 dataset

train\_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform\_train)

test\_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform\_test)

train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=False)

# CNN Model

class CNN(nn.Module):

def \_\_init\_\_(self):

super(CNN, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(3, 32, kernel\_size=3, padding=1)

self.conv2 = nn.Conv2d(32, 64, kernel\_size=3, padding=1)

self.pool = nn.MaxPool2d(2, 2)

self.dropout = nn.Dropout(0.3)

self.fc1 = nn.Linear(64 \* 8 \* 8, 256)

self.fc2 = nn.Linear(256, 10)

def forward(self, x):

x = self.pool(F.relu(self.conv1(x))) # 32x32 → 16x16

x = self.pool(F.relu(self.conv2(x))) # 16x16 → 8x8

x = x.view(-1, 64 \* 8 \* 8)

x = self.dropout(x)

x = F.relu(self.fc1(x))

x = self.fc2(x)

return x

# Device and setup

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model = CNN().to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

train\_losses, val\_losses, train\_acc, val\_acc = [], [], [], []

# Training and Validation Loop

for epoch in range(10):

model.train()

running\_loss, correct, total = 0.0, 0, 0

for images, labels in train\_loader:

images, labels = images.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(images)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

\_, predicted = outputs.max(1)

total += labels.size(0)

correct += predicted.eq(labels).sum().item()

train\_losses.append(running\_loss / len(train\_loader))

train\_acc.append(100. \* correct / total)

# Validation

model.eval()

val\_loss, correct, total = 0.0, 0, 0

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

loss = criterion(outputs, labels)

val\_loss += loss.item()

\_, predicted = outputs.max(1)

total += labels.size(0)

correct += predicted.eq(labels).sum().item()

val\_losses.append(val\_loss / len(test\_loader))

val\_acc.append(100. \* correct / total)

print(f"Epoch {epoch+1}: Train Loss: {train\_losses[-1]:.4f}, Train Acc: {train\_acc[-1]:.2f}% | Val Loss: {val\_losses[-1]:.4f}, Val Acc: {val\_acc[-1]:.2f}%")

# Plotting Loss and Accuracy Curves

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(train\_losses, label='Train Loss')

plt.plot(val\_losses, label='Validation Loss')

plt.title('Loss Curve')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(train\_acc, label='Train Accuracy')

plt.plot(val\_acc, label='Validation Accuracy')

plt.title('Accuracy Curve')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Final Evaluation

model.eval()

all\_preds, all\_labels = [], []

with torch.no\_grad():

for images, labels in test\_loader:

images = images.to(device)

outputs = model(images)

\_, preds = torch.max(outputs, 1)

all\_preds.extend(preds.cpu().numpy())

all\_labels.extend(labels.numpy())

acc = accuracy\_score(all\_labels, all\_preds)

print(f"\nTest Accuracy: {acc \* 100:.2f}%\n")

print("Classification Report:\n")

print(classification\_report(all\_labels, all\_preds, target\_names=test\_dataset.classes))

# Confusion Matrix

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(10, 8))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=test\_dataset.classes, yticklabels=test\_dataset.classes)

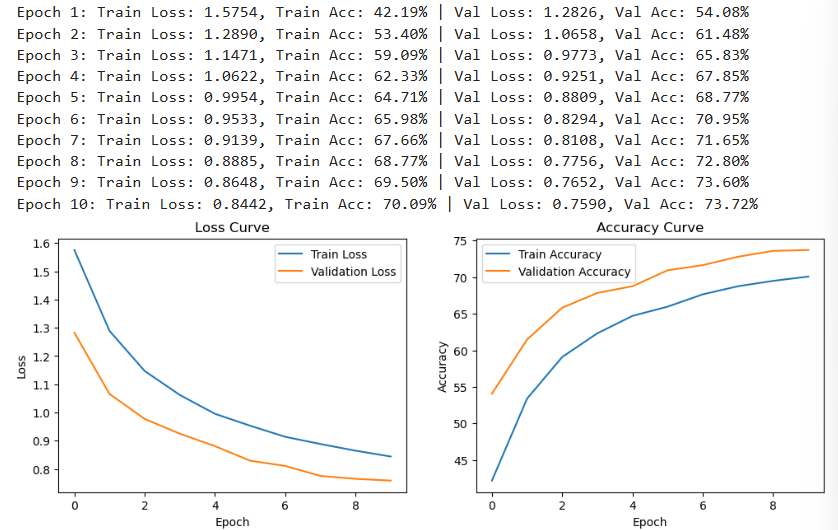
plt.xlabel('Predicted')

