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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import matplotlib.pyplot as plt
import tensorflow
from tensorflow.keras import Sequential,Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import Image
from pathlib import Path
from tensorflow.keras.utils import image_dataset_from_directory
import seaborn as sns
```

```
!pip install tf_keras_vis
```

```
Requirement already satisfied: tf_keras_vis in /usr/local/lib/python3.10/dist-packages (0.8.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from tf_keras_vis) (1.11.3)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from tf_keras_vis) (9.4.0)
Requirement already satisfied: deprecated in /usr/local/lib/python3.10/dist-packages (from tf_keras_vis) (1.2.14)
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packages (from tf_keras_vis) (2.31.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tf_keras_vis) (23.2)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (from deprecated->tf_keras_vis) (1.15.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from imageio->tf_keras_vis) (1.23.5)
```

```
from tf_keras_vis.saliency import Saliency
from tf_keras_vis.gradcam import Gradcam
from tf_keras_vis.utils.scores import BinaryScore
from tf_keras_vis.utils.model_modifiers import ReplaceToLinear
from matplotlib import cm
```

```
train_directory = Path('/content/drive/MyDrive/classifying building/train_another') # the training data; 5000 images of each class(damage/
validation_directory = Path('/content/drive/MyDrive/classifying building/validation_another') #the validation data; 1000 images of each cl
unbalanced_test_directory = Path('/content/drive/MyDrive/classifying building/test_another') # 8000/1000 images of damaged/undamaged class
balanced_test_directory = Path('/content/drive/MyDrive/classifying building/test') # the balanced test data; 1000 images of each class(dam
```

```
labels_train,images_train,labels_test_another,images_test_another,labels_test,images_test = [],[],[],[],[],[]
for i in ['damage','no_damage']:
    files_train = os.listdir(f'//content/drive/MyDrive/classifying building/train_another/{i}')
    files_test_another =  os.listdir(f'//content/drive/MyDrive/classifying building/test_another/{i}')
    files_test =  os.listdir(f'//content/drive/MyDrive/classifying building/test/{i}')
    images_train.append(files_train)
    images_test_another.append(files_test_another)
    images_test.append(files_test)
    for l in range(len(files_train)):
        labels_train.append(i)
    for l in range(len(files_test)):
        labels_test_another.append(i)
    for l in range(len(files_test_another)):
        labels_test.append(i)
print("Number of items in the training set : {}".format(len(labels_train)))
print("Number of items in the test_another dataset : {}".format(len(labels_test_another)))
print("Number of items in the test dataset : {}".format(len(labels_test)))
```

```
Number of items in the training set : 20
Number of items in the test_another dataset : 20
Number of items in the test dataset : 20
```

```
label_en = LabelEncoder()
labels_all = [0]*3
labels_all[0] = label_en.fit_transform(labels_train)
labels_all[1] = label_en.transform(labels_test_another)
labels_all[2] = label_en.transform(labels_test)

```

```
print(labels_all[1])
```

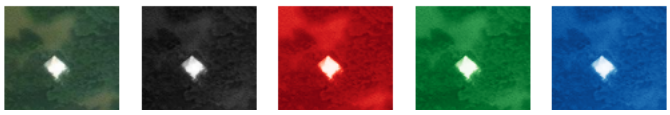
```
[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1]
```

```
img1 = Image.open('/content/drive/MyDrive/classifying building/train_another/damage/-93.55964_30.895018.jpeg') #train_damaged
plt.imshow(img1)
plt.title(f'Actual label = {labels_train[0]}')
plt.axis('off')
plt.show()
```


```
img2 = Image.open('//content/drive/MyDrive/classifying building/train_another/no_damage/-95.0701_29.830762.jpeg') #train_undamaged
plt.imshow(img2)
plt.title(f'Actual label = {labels_train[10]}')
plt.axis('off')
plt.show()
```


