

```
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 1.1 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->efficientnet) (1.24.3)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras-applications<=1.0.8,>=1.0.7->efficientnet) (3.10.0)
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->efficientnet) (10.0.0)
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) (2023.12.9)
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), 0), dtype='float32'))
        Y.append(label)

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



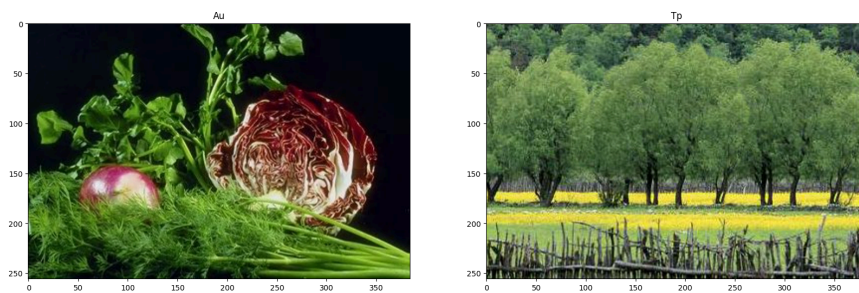
McAfee | WebAdvisor



Your download's being scanned.
We'll let you know if there's an issue.

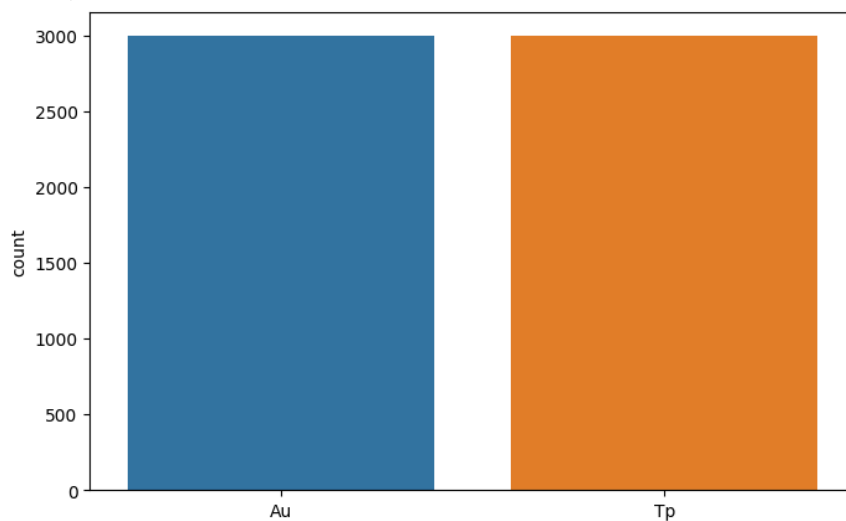
```
100%|██████████| 3000/3000 [02:47<00:00, 17.93it/s]
100%|██████████| 3000/3000 [04:12<00:00, 11.88it/s]
```

```
#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'>



McAfee | WebAdvisor



Your download's being scanned.
We'll let you know if there's an issue.

```
#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)
```

✓ VGG19

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

```
# Load the VGG model
vgg_conv = VGG19(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
```

```
# Create the model
model = models.Sequential()
# Freeze the layers the first layers
for layer in vgg_conv.layers[:-5]:
    layer.trainable = False
# Check the trainable status of the individual layers
for layer in vgg_conv.layers:
    print(layer, layer.trainable)
```

```
model.add(vgg_conv)
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.50))
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.50))
model.add(layers.Dense(2, activation='softmax'))
```

```
model.summary()
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_n80134624/80134624 [=====] - 1s 0us/step

```
<keras.src.engine.input_layer.InputLayer object at 0x7ddd4c5759f0> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c575690> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c597310> False
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7ddd4c45ff70> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c45f910> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c37c9a0> False
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7ddd4c37df30> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c37e7a0> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c37d630> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c37f250> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c37d690> False
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7ddd4c399b40> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c6365c0> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd5238eb00> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c37d660> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c399b70> False
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7ddd4c39a950> False
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c39a650> True
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c45fc10> True
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c45e4d0> True
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7ddd4c3989a0> True
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7ddd4c39a890> True
Model: "sequential"
```

Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 4, 4, 512)	20024384
flatten (Flatten)	(None, 8192)	0



McAfee | WebAdvisor

Your download's being scanned.
We'll let you know if there's an issue.

dense (Dense)	(None, 1024)	8389632
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1049600
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 2)	2050

```

=====
Total params: 29465666 (112.40 MB)
Trainable params: 18880514 (72.02 MB)
Non-trainable params: 10585152 (40.38 MB)
=====

```

```

optimizer = optimizers.Adagrad()
model.compile(optimizer=optimizer,loss="categorical_crossentropy",metrics=["accuracy"])

```

