Loading & Preprocessing

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /content/drive
```

```
!git clone https://github.com/abhinavsagar/breast.git
```

```
Cloning into 'breast'...
remote: Enumerating objects: 1198, done.
remote: Counting objects: 100% (1198/1198), done.
remote: Compressing objects: 100% (1196/1196), done.
remote: Total 1198 (delta 1), reused 1192 (delta 0), pack-reused 0
Receiving objects: 100% (1198/1198), 569.51 MiB | 15.31 MiB/s, done.
Resolving deltas: 100% (1/1), done.
Checking out files: 100% (1186/1186), done.
```

```
cd "drive/My Drive/"
```

```
/content/drive/My Drive
```

```
import json
import math
import os

import cv2
from PIL import Image
import numpy as np
from keras import layers
from keras.applications import ResNet50,MobileNet, DenseNet201, InceptionV3, NASNetLarge, Inc
from keras.callbacks import Callback, ModelCheckpoint, ReduceLROnPlateau, TensorBoard
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.optimizers import Adam
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, accuracy_score
import scipy
from tqdm import tqdm
import tensorflow as tf
from keras import backend as K
import gc
from functools import partial
from sklearn import metrics
from collections import Counter
import json
import itertools
import numpy as np

%matplotlib inline
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you [upgrade](https://www.tensorflow.org/guide/migrate) now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: [more info](https://colab.research.google.com/notebooks/tensorflow_version.ipynb).

```
!ls "/content"
```

```
drive  sample_data
```

```
#Transfer 'jpg' images to an array IMG
def Dataset_loader(DIR, RESIZE, sigmaX=10):
    IMG = []
    read = lambda imname: np.asarray(Image.open(imname).convert("RGB")) ## Lambda Function tc
    for IMAGE_NAME in tqdm(os.listdir(DIR)):
        PATH = os.path.join(DIR,IMAGE_NAME)  ## Generate path for each image in a folder
        _, ftype = os.path.splitext(PATH)
        if ftype == ".png":
            img = read(PATH)

            img = cv2.resize(img, (RESIZE,RESIZE))

            IMG.append(np.array(img))
    return IMG

benign_train = np.array(Dataset_loader('./Colab Notebooks/data/training/benign',224))
malign_train = np.array(Dataset_loader('./Colab Notebooks/data/training/malign',224))
# benign_test = np.array(Dataset_loader('data/validation/benign',224))
# malign_test = np.array(Dataset_loader('data/validation/malignant',224))
```

```
100%|██████████| 621/621 [03:27<00:00,  2.59it/s]
100%|██████████| 1138/1138 [06:11<00:00,  2.85it/s]
```

