**Multiclass Breast Cancer**

#!pip install tensorflow

import tensorflow as tf

print('tensorflow version used:',tf.\_\_version\_\_)

#!pip install matplotlib

#!pip install pandas

#!pip install scikit-learn

#!pip install tqdm

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import itertools

from sklearn.utils import resample

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

from sklearn.metrics import accuracy\_score

import tensorflow as tf

from tensorflow import keras

import os

from os import listdir

from tqdm import tqdm

import shutil

!pip install opendatasets --upgrade --quiet

#!pip install focal-loss

import opendatasets as od

import random

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Conv2D, MaxPool2D, Flatten, BatchNormalization

from tensorflow.keras.utils import to\_categorical

#from keras.utils import np\_utils

!pip install np\_utils

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

#from keras.optimizers import SGD, rmsprop\_v2

from sklearn.metrics import confusion\_matrix

from tensorflow.keras.models import Model

tf.keras.utils

#%matplotlib inline

print('Required Frame works are included')

cd "C:/Windows/System32/project/multiclass breastcancerclass"

pwd

train\_path="./multicalssbreastcancer\_dataset/train/"

valid\_path = "./multicalssbreastcancer\_dataset/valid/"

print('Input loaded successfully')

train\_datagen = ImageDataGenerator(

rescale = 1/255,

#samplewise\_center=True,

#samplewise\_std\_normalization=True,

rotation\_range=45,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

brightness\_range=[0.5,1.0],

shear\_range=0.2,

horizontal\_flip=True,

vertical\_flip=True

)

valid\_datagen = ImageDataGenerator(

rescale = 1/255,

rotation\_range=45,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

brightness\_range=[0.5,1.0],

shear\_range=0.2,

horizontal\_flip=True,

vertical\_flip=True

)

print('Augmentation Process successfully')

img\_size = 256

train\_batches = train\_datagen.flow\_from\_directory(directory=train\_path, target\_size=(img\_size,img\_size),

class\_mode = 'categorical', batch\_size=32, shuffle=True)

valid\_batches = valid\_datagen.flow\_from\_directory(directory=valid\_path, target\_size=(img\_size,img\_size),

class\_mode = 'categorical', batch\_size=32, shuffle = False)

len(train\_batches)

imgs, labels = next(train\_batches)

#!pip install matplotlib

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

# Read Images

img1 = mpimg.imread('C:/Windows/System32/project/multiclass breastcancerclass//multicalssbreastcancer\_dataset//train//A//copy-copy-copy-copy-copy-copy-copy-copy-copy-copy-SOB\_B\_A-14-22549CD-40-004.png')

img2 = mpimg.imread('C:/Windows/System32/project/multiclass breastcancerclass//multicalssbreastcancer\_dataset//train//DC//SOB\_M\_DC-14-2523-40-011.png')

img3 = mpimg.imread('C:/Windows/System32/project/multiclass breastcancerclass//multicalssbreastcancer\_dataset//train//F//copy-copy-copy-copy-copy-copy-copy-SOB\_B\_F-14-14134-100-018.png')

img4 = mpimg.imread('C:/Windows/System32/project/multiclass breastcancerclass//multicalssbreastcancer\_dataset//train//LC//copy-copy-copy-copy-copy-copy-copy-SOB\_M\_LC-14-15570-200-028.png')

img5 = mpimg.imread('C:/Windows/System32/project/multiclass breastcancerclass//multicalssbreastcancer\_dataset//train//MC//copy-copy-copy-copy-copy-copy-SOB\_M\_MC-14-16456-40-052.png')

img6 = mpimg.imread('C:/Windows/System32/project/multiclass breastcancerclass//multicalssbreastcancer\_dataset//train//PC//copy-copy-copy-copy-copy-copy-copy-copy-SOB\_M\_PC-14-9146-400-009.png')

img7 = mpimg.imread('C:/Windows/System32/project/multiclass breastcancerclass//multicalssbreastcancer\_dataset//train//PT//copy-copy-copy-copy-copy-copy-copy-copy-copy-copy-SOB\_B\_PT-14-21998AB-40-014.png')