```
img_data = np.array(img1)
img_data_shape = img_data.shape
print("Our NumPy array has the shape: {}".format(img_data_shape))
fig,ax = plt.subplots(1,5)
ax[0].imshow(img_data)
ax[0].axis('off')
ax[1].imshow(img_data[:,:,2], cmap='gray')
ax[1].axis('off')
ax[2].imshow(img_data[:,:,0], cmap=plt.cm.Reds_r)
ax[2].axis('off')
ax[3].imshow(img_data[:,:,1], cmap=plt.cm.Greens_r)
ax[3].axis('off')
ax[4].imshow(img_data[:,:,2], cmap=plt.cm.Blues_r)
ax[4].axis('off')
plt.show()
```

```
Our NumPy array has the shape: (128, 128, 3)
```



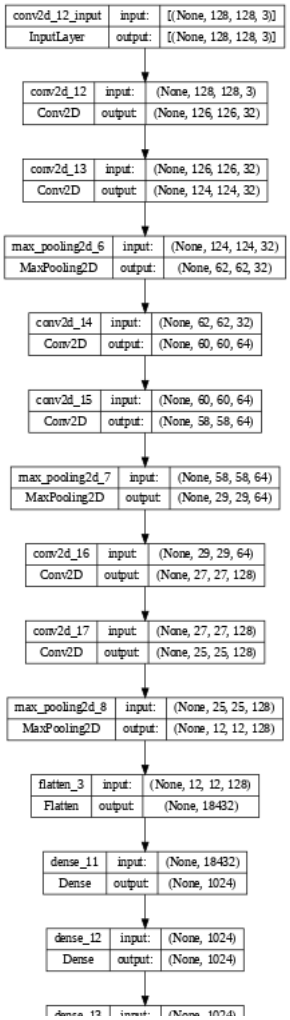
```
train_gen = ImageDataGenerator(rotation_range=10,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        rescale=1/255.0,
        brightness_range=[0.2,1.2])
validation_gen = ImageDataGenerator(rotation_range=10,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        rescale=1/255.0,
        brightness_range=[0.2,1.2])
test_unbalanced_gen =  ImageDataGenerator()
test_balanced_gen =  ImageDataGenerator()
```

```
train_data = train_gen.flow_from_directory(
    directory = train_directory,
    target_size = (128,128),
    class_mode = 'binary',
    color_mode='rgb',
    shuffle = True,
    batch_size=100)
val_data = validation_gen.flow_from_directory(
    directory = validation_directory,
    target_size = (128,128),
    class_mode = 'binary',
    color_mode='rgb',
    shuffle = True,
    batch_size=100)
unbalanced_data = test_unbalanced_gen.flow_from_directory(directory =unbalanced_test_directory,
                                                          target_size = (128,128),
                                                          class_mode = 'binary',
                                                          shuffle = False,
                                                          color_mode='rgb',
                                                          batch_size=100)
balanced_data = test_balanced_gen.flow_from_directory(directory =balanced_test_directory,
                                                          target_size = (128,128),
                                                          class_mode = 'binary',
                                                          color_mode='rgb',
                                                          shuffle =False,
                                                          batch_size=100)


```

```
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
```

```
from tensorflow.keras.layers import Conv2D,MaxPool2D,Dense,Flatten,BatchNormalization,Dropout
model = Sequential(name="Base_Model")
model.add(Conv2D(32,kernel_size =(3, 3), activation='relu',input_shape=(128,128,3)))
model.add(Conv2D(32,kernel_size =(3,3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(64,kernel_size =(3,3), activation='relu'))
model.add(Conv2D(64,kernel_size =(3,3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(128,kernel_size =(3,3), activation='relu'))
model.add(Conv2D(128,kernel_size =(3,3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(1024,activation='relu'))
model.add(Dense(1024,activation='relu'))
model.add(Dense(1,activation = 'sigmoid'))
tensorflow.keras.utils.plot_model(model, "base_model.png", show_shapes=True,dpi =50)
```




```
1 model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate = 1e-4), loss='binary_crossentropy', metrics=['accuracy'])
```

