Detector de Deepfake

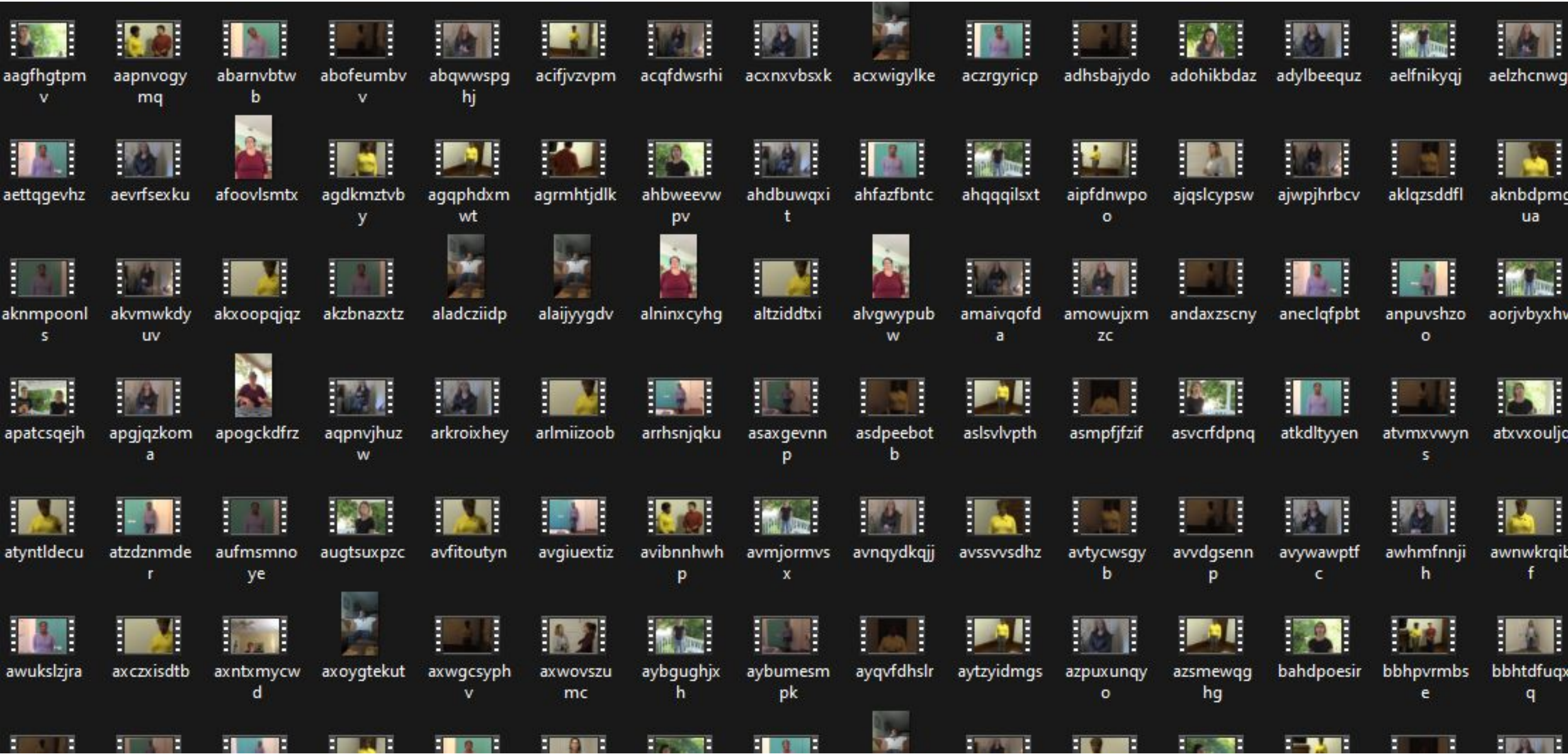
Henrique Kenzo Odaguiri - 148413
Jamil Anderson Mansur - 163812

Dataset

Deepfake Detection Challenge
https://www.kaggle.com/competitions/deepfake-detection-challenge/data

The data is comprised of .mp4 files, split into compressed sets of ~10GB apiece. A metadata.json accompanies each set of .mp4 files, and contains filename, label (REAL/FAKE), original and split columns, listed below under **Columns**.

