Code:

import tensorflow as tf

import tensorflow\_hub as hub

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

from tensorflow.keras.utils import get\_file

from sklearn.metrics import roc\_curve, auc, confusion\_matrix

from imblearn.metrics import sensitivity\_score, specificity\_score

import os

import glob

import zipfile

import random

# to get consistent results after multiple runs

tf.random.set\_seed(7)

np.random.seed(7)

random.seed(7)

# 0 for benign, 1 for malignant

class\_names = ["benign", "malignant"]

def download\_and\_extract\_dataset():

  # dataset from https://github.com/udacity/dermatologist-ai

  # 5.3GB

  train\_url = "https://s3-us-west-1.amazonaws.com/udacity-dlnfd/datasets/skin-cancer/train.zip"

  # 824.5MB

  valid\_url = "https://s3-us-west-1.amazonaws.com/udacity-dlnfd/datasets/skin-cancer/valid.zip"

  # 5.1GB

  test\_url  = "https://s3-us-west-1.amazonaws.com/udacity-dlnfd/datasets/skin-cancer/test.zip"

  for i, download\_link in enumerate([valid\_url, train\_url, test\_url]):

    temp\_file = f"temp{i}.zip"

    data\_dir = get\_file(origin=download\_link, fname=os.path.join(os.getcwd(), temp\_file))

    print("Extracting", download\_link)

    with zipfile.ZipFile(data\_dir, "r") as z:

      z.extractall("data")

    # remove the temp file

    os.remove(temp\_file)

# comment the below line if you already downloaded the dataset

download\_and\_extract\_dataset()



# preparing data

# generate CSV metadata file to read img paths and labels from it

def generate\_csv(folder, label2int):

    folder\_name = os.path.basename(folder)

    labels = list(label2int)

    # generate CSV file

    df = pd.DataFrame(columns=["filepath", "label"])

    i = 0

    for label in labels:

        print("Reading", os.path.join(folder, label, "\*"))

        for filepath in glob.glob(os.path.join(folder, label, "\*")):

            df.loc[i] = [filepath, label2int[label]]

            i += 1

    output\_file = f"{folder\_name}.csv"

    print("Saving", output\_file)

    df.to\_csv(output\_file)

# generate CSV files for all data portions, labeling nevus and seborrheic keratosis

# as 0 (benign), and melanoma as 1 (malignant)

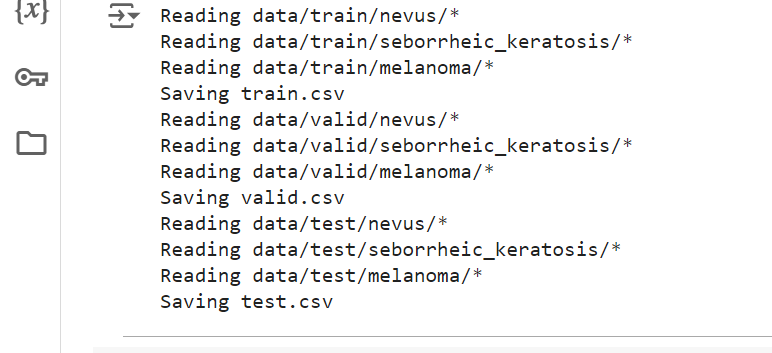
# you should replace "data" path to your extracted dataset path

# don't replace if you used download\_and\_extract\_dataset() function

generate\_csv("data/train", {"nevus": 0, "seborrheic\_keratosis": 0, "melanoma": 1})

generate\_csv("data/valid", {"nevus": 0, "seborrheic\_keratosis": 0, "melanoma": 1})

generate\_csv("data/test", {"nevus": 0, "seborrheic\_keratosis": 0, "melanoma": 1})



# loading data

train\_metadata\_filename = "train.csv"

valid\_metadata\_filename = "valid.csv"

# load CSV files as DataFrames

df\_train = pd.read\_csv(train\_metadata\_filename)

df\_valid = pd.read\_csv(valid\_metadata\_filename)

n\_training\_samples = len(df\_train)

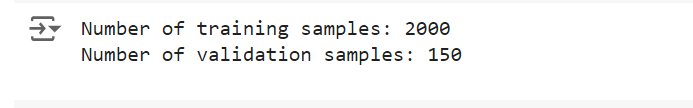
n\_validation\_samples = len(df\_valid)

print("Number of training samples:", n\_training\_samples)

print("Number of validation samples:", n\_validation\_samples)

train\_ds = tf.data.Dataset.from\_tensor\_slices((df\_train["filepath"], df\_train["label"]))

valid\_ds = tf.data.Dataset.from\_tensor\_slices((df\_valid["filepath"], df\_valid["label"]))



