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

```
Mounted at /content/drive
```

```
!pip install -U efficientnet
```

```
Collecting efficientnet
  Downloading efficientnet-1.1.1-py3-none-any.whl (18 kB)
Collecting keras-applications<=1.0.8,>=1.0.7 (from efficientnet)
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.7/50.7 kB 1.9 MB/s eta 0:00:00
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-packages (from efficientnet) (0.19.3)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/dist-packages (from keras-applications<=1.0.8,>=1.0.7->effi
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras-applications<=1.0.8,>=1.0.7->efficientnet
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (1.11.4)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (3.2.1)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image-
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (2.31.6)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (202
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (1.5.0
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (24.0)
Installing collected packages: keras-applications, efficientnet
Successfully installed efficientnet-1.1.1 keras-applications-1.0.8
```

```
#!unzip /content/drive/MyDrive/image_tampering_detection/CASIA2_DATA.zip -d /content/drive/MyDrive/image_tampering_detection/
```

```
import os
import io
import cv2
import random
import itertools
import numpy as np
from pylab import *
import pandas as pd
import seaborn as sns
from tqdm import tqdm
from PIL import Image
from keras import models
from keras import layers
from keras import optimizers
from keras.models import Model
import matplotlib.pyplot as plt
from keras.models import Sequential
import efficientnet.keras as effnet
from keras.applications.vgg19 import VGG19
from keras.utils import to_categorical
from PIL import Image, ImageChops, ImageEnhance
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import RMSprop,Adam
from tensorflow.keras.applications.densenet import DenseNet201
from sklearn.metrics import confusion_matrix,classification_report
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.layers import BatchNormalization, AveragePooling2D, GlobalAveragePooling2D
```

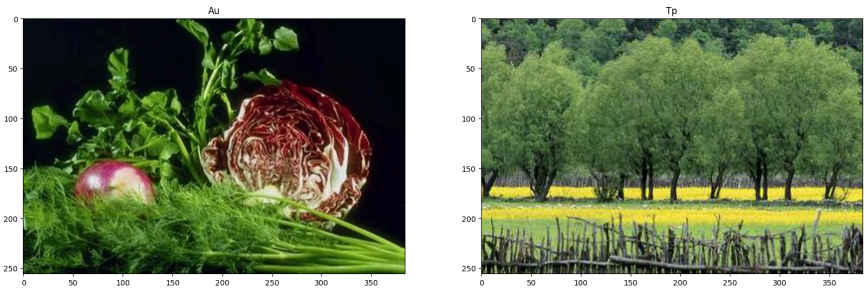
```
import warnings
warnings.filterwarnings('ignore')
```

```
def image_loading(path, quality):
    image = Image.open(path).convert('RGB')
    return image
```

```
#data loading
labels = ['Au', 'Tp']
image_size=128
X = []
Y = []

for label in labels:
    trainPath = os.path.join('/content/drive/MyDrive/image_tampering_detection/CASIA2_DATA',label)
    for file in tqdm(os.listdir(trainPath)[:3000]):
        X.append(array(image_loading(os.path.join(trainPath, file), 90).resize((128, 128))).flatten() / 255.0)
        Y.append(label)

X = np.array(X)
X = X.reshape(-1, 128, 128, 3)
```