train_sample_videos (401 files)



aagfhgtpmv aapnvogymq abarnvbtwb abofeumbvv abqwwspghj acifjvzvpm acqfdwsrhi acxnxvbsxk acxwigylke aczrgyricp adhsbajydo adohikbdaz adylbeequz aelfnikyqj aelzhcnwg

aettqgevhz aevrfsexku afoovlsmtx agdkmztvby agqphdxmwt agrmhtjdlk ahbweevwpv ahdbuwqxit ahfazfbntc ahqqqilsxt aipfdnwpoo ajqslcypsw ajwpjhrbcv aklqzsddfl aknbdpmgua

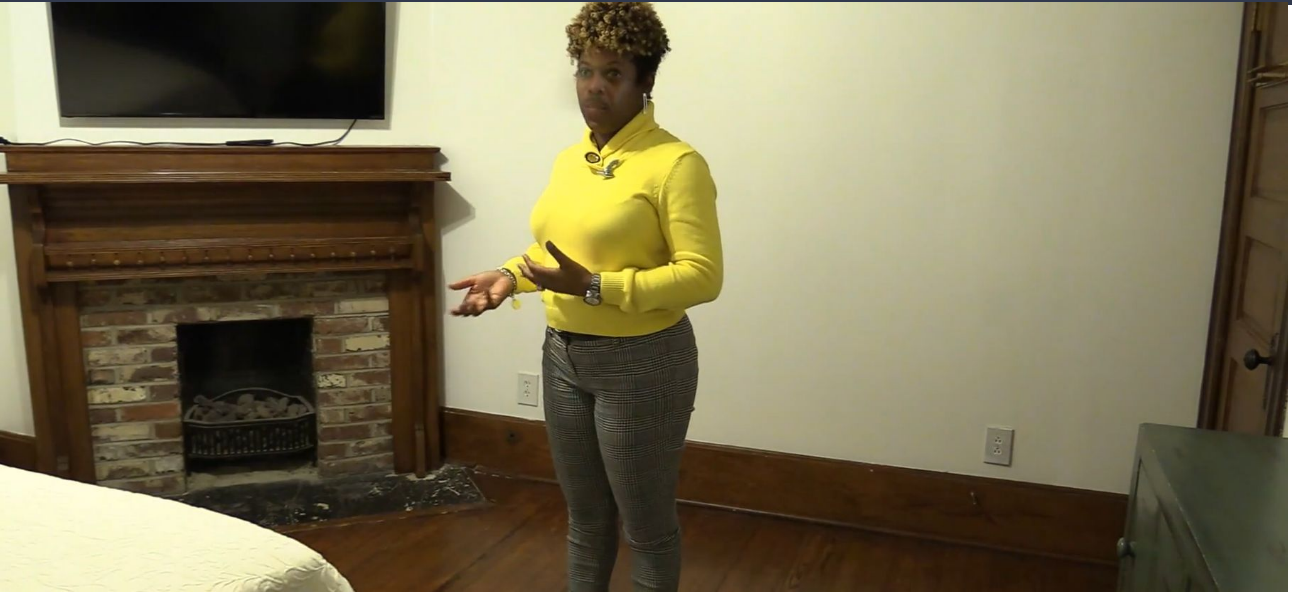
aknmpoonls akvmwkdyuv akxoopqjqz akzbnazxtz aladcziidp alaijyygdv alninxcyhg altziddtxi alvgwypubw amaivqofda amowujxmzc andaxzscny aneclqfpbt anpuvshzoo aorjvbyxhv

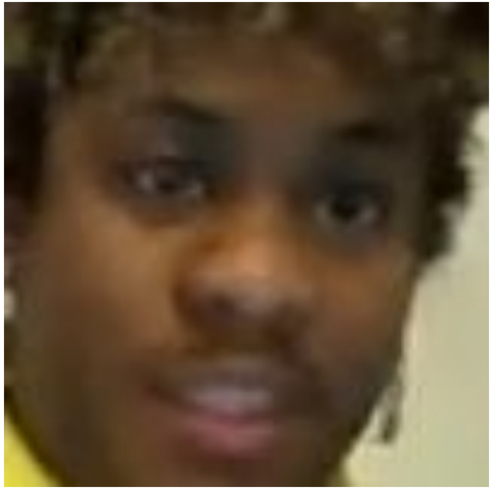
apatcsqejh apgjqzkoma apogckdfrz aqpnvjhuzw arkroixhey arlmiizoob arrhsnjqku asaxgevnnp asdpeebotb aslsvlvpth asmpfjfzif asvcrfdpnq atkdltyyen atvmxvwyns atxvxoulj

atyntldecu atzdznmder aufmsmnoye augtsuxpzc avfitoutyn avgiuextiz avibnnhwhp avmjormvsx avnqydkqjj avssvvsdhz avtycwsgyb avvdgsennp avywawptfc awhmfnnjih awnwkrqibf

