**Along with ImageNet weights, matrices, graphs, and generators to control RAM**

import tensorflow as tf

import os

import cv2

from tqdm import tqdm

import numpy as np

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

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.applications.vgg16 import VGG16

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Install seaborn

!pip install seaborn

# Download the data

!gdown 13ndWKp\_qe5aD1LCOEkdVf\_tkvLCMBPRJ        # For 3316 samples

#!gdown 1FiEdQcfa3W\_8ZrLExftnrVNGY4e-GC8I # For 3316 non\_emoj samples

import zipfile

file\_name = "/content/Test.zip"

with zipfile.ZipFile(file\_name, 'r') as zip\_ref:

    zip\_ref.extractall()

    print('Done')

data\_dir\_class1 = '/content/0'  # Path to the directory containing class 1 images

data\_dir\_class2 = '/content/1'  # Path to the directory containing class 2 images

# Create data generator for training and testing

batch\_size = 16

input\_shape = (224, 224)

train\_datagen = ImageDataGenerator(rescale=1.0/255)

test\_datagen = ImageDataGenerator(rescale=1.0/255)

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

    '/content/',

    target\_size=input\_shape,

    batch\_size=batch\_size,

    classes=['0', '1'],  # Class names should match the subdirectory names in the data directory

    class\_mode='categorical',  # For binary classification, use class\_mode='binary'

    shuffle=True,

)

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

    '/content/',

    target\_size=input\_shape,

    batch\_size=batch\_size,

    classes=['0', '1'],  # Class names should match the subdirectory names in the data directory

    class\_mode='categorical',  # For binary classification, use class\_mode='binary'

    shuffle=False,

)

# Load pre-trained VGG-16 model without the top layer

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

model = Sequential()

model.add(base\_model)

model.add(Flatten())

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='sigmoid'))

model.compile(optimizer=tf.keras.optimizers.SGD(learning\_rate=1e-6, momentum=0.9),

              loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model using the data generator

epochs = 40

history = model.fit(train\_generator,

                    steps\_per\_epoch=len(train\_generator),

                    validation\_data=test\_generator,

                    validation\_steps=len(test\_generator),

                    epochs=epochs)

# Evaluate the model on the test set using the data generator

test\_steps = len(test\_generator)

test\_loss, test\_accuracy = model.evaluate(test\_generator, steps=test\_steps)

print("Test accuracy:", test\_accuracy)

# Calculate predictions

y\_pred = model.predict(test\_generator)

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

# Convert one-hot encoded labels back to integers

Y\_test = test\_generator.classes

# Calculate evaluation metrics

accuracy = accuracy\_score(Y\_test, y\_pred)

precision = precision\_score(Y\_test, y\_pred)

recall = recall\_score(Y\_test, y\_pred)

f1 = f1\_score(Y\_test, y\_pred)

confusion\_mat = confusion\_matrix(Y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 score:", f1)

print("Confusion Matrix:")

print(confusion\_mat)

# Plot Confusion Matrix

cm = confusion\_matrix(Y\_test, y\_pred)

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

sns.heatmap(cm, annot=True, fmt=".0f", linewidths=0.5, square=True, cmap="Blues")

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.savefig('/content/confusion\_matrix.pdf', format='pdf')  # Save the Confusion Matrix as PDF

plt.show()

# Plot training and validation accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.savefig('/content/accuracy\_plot.pdf', format='pdf')  # Save the Accuracy Plot as PDF

plt.show()

# Plot training and validation loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.savefig('/content/loss\_plot.pdf', format='pdf')  # Save the Loss Plot as PDF

plt.show()

**With its own weights, matrix and graph, and without Generator**

import tensorflow as tf

import os

import cv2

from tqdm import tqdm

import numpy as np

from sklearn import preprocessing

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

from tensorflow.keras.models import Sequential

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

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

from matplotlib.backends.backend\_pdf import PdfPages

# Download the data

#!gdown 13ndWKp\_qe5aD1LCOEkdVf\_tkvLCMBPRJ       # For 3316 samples

#!gdown 1FiEdQcfa3W\_8ZrLExftnrVNGY4e-GC8I       # For 3316 non\_emoj samples

#!gdown 1qbaV7ZW6ccZDnK6lN\_qHQYgkr8tTn1w0       # For 167290 samples Compressed     Dataset\_C

#!gdown 14ZLbnCYAmEnK\_0xPlVvKL\_-adilqxoX9       # For 83644 samples Compressed      Dataset\_C\_H

#!gdown 11YiYReHcSeyVoe6I8fIHK-\_RJwYp2cgK       # For 41822 samples Compressed      Dataset\_C\_Q

!gdown 1\_SLbu\_FqCD5GVsFjPGaMm-54Li3npSgV       # For 41822 samples Compressed      Dataset\_C\_E

