**Project\_binary (VGG-16)**

# Basic Libraries

#!pip install scikit-learn

#!pip install https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-1.8.0-py3-none-any.whl

#!pip install seaborn

import numpy as np

import random

from os import listdir

from PIL import Image

#!pip install np\_utils

#!pip install keras

#!pip install tensorflow

# Preprocessing/Visualization

import matplotlib.pyplot as plt

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

from sklearn.utils import shuffle

#from keras.utils import to\_categorical

#from keras.utils.to\_categorical

# Model Creation

import tensorflow as tf

from tensorflow.keras.utils import to\_categorical

from tensorflow import keras

from tensorflow.keras import layers

# Evaluation Metrics

import seaborn as sns

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

base\_path = "C://Windows//System32//project//project\_binay classification//breast-histopathology-images//IDC\_regular\_ps50\_idx5//"

files = listdir(base\_path)

# Find the total length of data/Find out how many patients are there

print("Total Number of Patients: "+ str(len(files)))

# Saving the data into an array [image\_path, class]

dataset = []

for i in range(len(files)):

patient\_id = files[i]

for c in [0,1]:

patient\_path = base\_path + patient\_id

class\_path = patient\_path + '/' + str(c) + '/'

subfiles = listdir(class\_path)

for pic in subfiles:

image\_path = class\_path + pic

dataset.append([image\_path,c])

print("Total Number of Images: " + str(len(dataset)))

# How each data is stored

dataset[0]

total\_length = len(dataset)

limit = total\_length/4

dataset = dataset[:int(limit)]

len(dataset)

# Get the size

# Load the image

image\_path = dataset[0][0]

label = dataset[0][1]

image = Image.open(image\_path)

# Get the size (dimensions) of the image

image\_width, image\_height = image.size

print(f"Image Width: {image\_width} pixels")

print(f"Image Height: {image\_height} pixels")

# Separate the data by class

NCdata = [img for img, label in dataset if label == 0]

Cdata = [img for img, label in dataset if label == 1]

NClabels = [label for img, label in dataset if label == 0]

Clabels = [label for img, label in dataset if label == 1]

# Get a sample of images from each type of dataset

negativeSample = random.sample(NCdata, 50)

positiveSample = random.sample(Cdata, 50)

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

img1 = mpimg.imread('C://Windows//System32//project//project\_binay classification//breast-histopathology-images//IDC\_regular\_ps50\_idx5//8863//0//8863\_idx5\_x101\_y1251\_class0.png')

img2 = mpimg.imread('C://Windows//System32//project//project\_binay classification//breast-histopathology-images//IDC\_regular\_ps50\_idx5//8863//0//8863\_idx5\_x101\_y1301\_class0.png')

img3 = mpimg.imread('C://Windows//System32//project//project\_binay classification//breast-histopathology-images//IDC\_regular\_ps50\_idx5//8863//0//8863\_idx5\_x151\_y1301\_class0.png')

img4 = mpimg.imread('C://Windows//System32//project//project\_binay classification//breast-histopathology-images//IDC\_regular\_ps50\_idx5//8863//0//8863\_idx5\_x201\_y1301\_class0.png')

fig = plt.figure(figsize=(4,5))

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

plt.title('noncancer image')

ax1.imshow(img1)

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

plt.title('noncancer image')

ax2.imshow(img2)

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

plt.title('noncancer image')

ax3.imshow(img3)

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

plt.title('noncancer image')

ax4.imshow(img4)

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

img1 = mpimg.imread('C://Windows//System32//project//project\_binay classification//breast-histopathology-images//IDC\_regular\_ps50\_idx5//8863//1//8863\_idx5\_x1001\_y801\_class1.png')

img2 = mpimg.imread('C://Windows//System32//project//project\_binay classification//breast-histopathology-images//IDC\_regular\_ps50\_idx5//8863//1//8863\_idx5\_x1001\_y851\_class1.png')

img3 = mpimg.imread('C://Windows//System32//project//project\_binay classification//breast-histopathology-images//IDC\_regular\_ps50\_idx5//8863//1//8863\_idx5\_x1001\_y901\_class1.png')

img4 = mpimg.imread('C://Windows//System32//project//project\_binay classification//breast-histopathology-images//IDC\_regular\_ps50\_idx5//8863//1//8863\_idx5\_x1001\_y951\_class1.png')

fig = plt.figure(figsize=(4,5))