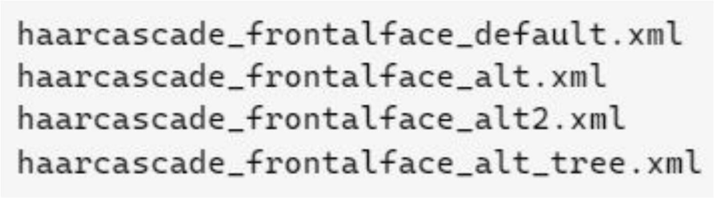
awukslzjra axczxisdtb axntxmycwd axoygtekut axwgcsyphv axwovszumc aybgughjxh aybumesmpk ayqvfdhslr aytzyidmgs azpuxunqyo azsmewqghg bahdpoesir bbhpvrmbse bbhtdfuqxq

Detecção e recorte de faces





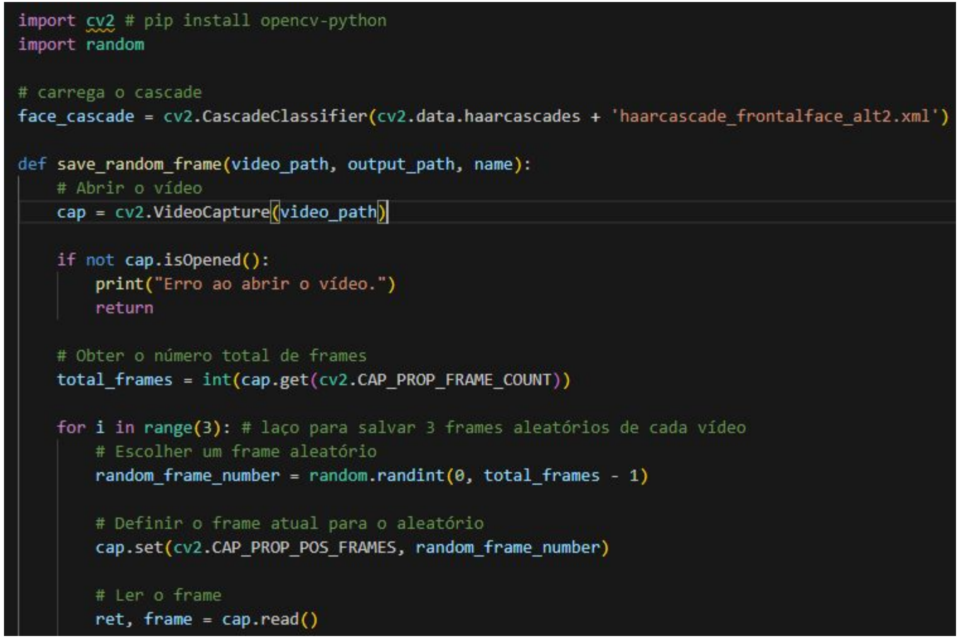
Detecção e recorte de faces