#!gdown 1VhpWHMQaYqecBmd2QTKMFIY0m\_9hbBy1       # For 41822 samples Compressed      Dataset\_C\_E\_non\_emoj

import zipfile

file\_name = "/content/Dataset\_C\_E.zip"

with zipfile.ZipFile(file\_name, 'r') as zip\_ref:

    zip\_ref.extractall()

    print('Done')

data\_dir\_class1 = '/content/0'  # Path to the directory containing class 1 images

data\_dir\_class2 = '/content/1'  # Path to the directory containing class 2 images

X = []

Y = []

# Load images from class 1 directory

for i in tqdm(os.listdir(data\_dir\_class1)):

    img = cv2.imread(os.path.join(data\_dir\_class1, i))

    img = cv2.resize(img, (224, 224))

    X.append(img)

    Y.append(0)  # Assign label 0 for class 1

# Load images from class 2 directory

for i in tqdm(os.listdir(data\_dir\_class2)):

    img = cv2.imread(os.path.join(data\_dir\_class2, i))

    img = cv2.resize(img, (224, 224))

    X.append(img)

    Y.append(1)  # Assign label 1 for class 2

le = preprocessing.LabelEncoder()

Y = le.fit\_transform(Y)

Y = tf.keras.utils.to\_categorical(Y, num\_classes=2)

X = np.array(X)

Y = np.array(Y)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=42)

print("Shape of an image in X\_train: ", X\_train[0].shape)

print("Shape of an image in X\_test: ", X\_test[0].shape)

model = Sequential()

# Add convolutional layers

model.add(Conv2D(64, (3, 3), activation='relu', input\_shape=(224, 224, 3)))

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

model.add(Conv2D(128, (3, 3), activation='relu'))

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

model.add(Conv2D(256, (3, 3), activation='relu'))

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

model.add(Flatten())

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='sigmoid'))

model.compile(optimizer=tf.keras.optimizers.SGD(learning\_rate=1e-6, momentum=0.9), loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), batch\_size=16, epochs=100, verbose=1)

# Calculate predictions

y\_pred = model.predict(X\_test)

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

# Convert one-hot encoded labels back to integers

Y\_test = np.argmax(Y\_test, axis=1)

# Calculate evaluation metrics

accuracy = accuracy\_score(Y\_test, y\_pred)

precision = precision\_score(Y\_test, y\_pred)

recall = recall\_score(Y\_test, y\_pred)

f1 = f1\_score(Y\_test, y\_pred)

confusion\_mat = confusion\_matrix(Y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 score:", f1)

print("Confusion Matrix:")

print(confusion\_mat)

# Create a PDF file for saving the plots and the confusion matrix

pdf\_file = "/content/output\_plots.pdf"

pdf\_pages = PdfPages(pdf\_file)

# Plot Confusion Matrix

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

sns.heatmap(confusion\_mat, annot=True, fmt=".0f", linewidths=0.5, square=True, cmap="Blues")

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

pdf\_pages.savefig()  # Save the Confusion Matrix plot to PDF

plt.close()

# Plot training and validation accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(['Train', 'Validation'], loc='upper left')

pdf\_pages.savefig()  # Save the Accuracy Plot to PDF

plt.close()

# Plot training and validation loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(['Train', 'Validation'], loc='upper left')

pdf\_pages.savefig()  # Save the Loss Plot to PDF

plt.close()

# Save the model weights

#model.save\_weights("/content/model\_weights.h5")

# Close the PDF file

pdf\_pages.close()

**With imagenet weights, matrix and graph, and without Generator**

import tensorflow as tf

import os

import cv2

from tqdm import tqdm

import numpy as np

from sklearn import preprocessing

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

from tensorflow.keras.applications import VGG16

from tensorflow.keras.models import Sequential

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

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

from matplotlib.backends.backend\_pdf import PdfPages

# Download the data

# Download the data

# Download the data

!gdown 1i3Rq9HwXYmS7v\_2ndNNVIou1CrwRYdus        # For 10000 samples         Dataset\_10000

#!gdown 13ndWKp\_qe5aD1LCOEkdVf\_tkvLCMBPRJ       # For 3316 samples

#!gdown 1FiEdQcfa3W\_8ZrLExftnrVNGY4e-GC8I       # For 3316 non\_emoj samples

#!gdown 1qbaV7ZW6ccZDnK6lN\_qHQYgkr8tTn1w0       # For 167290 samples Compressed     Dataset\_C

#!gdown 14ZLbnCYAmEnK\_0xPlVvKL\_-adilqxoX9       # For 83644 samples Compressed      Dataset\_C\_H

#!gdown 11YiYReHcSeyVoe6I8fIHK-\_RJwYp2cgK       # For 41822 samples Compressed      Dataset\_C\_Q

#!gdown 1\_SLbu\_FqCD5GVsFjPGaMm-54Li3npSgV       # For 41822 samples Compressed      Dataset\_C\_E