```
1 history_1 = model.fit(train_data,validation_data=val_data,epochs=5)
```

```
Epoch 1/5
1/1 [==============================] - 3s 3s/step - loss: 0.6695 - accuracy: 0.7500 - val_loss: 0.6631 - val_accuracy: 0.6500
Epoch 2/5
1/1 [==============================] - 3s 3s/step - loss: 0.6757 - accuracy: 0.6000 - val_loss: 0.6609 - val_accuracy: 0.6500
Epoch 3/5
1/1 [==============================] - 4s 4s/step - loss: 0.6735 - accuracy: 0.6000 - val_loss: 0.6448 - val_accuracy: 0.5000
Epoch 4/5
1/1 [==============================] - 3s 3s/step - loss: 0.6869 - accuracy: 0.5000 - val_loss: 0.6512 - val_accuracy: 0.6000
Epoch 5/5
1/1 [==============================] - 3s 3s/step - loss: 0.6777 - accuracy: 0.5500 - val_loss: 0.6543 - val_accuracy: 0.6500
```

```
1 from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
2 def metrices(model) :
3     y_preds_unbalanced = [1 if prob >= 0.5 else 0 for prob in model.predict(unbalanced_data)]
4     y_preds_balanced = [1 if prob >= 0.5 else 0 for prob in model.predict(balanced_data)]
5     y_true_unbalanced = unbalanced_data.classes
6     y_true_balanced = balanced_data.classes
7     precision_unbalanced = precision_score(y_true_unbalanced, y_preds_unbalanced)
8     recall_unbalanced = recall_score(y_true_unbalanced, y_preds_unbalanced)
9     f1_unbalanced = f1_score(y_true_unbalanced, y_preds_unbalanced)
10    roc_auc_unbalanced= roc_auc_score(y_true_unbalanced, y_preds_unbalanced)
11    precision_balanced = precision_score(y_true_balanced, y_preds_balanced)
12    recall_balanced = recall_score(y_true_balanced, y_preds_balanced)
13    f1_balanced = f1_score(y_true_balanced, y_preds_balanced)
14    roc_auc_balanced = roc_auc_score(y_true_balanced, y_preds_balanced)
15    print('Precision on Unbalanced Test Dataset : {}'.format(precision_unbalanced ))
16    print('Precision on Balanced Test Dataset : {}'.format(precision_balanced))
17    print('Recall on Unbalanced Test Dataset : {}'.format(recall_unbalanced))
18    print('Recall on Balanced Test Dataset : {}'.format(recall_balanced))
19    print('F1 Score on Unbalanced Test Dataset : {}'.format(f1_unbalanced))
20    print('F1 Score on Balanced Test Dataset : {}'.format(f1_balanced))
21    print('AUC score  on Unbalanced Test Dataset : {}'.format(roc_auc_unbalanced))
```

```
    print('AUC score on Balanced Test Dataset : {}'.format(roc_auc_balanced))
    return precision_unbalanced,precision_balanced, recall_unbalanced,recall_balanced,f1_unbalanced,f1_balanced ,roc_auc_unbalanced,roc_au
```

```
test_unbalanced_acc_1 = model.evaluate(unbalanced_data,verbose=0)
print('Test Accuracy of Specified Base Model on Unbalanced Test Dataset : {}'.format(test_unbalanced_acc_1[1] ))
test_balanced_acc_1= model.evaluate(balanced_data,verbose =0)
print('Test Accuracy of Specified Base Model on Balanced Test Dataset : {}'.format(test_balanced_acc_1[1] ))
metrics_1 = metrices(model)
```

```
Test Accuracy of Specified Base Model on Unbalanced Test Dataset : 0.5
Test Accuracy of Specified Base Model on Balanced Test Dataset : 0.6499999761581421
WARNING:tensorflow:5 out of the last 8 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7c36bc3b4430> trig
1/1 [==============================] - 1s 556ms/step
1/1 [==============================] - 0s 467ms/step
Precision on Unbalanced Test Dataset : 0.5
Precision on Balanced Test Dataset : 0.5882352941176471
Recall on Unbalanced Test Dataset : 1.0
Recall on Balanced Test Dataset : 1.0
F1 Score on Unbalanced Test Dataset : 0.6666666666666666
F1 Score on Balanced Test Dataset : 0.7407407407407407
AUC score  on Unbalanced Test Dataset : 0.5
AUC score on Balanced Test Dataset : 0.65
```

