**1)CNN**

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

# Load the CIFAR-10 dataset

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

# Normalize the data

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

# One-hot encode the labels

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Define the CNN model

def cnn():

model = models.Sequential()

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

model.add(layers.MaxPooling2D(pool\_size=(2, 2)))

model.add(layers.Dropout(0.25))

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

model.add(layers.MaxPooling2D(pool\_size=(2, 2)))

model.add(layers.Dropout(0.25))

model.add(layers.Flatten())

model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dropout(0.25))

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

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

return model

# Instantiate and compile the model

model = cnn()

# Train the model

history = model.fit(x\_train, y\_train, epochs=10, validation\_split=0.2, verbose=1)

# Plot Training and Validation Accuracy

import matplotlib.pyplot as plt

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

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

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Evaluate the model on test data

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy :{accuracy\*100:.2f}%")

**--------------------------------------------------------------------------------------------------------------------------------------**

2)

import os

import numpy as np

from tensorflow.keras.datasets import cifar10

from PIL import Image

import warnings

#warnings.filterwarnings('ignore')

# Create directories to save the images

train\_dir = 'cifar10\_tiff/train'

test\_dir = 'cifar10\_tiff/test'

os.makedirs(train\_dir, exist\_ok=True)

os.makedirs(test\_dir, exist\_ok=True)

# Load CIFAR-10 dataset

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

# Define class labels for CIFAR-10

class\_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Helper function to save images in .tiff format

def save\_images(images, labels, directory):

for i, (image\_array, label) in enumerate(zip(images, labels)):

# Convert numpy array to PIL image

image = Image.fromarray(image\_array)

# Define the label name

label\_name = class\_labels[int(label)]

# Create a subdirectory for each class

label\_dir = os.path.join(directory, label\_name)

os.makedirs(label\_dir, exist\_ok=True)

# Save the image in .tiff format

image\_path = os.path.join(label\_dir, f"{label\_name}\_{i}.tiff")

image.save(image\_path, format='TIFF')

# Save training and test images as .tiff

save\_images(x\_train, y\_train, train\_dir)

save\_images(x\_test, y\_test, test\_dir)

print("Images have been successfully saved as .tiff files.")

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Set paths

train\_dir = 'cifar10\_tiff/train'

test\_dir = 'cifar10\_tiff/test'

# Define ImageDataGenerator with data augmentation for the training set

train\_datagen = ImageDataGenerator(

rescale=1.0/255.0,

rotation\_range=15,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

horizontal\_flip=True,

validation\_split=0.2 # Split 20% of training data for validation

)

# Define ImageDataGenerator for the test set (only rescaling)

test\_datagen = ImageDataGenerator(rescale=1.0/255.0)

# Load training data

train\_data = train\_datagen.flow\_from\_directory(

directory=train\_dir,

target\_size=(32, 32), # CIFAR-10 images are 32x32 pixels

batch\_size=32,

class\_mode='categorical',

subset='training'

)

# Load validation data

validation\_data = train\_datagen.flow\_from\_directory(

directory=train\_dir,

target\_size=(32, 32),

batch\_size=32,

class\_mode='categorical',

subset='validation'

)

# Load test data

test\_data = test\_datagen.flow\_from\_directory(

directory=test\_dir,

target\_size=(32, 32),

batch\_size=32,

class\_mode='categorical',

shuffle=False

)

model = Sequential([

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

MaxPooling2D((2, 2)),

Dropout(0.25),

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

MaxPooling2D((2, 2)),

Dropout(0.25),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Dropout(0.25),

Flatten(),

Dense(256, activation='relu'),

Dropout(0.5),

Dense(10, activation='softmax') # 10 classes for CIFAR-10

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

model.summary()

from tensorflow.keras.callbacks import EarlyStopping

# Early stopping callback

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

#if patience=5, training will continue for up to 5 epochs without improvement in val\_loss. If there’s still no improvement after 5 epochs, training stops.

#This is helpful to avoid stopping too early if there’s a chance for improvement.

#When set to True, this parameter restores the model weights to those from the epoch with the best performance on the monitored metric (here, val\_loss).

#This ensures that the model doesn’t retain the weights from the last epoch, which may not be the best

history = model.fit(train\_data,validation\_data=validation\_data,epochs=5,callbacks=[early\_stopping])

# Evaluate on test data

test\_loss, test\_accuracy = model.evaluate(test\_data)

print(f"Test accuracy: {test\_accuracy \* 100:.2f}%")

# Plot training history

import matplotlib.pyplot as plt

# Plot accuracy

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

plt.subplot(1, 2, 1)

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

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

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy')

# Plot loss

plt.subplot(1, 2, 2)

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

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

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

# Get predictions

predictions = model.predict(test\_data)

predicted\_classes = tf.argmax(predictions, axis=1)

# Actual classes from the test data generator

true\_classes = test\_data.classes

# Accuracy by comparing predicted and actual classes

accuracy = np.mean(predicted\_classes == true\_classes)

print(f"Prediction accuracy on test set: {accuracy \* 100:.2f}%")

# Print actual and predicted classes

print("Actual classes:", true\_classes)

print("Predicted classes:", predicted\_classes.numpy())from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

# Classification report

print("\nClassification Report:")

print(classification\_report(true\_classes, predicted\_classes, target\_names=class\_labels))

# Confusion matrix

conf\_matrix = confusion\_matrix(true\_classes, predicted\_classes)

print(conf\_matrix)