```
from keras.models import load_model
import tensorflow as tf
from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score, ConfusionMatrixDisplay,
import time

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

```
# Montando o google drive para acessar imagens
from google.colab import drive
drive.mount('/content/gdrive')
path = "/content/gdrive/MyDrive/Colab Notebooks/"
```

```
    Mounted at /content/gdrive
```

Tamanho das imagens

```
IMAGE_SIZE = [400, 400]
```

```
train_path = path+'ebhi-split-2categorias/train'
valid_path = path+'ebhi-split-2categorias/val'
test_path = path+'ebhi-split-2categorias/test'
```

Declarando o modelo VGG16

```
vgg = VGG16(input_shape=IMAGE_SIZE + [3],
            weights='imagenet', include_top=False)
```

```
    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_
    58889256/58889256 [==============================] - 0s 0us/step
```

Não treina camadas

```
for layer in vgg.layers:
    layer.trainable = False
```

Adicionando as nossas camadas

```
x = Flatten()(vgg.output)
# x = Dense(1000, activation='relu')(x)
#prediction = Dense(len(folders), activation='softmax')(x)
prediction = Dense(1, activation='sigmoid')(x)
```

Criando o modelo

```
model = Model(inputs=vgg.input, outputs=prediction)
```

Visualizando a estrutura do modelo

```
model.summary()
```

```
    Model: "model"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #   
    =================================================================
     input_1 (InputLayer)        [(None, 400, 400, 3)]     0         
                                                                     
     block1_conv1 (Conv2D)       (None, 400, 400, 64)      1792      
                                                                     
     block1_conv2 (Conv2D)       (None, 400, 400, 64)      36928     
```

```
 block1_pool (MaxPooling2D)  (None, 200, 200, 64)      0

 block2_conv1 (Conv2D)       (None, 200, 200, 128)     73856

 block2_conv2 (Conv2D)       (None, 200, 200, 128)     147584

 block2_pool (MaxPooling2D)  (None, 100, 100, 128)     0

 block3_conv1 (Conv2D)       (None, 100, 100, 256)     295168

 block3_conv2 (Conv2D)       (None, 100, 100, 256)     590080

 block3_conv3 (Conv2D)       (None, 100, 100, 256)     590080

 block3_pool (MaxPooling2D)  (None, 50, 50, 256)       0

 block4_conv1 (Conv2D)       (None, 50, 50, 512)       1180160

 block4_conv2 (Conv2D)       (None, 50, 50, 512)       2359808

 block4_conv3 (Conv2D)       (None, 50, 50, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 25, 25, 512)       0

 block5_conv1 (Conv2D)       (None, 25, 25, 512)       2359808

 block5_conv2 (Conv2D)       (None, 25, 25, 512)       2359808

 block5_conv3 (Conv2D)       (None, 25, 25, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 12, 12, 512)       0

 flatten (Flatten)           (None, 73728)             0

 dense (Dense)               (None, 1)                 73729

=================================================================
Total params: 14788417 (56.41 MB)
Trainable params: 73729 (288.00 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

Declarando para o modelo as funções de custo e otimização

```
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    optimizer='adam',
    metrics=['accuracy']
)
```

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
```

```
test_datagen = ImageDataGenerator(rescale=1./255)
valid_datagen = ImageDataGenerator(rescale=1./255)
```

```
training_set = train_datagen.flow_from_directory(train_path,
                                                 target_size=(400, 400),
                                                 batch_size=32,
                                                 class_mode='binary')
```

```
Found 1163 images belonging to 2 classes.
```

```
test_set = test_datagen.flow_from_directory(test_path,
                                            target_size=(400, 400),
                                            batch_size=32,
                                            shuffle=False,
                                            class_mode='binary',classes=['ANORMAL','NORMAL'])
```

```
Found 390 images belonging to 2 classes.
```

```
valid_set = valid_datagen.flow_from_directory(valid_path,
                                              target_size=(400, 400),
                                              batch_size=32,
                                              class_mode='binary')
```