```
val_img1 = Image.open('/content/drive/MyDrive/classifying building/validation_another/damage/-93.558326_30.895248.jpeg')
val_img2 = Image.open('/content/drive/MyDrive/classifying building/validation_another/damage/-93.563851_30.894492.jpeg')
val_img3 = Image.open('/content/drive/MyDrive/classifying building/validation_another/no_damage/-95.06438_30.037894.jpeg')
val_img4 = Image.open('/content/drive/MyDrive/classifying building/validation_another/no_damage/-95.07518_29.829121999999998.jpeg')
val_img_labels = [0,0,1,1]
val_imgs = [val_img1,val_img2,val_img3,val_img4]
```

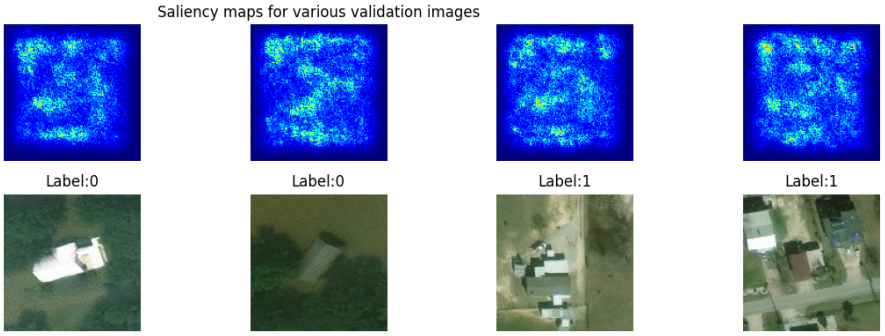
```
val_img_data = []
for img_data in val_imgs:
    data = np.array(img_data)
    data = data/255.0
    data = np.expand_dims(data,axis=0)
    val_img_data.append(data)
```

```
def plot_featuremaps(img,activations,layer_names):
    fig, axs = plt.subplots(ncols=4, nrows=4,figsize = (6,6))
    gs = axs[1, 2].get_gridspec()
    for ax in axs[1:-1, 1:-1]:
        ax[0].remove()
        ax[1].remove()
    axbig = fig.add_subplot(gs[1:-1, 1:-1])
    axbig.imshow(img.squeeze()+0.5)
    axbig.axis('off')
    for i, axis in enumerate(axs.ravel()):
        axis.imshow(activations.squeeze()[:,:,i], cmap='gray')
        axis.axis('off')
    fig.tight_layout()
    fig.suptitle(f'Feature maps for {layer_names[0]}',y=1.05);
first_conv_layer_output = model.layers[0].output
activation_model = Model(inputs=model.input,outputs=first_conv_layer_output)
activations = activation_model.predict(val_img_data[0])
plot_featuremaps(val_img_data[0],activations,[model.layers[0].name])
```

```
WARNING:tensorflow:6 out of the last 10 calls to <function Model.make_predict_function.
1/1 [==============================] - 0s 52ms/step
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB dat
```


```
def score_function(index):
    if (index==0 or index==1):
        score = BinaryScore(0.0)
    else:
        score = BinaryScore(1.0)
    return score
```

```
fig,ax = plt.subplots(2,4,figsize=(12, 4))
for i in range(4):
    saliency_model = Saliency(model,model_modifier=ReplaceToLinear(),clone=True)
    saliency_map = saliency_model(score_function(i),val_img_data[i])
    ax[1,i].imshow(val_imgs[i])
    ax[1,i].set_title(f'Label:{val_img_labels[i]}')
    ax[1,i].axis('off')
    ax[0,i].imshow(saliency_map[0], cmap='jet')
    ax[0,i].set_title('Saliency maps for various validation images')
    ax[0,i].axis('off')
plt.tight_layout()
plt.show()
```