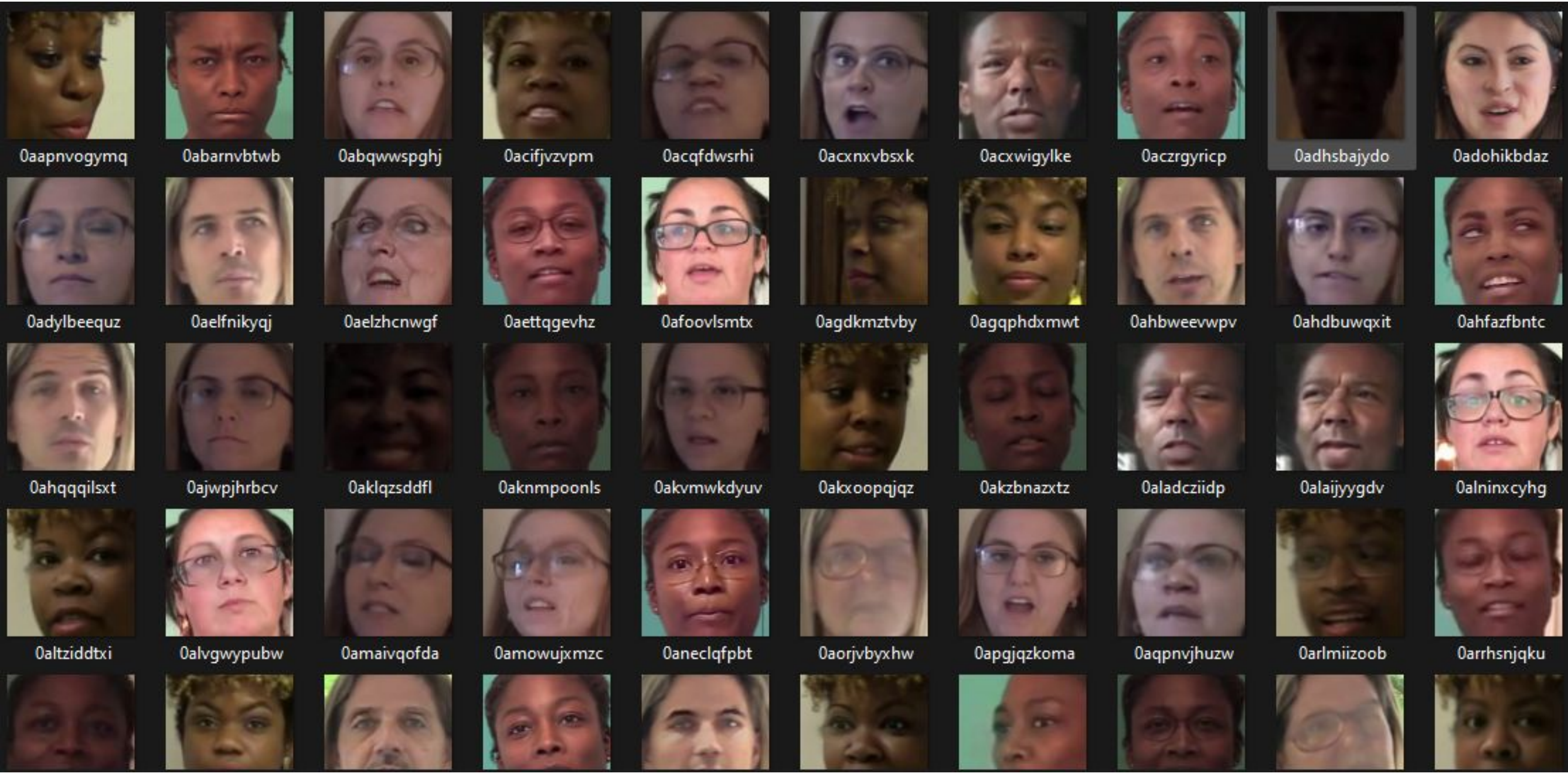
```
haarcascade_frontalface_default.xml
haarcascade_frontalface_alt.xml
haarcascade_frontalface_alt2.xml
haarcascade_frontalface_alt_tree.xml
```

import cv2 # pip install opencv-python

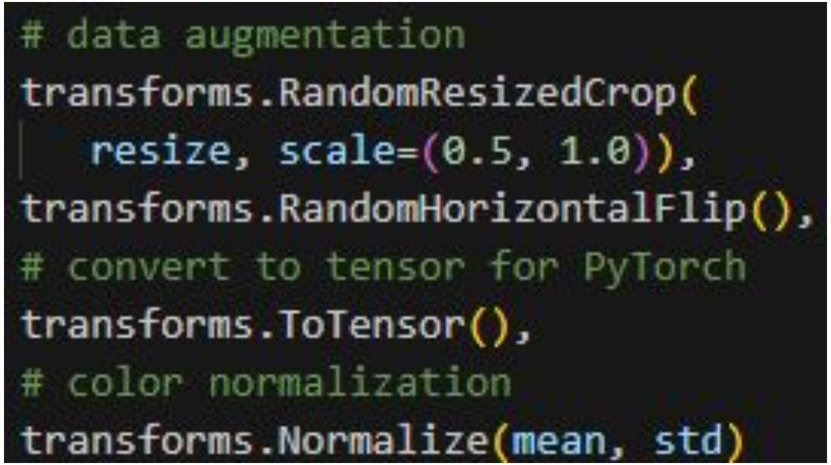
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_alt2.xml')



