**Lung\_Colon EfficientNet B2**

from keras.layers import Input, Lambda, Dense, Flatten

from keras.models import Model

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

import numpy as np

from glob import glob

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.layers import Dense, Activation

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.metrics import categorical\_crossentropy

from tensorflow.keras.applications import imagenet\_utils

from sklearn.metrics import confusion\_matrix

import itertools

import os

import shutil

import random

import matplotlib.pyplot as plt

from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

from tensorflow.keras import layers

from sklearn.metrics import accuracy\_score

from sklearn import metrics

from sklearn.metrics import recall\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import f1\_score

print('Required Frame works are included')

def initiateGenerator(path):

base\_path = path

print("\nTotal : ", end=" ")

train\_dataset = tf.keras.preprocessing.image\_dataset\_from\_directory(batch\_size=32, directory=base\_path)

train\_datagen = ImageDataGenerator(validation\_split=0.3)

print("\nFor Training : ", end=" ")

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

base\_path,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical', subset='training')

print("\nFor Val : ", end=" ")

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

base\_path,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical',

subset='validation', shuffle=False)

class\_names = train\_dataset.class\_names

noOfClasses = len(class\_names)

print("\nNo of Classes : ", noOfClasses)

print("Classes : ", class\_names)

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

for images, labels in train\_dataset.take(1):

for i in range(noOfClasses):

ax = plt.subplot(4, 4, i + 1)

plt.imshow(images[i].numpy().astype("uint8"))

plt.title(class\_names[labels[i]])

plt.axis("off")

for image\_batch, labels\_batch in train\_dataset:

print("Image Shape : ",image\_batch.shape)

break

return noOfClasses,class\_names, train\_generator, validation\_generator

print('function for train\_generator,validation\_generator')

def initiateNormalize():

AUTOTUNE = tf.data.AUTOTUNE

train\_ds = train\_generator.cache().shuffle(1000).prefetch(buffer\_size=AUTOTUNE)

val\_ds = val\_generator.cache().prefetch(buffer\_size=AUTOTUNE)

normalization\_layer = layers.Rescaling(1./255)

normalized\_ds = train\_ds.map(lambda x, y: (normalization\_layer(x), y))

image\_batch, labels\_batch = next(iter(normalized\_ds))

first\_image = image\_batch[0]

print(np.min(first\_image), np.max(first\_image))

print('function - data normalize')

def initiateModel(noOfClasses):

modelInput = tf.keras.applications.EfficientNetB2(

input\_shape=IMAGE\_SIZE + [3],

include\_top=False,

weights="imagenet"

)

for layer in modelInput.layers:

layer.trainable = False

x = Flatten()(modelInput.output)

prediction = Dense(noOfClasses, activation='softmax')(x)

model = Model(inputs=modelInput.input, outputs=prediction)

return model

def modelSummary(model):

model.summary()

print('function - model creation - EfficientNetB2')

def initiateParams(className, model, lr):

opt = tf.keras.optimizers.Adam(learning\_rate=lr)

model.compile(optimizer=opt,

loss='categorical\_crossentropy',

metrics=['accuracy'])

annealer = ReduceLROnPlateau(monitor='val\_accuracy', factor=0.5, patience=5, verbose=1, min\_lr=1e-3)

checkpoint = ModelCheckpoint(className + 'EfficientNetB2.h5', verbose=1, save\_best\_only=True)

return model, annealer, checkpoint

print('function - parameter passing to model - EfficientNetB2')

def modelFit(model, annealer, checkpoint, epochs=20, batchSize = 256):

history = model.fit(

train\_generator,

validation\_data=validation\_generator,

epochs=epochs,

batch\_size=batchSize,

callbacks=[annealer, checkpoint],

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

validation\_steps=len(validation\_generator)

)

return history

print('function - model training')

def plotOutput(history, className, epochs):

acc = history.history['accuracy']

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

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(epochs)

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

plt.subplot(3, 3, 1)

plt.plot(epochs\_range, acc, label='Training Accuracy')

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

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

plt.subplot(3, 3, 2)

plt.plot(epochs\_range, loss, label='Training Loss')

plt.plot(epochs\_range, val\_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()

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

plt.plot(epochs\_range, acc, label='Training Accuracy')

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

plt.plot(epochs\_range, loss, label='Training Loss')

plt.plot(epochs\_range, val\_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss & Training and Validation Accuracy')

plt.show()

print('function for model output plot')

def evalModel(model):

evl = model.evaluate(validation\_generator)

acc = evl[1]\*100

msg=f'Accuracy on the Test Set = {acc:5.2f} %'

print(msg)

def saveModel(model, className):

model.save(className + " - EfficientNetB2.h5")

print("Model Saved!")