```
from tensorflow.keras.layers import Conv2D,MaxPool2D,Dense,Flatten
from tensorflow.keras import regularizers
model_reg = Sequential(name="Regularized_Model")
model_reg.add(Conv2D(32,kernel_size =(3, 3), activation='relu',input_shape=(128,128,3)))
model_reg.add(Conv2D(32,kernel_size =(3,3), activation='relu'))
model_reg.add(BatchNormalization())
model_reg.add(MaxPool2D(pool_size=(2, 2)))
model_reg.add(Conv2D(64,kernel_size =(3,3), activation='relu'))
model_reg.add(Conv2D(64,kernel_size =(3,3), activation='relu'))
model_reg.add(BatchNormalization())
model_reg.add(MaxPool2D(pool_size=(2, 2)))
model_reg.add(Conv2D(128,kernel_size =(3,3), activation='relu'))
model_reg.add(Conv2D(128,kernel_size =(3,3), activation='relu'))
model_reg.add(BatchNormalization())
model_reg.add(MaxPool2D(pool_size=(2, 2)))
model_reg.add(Flatten())
model_reg.add(Dense(1024,activation='relu'))
```

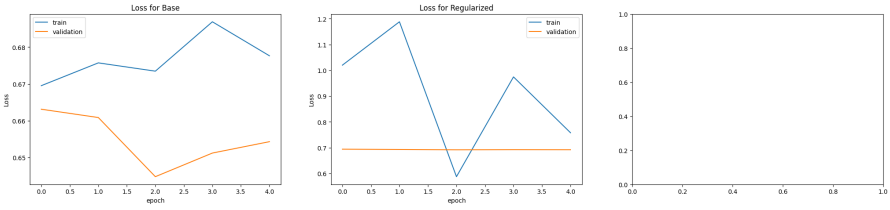
```
model_reg.add(Dropout(0.05))
model_reg.add(Dense(1024,activation='relu'))
model_reg.add(Dropout(0.05))
model_reg.add(Dense(1,activation = 'sigmoid'))
```

```
model_reg.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate = 1e-4), loss='binary_crossentropy', metrics=['accuracy'])
```

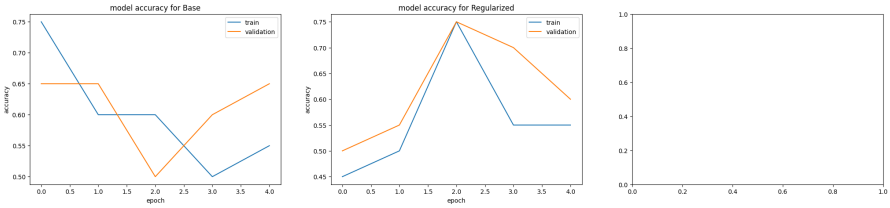
```
history_2 = model_reg.fit(train_data,validation_data=val_data,epochs=5)
```

```
Epoch 1/5
1/1 [==============================] - 7s 7s/step - loss: 1.0202 - accuracy: 0.4500 - val_loss: 0.6942 - val_accuracy: 0.5000
Epoch 2/5
1/1 [==============================] - 3s 3s/step - loss: 1.1884 - accuracy: 0.5000 - val_loss: 0.6927 - val_accuracy: 0.5500
Epoch 3/5
1/1 [==============================] - 3s 3s/step - loss: 0.5874 - accuracy: 0.7500 - val_loss: 0.6917 - val_accuracy: 0.7500
Epoch 4/5
1/1 [==============================] - 2s 2s/step - loss: 0.9746 - accuracy: 0.5500 - val_loss: 0.6922 - val_accuracy: 0.7000
Epoch 5/5
1/1 [==============================] - 5s 5s/step - loss: 0.7578 - accuracy: 0.5500 - val_loss: 0.6919 - val_accuracy: 0.6000
```

```
fig, ax = plt.subplots(1,3,figsize = (25,5))
ax[0].plot(history_1.history['loss'])
ax[0].plot(history_1.history['val_loss'])
ax[0].legend(['train', 'validation'])
ax[0].set_xlabel('epoch')
ax[0].set_ylabel('Loss')
ax[0].set_title('Loss for Base')
ax[1].plot(history_2.history['loss'])
ax[1].plot(history_2.history['val_loss'])
ax[1].legend(['train', 'validation'])
ax[1].set_xlabel('epoch')
ax[1].set_ylabel('Loss')
ax[1].set_title('Loss for Regularized')
plt.show()
```




