Resnet with attention layer

Code

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_auc\_score, roc\_curve, precision\_recall\_curve, f1\_score, auc

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.applications import ResNet50

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

from tensorflow.keras.metrics import BinaryAccuracy, Recall, Precision

from tensorflow.keras.optimizers import Adam

tf.keras.backend.clear\_session()

data = keras.utils.image\_dataset\_from\_directory("assets/brain\_tumor\_dataset")

batch = data.as\_numpy\_iterator().next()

fig, ax = plt.subplots(3, 5, figsize=(15, 10))

ax = ax.flatten()

for idx, img in enumerate(batch[0][:15]):

ax[idx].imshow(img.astype(int))

ax[idx].title.set\_text(batch[1][idx])

data = data.map(lambda x, y: (x / 255, y))

batch = data.as\_numpy\_iterator().next()

print("Minimum value of the scaled data:", batch[0].min())

print("Maximum value of the scaled data:", batch[0].max())

print("There are", len(data), "batches in our data")

train\_size = int(len(data) \* 0.6)

val\_size = int(len(data) \* 0.2) + 1

test\_size = int(len(data) \* 0.2) + 1

print("Train Size:", train\_size)

print("Validation Size:", val\_size)

print("Test Size:", test\_size)

print("Sum of Train, Validation, and Test sizes is equal to:", train\_size + val\_size + test\_size)

train = data.take(train\_size)

val = data.skip(train\_size).take(val\_size)

test = data.skip(train\_size + val\_size).take(test\_size)

batch = data.as\_numpy\_iterator().next()

# Using ResNet-50 as the base model

base\_model = ResNet50(weights="imagenet", include\_top=False, input\_shape=(256, 256, 3))

# Freeze the layers of the pre-trained ResNet-50 model

for layer in base\_model.layers:

layer.trainable = False

# Create an attention layer

attention = Attention()([base\_model.output, base\_model.output])

# Flatten layer

flat = Flatten()(attention)

# Dense layers for classification

dense1 = Dense(128, activation="relu")(flat)

output = Dense(1, activation="sigmoid")(dense1)

# Build the final model

model = keras.Model(inputs=base\_model.input, outputs=output)

model.compile(optimizer=Adam(), loss=keras.losses.BinaryCrossentropy(), metrics=["accuracy"])

model.summary()

history = model.fit(train, epochs=35, validation\_data=val)

fig, ax = plt.subplots(2, 1, figsize=(10, 8))

ax[0].plot(history.history["loss"], label="Train")

ax[0].plot(history.history["val\_loss"], label="Validation")

ax[0].title.set\_text("Loss")

ax[0].legend()

ax[1].plot(history.history["accuracy"], label="Train")

ax[1].plot(history.history["val\_accuracy"], label="Validation")

ax[1].title.set\_text("Accuracy")

ax[1].legend()

plt.show()

bin\_acc = BinaryAccuracy()

recall = Recall()

precision = Precision()

y\_true\_list = []

y\_pred\_list = []

for batch in test.as\_numpy\_iterator():

X, y = batch

yhat = model.predict(X)

bin\_acc.update\_state(y, yhat)

recall.update\_state(y, yhat)

precision.update\_state(y, yhat)

y\_true\_list.append(y)

y\_pred\_list.append(yhat)

y\_true = np.concatenate(y\_true\_list, axis=0)

y\_pred = np.concatenate(y\_pred\_list, axis=0)

# Calculate F1 Score

f1 = f1\_score(y\_true, (y\_pred > 0.5).astype(int))

# Calculate ROC-AUC score

roc\_auc = roc\_auc\_score(y\_true, y\_pred)

# Calculate ROC curve

fpr, tpr, \_ = roc\_curve(y\_true, y\_pred)

# Calculate precision-recall curve

precision, recall, \_ = precision\_recall\_curve(y\_true, y\_pred)

# Print all metrics

print("Accuracy:", bin\_acc.result().numpy())

print("Recall:", recall.result().numpy())

print("Precision:", precision.result().numpy())

print("F1 Score:", f1)

print("ROC-AUC Score:", roc\_auc)

# Plot ROC curve

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

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')

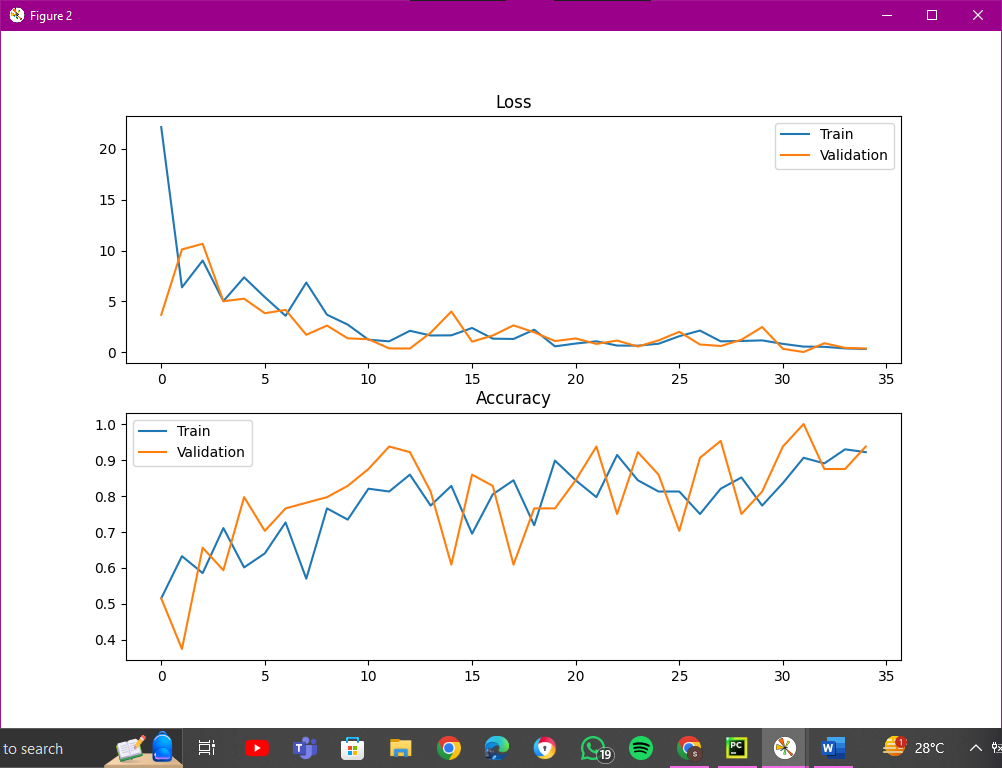
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend()

plt.show()



Evaluation Metrics

Accuracy: 0.93442625

Recall: 1.0

Precision: 0.9111111

F1 Score: 0.9534883720930233

ROC-AUC Score: 0.998780487804878