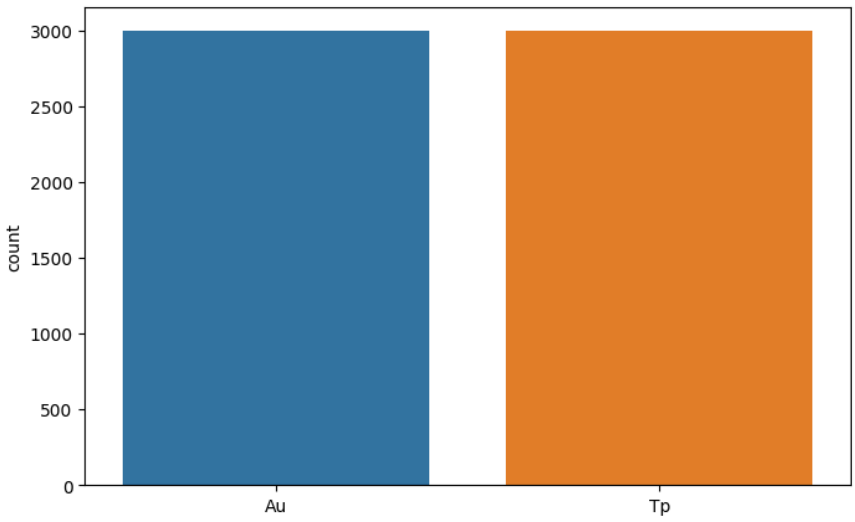
```
100%|██████████| 3000/3000 [03:22<00:00, 14.79it/s]
100%|██████████| 3000/3000 [03:38<00:00, 13.75it/s]
```

```
#Data Visualization
import matplotlib.image as mpimg
plt.figure(figsize = (20, 40))
image_count = 1
BASE_URL = '/content/drive/MyDrive/image_tampering_detection/CASIA2_DATA/'
for directory in labels:
    if directory[0] != '.':
        for i, file in enumerate(os.listdir(BASE_URL + directory)):
            if i == 1:
                break
            else:
                fig = plt.subplot(1, 2, image_count)
                image_count += 1
                image = mpimg.imread(BASE_URL + directory + '/' + file)
                plt.imshow(image)
                plt.title(directory)
```




```
#count plot
plt.figure(figsize = (8, 5))
plt.xticks(rotation=0)
sns.countplot(x=Y,palette=sns.color_palette())
```

```
<Axes: ylabel='count'>
```



```
#converting categorical class to number
y = []
for i in Y:
  if i == 'Tp':
    y.append(1)
  else:
    y.append(0)

Y = y
Y = to_categorical(Y, 2)
```

```
print("shape of image: ",X.shape)
print("shape of target class: ",Y.shape)
```

```
shape of image:  (6000, 128, 128, 3)
shape of target class:  (6000, 2)
```

```
#splitting data into train and test with ratio of 80:10:10
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=5, shuffle=True)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=5, shuffle=True)
```

EfficientNet-B2

```
epochs = 10
batch_size = 20
```

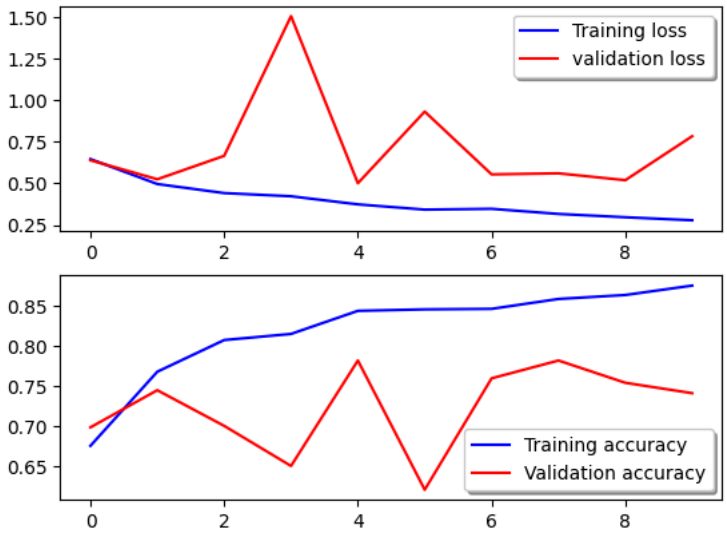
```
base_model = effnet.EfficientNetB2(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
model = base_model.output
model = GlobalAveragePooling2D()(model)
model = Dropout(0.4)(model)
model = Dense(2, activation='softmax')(model)
model = Model(inputs = base_model.input, outputs=model)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
 dropout (Dropout)           (None, 1408)              0         ['global_average_pooling2d[0][
                                                                 0]']

 dense (Dense)               (None, 2)                 2818      ['dropout[0][0]']

==================================================================================================
Total params: 7771380 (29.65 MB)
Trainable params: 7703812 (29.39 MB)
Non-trainable params: 67568 (263.94 KB)
```

```
history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(X_val, y_val), verbose=1)
```

```
Epoch 1/10
243/243 [==============================] - 83s 104ms/step - loss: 0.6465 - accuracy: 0.6751 - val_loss: 0.6393 - val_accuracy: 0.698
Epoch 2/10
243/243 [==============================] - 21s 86ms/step - loss: 0.4963 - accuracy: 0.7675 - val_loss: 0.5250 - val_accuracy: 0.744
Epoch 3/10
243/243 [==============================] - 22s 90ms/step - loss: 0.4418 - accuracy: 0.8070 - val_loss: 0.6658 - val_accuracy: 0.700
Epoch 4/10
243/243 [==============================] - 22s 90ms/step - loss: 0.4227 - accuracy: 0.8146 - val_loss: 1.5066 - val_accuracy: 0.650
Epoch 5/10
243/243 [==============================] - 22s 89ms/step - loss: 0.3740 - accuracy: 0.8434 - val_loss: 0.5013 - val_accuracy: 0.781
Epoch 6/10
243/243 [==============================] - 22s 90ms/step - loss: 0.3424 - accuracy: 0.8453 - val_loss: 0.9322 - val_accuracy: 0.620
Epoch 7/10
243/243 [==============================] - 22s 90ms/step - loss: 0.3472 - accuracy: 0.8459 - val_loss: 0.5544 - val_accuracy: 0.759
Epoch 8/10
243/243 [==============================] - 21s 88ms/step - loss: 0.3163 - accuracy: 0.8582 - val_loss: 0.5607 - val_accuracy: 0.781
Epoch 9/10
243/243 [==============================] - 22s 91ms/step - loss: 0.2964 - accuracy: 0.8632 - val_loss: 0.5195 - val_accuracy: 0.753
Epoch 10/10
243/243 [==============================] - 22s 90ms/step - loss: 0.2782 - accuracy: 0.8749 - val_loss: 0.7841 - val_accuracy: 0.740
```