print('function - model evaluation and model save to drive')

def plot\_confusion\_matrix(cm,

target\_names,

title='Confusion matrix',

cmap=None,

normalize=True):

import matplotlib.pyplot as plt

import numpy as np

import itertools

accuracy = np.trace(cm) / float(np.sum(cm))

misclass = 1 - accuracy

if cmap is None:

cmap = plt.get\_cmap('Blues')

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

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

if target\_names is not None:

tick\_marks = np.arange(len(target\_names))

plt.xticks(tick\_marks, target\_names, rotation=45)

plt.yticks(tick\_marks, target\_names)

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

thresh = cm.max() / 1.5 if normalize else cm.max() / 2

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

if normalize:

plt.text(j, i, "{:0.4f}".format(cm[i, j]),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

else:

plt.text(j, i, "{:,}".format(cm[i, j]),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclass))

plt.show()

plt.savefig(title + '.png')

print('function to plot confusion\_matrix')

from sklearn import metrics

def callPlot(model, className, classes):

y\_true = validation\_generator.classes

print("True : ", (y\_true))

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

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

print("Predicted : ", (y\_pred))

conf\_mat = confusion\_matrix(y\_true, y\_pred)

TPA=conf\_mat[0][0]

TPB=conf\_mat[1][1]

TPC=conf\_mat[2][2]

TPD=conf\_mat[3][3]

TPE=conf\_mat[4][4]

FPA=conf\_mat[0][1]

FPB=conf\_mat[1][2]

FPC=conf\_mat[2][3]

FPD=conf\_mat[3][4]

FNB=conf\_mat[1][0]

FNC=conf\_mat[2][1]

FND=conf\_mat[3][2]

FNE=conf\_mat[4][3]

ACC=round((TPA+TPB+TPC+TPD+TPE)/(TPA+TPB+TPC+TPD+TPE+FPA+FPB+FPC+FPD+FNB+FNC+FND+FNE),3)

P=round((TPA+TPB+TPC+TPD+TPE)/((TPA+TPB+TPC+TPD+TPE)+(FPA+FPB+FPC+FPD)),2)

RC=round((TPA+TPB+TPC+TPD+TPE)/((TPA+TPB+TPC+TPD+TPE)+(FNB+FNC+FND+FNE)),2)

F1=round(2\*(TPA+TPB+TPC+TPD+TPE)/(2\*(TPA+TPB+TPC+TPD+TPE)+(FPA+FPB+FPC+FPD)+(FNB+FNC+FND+FNE)),3)

print(ACC,P,RC,F1)

plot\_confusion\_matrix(cm = conf\_mat,

normalize = False,

target\_names = classes,

title = className + "Confusion Matrix")

print(f"Accuracy :{ACC}")

print(f"Precision: {P}")

print(f"Recall: {RC}")

print(f"F1 score: {F1}")

# Create a figure object

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

# Add a subplot to the figure

ax = fig.add\_subplot(2,1,1)

# Create the bar plot

bars = ax.bar(['Precision','Recall','Accuracy','F1 Score'],[ACC,P,RC,F1])

# Loop through the bars and add annotations

for bar in bars:

height = bar.get\_height()

ax.annotate(f'{height}', xy=(bar.get\_x() + bar.get\_width() / 2, height), xytext=(0, 3),

textcoords="offset points", ha='center', va='bottom')

# Show the plot

plt.title('Performance metrics')

plt.show()

print('function to plot True,Predicted,Performance metrics')

import os

mpath = "C://Windows//System32//project//Multiclass cancer//Multi Cancer"

classPaths = os.listdir(mpath)

IMAGE\_SIZE = [224, 224]

img\_height = 224

img\_width = 224

noOfClasses = 0

gEpochs = 20

lr = 0.001

for i in classPaths:

print(i)

print('load input from google drive')

className = 'Lung and Colon Cancer '

cpath = os.path.join(mpath, 'Lung and Colon Cancer')

noOfClasses, class\_names, train\_generator, validation\_generator = initiateGenerator(cpath)

curModel = initiateModel(noOfClasses)

modelSummary(curModel)

curModel, annealer, checkpoint = initiateParams(className, curModel, lr)

curHistory = modelFit(curModel, annealer, checkpoint, epochs=gEpochs, batchSize=256)

plotOutput(curHistory, className, gEpochs)

#evalModel(curModel)

#saveModel(curModel, className)

#callPlot(curModel, className, class\_names)

evalModel(curModel)

saveModel(curModel, className)

callPlot(curModel, className, class\_names)

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

from tensorflow.keras.utils import to\_categorical

num\_classes = 5

model=curModel

# Assuming you have a TensorFlow model and validation data

# Example:

# model = tf.keras.models.load\_model('your\_model.h5')

# valid\_batches = your\_validation\_data\_generator

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

# Get predictions from the model

predictions = y\_pred # Make sure y\_pred is a matrix of predicted probabilities

# Convert one-hot encoded labels to integers

y\_true = validation\_generator.classes

# Convert integer labels to one-hot encoding

y\_true\_one\_hot = to\_categorical(y\_true, num\_classes=num\_classes)