# preprocess data

def decode\_img(img):

  # convert the compressed string to a 3D uint8 tensor

  img = tf.image.decode\_jpeg(img, channels=3)

  # Use `convert\_image\_dtype` to convert to floats in the [0,1] range.

  img = tf.image.convert\_image\_dtype(img, tf.float32)

  # resize the image to the desired size.

  return tf.image.resize(img, [299, 299])

def process\_path(filepath, label):

  # load the raw data from the file as a string

  img = tf.io.read\_file(filepath)

  img = decode\_img(img)

  return img, label

valid\_ds = valid\_ds.map(process\_path)

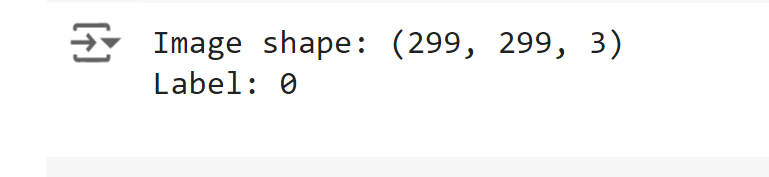
train\_ds = train\_ds.map(process\_path)

# test\_ds = test\_ds

for image, label in train\_ds.take(1):

    print("Image shape:", image.shape)

    print("Label:", label.numpy())



# training parameters

batch\_size = 64

optimizer = "rmsprop"

def prepare\_for\_training(ds, cache=True, batch\_size=64, shuffle\_buffer\_size=1000):

  if cache:

    if isinstance(cache, str):

      ds = ds.cache(cache)

    else:

      ds = ds.cache()

  # shuffle the dataset

  ds = ds.shuffle(buffer\_size=shuffle\_buffer\_size)

  # Repeat forever

  ds = ds.repeat()

  # split to batches

  ds = ds.batch(batch\_size)

  # `prefetch` lets the dataset fetch batches in the background while the model

  # is training.

  ds = ds.prefetch(buffer\_size=tf.data.experimental.AUTOTUNE)

  return ds

valid\_ds = prepare\_for\_training(valid\_ds, batch\_size=batch\_size, cache="valid-cached-data")

train\_ds = prepare\_for\_training(train\_ds, batch\_size=batch\_size, cache="train-cached-data")

batch = next(iter(valid\_ds))

def show\_batch(batch):

  plt.figure(figsize=(12,12))

  for n in range(25):

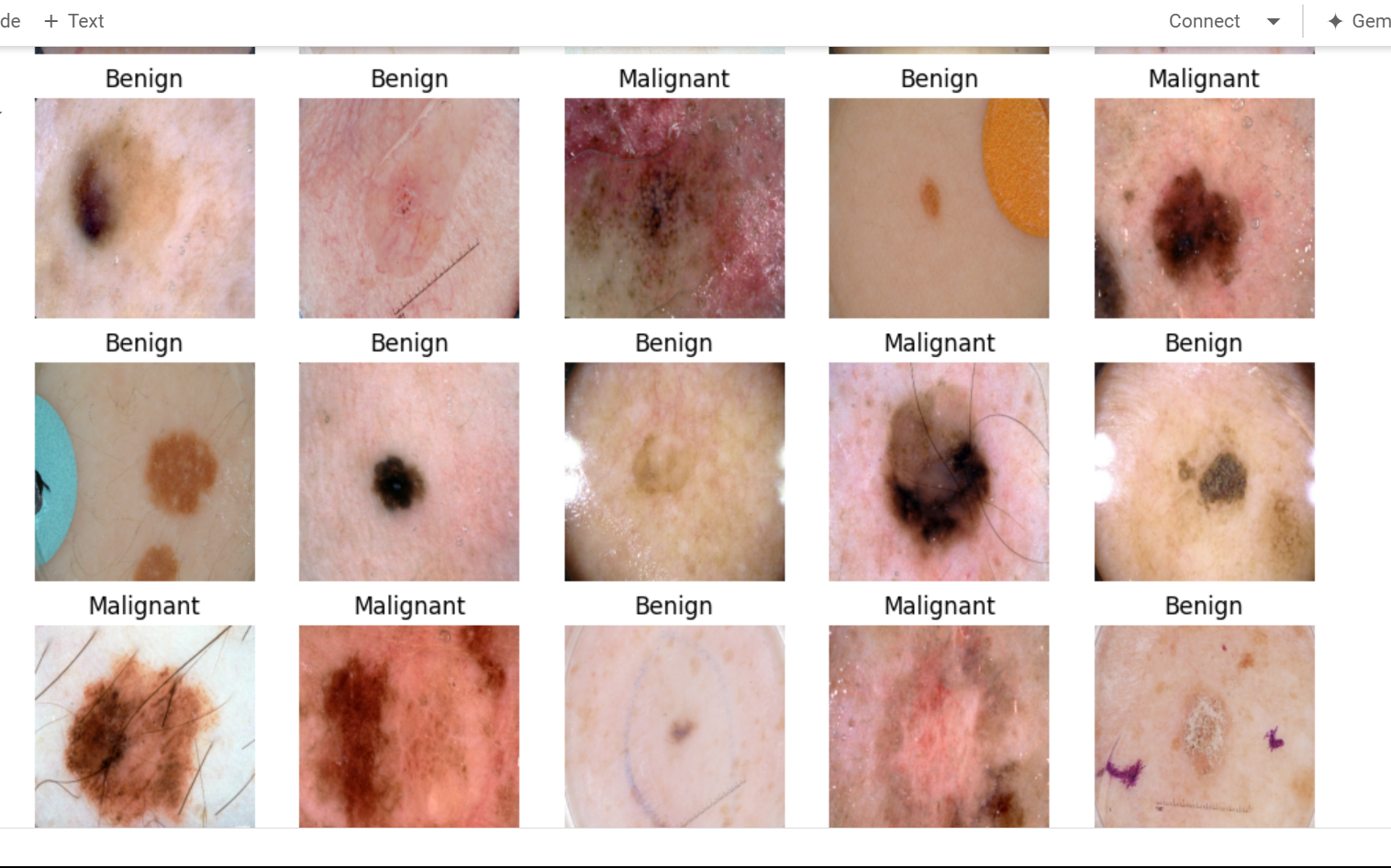
      ax = plt.subplot(5,5,n+1)