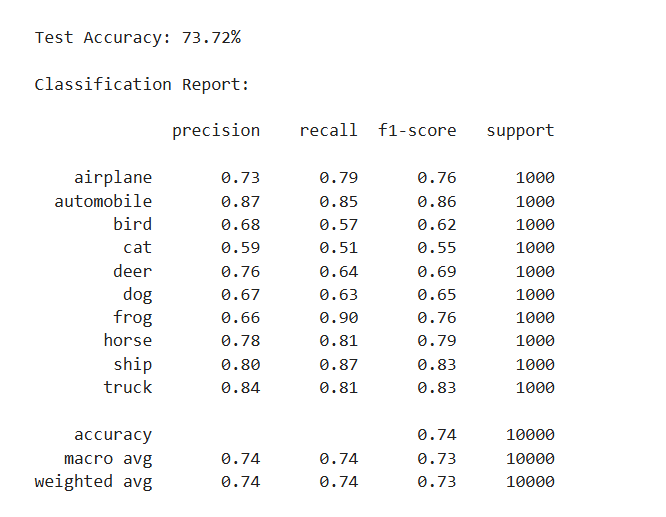
plt.ylabel('True')

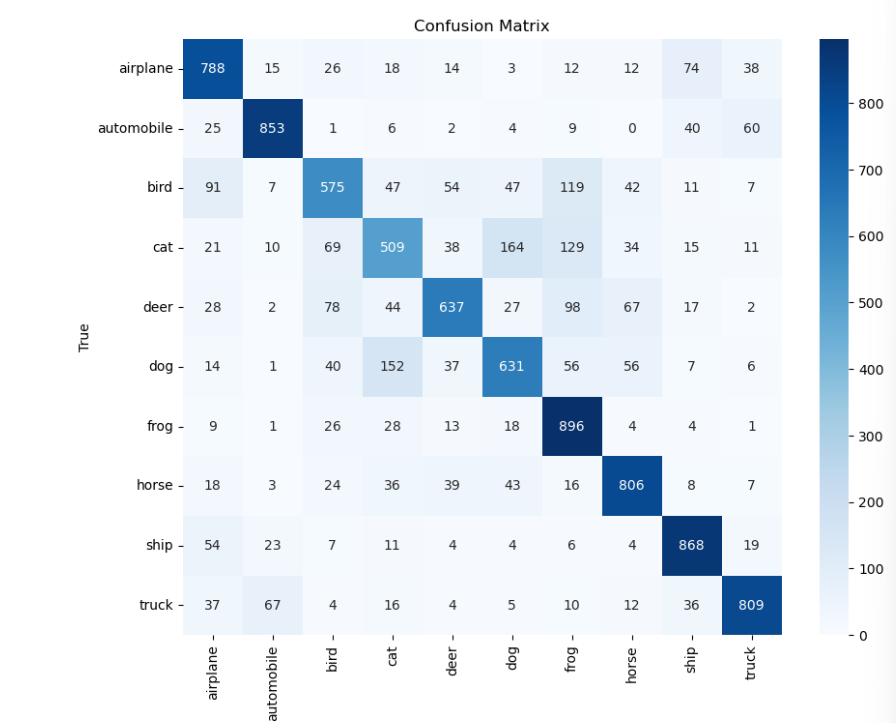
plt.title('Confusion Matrix')

plt.show()

Output :







# Show correct vs incorrect samples

correct\_samples = np.array(all\_preds) == np.array(all\_labels)

fig, axs = plt.subplots(2, 5, figsize=(12, 5))

fig.suptitle('Correctly Classified vs Misclassified Samples')

correct\_idx = np.where(correct\_samples)[0][:5]

wrong\_idx = np.where(~correct\_samples)[0][:5]

for i, idx in enumerate(correct\_idx):

img, label = test\_dataset[idx]

axs[0, i].imshow(img.permute(1, 2, 0) \* 0.5 + 0.5)

axs[0, i].set\_title(f"True: {test\_dataset.classes[all\_labels[idx]]}\nPred: {test\_dataset.classes[all\_preds[idx]]}")

axs[0, i].axis('off')

for i, idx in enumerate(wrong\_idx):

img, label = test\_dataset[idx]

axs[1, i].imshow(img.permute(1, 2, 0) \* 0.5 + 0.5)