```
Found 387 images belonging to 2 classes.
```

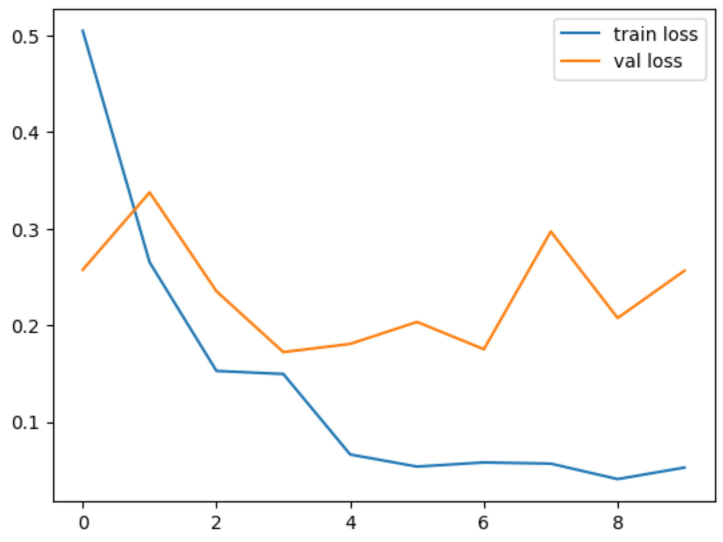
Treinando o modelo

```
r = model.fit(
    training_set,
    validation_data=valid_set,
    epochs=10,
    steps_per_epoch=len(training_set),
    validation_steps=len(valid_set)
)
model.save('hist_model_vgg.h5')
```

```
Epoch 1/10
37/37 [==============================] - 573s 15s/step - loss: 0.5050 - accuracy: 0.7876 - val_loss: 0.2576 - val_accura
Epoch 2/10
37/37 [==============================] - 99s 3s/step - loss: 0.2653 - accuracy: 0.8874 - val_loss: 0.3377 - val_accurac
Epoch 3/10
37/37 [==============================] - 97s 3s/step - loss: 0.1528 - accuracy: 0.9304 - val_loss: 0.2353 - val_accurac
Epoch 4/10
37/37 [==============================] - 88s 2s/step - loss: 0.1496 - accuracy: 0.9398 - val_loss: 0.1723 - val_accurac
Epoch 5/10
37/37 [==============================] - 87s 2s/step - loss: 0.0662 - accuracy: 0.9776 - val_loss: 0.1808 - val_accurac
Epoch 6/10
37/37 [==============================] - 86s 2s/step - loss: 0.0537 - accuracy: 0.9854 - val_loss: 0.2035 - val_accurac
Epoch 7/10
37/37 [==============================] - 86s 2s/step - loss: 0.0580 - accuracy: 0.9828 - val_loss: 0.1752 - val_accurac
Epoch 8/10
37/37 [==============================] - 98s 3s/step - loss: 0.0568 - accuracy: 0.9828 - val_loss: 0.2971 - val_accurac
Epoch 9/10
37/37 [==============================] - 87s 2s/step - loss: 0.0407 - accuracy: 0.9888 - val_loss: 0.2076 - val_accurac
Epoch 10/10
37/37 [==============================] - 86s 2s/step - loss: 0.0527 - accuracy: 0.9794 - val_loss: 0.2566 - val_accurac
```

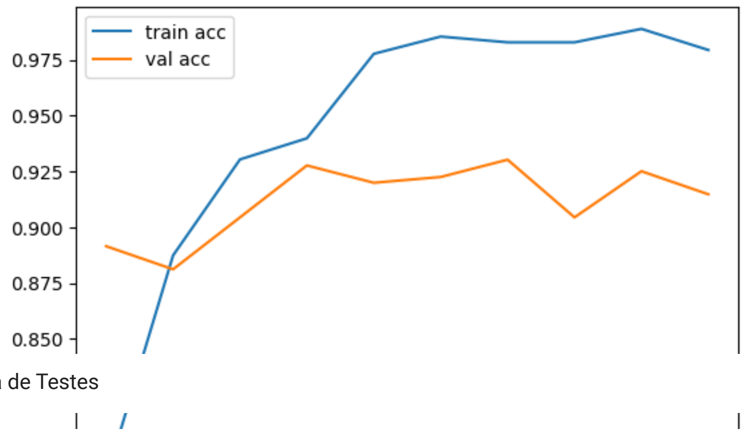
Perda do treino

