```
from google.colab import drive
drive.mount('/content/drive')
import os
workdir_path = '/content/drive/My Drive/Fugitivas' #OK!
os.chdir(workdir_path)
```

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

Clique duas vezes (ou pressione "Enter") para editar

Importando base do API Keras (que realiza interligação entre o Tensoflow e a base de dados. E utilizando uma CNN já treinada (*transfer learning*)

```
import sys
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
# from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import cv2
import os
import glob
from imutils import paths
```

```
INIT_LR = 1e-3
EPOCHS = 50
BS = 8
```

O dataset de imagens foi gerado a partir de videos de detecção de vazamentos em componentes de linhas de processo (flanges e caps). Foram realizados recortes de figuras nos videos que forma salvas em .jpg

```
# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
dataset_path = 'Dataset2'
print("[INFO] loading images...")
imagePaths = list(paths.list_images(dataset_path))
data = []
labels = []
```

```
[INFO] loading images...
```

```
# loop over the image paths
for imagePath in imagePaths:
    # extract the class label from the filename
    label = imagePath.split(os.path.sep)[-2]

    # load the image, swap color channels, and resize it to be a fixed
    # 224x224 pixels while ignoring aspect ratio
    image = cv2.imread(imagePath)
```

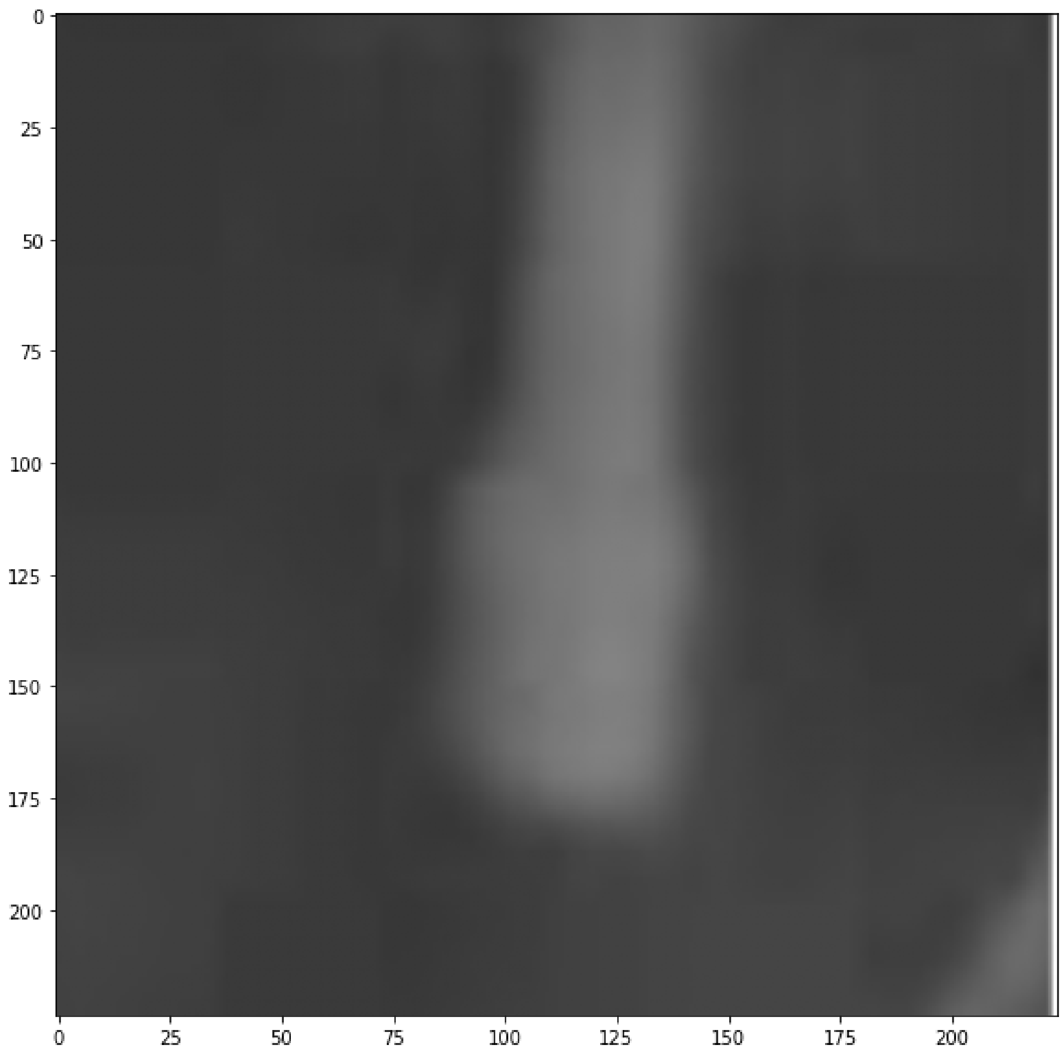
```
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (224, 224))

    # update the data and labels lists, respectively
    data.append(image)
    labels.append(label)
print("labels: ", np.unique(labels))
```

```
labels:  ['COMfugitivas' 'SEMfugitivas']
```

```
# convert the data and labels to NumPy arrays while scaling the pixel
# intensities to the range [0, 255]
data = np.array(data) / 255.0
labels = np.array(labels)
```

```
plt.figure(figsize=(10, 10))
plt.imshow(data[labels=='SEMfugitivas'][5])
plt.show()
```




```
plt.figure(figsize=(10, 10))
plt.imshow(data[labels=='COMfugitivas'][5])
plt.show()
```


```
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
```

```
# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
                                                  test_size=0.20, stratify=labels, random_state=42)
```