Create Label

```
# Skin Cancer: Malignant vs. Benign
# Create labels
benign_train_label = np.zeros(len(benign_train))
malign_train_label = np.ones(len(malign_train))
# benign_test_label = np.zeros(len(benign_test))
# malign_test_label = np.ones(len(malign_test))

# Merge data
X_train = np.concatenate((benign_train, malign_train), axis = 0)
Y_train = np.concatenate((benign_train_label, malign_train_label), axis = 0)
# X_test = np.concatenate((benign_test, malign_test), axis = 0)
# Y_test = np.concatenate((benign_test_label, malign_test_label), axis = 0)

# Shuffle train data
s = np.arange(X_train.shape[0])
np.random.shuffle(s)
X_train = X_train[s]
Y_train = Y_train[s]

# Shuffle test data
# s = np.arange(X_test.shape[0])
# np.random.shuffle(s)
# X_test = X_test[s]
# Y_test = Y_test[s]

# To categorical
Y_train = to_categorical(Y_train, num_classes= 2)
# Y_test = to_categorical(Y_test, num_classes= 2)


y = np.array([1,0,1,1,1,0,0,0])
y_cat = to_categorical(y,num_classes=2)
print(y_cat)
```

```
[[0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

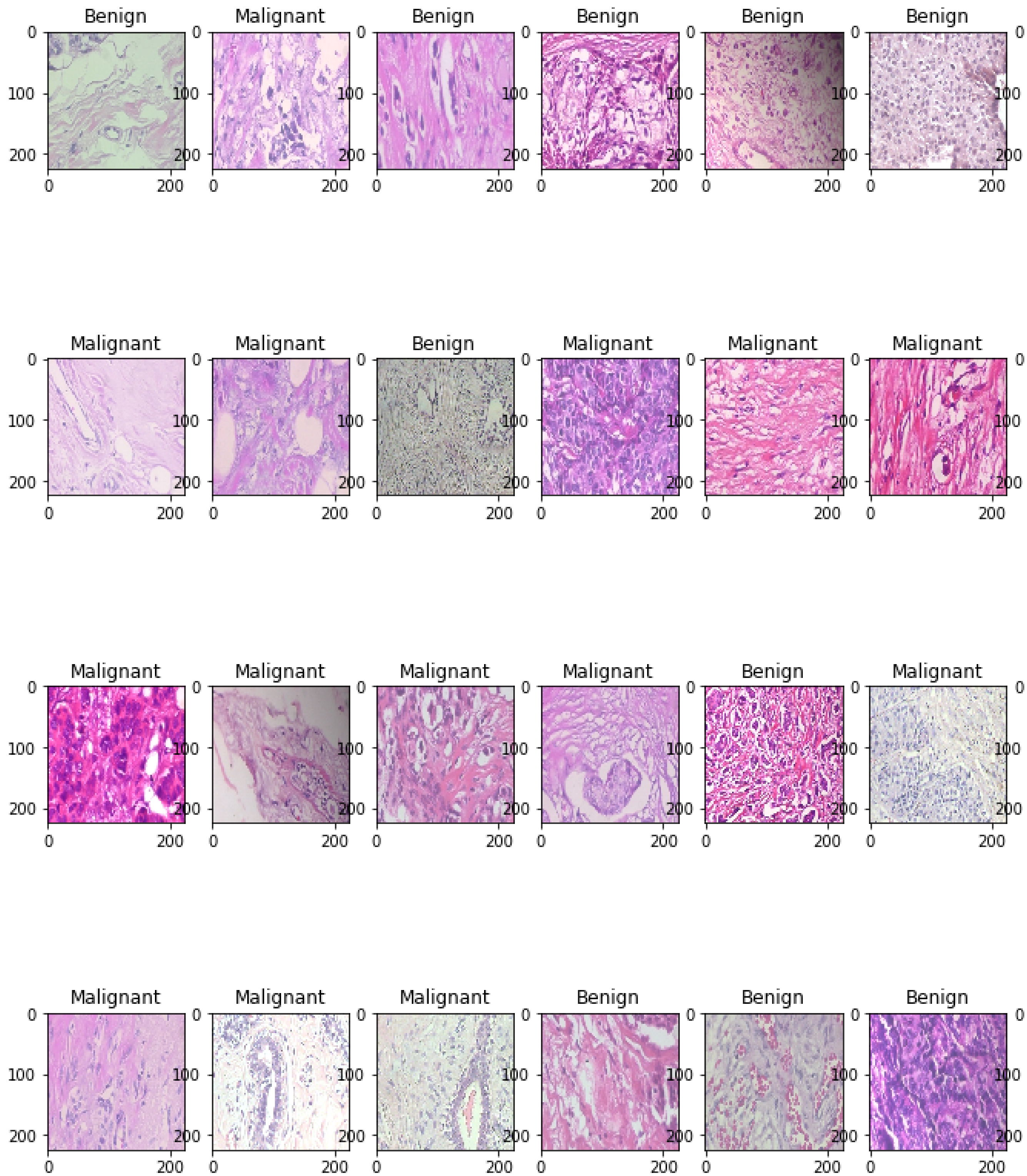
Train and Evalutation split

```
x_train, x_val, y_train, y_val = train_test_split(
    X_train, Y_train,
    test_size=0.2,
    random_state=11
)
```