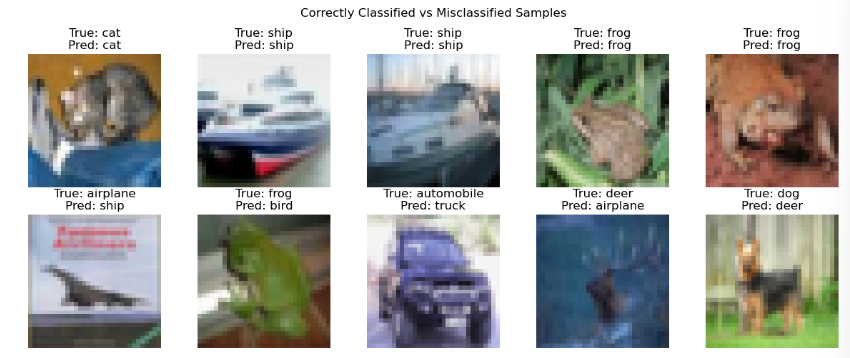
axs[1, i].set\_title(f"True: {test\_dataset.classes[all\_labels[idx]]}\nPred: {test\_dataset.classes[all\_preds[idx]]}")

axs[1, i].axis('off')

plt.tight\_layout()

plt.show()

Output :



import torch

from torchvision import transforms

from PIL import Image

import matplotlib.pyplot as plt

# Define the transform for the input image

user\_transform = transforms.Compose([

transforms.Resize((32, 32)),

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

# Function to predict the image using the trained model

def predict\_user\_image(img\_path, model, test\_dataset, device):

img = Image.open(img\_path).convert('RGB')

input\_tensor = user\_transform(img).unsqueeze(0).to(device)

model.eval()

with torch.no\_grad():

output = model(input\_tensor)

\_, predicted = torch.max(output, 1)

pred\_class = test\_dataset.classes[predicted.item()]

plt.imshow(img)

plt.title(f"Model Prediction: {pred\_class}")

plt.axis('off')

plt.show()

# Example usage with the corrected path

img\_path = "C:\\Users\\kakad\\OneDrive\\Desktop\\car-cars-lamborghini-aventador-63764.jpg"

predict\_user\_image(img\_path, model, test\_dataset, device)

Output :



Name : Shreya Kakade

Roll No : TEAD22541

Assignment No : 13

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.preprocessing import image

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

from PIL import Image

import webbrowser

import os

# Load MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Normalize images to [0, 1]

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Reshape for Conv2D input: (batch\_size, height, width, channels)

x\_train = x\_train.reshape(-1, 28, 28, 1)

x\_test = x\_test.reshape(-1, 28, 28, 1)

# Build the CNN model

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax') # 10 classes (digits 0–9)

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

history = model.fit(x\_train, y\_train, epochs=5, validation\_data=(x\_test, y\_test))

# Evaluate on test set

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f"\nTest Accuracy: {test\_acc:.4f}")

# Plot Accuracy and Loss

plt.figure(figsize=(12, 4))

# Accuracy plot

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

# Loss plot

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.tight\_layout()

plt.show()

# Prediction on custom image (only "th (3).jpeg" path)

image\_paths = [

"C:/Users/kakad/OneDrive/Desktop/th (3).jpeg" # Only this image path is kept

]

for path in image\_paths:

try:

img = Image.open(path).convert('L') # Grayscale

img = img.resize((28, 28))

img = 255 - np.array(img) # Invert background

img = img / 255.0

img\_input = img.reshape(1, 28, 28, 1)

# Display image

plt.imshow(img, cmap='gray')

plt.title(f"Image: {os.path.basename(path)}")

plt.axis('off')

plt.show()

prediction = model.predict(img\_input)

predicted\_class = np.argmax(prediction)

print(f"Model Prediction for {os.path.basename(path)}: {predicted\_class}")

except FileNotFoundError:

print(f"Image not found: {path}")

# Open the HTML file in default browser

html\_path = "C:/Users/kakad/OneDrive/Desktop/view (1).html"

if os.path.exists(html\_path):

webbrowser.open(f"file:///{html\_path}")

else:

print(f"HTML file not found: {html\_path}")

Output :