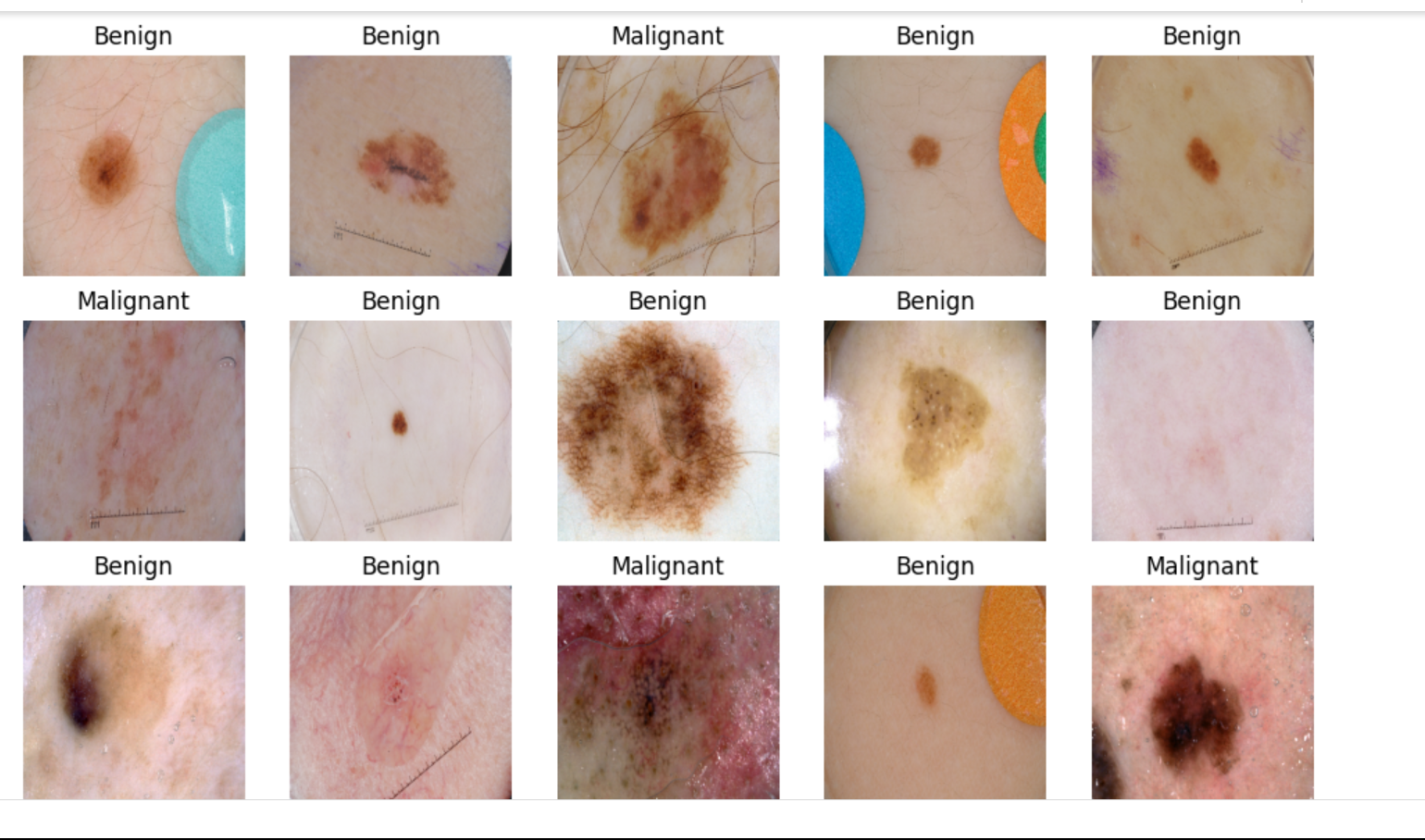
      plt.imshow(batch[0][n])