# Compute ROC curve and area under the curve (AUC) for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(num\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_true\_one\_hot[:, i], predictions[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves for individual classes

plt.figure(figsize=(10, 6))

for i in range(num\_classes):

plt.plot(fpr[i], tpr[i], lw=2, label=f'ROC curve - Class {i} (AUC = {roc\_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curves for lung and colon cancer classification using EfficientNetB2')

plt.legend(loc='lower right')

plt.show()

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

from tensorflow.keras.utils import to\_categorical

num\_classes = 5

model=curModel

# Assuming you have a TensorFlow model and validation data

# Example:

# model = tf.keras.models.load\_model('your\_model.h5')

# valid\_batches = your\_validation\_data\_generator

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

# Get predictions from the model

predictions = y\_pred # Make sure y\_pred is a matrix of predicted probabilities

# Convert one-hot encoded labels to integers

y\_true = validation\_generator.classes

# Convert integer labels to one-hot encoding

y\_true\_one\_hot = to\_categorical(y\_true, num\_classes=num\_classes)

# Compute ROC curve and area under the curve (AUC) for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

# Ensure that the loop does not access indices beyond the bounds of the predictions array

for i in range(min(num\_classes, predictions.shape[1])):

fpr[i], tpr[i], \_ = roc\_curve(y\_true\_one\_hot[:, i], predictions[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Micro-average ROC curve and AUC

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_true\_one\_hot.ravel(), predictions.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot the micro-average ROC curve

plt.figure(figsize=(10, 6))

plt.plot(fpr["micro"], tpr["micro"], color='darkorange', lw=2, label=f'ROC curve (micro-average AUC = {roc\_auc["micro"]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curves for lung and colon cancer classification using EfficientNetB2 (Micro-average)')

plt.legend(loc='lower right')

plt.show()

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

from tensorflow.keras.utils import to\_categorical

num\_classes = 5

model=curModel

# Assuming you have a TensorFlow model and validation data

# Example:

# model = tf.keras.models.load\_model('your\_model.h5')

# valid\_batches = your\_validation\_data\_generator

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

# Get predictions from the model

predictions = y\_pred # Make sure y\_pred is a matrix of predicted probabilities

# Convert one-hot encoded labels to integers

y\_true = validation\_generator.classes

# Convert integer labels to one-hot encoding

y\_true\_one\_hot = to\_categorical(y\_true, num\_classes=num\_classes)

# Compute ROC curve and area under the curve (AUC) for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

# Ensure that the loop does not access indices beyond the bounds of the predictions array

for i in range(min(num\_classes, predictions.shape[1])):

fpr[i], tpr[i], \_ = roc\_curve(y\_true\_one\_hot[:, i], predictions[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Macro-average ROC curve and AUC

fpr["macro"], tpr["macro"], \_ = roc\_curve(y\_true\_one\_hot.ravel(), predictions.ravel())

roc\_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot the micro-average ROC curve

plt.figure(figsize=(10, 6))

plt.plot(fpr["macro"], tpr["macro"], color='darkorange', lw=2, label=f'ROC curve (macro-average AUC = {roc\_auc["macro"]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curves for lung and colon cancer classification using EfficientNetB2 (Macro-average)')

plt.legend(loc='lower right')

plt.show()

className = 'Breast Cancer '

cpath = os.path.join(mpath, 'Breast Cancer')

noOfClasses, class\_names, train\_generator, validation\_generator = initiateGenerator(cpath)

curModel = initiateModel(noOfClasses)

modelSummary(curModel)

curModel, annealer, checkpoint = initiateParams(className, curModel, lr)

curHistory = modelFit(curModel, annealer, checkpoint, epochs=gEpochs, batchSize=256)

plotOutput(curHistory, className, gEpochs)

#evalModel(curModel)

#saveModel(curModel, className)

#callPlot(curModel, className, class\_names)

from sklearn import metrics

def callPlot1(model, className, classes):

y\_true = validation\_generator.classes

print("True : ", (y\_true))

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

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

print("Predicted : ", (y\_pred))

conf\_mat = confusion\_matrix(y\_true, y\_pred)

TP=conf\_mat[0][0]

FN=conf\_mat[0][1]

FP=conf\_mat[1][0]

TN=conf\_mat[1][1]

ACC=round((TP+TN)/(TP+TN+FP+FN),3)

P=round(TP/(TP+FP),3)

RC=round(TP/(TP+FN),3)

F1=(2\*P\*RC)/(P+RC)