```
)
```

Display Some Images

```
# # Display first 15 images of moles, and how they are classified
w=60
h=40
fig=plt.figure(figsize=(15, 15))
columns = 8
rows = 4

for i in range(1, columns*rows +1):
    ax = fig.add_subplot(rows, columns, i)
    if np.argmax(Y_train[i]) == 0:
        ax.title.set_text('Benign')
    else:
        ax.title.set_text('Malignant')
    plt.imshow(x_train[i], interpolation='nearest')
plt.show()
```



Data Generator

```
BATCH_SIZE = 16

# Using original generator
train_generator = ImageDataGenerator(
```

```
        zoom_range=2,  # set range for random zoom
        rotation_range = 90,
        horizontal_flip=True,  # randomly flip images
        vertical_flip=True,  # randomly flip images
    )
```

Model: ResNet50

```
def build_model(backbone, lr=1e-4):
    model = Sequential()
    model.add(backbone)
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Dropout(0.5))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(2, activation='softmax'))


    model.compile(
        loss='binary_crossentropy',
        optimizer=Adam(lr=lr),
        metrics=['accuracy']
    )

    return model


K.clear_session()
gc.collect()

resnet = InceptionV3(
    weights='imagenet',
    include_top=False,
    input_shape=(224,224,3)
)
```

```
model = build_model(resnet ,lr = 1e-4)
model.summary()
```

```
# Learning Rate Reducer
learn_control = ReduceLROnPlateau(monitor='val_acc', patience=5,
                                  verbose=1,factor=0.2, min_lr=1e-7)

