**LENET**

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# 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 LeNet model

def lenet():

model = models.Sequential()

# Layer 1: Convolution + Pooling

model.add(layers.Conv2D(6, (5, 5), activation='relu', input\_shape=(32, 32, 3), padding='same'))

model.add(layers.AveragePooling2D(pool\_size=(2, 2)))

# Layer 2: Convolution + Pooling

model.add(layers.Conv2D(16, (5, 5), activation='relu'))

model.add(layers.AveragePooling2D(pool\_size=(2, 2)))

# Flatten and Fully Connected Layers

model.add(layers.Flatten())

model.add(layers.Dense(120, activation='relu'))

model.add(layers.Dense(84, activation='relu'))

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

# Compile the model

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

return model

# Instantiate the model

model = lenet()

# Train the model

history = model.fit(x\_train, y\_train, epochs=3, validation\_split=0.2, batch\_size=64, verbose=1)

# Plot Training and Validation Accuracy

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}%")

import os

import numpy as np

import pandas as pd

from PIL import Image

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

from keras.models import Sequential

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

from keras.optimizers import Adam

import matplotlib.pyplot as plt

from matplotlib import image

os.listdir(r"C:\Users\sudik\DLVS\wildfire-prediction-dataset\train\nowildfire")

from matplotlib import image

from PIL import Image

import numpy as np

import pandas as pd

import os

def Load\_Images(path):

folders = os.listdir(path)

data = []

label = []

if '.DS\_Store' in folders:

folders.remove('.DS\_Store')

for i in folders:

images = os.listdir(path + '/' + i)

if '.DS\_Store' in images:

images.remove('.DS\_Store')

for j in images:

try:

im = Image.open(path + '/' + i + '/' + j).convert('RGB')

im = im.resize((32, 32), Image.NEAREST) # Resize to 32x32 for LeNet input

ar = np.array(im)

data.append(ar)

# Binary label assignment for 'NORMAL' and 'PNEUMONIA'

if i == 'nowildfire':

label.append(0)

else:

label.append(1)

except OSError as e:

print(f"Skipping file {j} due to error: {e}")

data = np.array(data) / 255.0 # Normalize images to [0, 1]

tgt = np.array(label)

return data, tgt

os.listdir(r'C:\Users\sudik\DLVS\wildfire-prediction-dataset\train')

# Load training and testing data

train\_x, train\_y = Load\_Images(r'C:\Users\sudik\DLVS\wildfire-prediction-dataset\train')

test\_x, test\_y = Load\_Images(r'C:\Users\sudik\DLVS\wildfire-prediction-dataset\test')

train\_x

train\_y

train\_x.shape,train\_y.shapetrain\_x.shape,train\_y.shape

# Normalize image data

train\_x, test\_x = train\_x / 255.0, test\_x / 255.0

from keras.models import Sequential

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

# Define activation function

act = 'relu'

# Initialize the model

model = Sequential()

# LeNet Architecture with similar structure

model.add(Conv2D(6, (5, 5), activation=act, input\_shape=(32, 32, 3), padding='same'))

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Conv2D(16, (5, 5), activation=act, padding='valid'))

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Dropout(0.2)) # Optional dropout for regularization

# Additional convolutional layers to mimic the complexity

model.add(Conv2D(32, (3, 3), activation=act, padding='same'))

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Dropout(0.2))

# Flatten layer to convert 2D matrix to 1D vector

model.add(Flatten())

# Fully connected layers

model.add(Dense(120, activation=act))

model.add(Dropout(0.5))

model.add(Dense(84, activation=act))

model.add(Dropout(0.5))

# Output layer for 3-class classification (adjust for the number of classes in your problem)

model.add(Dense(3, activation='softmax'))

# Display model summary

model.summary()

# Compile model

opt = Adam()

model.compile(optimizer=opt, loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

hist=model.fit(train\_x,train\_y,epochs=20)

# Access the training and validation accuracy from the history object

train\_accuracy = hist.history['accuracy']

val\_accuracy = hist.history['val\_accuracy']

# Display the final accuracy

print(f"Final Training Accuracy: {train\_accuracy[-1] \* 100:.2f}%")

print(f"Final Validation Accuracy: {val\_accuracy[-1] \* 100:.2f}%")

test\_x,test\_y=Load\_Images(r'C:\Users\sudik\DLVS\wildfire-prediction-dataset\test')

test\_x

test\_y

# Evaluate model on test data

score = model.evaluate(test\_x, test\_y)

accuracy = 100 \* score[1]

print("Test Accuracy: %.4f%%" % accuracy)

# Function to preprocess single image for prediction

def image\_preprocess(path):

im = Image.open(path).convert('RGB')

im = im.resize((32, 32), Image.NEAREST)

unseen\_ar = np.array(im)

unseen\_ar = unseen\_ar.reshape(1, unseen\_ar.shape[0], unseen\_ar.shape[1], unseen\_ar.shape[2])

return unseen\_ar

# Display example image

im = image.imread(r'C:\Users\sudik\DLVS\wildfire-prediction-dataset\valid\nowildfire\-73.63276,45.599111.jpg')

plt.imshow(im)

img\_unseen=image\_preprocess(r"C:\Users\sudik\DLVS\wildfire-prediction-dataset\valid\nowildfire\-73.71373,45.493494.jpg")

img\_unseen.shape

model.predict(img\_unseen).argmax()

image=image\_preprocess(r"C:\Users\sudik\DLVS\wildfire-prediction-dataset\valid\wildfire\-64.23,51.0558.jpg")

image.shape

model.predict(image).argmax()

# Plot training history

import matplotlib.pyplot as plt

# Plot accuracy

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

plt.subplot(1, 2, 1)

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

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

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

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

# Predict on training data to compute confusion matrix and classification report

y\_pred = model.predict(train\_x)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

# Confusion matrix and classification report

cm = confusion\_matrix(train\_y, y\_pred\_classes)

print("Confusion Matrix:\n", cm)

cr = classification\_report(train\_y, y\_pred\_classes)

print("Classification Report:\n", cr)