Utilização de Data Augmentation de forma que o modelo generalize melhor e treine com mais dados

```
# initialize the training data augmentation object
trainAug = ImageDataGenerator(
    rotation_range=30,
    fill_mode="nearest")
```

Clique duas vezes (ou pressione "Enter") para editar

```
# load the VGG16 network, ensuring the head FC layer sets are left
# off
baseModel = VGG16(weights="imagenet", include_top=False,
                  input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

```
# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
              metrics=["accuracy"])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_t
58892288/58889256 [==============================] - 0s 0us/step
[INFO] compiling model...
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2.py:375: UserWa
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

```
baseModel = VGG16(weights="imagenet", include_top=False,
                  input_tensor=Input(shape=(224, 224, 3)))

f1 = baseModel.layers[1].output
f2 = baseModel.layers[2].output
f3 = baseModel.layers[4].output
f4 = baseModel.layers[5].output
feature_maps = Model(inputs=baseModel.input, outputs=[f1, f2, f3, f4])
```

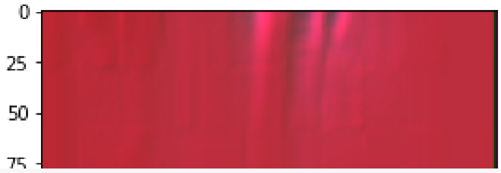
```
y = np.argmax(trainY, axis=-1)
```

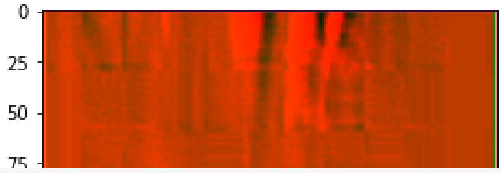
```
feat1, feat2, feat3, feat4 = feature_maps.predict(trainX[y==1][0:1])
```