      plt.title(class\_names[batch[1][n].numpy()].title())

      plt.axis('off')

show\_batch(batch)





# building the model

# InceptionV3 model & pre-trained weights

module\_url = "https://tfhub.dev/google/tf2-preview/inception\_v3/feature\_vector/4"

m = tf.keras.Sequential([

    hub.KerasLayer(module\_url, output\_shape=[2048], trainable=False),

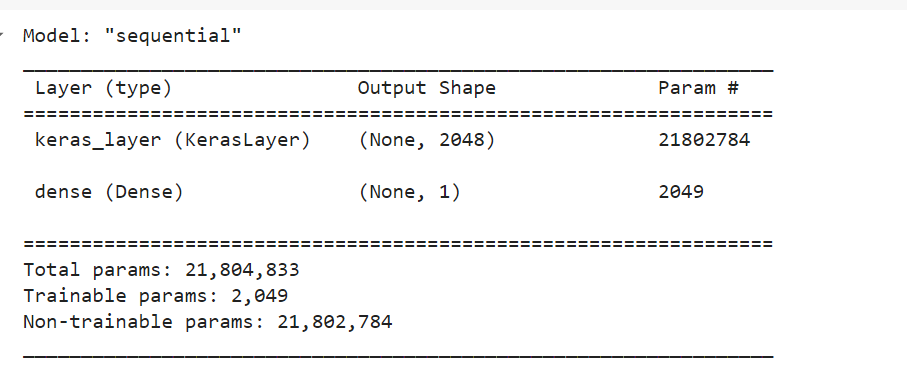
    tf.keras.layers.Dense(1, activation="sigmoid")

])

m.build([None, 299, 299, 3])

m.compile(loss="binary\_crossentropy", optimizer=optimizer, metrics=["accuracy"])

m.summary()



model\_name = f"benign-vs-malignant\_{batch\_size}\_{optimizer}"

tensorboard = tf.keras.callbacks.TensorBoard(log\_dir=os.path.join("logs", model\_name))