```
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 [=====] - 29s 61ms/step - loss: 0.7130 - accuracy: 0.5872 - val_loss: 0.6005 - val_accuracy: 0.6852
Epoch 2/10
243/243 [=====] - 14s 57ms/step - loss: 0.6205 - accuracy: 0.6574 - val_loss: 0.5718 - val_accuracy: 0.7093
Epoch 3/10
243/243 [=====] - 14s 59ms/step - loss: 0.5779 - accuracy: 0.7019 - val_loss: 0.5828 - val_accuracy: 0.6885
Epoch 4/10
243/243 [=====] - 14s 59ms/step - loss: 0.5526 - accuracy: 0.7251 - val_loss: 0.5796 - val_accuracy: 0.6907
Epoch 5/10
243/243 [=====] - 15s 62ms/step - loss: 0.5229 - accuracy: 0.7407 - val_loss: 0.5482 - val_accuracy: 0.7444
Epoch 6/10
243/243 [=====] - 16s 65ms/step - loss: 0.4907 - accuracy: 0.7611 - val_loss: 0.5561 - val_accuracy: 0.7311
Epoch 7/10
243/243 [=====] - 15s 62ms/step - loss: 0.4692 - accuracy: 0.7765 - val_loss: 0.5648 - val_accuracy: 0.7255
Epoch 8/10
243/243 [=====] - 15s 63ms/step - loss: 0.4298 - accuracy: 0.8004 - val_loss: 0.5758 - val_accuracy: 0.7385
Epoch 9/10
243/243 [=====] - 15s 63ms/step - loss: 0.3999 - accuracy: 0.8142 - val_loss: 0.6324 - val_accuracy: 0.6981
Epoch 10/10
243/243 [=====] - 15s 64ms/step - loss: 0.3778 - accuracy: 0.8319 - val_loss: 0.5819 - val_accuracy: 0.7136

```

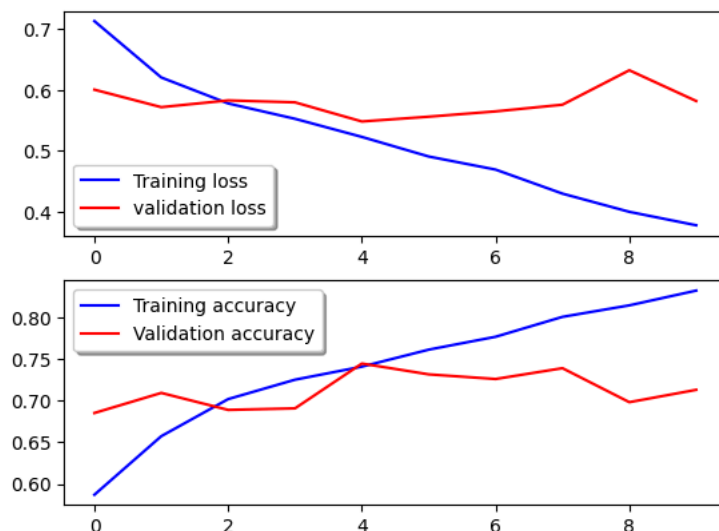
#Accuracy & Loss Graph

```

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)

```

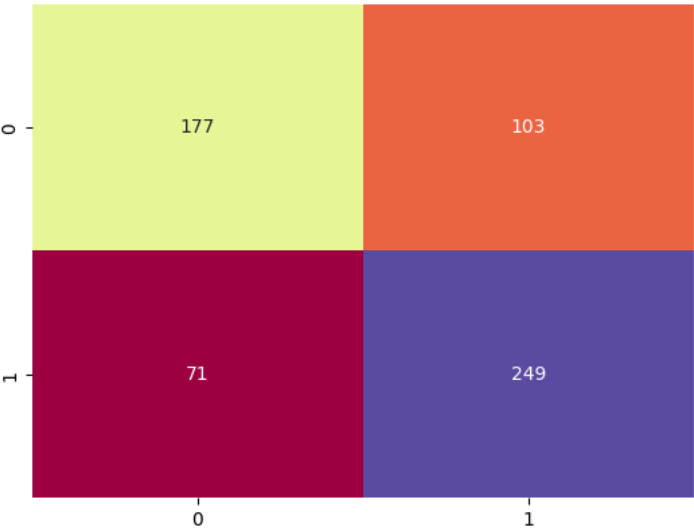
```
19/19 [=====] - 6s 169ms/step
```



McAfee | WebAdvisor


Your download's being scanned.
We'll let you know if there's an issue.

```
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	0.71	0.63	0.67	280
1	0.71	0.78	0.74	320
accuracy			0.71	600
macro avg	0.71	0.71	0.71	600
weighted avg	0.71	0.71	0.71	600



McAfee

WebAdvisor

×

Your download's being scanned.
We'll let you know if there's an issue.