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

plt.title('cancer image')

ax1.imshow(img1)

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

plt.title('cancer image')

ax2.imshow(img2)

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

plt.title('cancer image')

ax3.imshow(img3)

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

plt.title('cancer image')

ax4.imshow(img4)

# Get the class distribution

labels = ["Non-Cancer", "Cancer"]

counts = [len(NCdata), len(Cdata)]

total\_samples = sum(counts)

percentages = [(count / total\_samples) \* 100 for count in counts]

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

plt.bar(labels, counts)

plt.xlabel("Class")

plt.ylabel("Count")

plt.title("Class Distribution")

plt.xticks(rotation=45)

plt.show()

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

plt.pie(percentages, labels=labels, autopct='%1.1f%%', startangle=140)

plt.title("Class Distribution (Percentage)")

plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()

# Resizing using PIL Image

desired\_size = (50,50)

resizedNC = []

resizedC = []

for image\_path in NCdata:

image = Image.open(image\_path)

nimage = image.resize(desired\_size, Image.LANCZOS) # Resize with anti-aliasing for better quality

resizedNC.append(nimage)

for image\_path in Cdata:

image = Image.open(image\_path)

cimage = image.resize(desired\_size, Image.LANCZOS) # Resize with anti-aliasing for better quality

resizedC.append(cimage)

# Normalize the Dataset pixel values to [0, 1] range

NCdataset = np.array([np.array(image) / 255.0 for image in resizedNC])

Cdataset = np.array([np.array(image) / 255.0 for image in resizedC])

# Shuffle the dataset

NCdataset = shuffle(NCdataset, random\_state=42)

Cdataset = shuffle(Cdataset, random\_state=42)

# Get the Shape of all dataset

print('NCdataset shape : {}' .format(NCdataset.shape))

print('Cdataset shape : {}' .format(Cdataset.shape))

# Split the data

# Split each dataset into training data and temporary data - 70:30

NCtrain, NCtemp, NCtrain\_labels, NCtemp\_labels = train\_test\_split(

NCdataset, NClabels, test\_size=0.3, stratify=NClabels, random\_state=42

)

# Split the Cancer data

Ctrain, Ctemp, Ctrain\_labels, Ctemp\_labels = train\_test\_split(

Cdataset, Clabels, test\_size=0.3, stratify=Clabels, random\_state=42

)

# Use the temporary data to split into Validation and Testing Data - 15:15

NCval, NCtest, NCval\_labels, NCtest\_labels = train\_test\_split(

NCtemp, NCtemp\_labels, test\_size=0.5, stratify=NCtemp\_labels, random\_state=42

)

Cval, Ctest, Cval\_labels, Ctest\_labels = train\_test\_split(

Ctemp, Ctemp\_labels, test\_size=0.5, stratify=Ctemp\_labels, random\_state=42

)

# Combine the two Non-Cancer Data and the Cancer Data to make one train\_data, val\_data, test\_data

train\_data = np.concatenate((NCtrain, Ctrain), axis=0)

train\_labels = np.concatenate((NCtrain\_labels, Ctrain\_labels), axis=0)

val\_data = np.concatenate((NCval, Cval), axis=0)

val\_labels = np.concatenate((NCval\_labels, Cval\_labels), axis=0)

test\_data = np.concatenate((NCtest, Ctest), axis=0)

test\_labels = np.concatenate((NCtest\_labels, Ctest\_labels), axis=0)