# saves model checkpoint whenever we reach better weights

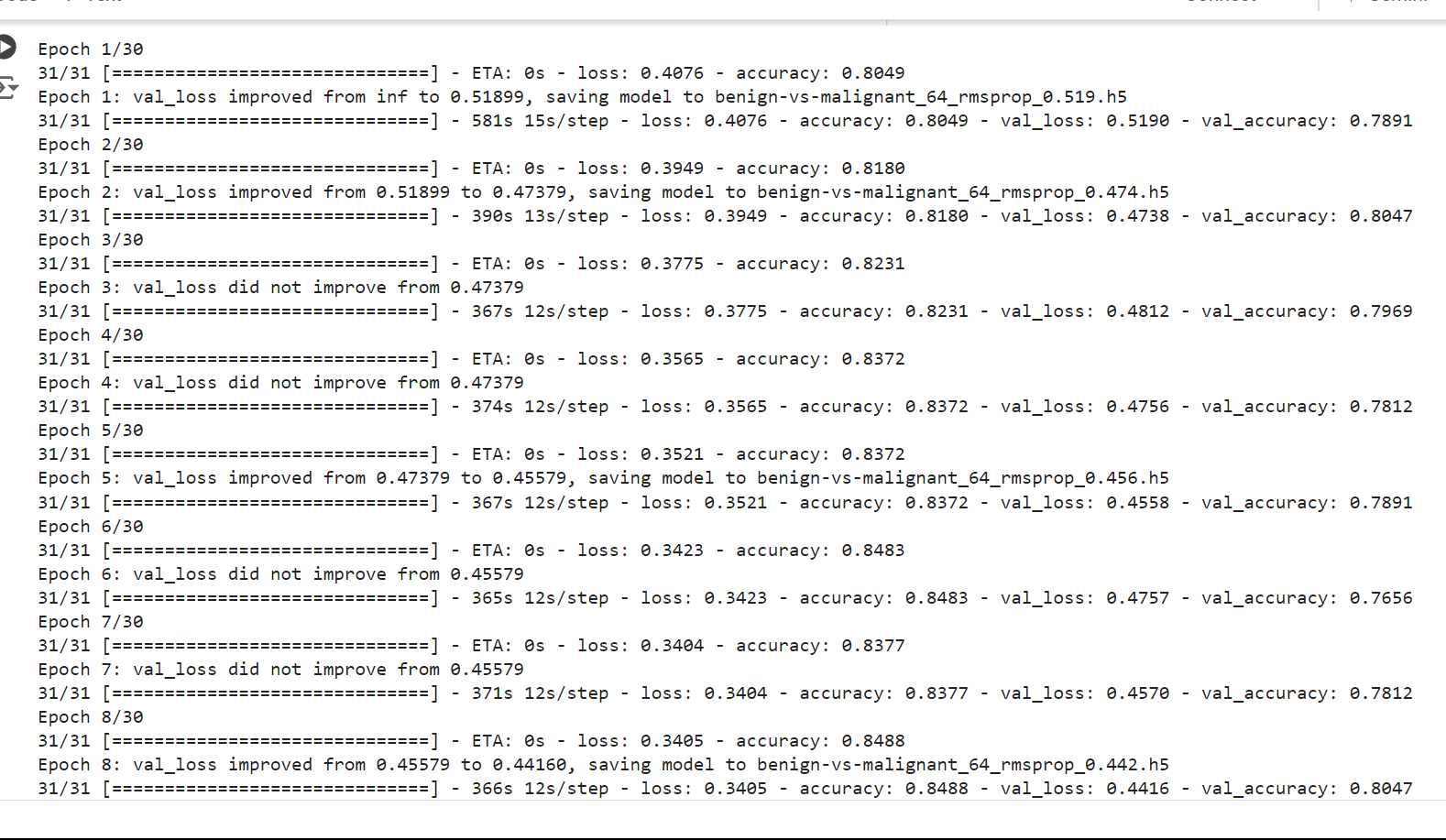
modelcheckpoint = tf.keras.callbacks.ModelCheckpoint(model\_name + "\_{val\_loss:.3f}.h5", save\_best\_only=True, verbose=1)

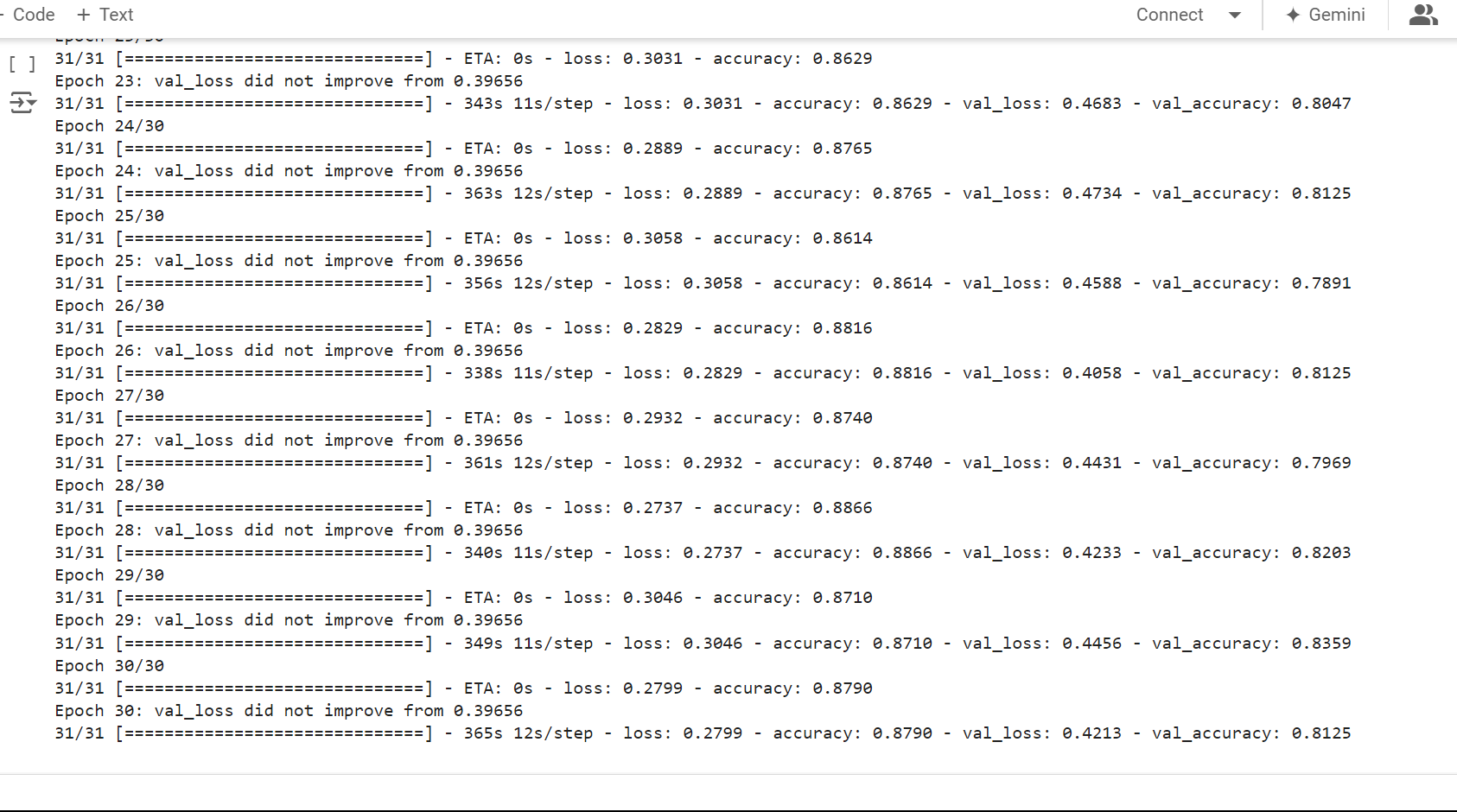
history = m.fit(train\_ds, validation\_data=valid\_ds,