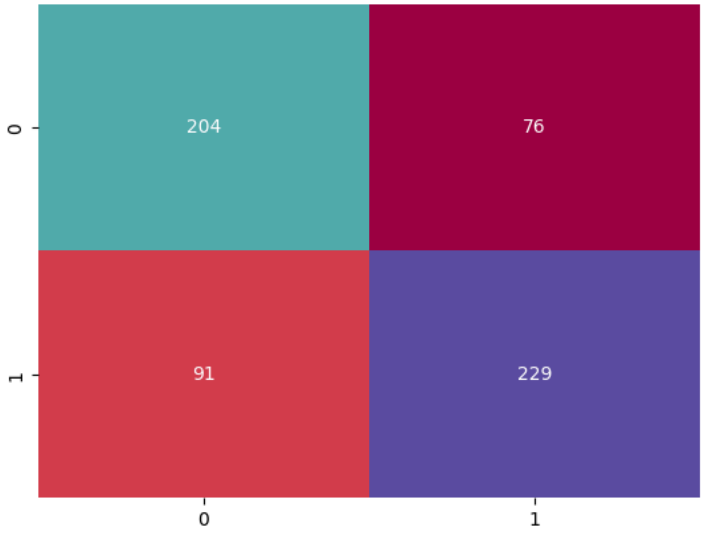
```
#Accuracy & Loss Graph
fig, ax = plt.subplots(2, 1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='r', label="validation loss", axes=ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='r', label="Validation accuracy")
legend_ = ax[1].legend(loc='best', shadow=True)
```




```
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)
```

```
19/19 [==============================] - 5s 95ms/step
```

```
confusion_mtx = confusion_matrix(y_true, y_pred_classes)
#plt.figure(figsize=(10,10))
sns.heatmap(confusion_mtx, annot = True, cbar = False, cmap=cm.get_cmap("Spectral"), fmt="d");
```




```
print(classification_report(y_true, y_pred_classes))
```

```
                  precision    recall  f1-score   support

               0       0.69      0.73      0.71       280
               1       0.75      0.72      0.73       320

        accuracy                           0.72       600
       macro avg       0.72      0.72      0.72       600
    weighted avg       0.72      0.72      0.72       600
```