# Checkpoint
filepath="weights.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, moc
```

Training & Evaluation

```
history = model.fit_generator(
    train_generator.flow(x_train, y_train, batch_size=BATCH_SIZE),
    steps_per_epoch=x_train.shape[0] / BATCH_SIZE,
    epochs=20,
    validation_data=(x_val, y_val),
    callbacks=[learn_control, checkpoint]
)
```

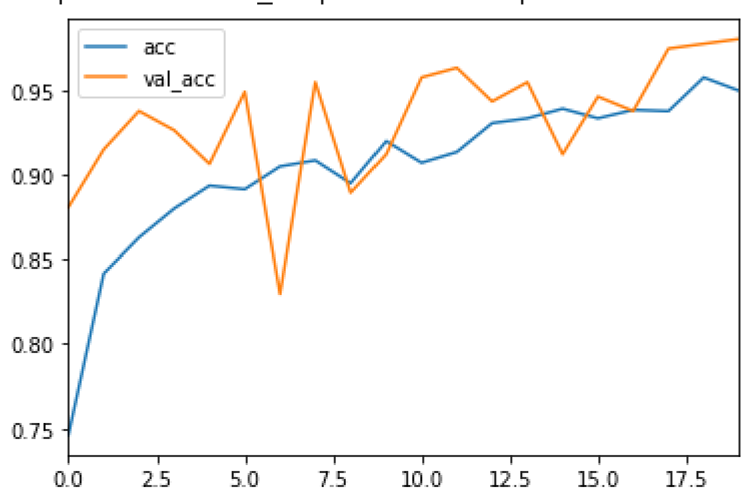
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

Epoch 1/20
88/87 [==============================] - 175s 2s/step - loss: 0.5511 - acc: 0.7464 - val

Epoch 00001: val_acc improved from -inf to 0.88068, saving model to weights.best.hdf5
Epoch 2/20
88/87 [==============================] - 78s 883ms/step - loss: 0.3892 - acc: 0.8414 - v

Epoch 00002: val_acc improved from 0.88068 to 0.91477, saving model to weights.best.hdf5
Epoch 3/20
88/87 [==============================] - 78s 885ms/step - loss: 0.3307 - acc: 0.8628 - v

Epoch 00003: val_acc improved from 0.91477 to 0.93750, saving model to weights.best.hdf5
Epoch 4/20
88/87 [==============================] - 78s 886ms/step - loss: 0.2960 - acc: 0.8799 - v

Epoch 00004: val_acc did not improve from 0.93750
Epoch 5/20
88/87 [==============================] - 78s 885ms/step - loss: 0.2708 - acc: 0.8935 - v

Epoch 00005: val_acc did not improve from 0.93750
Epoch 6/20
88/87 [==============================] - 78s 888ms/step - loss: 0.2716 - acc: 0.8913 - v

Epoch 00006: val_acc improved from 0.93750 to 0.94886, saving model to weights.best.hdf5
Epoch 7/20
88/87 [==============================] - 78s 886ms/step - loss: 0.2371 - acc: 0.9048 - v

Epoch 00007: val_acc did not improve from 0.94886
Epoch 8/20
88/87 [==============================] - 80s 906ms/step - loss: 0.2234 - acc: 0.9083 - v

Epoch 00008: val_acc improved from 0.94886 to 0.95455, saving model to weights.best.hdf5
Epoch 9/20
88/87 [==============================] - 81s 916ms/step - loss: 0.2646 - acc: 0.8948 - v

Epoch 00009: val_acc did not improve from 0.95455
Epoch 10/20
88/87 [==============================] - 80s 906ms/step - loss: 0.2115 - acc: 0.9196 - v

Epoch 00010: val_acc did not improve from 0.95455
Epoch 11/20
88/87 [==============================] - 81s 916ms/step - loss: 0.2272 - acc: 0.9069 - v

Epoch 00011: val_acc improved from 0.95455 to 0.95739, saving model to weights.best.hdf5
Epoch 12/20
88/87 [==============================] - 78s 883ms/step - loss: 0.2205 - acc: 0.9133 - v

Epoch 00012: val_acc improved from 0.95739 to 0.96307, saving model to weights.best.hdf5
Epoch 13/20
88/87 [==============================] - 78s 883ms/step - loss: 0.1691 - acc: 0.9304 - v

Epoch 00013: val_acc did not improve from 0.96307
Epoch 14/20
```

```
88/87 [==============================] - 78s 882ms/step - loss: 0.1824 - acc: 0.9332 - v

Epoch 00014: val_acc did not improve from 0.96307
Epoch 15/20
88/87 [==============================] - 78s 885ms/step - loss: 0.1669 - acc: 0.9389 - v

Epoch 00015: val_acc did not improve from 0.96307
Epoch 16/20
88/87 [==============================] - 78s 884ms/step - loss: 0.1714 - acc: 0.9332 - v

Epoch 00016: val_acc did not improve from 0.96307
Epoch 17/20
88/87 [==============================] - 78s 886ms/step - loss: 0.1774 - acc: 0.9382 - v

Epoch 00017: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.

Epoch 00017: val_acc did not improve from 0.96307
Epoch 18/20
88/87 [==============================] - 78s 887ms/step - loss: 0.1667 - acc: 0.9375 - v

Epoch 00018: val_acc improved from 0.96307 to 0.97443, saving model to weights.best.hdf5
Epoch 19/20
88/87 [==============================] - 79s 900ms/step - loss: 0.1224 - acc: 0.9574 - v

Epoch 00019: val_acc improved from 0.97443 to 0.97727, saving model to weights.best.hdf5
Epoch 20/20
88/87 [==============================] - 81s 924ms/step - loss: 0.1087 - acc: 0.9496 - v

Epoch 00020: val_acc improved from 0.97727 to 0.98011, saving model to weights.best.hdf5
```

```
with open('history.json', 'w') as f:
    json.dump(str(history.history), f)
```

```
history_df = pd.DataFrame(history.history)
history_df[['acc', 'val_acc']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fec952948d0>
```