Clique duas vezes (ou pressione "Enter") para editar

```
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
axs[0, 0].imshow(feat1[0, :, :, 0:3])
axs[0, 1].imshow(feat2[0, :, :, 6:9])
axs[1, 0].imshow(feat3[0, :, :, 0:3])
axs[1, 1].imshow(feat4[0, :, :, 3:6])
# plt.subplots
# plt.imshow(feat1[0, :, :, 0:3])
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integ
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integ
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integ
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integ
```






```
# train the head of the network
print("[INFO] training head...")
H = model.fit_generator(
    trainAug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)
```

```
Epoch 22/50
3/3 [==============================] - 9s 3s/step - loss: 0.5160 - accuracy: 0.9565 - val_loss: 0.603
Epoch 23/50
3/3 [==============================] - 9s 3s/step - loss: 0.5825 - accuracy: 0.7826 - val_loss: 0.598
Epoch 24/50
3/3 [==============================] - 9s 3s/step - loss: 0.5067 - accuracy: 0.9565 - val_loss: 0.593
Epoch 25/50
3/3 [==============================] - 9s 3s/step - loss: 0.5541 - accuracy: 0.7391 - val_loss: 0.586
Epoch 26/50
3/3 [==============================] - 9s 3s/step - loss: 0.4992 - accuracy: 0.9130 - val_loss: 0.581
Epoch 27/50
3/3 [==============================] - 9s 3s/step - loss: 0.4531 - accuracy: 1.0000 - val_loss: 0.576
Epoch 28/50
3/3 [==============================] - 9s 3s/step - loss: 0.4549 - accuracy: 0.8333 - val_loss: 0.571
Epoch 29/50
3/3 [==============================] - 9s 3s/step - loss: 0.4643 - accuracy: 0.9130 - val_loss: 0.566
Epoch 30/50
3/3 [==============================] - 9s 3s/step - loss: 0.5020 - accuracy: 0.8696 - val_loss: 0.562
Epoch 31/50
3/3 [==============================] - 9s 3s/step - loss: 0.4816 - accuracy: 0.8333 - val_loss: 0.556
Epoch 32/50
3/3 [==============================] - 9s 3s/step - loss: 0.4541 - accuracy: 0.8696 - val_loss: 0.550
Epoch 33/50
3/3 [==============================] - 9s 3s/step - loss: 0.4664 - accuracy: 0.8333 - val_loss: 0.543
Epoch 34/50
3/3 [==============================] - 9s 3s/step - loss: 0.4976 - accuracy: 0.8696 - val_loss: 0.538
Epoch 35/50
3/3 [==============================] - 9s 3s/step - loss: 0.4087 - accuracy: 0.9167 - val_loss: 0.534
Epoch 36/50
3/3 [==============================] - 9s 3s/step - loss: 0.3954 - accuracy: 0.9565 - val_loss: 0.529
Epoch 37/50
3/3 [==============================] - 9s 3s/step - loss: 0.4816 - accuracy: 0.8696 - val_loss: 0.523
Epoch 38/50
3/3 [==============================] - 9s 3s/step - loss: 0.4037 - accuracy: 0.8261 - val_loss: 0.517
Epoch 39/50
3/3 [==============================] - 9s 3s/step - loss: 0.3903 - accuracy: 0.9565 - val_loss: 0.511
Epoch 40/50
3/3 [==============================] - 9s 3s/step - loss: 0.4207 - accuracy: 0.7917 - val_loss: 0.505
Epoch 41/50
3/3 [==============================] - 9s 3s/step - loss: 0.3791 - accuracy: 0.9130 - val_loss: 0.499
Epoch 42/50
3/3 [==============================] - 9s 3s/step - loss: 0.4406 - accuracy: 0.7826 - val_loss: 0.493
Epoch 43/50
3/3 [==============================] - 9s 3s/step - loss: 0.3500 - accuracy: 0.8696 - val_loss: 0.487
Epoch 44/50
3/3 [==============================] - 9s 3s/step - loss: 0.4418 - accuracy: 0.8261 - val_loss: 0.481
Epoch 45/50
3/3 [==============================] - 9s 3s/step - loss: 0.3826 - accuracy: 0.9130 - val_loss: 0.475
Epoch 46/50
3/3 [==============================] - 9s 3s/step - loss: 0.3310 - accuracy: 0.9583 - val_loss: 0.469
Epoch 47/50
3/3 [==============================] - 9s 3s/step - loss: 0.3159 - accuracy: 0.9565 - val_loss: 0.464
```

```
Epoch 48/50
3/3 [==============================] - 9s 3s/step - loss: 0.3352 - accuracy: 0.9130 - val_loss: 0.457
Epoch 49/50
3/3 [==============================] - 9s 3s/step - loss: 0.3454 - accuracy: 0.9565 - val_loss: 0.450
Epoch 50/50
3/3 [==============================] - 9s 3s/step - loss: 0.2272 - accuracy: 1.0000 - val_loss: 0.444
```

```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
                            target_names=lb.classes_))

# compute the confusion matrix and and use it to derive the raw
# accuracy, sensitivity, and specificity
cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])

# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy on Fugitivas Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()
# plt.savefig(args["plot"])

# serialize the model to disk
# print("[INFO] saving Fugitivas detector model...")
# model.save(args["model"], save_format="h5")
```

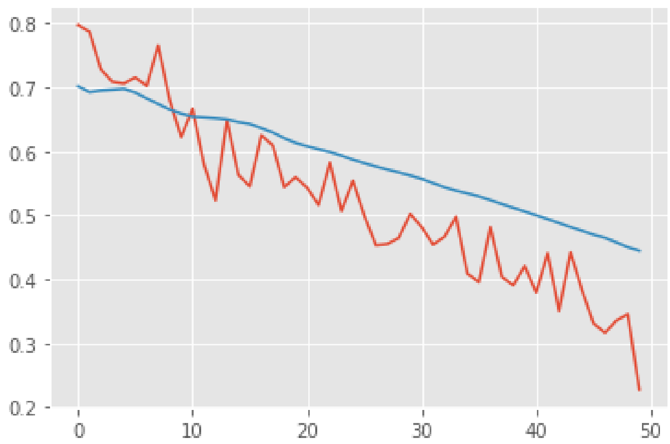
```
[INFO] evaluating network...
              precision    recall  f1-score   support

COMfugitivas       1.00      0.75      0.86         4
SEMfugitivas       0.80      1.00      0.89         4

    accuracy                           0.88         8
   macro avg       0.90      0.88      0.87         8
weighted avg       0.90      0.88      0.87         8

[[3 1]
 [0 4]]
acc: 0.8750
sensitivity: 0.7500
specificity: 1.0000
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-22-f29b7c00ae8e> in <module>()
     31 plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
     32 plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
---> 33 plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
     34 plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
     35 plt.title("Training Loss and Accuracy on Fugitivas Dataset")

KeyError: 'acc'
```



✓ 0s conclusão: 17:19