```
fig, ax = plt.subplots(1,3,figsize = (25,5))
ax[0].plot(history_1.history['accuracy'])
ax[0].plot(history_1.history['val_accuracy'])
ax[0].legend(['train', 'validation'])
ax[0].set_xlabel('epoch')
ax[0].set_ylabel('accuracy')
ax[0].set_title('model accuracy for Base')
ax[1].plot(history_2.history['accuracy'])
ax[1].plot(history_2.history['val_accuracy'])
ax[1].legend(['train', 'validation'])
ax[1].set_xlabel('epoch')
ax[1].set_ylabel('accuracy')
ax[1].set_title('model accuracy for Regularized')
plt.show()
```




```
test_unbalanced_acc_2 = model_reg.evaluate(unbalanced_data,verbose=0)
print('Test Accuracy of Regularised Model on Unbalanced Test Dataset : {}'.format(test_unbalanced_acc_2[1] ))
```

```
test_balanced_acc_2= model_reg.evaluate(balanced_data,verbose =0)
print('Test Accuracy of Regularised Model on Balanced Test Dataset : {}'.format(test_balanced_acc_2[1] ))
metrics_2 = metrices(model_reg)
```

```
Test Accuracy of Regularised Model on Unbalanced Test Dataset : 0.5
Test Accuracy of Regularised Model on Balanced Test Dataset : 0.5
1/1 [==============================] - 1s 648ms/step
1/1 [==============================] - 0s 490ms/step
Precision on Unbalanced Test Dataset : 0.5
Precision on Balanced Test Dataset : 0.5
Recall on Unbalanced Test Dataset : 1.0
Recall on Balanced Test Dataset : 1.0
F1 Score on Unbalanced Test Dataset : 0.6666666666666666
F1 Score on Balanced Test Dataset : 0.6666666666666666
AUC score  on Unbalanced Test Dataset : 0.5
AUC score on Balanced Test Dataset : 0.5
```

```
from tensorflow.keras.applications.resnet import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.applications import imagenet_utils
from tensorflow.keras.layers import Dense,GlobalAveragePooling2D
from tensorflow.keras.applications.resnet50 import preprocess_input as prep_res
ResNmodel = ResNet50(input_shape=(128,128,3),weights='imagenet',include_top = False)
ResNmodel = Model(inputs=ResNmodel.input, outputs=ResNmodel.layers[-1].output)
ResNmodel.layers[-1].output
```

```
<KerasTensor: shape=(None, 4, 4, 2048) dtype=float32 (created by layer 'conv5_block3_out')>
```

```
x=ResNmodel.output
x=Flatten()(x)
x = Dropout(rate = 0.5)(x)
x=Dense(2048,activation= 'relu')(x)
x = Dropout(rate = 0.5)(x)
x=Dense(1024,activation = 'relu')(x)
x = Dropout(rate = 0.5)(x)
x=Dense(512,activation = 'relu')(x)
x = Dropout(rate = 0.5)(x)
x= Dense(128,activation = 'relu')(x)
preds=Dense(1,activation = 'sigmoid')(x)
```

```
model_resn=Model(ResNmodel.input,preds)
```

```
def data_generator(preprocess_input,target_size):
        datagen=ImageDataGenerator(preprocessing_function=preprocess_input,rotation_range=10, # rotation
                                   width_shift_range=0.2, # horizontal shift
                                   height_shift_range=0.3, # vertical shift
                                   zoom_range=0.4, # zoom
                                   horizontal_flip=True,# horizontal flip
                                    #re-scaling
                                   brightness_range=[0.3,1.1]) # brightness)

        datagen_2=ImageDataGenerator(preprocessing_function=preprocess_input,rotation_range=10, # rotation
                                   width_shift_range=0.2, # horizontal shift
                                   height_shift_range=0.3, # vertical shift
                                   zoom_range=0.4, # zoom
                                   horizontal_flip=True,# horizontal flip
                                    #re-scaling
                                   brightness_range=[0.3,1.1])
        test_unbalanced_gen_2 =  ImageDataGenerator(preprocessing_function=preprocess_input)
        test_balanced_gen_2 =  ImageDataGenerator(preprocessing_function=preprocess_input)

        train_data=datagen.flow_from_directory(  directory = train_directory,
                                       target_size = target_size,
                                           class_mode = 'binary',