```
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.savefig('LossVal_loss_vgg')
plt.show()
```



Acurácias do treino

```
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.savefig('AccVal_acc_vgg')
plt.show()
```



Etapa de Testes

```
vgg_model = model

t = time.time()
# Usando o modelo para predição das amostras de teste
y_pred = vgg_model.predict(test_set)
# Reset
test_set.reset()
#loss, acc = vgg_model.evaluate(test_set)
#aux = np.argmax(aux, axis=1)
y_pred = np.where(y_pred > 0.5, 1, 0).flatten()
print("y predito:")
print(y_pred)
y_true = test_set.classes
print("y real:")
print(y_true)
# Método para calcular o valor F1-Score
print('F1-Score: {}'.format(f1_score(y_true, y_pred, average='macro')))
# Método para calcular a Precision
print('Precision : {}'.format(precision_score(y_true, y_pred, average='macro')))
# Método para calcular o Recall
print('Recall: {}'.format(recall_score(y_true, y_pred, average='macro')))

print('Matriz de Confusão:')
cm = confusion_matrix(y_true, y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['Anormal','Normal'])
cm_display.plot()
plt.savefig('Matriz-vgg16')
plt.show()

print ('Accuracy score: ', accuracy_score(y_true, y_pred))
#print('Acuracia obtida com o Vgg16 no Conjunto de Teste EBHI: {:.2f}'.format(
#    acc))
```

```
13/13 [==============================] - 187s 16s/step
y predito:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 1 1 0 1 0 1 1 1 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1
 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
y real:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Avaliando no UnitoPatho

```
Precision : 0.917581257914732
```

```
from PIL import Image
from skimage import transform
#funcao carregar imagem
def load(filename):
   np_image = Image.open(filename)
   np_image = np.array(np_image).astype('float32')/255
   np_image = transform.resize(np_image, (400, 400, 3))
   np_image = np.expand_dims(np_image, axis=0)
   return np_image


#pred2 = load(path+"dataset-unitopatho/ANORMAL/54-B2-TAHG.ndpi_ROI__mpp0.44_reg000_crop_sk00000_(73992,7343,1812,1812).png"
#pred2 = vgg_model.predict(pred2)
#print(np.where(pred2 > 0.5, 'Normal', 'Anormal').flatten())


test_path_uni = path+'/dataset-unitopatho/'
test_datagen_uni = ImageDataGenerator(rescale=1./255)
test_set_uni = test_datagen_uni.flow_from_directory(test_path_uni,
                                            target_size=(400, 400),
                                            batch_size=32,
                                            shuffle=False,
                                            class_mode='binary',classes=['ANORMAL','NORMAL'])



y_pred = vgg_model.predict(test_set_uni)
# Reset
test_set_uni.reset()
#loss, acc = vgg_model.evaluate(test_set_uni)
#aux = np.argmax(aux, axis=1)
y_pred = np.where(y_pred > 0.5, 1, 0).flatten()
print("y predito:")
print(y_pred)
y_true = test_set_uni.classes
print("y real:")
print(y_true)
# Método para calcular o valor F1-Score
print('F1-Score: {}'.format(f1_score(y_true, y_pred, average='macro')))
# Método para calcular a Precision
print('Precision : {}'.format(precision_score(y_true, y_pred, average='macro')))
# Método para calcular o Recall
print('Recall: {}'.format(recall_score(y_true, y_pred, average='macro')))


print('Matriz de Confusão:')
cm = confusion_matrix(y_true, y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['Anormal','Normal'])
cm_display.plot()
plt.savefig('Matriz-vgg16-UNITOPATHO')
plt.show()

print ('Accuracy score: ', accuracy_score(y_true, y_pred))
#print('Acuracia obtida com o Vgg16 no Conjunto de Teste UNITOPATHO: {:.2f}'.format(
#    acc))
```

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 1 1 0
 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0]
y_real:
```