F1=round(F1, 3)

print(ACC,P,RC,F1)

plot\_confusion\_matrix(cm = conf\_mat,

normalize = False,

target\_names = classes,

title = className + "Confusion Matrix")

print(f"Accuracy :{ACC}")

print(f"Precision: {P}")

print(f"Recall: {RC}")

print(f"F1 score: {F1}")

# Create a figure object

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

# Add a subplot to the figure

ax = fig.add\_subplot(2,1,1)

# Create the bar plot

bars = ax.bar(['Precision','Recall','Accuracy','F1 Score'],[ACC,P,RC,F1])

# Loop through the bars and add annotations

for bar in bars:

height = bar.get\_height()

ax.annotate(f'{height}', xy=(bar.get\_x() + bar.get\_width() / 2, height), xytext=(0, 3),

textcoords="offset points", ha='center', va='bottom')

# Show the plot

plt.title('Performance metrics')

plt.show()

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

from tensorflow.keras.utils import to\_categorical

num\_classes = 2

model=curModel

# Assuming you have a TensorFlow model and validation data

# Example:

# model = tf.keras.models.load\_model('your\_model.h5')

# valid\_batches = your\_validation\_data\_generator

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

# Get predictions from the model

predictions = y\_pred # Make sure y\_pred is a matrix of predicted probabilities

# Convert one-hot encoded labels to integers

y\_true = validation\_generator.classes

# Convert integer labels to one-hot encoding

y\_true\_one\_hot = to\_categorical(y\_true, num\_classes=num\_classes)

# Compute ROC curve and area under the curve (AUC) for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(num\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_true\_one\_hot[:, i], predictions[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves for individual classes

plt.figure(figsize=(10, 6))

for i in range(num\_classes):

plt.plot(fpr[i], tpr[i], lw=2, label=f'ROC curve - Class {i} (AUC = {roc\_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curves for Multiclass Breast cancer classification using EfficientNetB2')

plt.legend(loc='lower right')

plt.show()

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

from tensorflow.keras.utils import to\_categorical

num\_classes = 2

model=curModel

# Assuming you have a TensorFlow model and validation data

# Example:

# model = tf.keras.models.load\_model('your\_model.h5')

# valid\_batches = your\_validation\_data\_generator

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

# Get predictions from the model

predictions = y\_pred # Make sure y\_pred is a matrix of predicted probabilities

# Convert one-hot encoded labels to integers

y\_true = validation\_generator.classes

# Convert integer labels to one-hot encoding

y\_true\_one\_hot = to\_categorical(y\_true, num\_classes=num\_classes)

# Compute ROC curve and area under the curve (AUC) for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

# Ensure that the loop does not access indices beyond the bounds of the predictions array

for i in range(min(num\_classes, predictions.shape[1])):

fpr[i], tpr[i], \_ = roc\_curve(y\_true\_one\_hot[:, i], predictions[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Micro-average ROC curve and AUC

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_true\_one\_hot.ravel(), predictions.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot the micro-average ROC curve

plt.figure(figsize=(10, 6))

plt.plot(fpr["micro"], tpr["micro"], color='darkorange', lw=2, label=f'ROC curve (micro-average AUC = {roc\_auc["micro"]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curves for Multiclass breast cancer classification using EfficientNetB2 (Micro-average)')

plt.legend(loc='lower right')

plt.show()

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

from tensorflow.keras.utils import to\_categorical

num\_classes = 2

model=curModel

# Assuming you have a TensorFlow model and validation data

# Example:

# model = tf.keras.models.load\_model('your\_model.h5')

# valid\_batches = your\_validation\_data\_generator

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

# Get predictions from the model

predictions = y\_pred # Make sure y\_pred is a matrix of predicted probabilities

# Convert one-hot encoded labels to integers

y\_true = validation\_generator.classes

# Convert integer labels to one-hot encoding

y\_true\_one\_hot = to\_categorical(y\_true, num\_classes=num\_classes)

# Compute ROC curve and area under the curve (AUC) for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

# Ensure that the loop does not access indices beyond the bounds of the predictions array

for i in range(min(num\_classes, predictions.shape[1])):

fpr[i], tpr[i], \_ = roc\_curve(y\_true\_one\_hot[:, i], predictions[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Macro-average ROC curve and AUC

fpr["macro"], tpr["macro"], \_ = roc\_curve(y\_true\_one\_hot.ravel(), predictions.ravel())

roc\_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot the micro-average ROC curve

plt.figure(figsize=(10, 6))

plt.plot(fpr["macro"], tpr["macro"], color='darkorange', lw=2, label=f'ROC curve (macro-average AUC = {roc\_auc["macro"]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curves for Multiclass breast cancer classification using EfficientNetB2 (Macro-average)')

plt.legend(loc='lower right')

plt.show()