                steps\_per\_epoch=n\_training\_samples // batch\_size,

                validation\_steps=n\_validation\_samples // batch\_size, verbose=1, epochs=65,

                callbacks=[tensorboard, modelcheckpoint])





# convert testing set to numpy array to fit in memory (don't do that when testing

# set is too large)

y\_test = np.zeros((n\_testing\_samples,))

X\_test = np.zeros((n\_testing\_samples, 299, 299, 3))

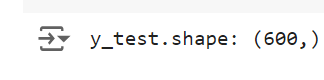
for i, (img, label) in enumerate(test\_ds.take(n\_testing\_samples)):

  # print(img.shape, label.shape)

  X\_test[i] = img

  y\_test[i] = label.numpy()

print("y\_test.shape:", y\_test.shape)



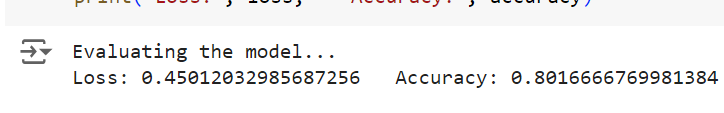
# load the weights with the least loss

m.load\_weights("benign-vs-malignant\_64\_rmsprop\_0.396.h5")

print("Evaluating the model...")

loss, accuracy = m.evaluate(X\_test, y\_test, verbose=0)

print("Loss:", loss, "  Accuracy:", accuracy)



def get\_predictions(threshold=None):

  """

  Returns predictions for binary classification given `threshold`

  For instance, if threshold is 0.3, then it'll output 1 (malignant) for that sample if

  the probability of 1 is 30% or more (instead of 50%)

  """

  y\_pred = m.predict(X\_test)

  if not threshold:

    threshold = 0.5

  result = np.zeros((n\_testing\_samples,))

  for i in range(n\_testing\_samples):

    # test melanoma probability

    if y\_pred[i][0] >= threshold:

      result[i] = 1

    # else, it's 0 (benign)

  return result

threshold = 0.23

# get predictions with 23% threshold

# which means if the model is 23% sure or more that is malignant,

# it's assigned as malignant, otherwise it's benign

y\_pred = get\_predictions(threshold)

def plot\_confusion\_matrix(y\_test, y\_pred):

  cmn = confusion\_matrix(y\_test, y\_pred)

  # Normalise

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

  # print it

  print(cmn)

  fig, ax = plt.subplots(figsize=(10,10))

  sns.heatmap(cmn, annot=True, fmt='.2f',

              xticklabels=[f"pred\_{c}" for c in class\_names],

              yticklabels=[f"true\_{c}" for c in class\_names],

              cmap="Blues"

              )