#!gdown 1VhpWHMQaYqecBmd2QTKMFIY0m\_9hbBy1       # For 41822 samples Compressed      Dataset\_C\_E\_non\_emoj

import zipfile

file\_name = "/content/Dataset\_C\_E.zip"

with zipfile.ZipFile(file\_name, 'r') as zip\_ref:

    zip\_ref.extractall()

    print('Done')

data\_dir\_class1 = '/content/0'  # Path to the directory containing class 1 images

data\_dir\_class2 = '/content/1'  # Path to the directory containing class 2 images

X = []

Y = []

# Load images from class 1 directory

for i in tqdm(os.listdir(data\_dir\_class1)):

    img = cv2.imread(os.path.join(data\_dir\_class1, i))

    img = cv2.resize(img, (224, 224))

    X.append(img)

    Y.append(0)  # Assign label 0 for class 1

# Load images from class 2 directory

for i in tqdm(os.listdir(data\_dir\_class2)):

    img = cv2.imread(os.path.join(data\_dir\_class2, i))

    img = cv2.resize(img, (224, 224))

    X.append(img)

    Y.append(1)  # Assign label 1 for class 2

le = preprocessing.LabelEncoder()

Y = le.fit\_transform(Y)

Y = tf.keras.utils.to\_categorical(Y, num\_classes=2)

X = np.array(X)

Y = np.array(Y)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=42)

print("Shape of an image in X\_train: ", X\_train[0].shape)

print("Shape of an image in X\_test: ", X\_test[0].shape)

# Load VGG16 with pre-trained ImageNet weights (excluding the top classification layers)

vgg\_model = VGG16(include\_top=False, weights='imagenet', input\_shape=(224, 224, 3))

# Make all layers of the VGG16 model trainable

for layer in vgg\_model.layers:

    layer.trainable = True

model = Sequential()

model.add(vgg\_model)

model.add(Flatten())

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='sigmoid'))

model.compile(optimizer=tf.keras.optimizers.SGD(learning\_rate=1e-6, momentum=0.9),

              loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test),

                    batch\_size=16, epochs=100, verbose=1)

# Calculate predictions

y\_pred = model.predict(X\_test)

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

# Convert one-hot encoded labels back to integers

Y\_test = np.argmax(Y\_test, axis=1)

# Calculate evaluation metrics

accuracy = accuracy\_score(Y\_test, y\_pred)

precision = precision\_score(Y\_test, y\_pred)

recall = recall\_score(Y\_test, y\_pred)

f1 = f1\_score(Y\_test, y\_pred)

confusion\_mat = confusion\_matrix(Y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 score:", f1)

print("Confusion Matrix:")

print(confusion\_mat)

# Create a PDF file for saving the plots and the confusion matrix

pdf\_file = "/content/output\_plots.pdf"

pdf\_pages = PdfPages(pdf\_file)

# Plot Confusion Matrix

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

sns.heatmap(confusion\_mat, annot=True, fmt=".0f", linewidths=0.5, square=True, cmap="Blues")

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

pdf\_pages.savefig()  # Save the Confusion Matrix plot to PDF

plt.close()

# Plot training and validation accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(['Train', 'Validation'], loc='upper left')

pdf\_pages.savefig()  # Save the Accuracy Plot to PDF

plt.close()

# Plot training and validation loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(['Train', 'Validation'], loc='upper left')

pdf\_pages.savefig()  # Save the Loss Plot to PDF

plt.close()

# Save the model weights

# model.save\_weights("/content/model\_weights.h5")

# Close the PDF file

pdf\_pages.close()

**Visualization of the vgg-16 model**

import matplotlib.pyplot as plt

from tensorflow.keras.models import Model

from matplotlib.backends.backend\_pdf import PdfPages

# Replace 'your\_layer\_name' with the actual name of your first convolutional layer

layer\_name = 'block1\_conv1'

# Create a new model that consists only of the specified convolutional layer

model\_layer = Model(inputs=vgg\_model.input, outputs=vgg\_model.get\_layer(layer\_name).output)

# Get the weights (kernels) of the first convolutional layer

conv1\_weights = model\_layer.get\_weights()[0]

# Create a PDF file for saving the plot

pdf\_file = "conv1\_kernels.pdf"

pdf\_pages = PdfPages(pdf\_file)

# Visualize the convolutional kernels as images

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

for i in range(conv1\_weights.shape[3]):

    plt.subplot(8, 8, i+1)  # Adjust the subplot arrangement as needed

    plt.imshow(conv1\_weights[:, :, 0, i], cmap='viridis')

    plt.axis('off')

    plt.title(f'Kernel {i + 1}')

# Save the plot to the PDF file

pdf\_pages.savefig()

# Close the PDF file

pdf\_pages.close()

#print(f'Kernel visualization saved as {pdf\_file}')