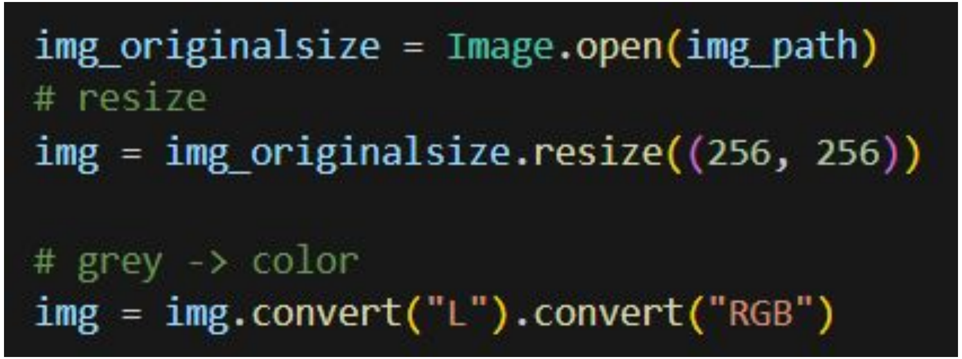
```python
import cv2 # pip install opencv-python
import random

# carrega o cascade
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_alt2.xml')

def save_random_frame(video_path, output_path, name):
    # Abrir o vídeo
    cap = cv2.VideoCapture(video_path)

    if not cap.isOpened():
        print("Erro ao abrir o vídeo.")
        return

    # Obter o número total de frames
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    for i in range(3): # laço para salvar 3 frames aleatórios de cada vídeo
        # Escolher um frame aleatório
        random_frame_number = random.randint(0, total_frames - 1)

        # Definir o frame atual para o aleatório
        cap.set(cv2.CAP_PROP_POS_FRAMES, random_frame_number)

        # Ler o frame
        ret, frame = cap.read()
```


0aapnvogymq
0abarnvbtwb
0abqwwspghj
0acifjvzvpm
0acqfdwsrhi
0acxnxvbsxk
0acxwigylke
0aczrgyricp
0adhsbajydo
0adohikbdaz
0adylbeequz
0aelfnikyqj
0aelzhcnwgf
0aettqgevhz
0afoovlsmtx
0agdkmztvby
0agqphdxmwt
0ahbweevwpv
0ahdbuwqxit
0ahfazfbntc
0ahqqqilsxt
0ajwpjhrbcv
0aklqzsddfl
0aknmpoonls
0akvmwkdyuv
0akxoopqjqz
0akzbnazxtz
0aladcziidp
0alaijyygdv
0alninxcyhg
0altziddtxi
0alvgwypubw
0amaivqofda
0amowujxmzc
0aneclqfpbt
0aorjvbyxhw
0apgjqzkoma
0aqpnvjhuzw
0arlmiizoob
0arrhsnjqku