                                           batch_size=16)

        validation_data=datagen_2.flow_from_directory( directory = validation_directory,
                                              target_size = target_size,
                                                  class_mode = 'binary',

                                                  batch_size=16)

        unbalanced_data = test_unbalanced_gen_2.flow_from_directory(directory =unbalanced_test_directory,
                                                            target_size = target_size,
```

```
                                                    class_mode = 'binary',

                                                    batch_size=16)

        balanced_data = test_balanced_gen_2.flow_from_directory(directory =balanced_test_directory,
                                                    target_size = target_size,
                                                    class_mode = 'binary',
                                                    batch_size=16)
        return train_data,validation_data,unbalanced_data,balanced_data
```

```
model_resn.compile(optimizer = tensorflow.keras.optimizers.Adam(2e-4),loss = tensorflow.keras.losses.BinaryCrossentropy(),metrics = ['accu
```

```
print(data_generator(prep_res,(128,128)))
```

```
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
(<keras.src.preprocessing.image.DirectoryIterator object at 0x7c368537e7d0>, <keras.src.preprocessing.image.DirectoryIterator object at
```

```
train_data_1,validation_data_1,unbalanced_data_1,balanced_data_1 = data_generator(prep_res,(128,128))
```

```
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
```

```
step_size_train=train_data_1.n//train_data_1.batch_size
step_size_validation=validation_data_1.n//validation_data_1.batch_size
model_resn.fit(train_data_1,validation_data=validation_data_1,epochs =5,batch_size = 64)
```

```
Epoch 1/5
2/2 [==============================] - 9s 4s/step - loss: 2.0607 - accuracy: 0.5500 - val_loss: 0.7000 - val_accuracy: 0.5500
Epoch 2/5
2/2 [==============================] - 11s 6s/step - loss: 1.3512 - accuracy: 0.8000 - val_loss: 0.5400 - val_accuracy: 0.8000
Epoch 3/5
2/2 [==============================] - 12s 6s/step - loss: 1.2763 - accuracy: 0.7000 - val_loss: 0.5355 - val_accuracy: 0.7000
Epoch 4/5
2/2 [==============================] - 11s 6s/step - loss: 1.1429 - accuracy: 0.7500 - val_loss: 1.0487 - val_accuracy: 0.6000
Epoch 5/5
2/2 [==============================] - 11s 7s/step - loss: 1.4270 - accuracy: 0.6500 - val_loss: 1.7432 - val_accuracy: 0.5000
<keras.src.callbacks.History at 0x7c36853e0640>
```

```
test_unbalanced_acc_3 = model_resn.evaluate(unbalanced_data_1,verbose=0)
print('Test Accuracy of model_mn on Unbalanced Test Dataset : {}'.format(test_unbalanced_acc_3[1] ))
test_balanced_acc_3= model_resn.evaluate(balanced_data_1,verbose =0)
print('Test Accuracy of model_mn on Balanced Test Dataset : {}'.format(test_balanced_acc_3[1] ))
metrics_3 = metrices(model_resn)
```

```
Test Accuracy of model_mn on Unbalanced Test Dataset : 0.5
Test Accuracy of model_mn on Balanced Test Dataset : 0.5
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 1s 1s/step
Precision on Unbalanced Test Dataset : 0.5
Precision on Balanced Test Dataset : 0.5
Recall on Unbalanced Test Dataset : 1.0
Recall on Balanced Test Dataset : 1.0
F1 Score on Unbalanced Test Dataset : 0.6666666666666666
F1 Score on Balanced Test Dataset : 0.6666666666666666
AUC score  on Unbalanced Test Dataset : 0.5
AUC score on Balanced Test Dataset : 0.5
```