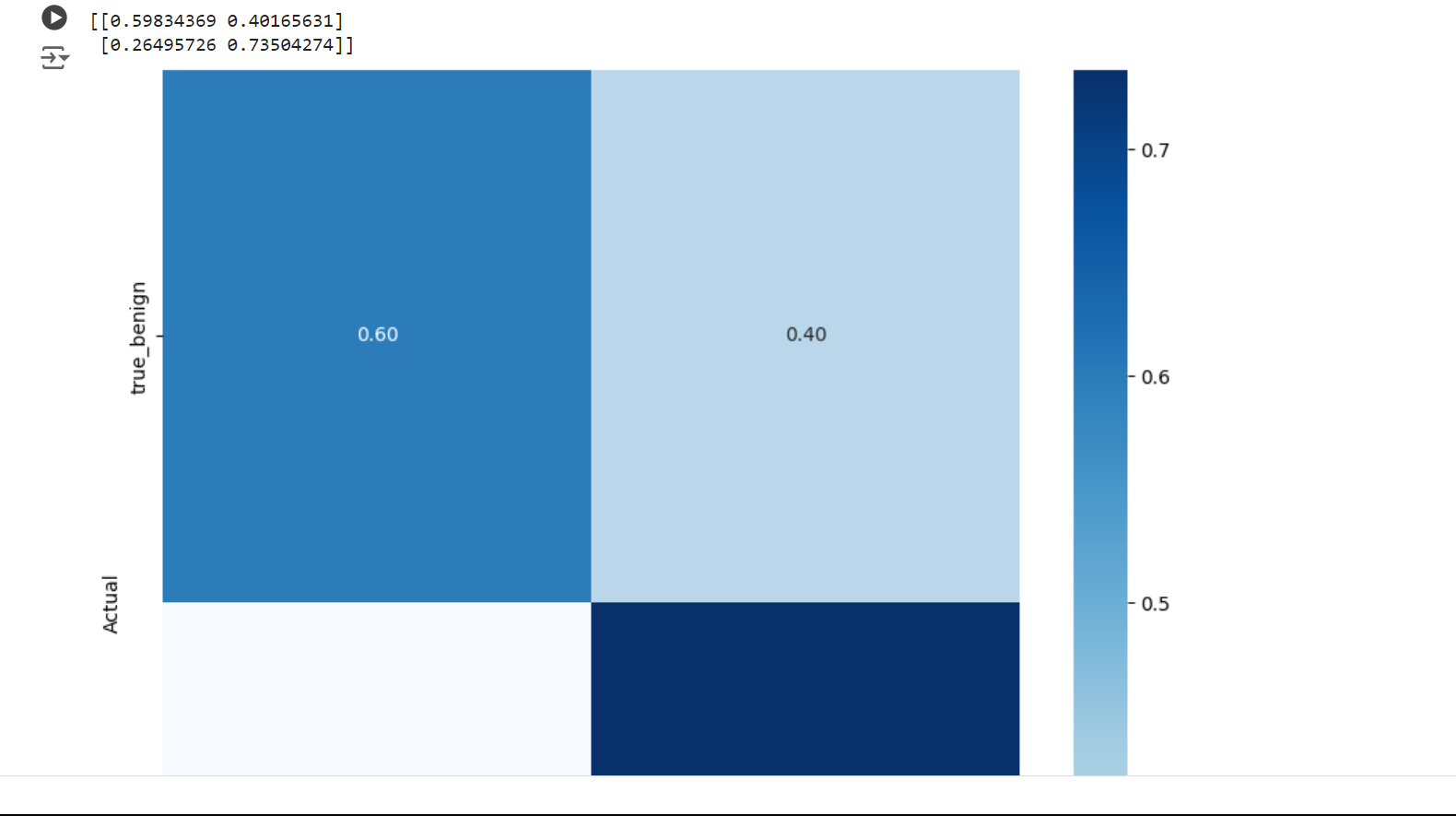
  plt.ylabel('Actual')

  plt.xlabel('Predicted')

  # plot the resulting confusion matrix

  plt.show()

plot\_confusion\_matrix(y\_test, y\_pred)



sensitivity = sensitivity\_score(y\_test, y\_pred)

specificity = specificity\_score(y\_test, y\_pred)

print("Melanoma Sensitivity:", sensitivity)

print("Melanoma Specificity:", specificity)



def plot\_roc\_auc(y\_true, y\_pred):

    """

    This function plots the ROC curves and provides the scores.