Aumento de dados e Normalização

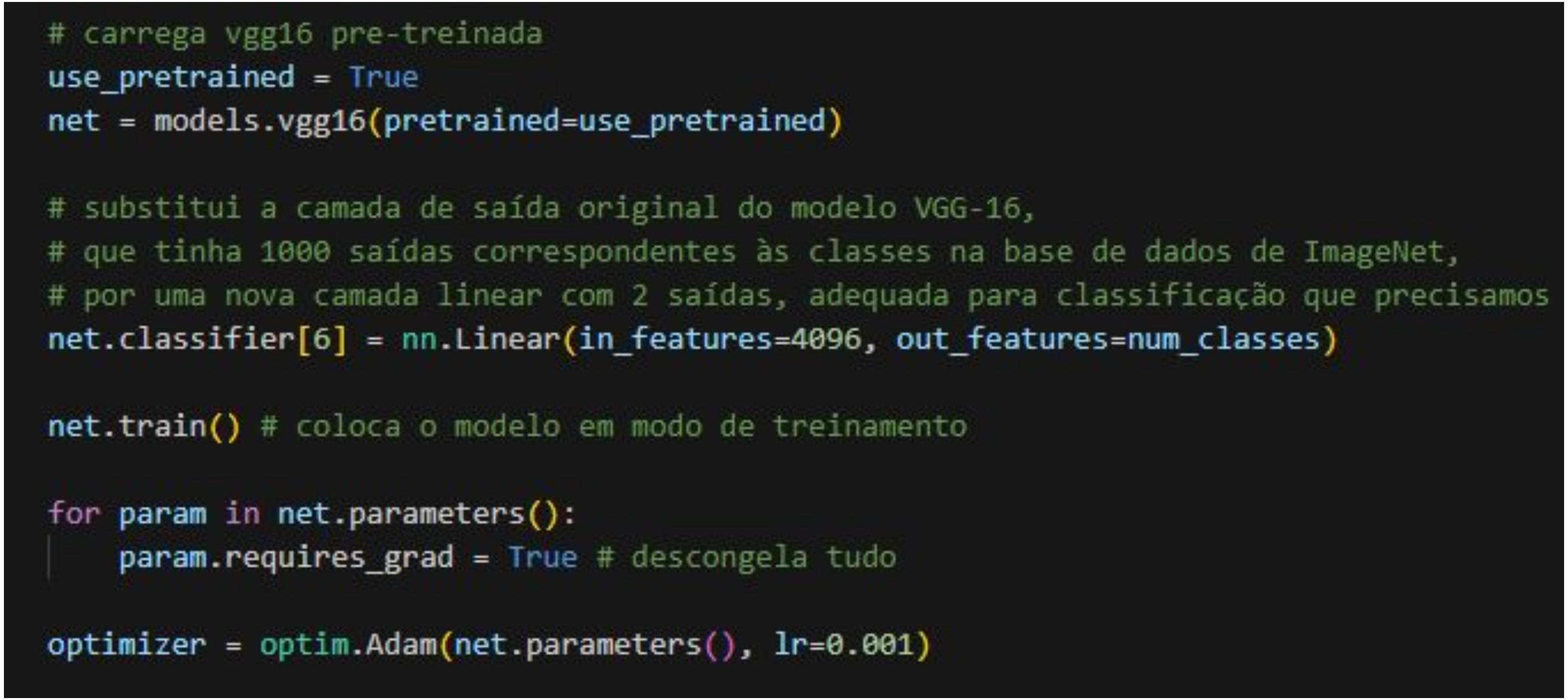


```
# data augmentation
transforms.RandomResizedCrop(
    resize, scale=(0.5, 1.0)),
transforms.RandomHorizontalFlip(),
# convert to tensor for PyTorch
transforms.ToTensor(),
# color normalization
transforms.Normalize(mean, std)
```



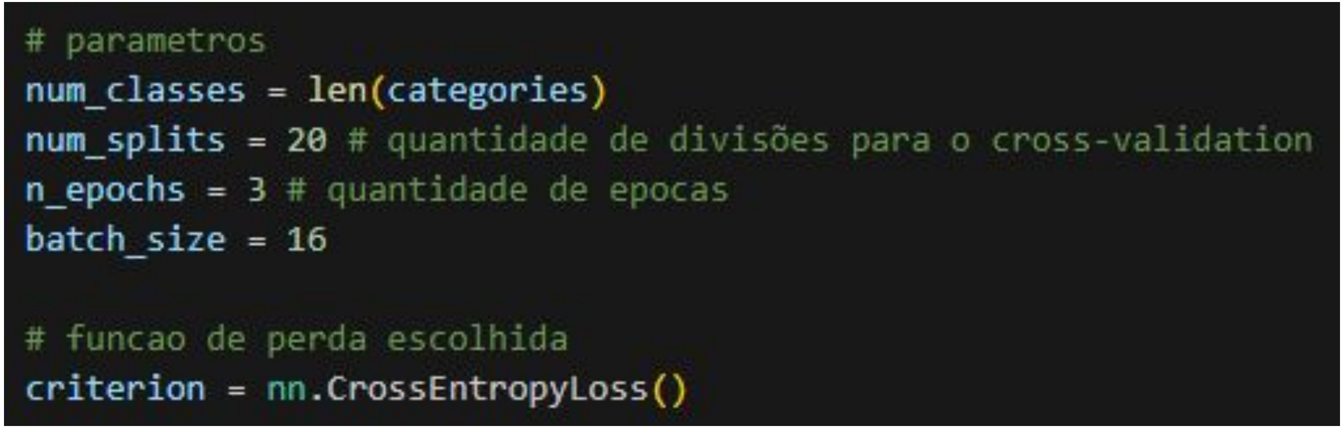
```
img_originalsize = Image.open(img_path)
# resize
img = img_originalsize.resize((256, 256))

# grey -> color
img = img.convert("L").convert("RGB")
```