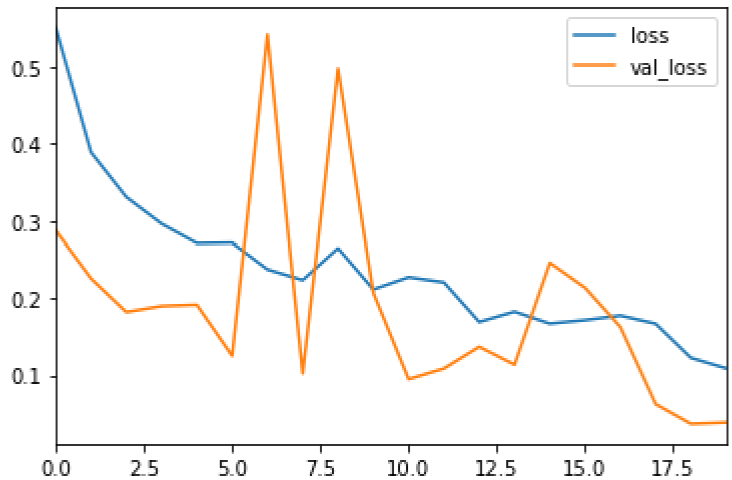

```
from google.colab import drive
drive.mount('/content/drive')
```

```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```
history_df = pd.DataFrame(history.history)
history_df[['loss', 'val_loss']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fec2f20c438>
```



Prediction

```
model.load_weights("weights.best.hdf5")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-daa1a4ee913c> in <module>()
----> 1 model.load_weights("weights.best.hdf5")

NameError: name 'model' is not defined
```

SEARCH STACK OVERFLOW

```
Y_val_pred = model.predict(x_val)
```

```
accuracy_score(np.argmax(y_val, axis=1), np.argmax(Y_val_pred, axis=1))
```

```
0.9801136363636364
```

```
Y_pred = model.predict(x_val)
```

```
tta_steps = 10
```

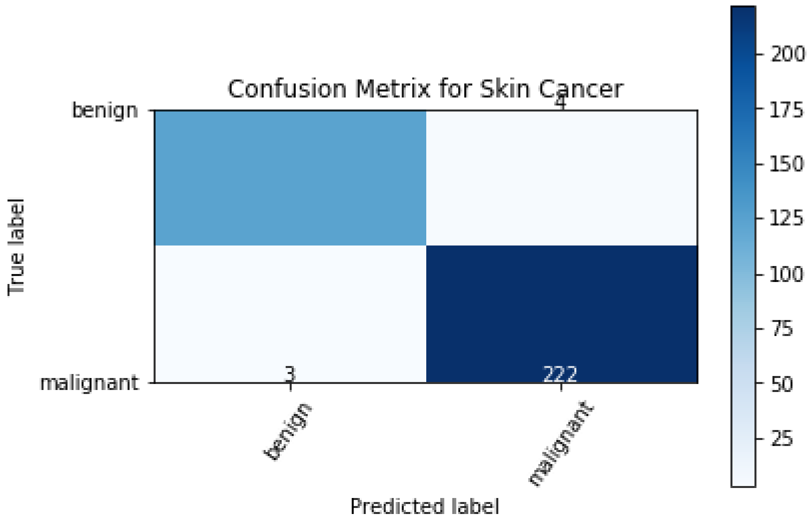
```
predictions = []

for i in tqdm(range(tta_steps)):
    preds = model.predict_generator(train_generator.flow(x_val, batch_size=BATCH_SIZE, shuffl
                                    steps = len(x_val)/BATCH_SIZE)

    predictions.append(preds)
    gc.collect()

Y_pred_tta = np.mean(predictions, axis=0)
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=55)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

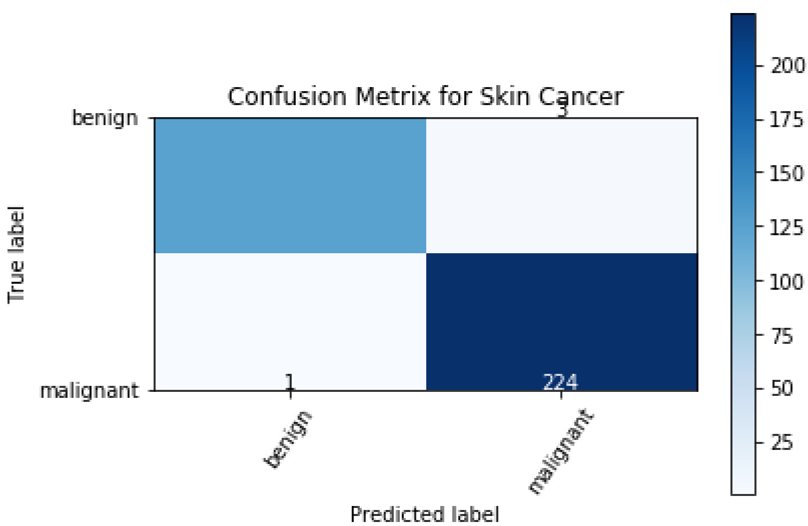
```
cm = confusion_matrix(np.argmax(y_val, axis=1), np.argmax(Y_pred, axis=1))