img8 = mpimg.imread('C:/Windows/System32/project/multiclass breastcancerclass//multicalssbreastcancer\_dataset//train/TA//copy-copy-copy-copy-copy-copy-copy-copy-SOB\_B\_TA-14-13200-40-007.png')

fig = plt.figure(figsize=(20,8))

ax1 = fig.add\_subplot(2,4,1)

plt.title('Input sample adenosis image')

ax1.imshow(img1)

ax2 = fig.add\_subplot(2,4,2)

plt.title('Input sample fibroadenoma image')

ax2.imshow(img2)

ax3 = fig.add\_subplot(2,4,3)

plt.title('Input sample phyllodes\_tumor image')

ax3.imshow(img3)

ax4 = fig.add\_subplot(2,4,4)

plt.title('Input sample tubular\_adenoma image')

ax4.imshow(img4)

ax5 = fig.add\_subplot(2,4,5)

plt.title('Input sample ductal\_carcinoma image')

ax5.imshow(img5)

ax6 = fig.add\_subplot(2,4,6)

plt.title('Input sample lobular\_carcinoma image')

ax6.imshow(img6)

ax7 = fig.add\_subplot(2,4,7)

plt.title('Input sample mucinous\_carcinoma image')

ax7.imshow(img7)

ax8 = fig.add\_subplot(2,4,8)

plt.title('Input sample papillary carcinoma mage')

ax8.imshow(img8)

k\_size = 5

model = Sequential([

Conv2D(filters=64, kernel\_size=(k\_size, k\_size), activation='relu', padding = 'same', input\_shape=(img\_size,img\_size,3)),

BatchNormalization(),

MaxPool2D(pool\_size=(2, 2), strides=3),

Conv2D(filters=128, kernel\_size=(k\_size, k\_size), activation='relu', padding = 'same'),

BatchNormalization(),

Conv2D(filters=128, kernel\_size=(k\_size, k\_size), activation='relu', padding = 'same'),

BatchNormalization(),

MaxPool2D(pool\_size=(2, 2), strides=2),

Dropout(0.25),

Conv2D(filters=256, kernel\_size=(k\_size, k\_size), activation='relu', padding = 'same'),

BatchNormalization(),

MaxPool2D(pool\_size=(2, 2), strides=2),

Conv2D(filters=512, kernel\_size=(k\_size, k\_size), activation='relu', padding = 'same'),

BatchNormalization(),

MaxPool2D(pool\_size=(2, 2), strides=2),

Dropout(0.25),

Conv2D(filters=512, kernel\_size=(k\_size, k\_size), activation='relu', padding = 'same'),

BatchNormalization(),

MaxPool2D(pool\_size=(2, 2), strides=2),

Flatten(),

Dense(units=512, activation='relu'),

BatchNormalization(),

Dropout(0.2),

Dense(units=8, activation='softmax')

])

print(model.summary())

model.compile(optimizer=tf.

keras.optimizers.Adam(learning\_rate=0.5E-4), loss='categorical\_crossentropy', metrics=['accuracy'])

class\_weight = {0: 17.8025,

1: 7.7995618838992335,

2: 17.45343137254902,

3: 13.881091617933723,

4: 2.292659368963297,

5: 12.625886524822695,

6: 9.987377279102384,

7: 14.128968253968255}

class\_weight

if not os.path.isdir('./breakhis\_chkpt'):

os.mkdir('./breakhis\_chkpt')

checkpoint\_filepath = os.path.join('./breakhis\_chkpt','{epoch:02d}-{val\_accuracy:.4f}.hdf5')

callback = tf.keras.callbacks.ModelCheckpoint(

filepath=checkpoint\_filepath,

save\_weights\_only=True,

monitor='val\_accuracy',

mode='max',

save\_best\_only=True)

#!pip install pillow

model.fit(x = train\_batches,steps\_per\_epoch=len(train\_batches), validation\_data=valid\_batches, callbacks=[callback],class\_weight = class\_weight, validation\_steps=len(valid\_batches),epochs = 60,verbose = 1)

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

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

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

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

plt.show()

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

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

plt.title('Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

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

plt.show()

predictions = model.predict(x = valid\_batches,steps = len(valid\_batches), verbose = 0)

predictions = np.argmax(predictions, axis = -1)

cm = confusion\_matrix(y\_true=valid\_batches.classes, y\_pred=predictions)

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion Matrix',

cmap=plt.cm.Blues):

if normalize:

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

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

plt.title(title)

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, ['A','F','PT','TA','DC','LC','MC','PC'])

plt.yticks(tick\_marks, classes)

plt.colorbar()

thresh = cm.max() / 2.