# Reformat the shape for the labels

train\_labels = to\_categorical(train\_labels, 2)

val\_labels = to\_categorical(val\_labels, 2)

test\_labels = to\_categorical(test\_labels, 2)

print('train\_data shape : {}' .format(train\_data.shape))

print('train\_labels shape : {}' .format(train\_labels.shape))

print('val\_data shape : {}' .format(val\_data.shape))

print('val\_labels shape : {}' .format(val\_labels.shape))

print('test\_data shape : {}' .format(test\_data.shape))

print('test\_labels shape : {}' .format(test\_labels.shape))

model = tf.keras.Sequential([

# Convolutional Layers

tf.keras.layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu', input\_shape = (50, 50, 3)),

tf.keras.layers.MaxPooling2D(strides = 2),

tf.keras.layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu'),

tf.keras.layers.MaxPooling2D((3, 3),strides = 2),

tf.keras.layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu'),

tf.keras.layers.MaxPooling2D((3, 3),strides =2),

tf.keras.layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu'),

tf.keras.layers.MaxPooling2D((3, 3),strides =2),

# Flatten Layer

tf.keras.layers.Flatten(),

# Fully Connected Layers

tf.keras.layers.Dense(128, activation = 'relu'),

tf.keras.layers.Dense(2, activation = 'softmax')

])

model.summary()

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001),loss='binary\_crossentropy',metrics=['accuracy'])

history = model.fit(train\_data, train\_labels, validation\_data = (val\_data, val\_labels), epochs = 80, batch\_size = 75) #25

model.evaluate(test\_data,test\_labels)

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

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

plt.title('Model Accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

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

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

plt.title('Model Loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

predict\_data = model.predict(test\_data)

predict\_labels = np.argmax(predict\_data, axis=1)

def convert\_to\_single\_label(one\_hot\_labels):

return np.argmax(one\_hot\_labels, axis=1)

# Convert train\_labels

true\_train\_labels = convert\_to\_single\_label(train\_labels)

# Convert val\_labels

true\_val\_labels = convert\_to\_single\_label(val\_labels)

# Convert test\_labels

true\_test\_labels = convert\_to\_single\_label(test\_labels)

import numpy as np

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

# Calculate accuracy

accuracy = accuracy\_score(true\_test\_labels, predict\_labels)

print(f'Accuracy: {accuracy:.2f}')

# Calculate precision

precision = precision\_score(true\_test\_labels, predict\_labels)

print(f'Precision: {precision:.2f}')

# Calculate recall

recall = recall\_score(true\_test\_labels, predict\_labels)

print(f'Recall: {recall:.2f}')

# Calculate F1-score

f1 = f1\_score(true\_test\_labels, predict\_labels)

print(f'F1-score: {f1:.2f}')

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(true\_test\_labels, predict\_labels)

f,ax = plt.subplots(figsize=(8, 8))

sns.heatmap(conf\_matrix, annot=True, linewidths=0.01,cmap="BuPu",linecolor="gray", fmt= '.1f',ax=ax)

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title("Confusion Matrix")

plt.show()

import numpy as np

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

# Calculate accuracy

accuracy = round(accuracy\_score(true\_test\_labels, predict\_labels),2)

# Calculate precision

precision = round(precision\_score(true\_test\_labels, predict\_labels),2)

# Calculate recall

recall =round(recall\_score(true\_test\_labels, predict\_labels),2)

# Calculate F1-score

f1 = round(f1\_score(true\_test\_labels, predict\_labels),2)

# 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'],[accuracy,precision,recall,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')

#!pip install scikit-learn matplotlib tensorflow

from sklearn.metrics import roc\_curve, auc

# Compute ROC curve and area under the curve (AUC)

fpr, tpr, thresholds = roc\_curve(true\_test\_labels, predict\_labels)

roc\_auc = auc(fpr, tpr)

# Plot the ROC curve

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

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.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) Curve for Binary breast cancer classification')

plt.legend(loc='lower right')

plt.show()