cm_plot_label =['benign', 'malignant']
plot_confusion_matrix(cm, cm_plot_label, title ='Confusion Metrix for Skin Cancer')
```

```
Confusion matrix, without normalization
[[123   4]
 [  3 222]]
```




```
cm = confusion_matrix(np.argmax(y_val, axis=1), np.argmax(Y_pred_tta, axis=1))

cm_plot_label =['benign', 'malignant']
plot_confusion_matrix(cm, cm_plot_label, title ='Confusion Metrix for Skin Cancer')
```

```
Confusion matrix, without normalization
[[124   3]
 [  1 224]]
```



Classification Report

```
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import classification_report
classification_report( np.argmax(y_val, axis=1), np.argmax(Y_pred_tta, axis=1))
```

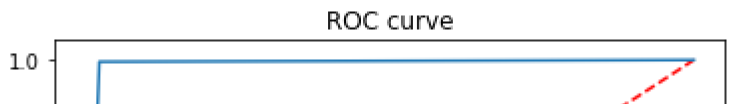
```
'              precision    recall  f1-score   support\n\n           0       0.99       6
```

```
print( 'precision    recall  f1-score   support\n\n           0       0.99      0.98      0.9
```

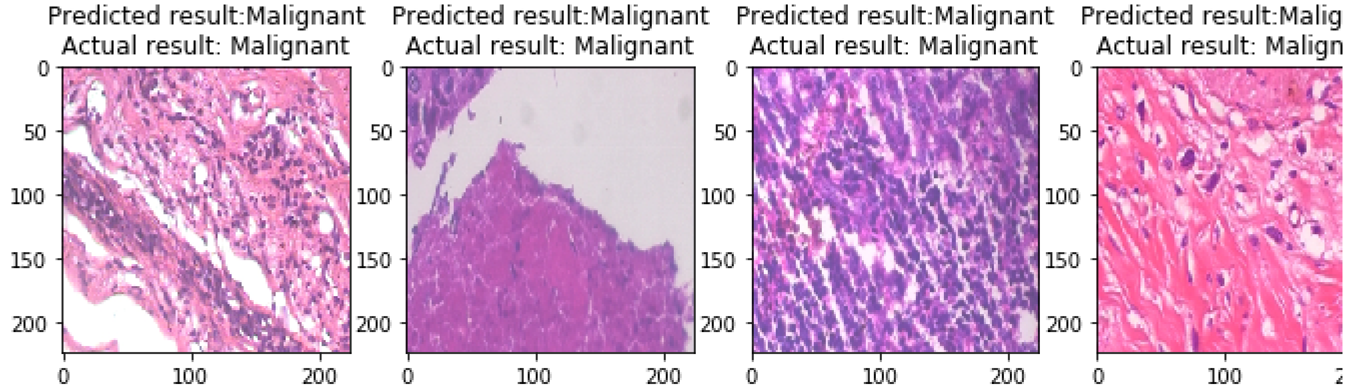
```
precision    recall  f1-score   support