    """

    # prepare for figure

    plt.figure()

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

    # obtain ROC AUC

    roc\_auc = auc(fpr, tpr)

    # print score

    print(f"ROC AUC: {roc\_auc:.3f}")

    # plot ROC curve

    plt.plot(fpr, tpr, color="blue", lw=2,

                label='ROC curve (area = {f:.2f})'.format(d=1, f=roc\_auc))

    plt.xlim([0.0, 1.0])

    plt.ylim([0.0, 1.05])

    plt.xlabel('False Positive Rate')

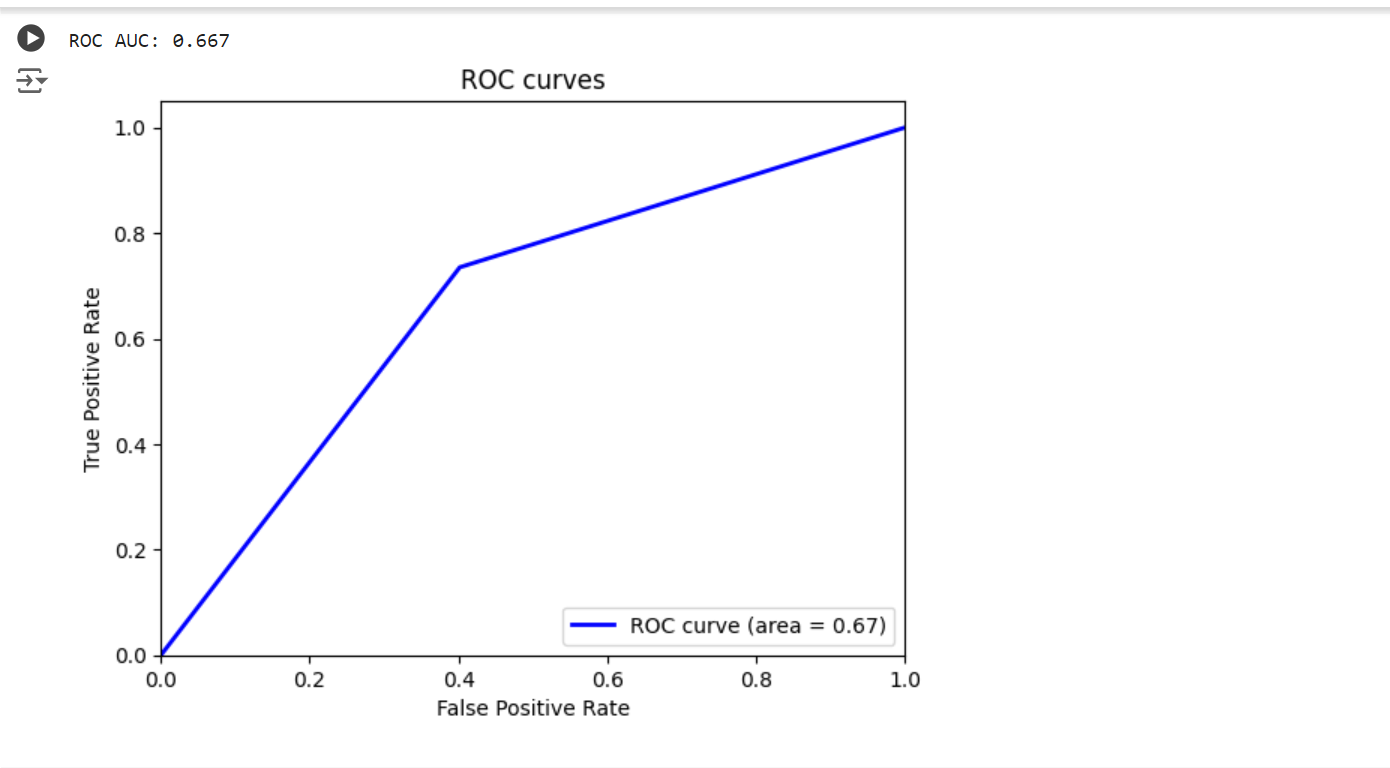
    plt.ylabel('True Positive Rate')

    plt.title('ROC curves')

    plt.legend(loc="lower right")

    plt.show()

plot\_roc\_auc(y\_test, y\_pred)



# a function given a function, it predicts the class of the image

def predict\_image\_class(img\_path, model, threshold=0.5):

  img = tf.keras.preprocessing.image.load\_img(img\_path, target\_size=(299, 299))

  img = tf.keras.preprocessing.image.img\_to\_array(img)

  img = tf.expand\_dims(img, 0) # Create a batch

  img = tf.keras.applications.inception\_v3.preprocess\_input(img)

  img = tf.image.convert\_image\_dtype(img, tf.float32)

  predictions = model.predict(img)

  score = predictions.squeeze()

  if score >= threshold:

    print(f"This image is {100 \* score:.2f}% malignant.")

  else:

    print(f"This image is {100 \* (1 - score):.2f}% benign.")

  plt.imshow(img[0])

  plt.axis('off')

  plt.show()

predict\_image\_class("data/test/melanoma/ISIC\_0013767.jpg", m)

predict\_image\_class("data/test/seborrheic\_keratosis/ISIC\_0012136.jpg", m)