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

plt.text(j, i, cm[i, j],

horizontalalignment="center",

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

plt.tight\_layout()

#plt.tight\_layout(pad=0.01)

plt.ylabel('True Label')

plt.xlabel('Predicted Label')

classes = ['A(adenosis)', 'F(fibroadenoma)', 'PT(phyllodes\_tumor)', 'TA(tubular\_adenoma)', 'DC(ductal carcinoma)',

'LC(lobular carcinoma)', 'MC(mucinous carcinoma)', 'PC( papillary carcinoma)']

plot\_confusion\_matrix(cm=cm, classes=classes, title='Confusion Matrix')

TPA=cm[0][0]

FPA=cm[0][1]

FNA=cm[0][3]

FNB=cm[1][0]

TPB=cm[1][1]

FPB=cm[1][2]

FPC=cm[2][0]

TPC=cm[2][2]

FNC=cm[2][3]

FND=cm[3][0]

FPD=cm[3][1]

TPD=cm[3][3]

FPE=cm[4][0]

FNE=cm[4][3]

TPE=cm[4][4]

FNF=cm[5][0]

FPF=cm[5][1]

TPF=cm[3][5]

FNG=cm[6][0]

FPG=cm[6][1]

TPG=cm[6][6]

FNH=cm[7][0]

FPH=cm[7][1]

TPH=cm[7][7]

A=(TPA+TPB+TPC+TPD+TPE+TPF+TPG+TPH)/(TPA+FPA+FNA+FNB+TPB+FPB+FPC+TPC+FNC+FND+FPD+TPD+FPE+FNE+TPE+FNF+FPF+TPF+FNG+FPG+TPG+FNH+FPH+TPH)

P=(TPA+TPB+TPC+TPD+TPE+TPF+TPG+TPH)/((TPA+TPB+TPC+TPD+TPE+TPF+TPG+TPH)+(FPA+FPB+FPC+FPD+FPE+FPF+FPG+FPH))

RC=(TPA+TPB+TPC+TPD+TPE+TPF+TPG+TPH)/((TPA+TPB+TPC+TPD+TPE+TPF+TPG+TPH)+(FNA+FNB+FNC+FND+FNE+FNF+FNG+FNH))

#F1=(TPA+TPB+TPC+TPD+TPE+TPF+TPG+TPH)/((TPA+TPB+TPC+TPD+TPE+TPF+TPG+TPH)+(FPA+FPB+FPC+FPD+FPE+FPF+FPG+FPH)+(FNA+FNB+FNC+FND+FNE+FNF+FNG+FNH))

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

A=round(A,3)

P=round(P,3)

RC=round(RC,3)

F1=round(F1,3)

print(A,P,RC,F1)

print(f"Accuracy :{A}")

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(['Accuracy','Precision','Recall','F1 Score'],[A,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')

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

import numpy as np

import tensorflow as tf

from tensorflow.keras.utils import to\_categorical

num\_classes = 8

# 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

# Get predictions from the model

predictions = model.predict(valid\_batches)

# Convert one-hot encoded labels to integers

y\_true = valid\_batches.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')

plt.legend(loc='lower right')

plt.show()

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

import numpy as np

import tensorflow as tf

from tensorflow.keras.utils import to\_categorical

num\_classes=8

# 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

# Get predictions from the model

predictions = model.predict(valid\_batches)

# Convert one-hot encoded labels to integers

y\_true = valid\_batches.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 (Micro-average)')

plt.legend(loc='lower right')

plt.show()

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

import numpy as np

import tensorflow as tf

from tensorflow.keras.utils import to\_categorical

num\_classes=8

# 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

# Get predictions from the model

predictions = model.predict(valid\_batches)

# Convert one-hot encoded labels to integers

y\_true = valid\_batches.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 macro-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 (Macro-average)')

plt.legend(loc='lower right')

plt.show()