import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

import keras

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import Dense, Conv2D, MaxPooling2D,GlobalAveragePooling2D, Flatten, Dropout, BatchNormalization, Input,AveragePooling2D

from glob import glob

from keras.models import Model

from keras.preprocessing.image import ImageDataGenerator

from keras.utils import load\_img, img\_to\_array

from keras.callbacks import ModelCheckpoint, EarlyStopping

from keras.models import load\_model

import os

from google.colab import drive

drive.mount('/content/drive')

#dataset of fake real recognition

train\_path = '/content/drive/MyDrive/mini project/Indian Currency Dataset/train'

validation\_path = '/content/drive/MyDrive/mini project/Indian Currency Dataset/test'

# no. of classes in recognition data

className = glob(train\_path + "/\*")

NumberofClass = len(className)

print("NumberofClass:", NumberofClass)

# data augumentation

train\_datagen = ImageDataGenerator(rescale=1/255)

validation\_datagen=ImageDataGenerator(rescale=1/255)

test\_datagen=ImageDataGenerator(rescale=1/255)

batch\_size = 10

# recognition data

train\_datagen = ImageDataGenerator(rescale=1/255,

shear\_range=0.3,

horizontal\_flip=True,

zoom\_range=0.3

)

val\_datagen = ImageDataGenerator(rescale=1/255)

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

train\_path,

target\_size=(224,224),

batch\_size=batch\_size,

color\_mode="rgb",

class\_mode="categorical"

)

val\_generator = val\_datagen.flow\_from\_directory(

validation\_path,

target\_size=(224,224),

batch\_size=batch\_size,

color\_mode="rgb",

class\_mode="categorical"

)

train\_generator.class\_indices

input\_shape=(224,224,3)

def building\_the\_model(input\_shape):

inputs = Input(shape=input\_shape)

model = (Conv2D(32, (3, 3),strides=2,padding='same', activation='relu',data\_format='channels\_last' ,input\_shape=input\_shape))(inputs)

model = (BatchNormalization())(model)

model = (Conv2D(64, (3, 3),strides=2,padding='same', activation='relu',data\_format='channels\_last', input\_shape=input\_shape))(model)

model = (BatchNormalization())(model)

model = (MaxPooling2D(pool\_size=(2, 2), strides=2,data\_format='channels\_last'))(model)

model = (Conv2D(128, (3, 3),strides=2,padding='same', activation='relu',data\_format='channels\_last', input\_shape=input\_shape))(model)

model = (BatchNormalization())(model)

model = (AveragePooling2D(pool\_size=(2, 2), strides=2,data\_format='channels\_last'))(model)

model = (Flatten())(model)

model = (Dense(2056))(model)

model = tf.keras.layers.Dropout(0.2)(model)

model = (Dense(512))(model)

model = tf.keras.layers.Dropout(0.2)(model)

model = (Dense(256))(model)

model = (Dense(2,activation="sigmoid"))(model)

# outputs = (Dense(16))(model)

X = model

# Bui4ld model

model = Model(inputs=inputs, outputs=X)

return model

model = building\_the\_model(input\_shape)

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

history = model.fit(train\_generator,

validation\_data = val\_generator,

batch\_size = 128,

epochs = 50)

# Plot training history

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.subplot(1, 2, 2)

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

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

plt.xlabel('Epochs')

val\_predictions = model.predict(val\_generator)

val\_predicted\_labels = np.argmax(val\_predictions, axis=1)

val\_true\_labels = val\_generator.classes

conf\_matrix = tf.math.confusion\_matrix(val\_true\_labels, val\_predicted\_labels)

conf\_matrix = conf\_matrix.numpy()

print("Confusion Matrix:")

print(conf\_matrix)

# Plot confusion matrix as a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=val\_generator.class\_indices.keys(),

yticklabels=val\_generator.class\_indices.keys())

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

plt.ylabel('Loss')

plt.legend()

plt.show()

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

val\_predictions = model.predict(val\_generator)

val\_predicted\_labels = np.argmax(val\_predictions, axis=1)

val\_true\_labels = val\_generator.classes

# Generate and print classification report

class\_names = list(train\_generator.class\_indices.keys())

print("Classification Report:")

print(classification\_report(val\_true\_labels, val\_predicted\_labels, target\_names=class\_names))