           0       0.99      0.98      0.98       127
           1       0.99      1.00      0.99       225

    accuracy                           0.99       352
   macro avg       0.99      0.99      0.99       352
weighted avg       0.99      0.99      0.99       352
```

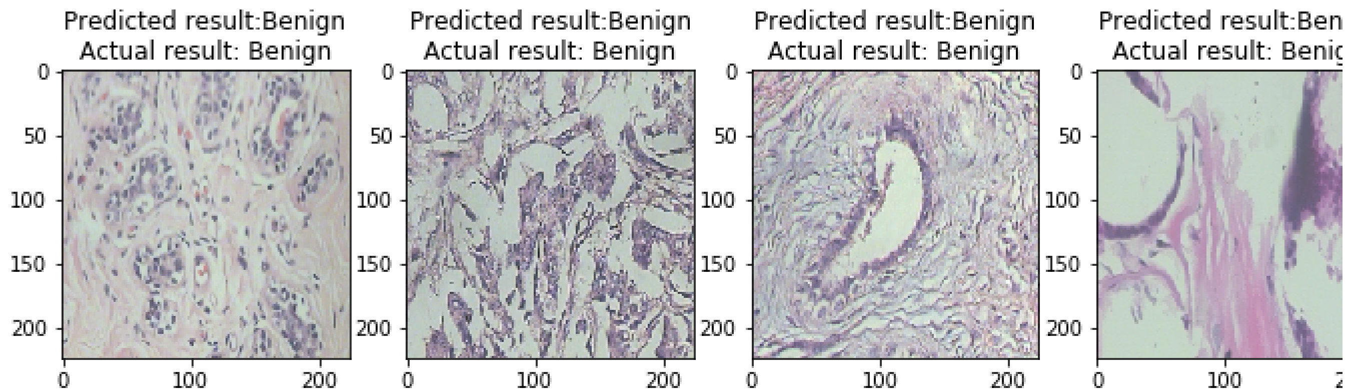
ROC and AUC

```
from sklearn.metrics import roc_auc_score, auc
from sklearn.metrics import roc_curve
roc_log = roc_auc_score(np.argmax(y_val, axis=1), np.argmax(Y_pred_tta, axis=1))
false_positive_rate, true_positive_rate, threshold = roc_curve(np.argmax(y_val, axis=1), np.a
area_under_curve = auc(false_positive_rate, true_positive_rate)

plt.plot([0, 1], [0, 1], 'r--')
plt.plot(false_positive_rate, true_positive_rate, label='AUC = {:.3f}'.format(area_under_curv
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
#plt.savefig(ROC_PLOT_FILE, bbox_inches='tight')
plt.close()
```




```
i=0
prop_class=[]
mis_class=[]

for i in range(len(y_val)):
    if(np.argmax(y_val[i])==np.argmax(Y_pred_tta[i])):
        prop_class.append(i)
    if(len(prop_class)==12):
        break

i=0
for i in range(len(y_val)):
    if(not np.argmax(y_val[i])==np.argmax(Y_pred_tta[i])):
        mis_class.append(i)
    if(len(mis_class)==12):
        break

# # Display first 8 images of benign
w=60
h=40
fig=plt.figure(figsize=(18, 10))
columns = 6
rows = 2

def Transfername(namecode):
    if namecode==0:
        return "Benign"
    else:
        return "Malignant"

for i in range(len(prop_class)):
    ax = fig.add_subplot(rows, columns, i+1)
    ax.set_title("Predicted result:"+ Transfername(np.argmax(Y_pred_tta[prop_class[i]]))
                       +"\n"+"Actual result: "+ Transfername(np.argmax(y_val[prop_class[i]]))
    plt.imshow(x_val[prop_class[i]], interpolation='nearest')
plt.show()
```


Predicted result:Malignant
Actual result: Malignant
Predicted result:Malignant
Actual result: Malignant
Predicted result:Malignant
Actual result: Malignant
Predicted result:Malig
Actual result: Malign
0
50
100
150
200
0
100
200


Predicted result:Benign
Actual result: Benign
Predicted result:Benign
Actual result: Benign
Predicted result:Benign
Actual result: Benign
Predicted result:Ben
Actual result: Benig
0
50
100
150
200
0
100
200