Arquitetura VGG16



```
# carrega vgg16 pre-treinada
use_pretrained = True
net = models.vgg16(pretrained=use_pretrained)

# substitui a camada de saída original do modelo VGG-16,
# que tinha 1000 saídas correspondentes às classes na base de dados de ImageNet,
# por uma nova camada linear com 2 saídas, adequada para classificação que precisamos
net.classifier[6] = nn.Linear(in_features=4096, out_features=num_classes)

net.train() # coloca o modelo em modo de treinamento

for param in net.parameters():
    param.requires_grad = True # descongela tudo

optimizer = optim.Adam(net.parameters(), lr=0.001)
```

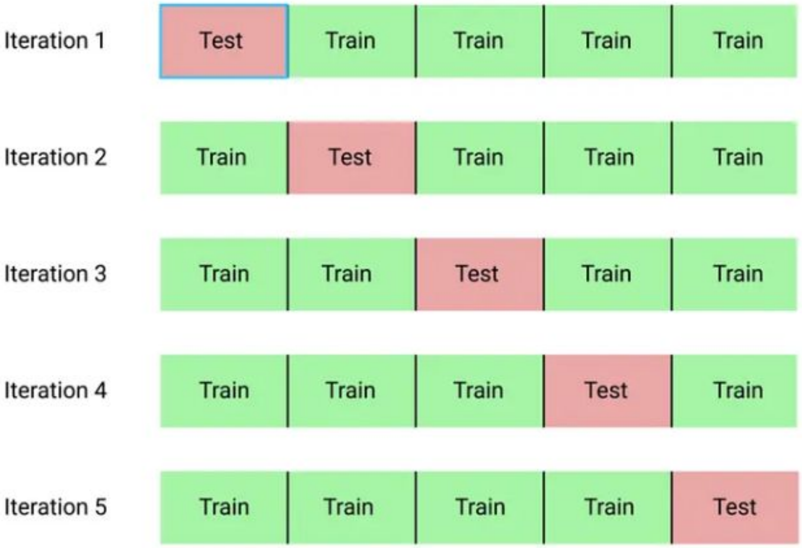
Parâmetros e função de perda



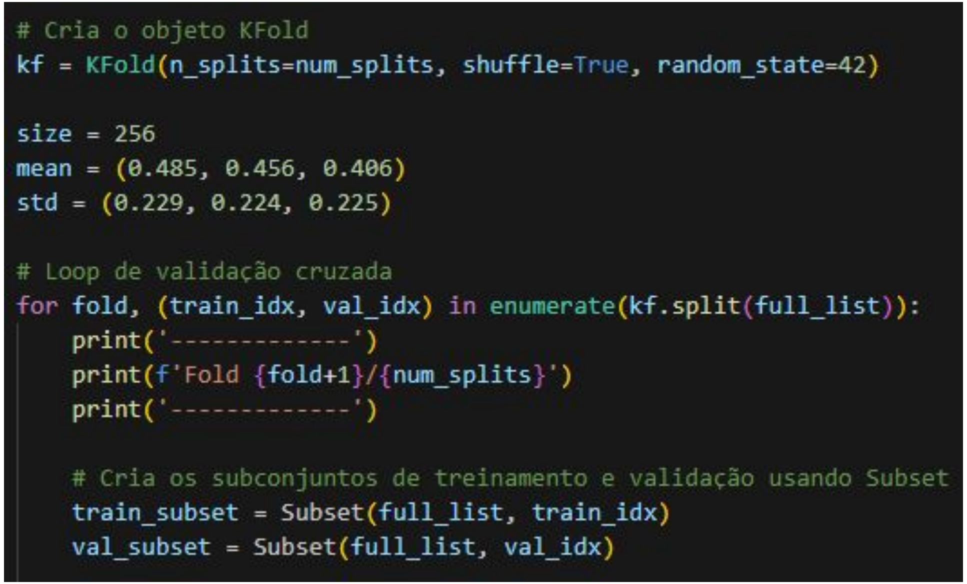
```
# parametros
num_classes = len(categories)
num_splits = 20 # quantidade de divisões para o cross-validation
n_epochs = 3 # quantidade de epocas
batch_size = 16

# funcao de perda escolhida
criterion = nn.CrossEntropyLoss()
```

K-folds

from sklearn.model_selection import KFold

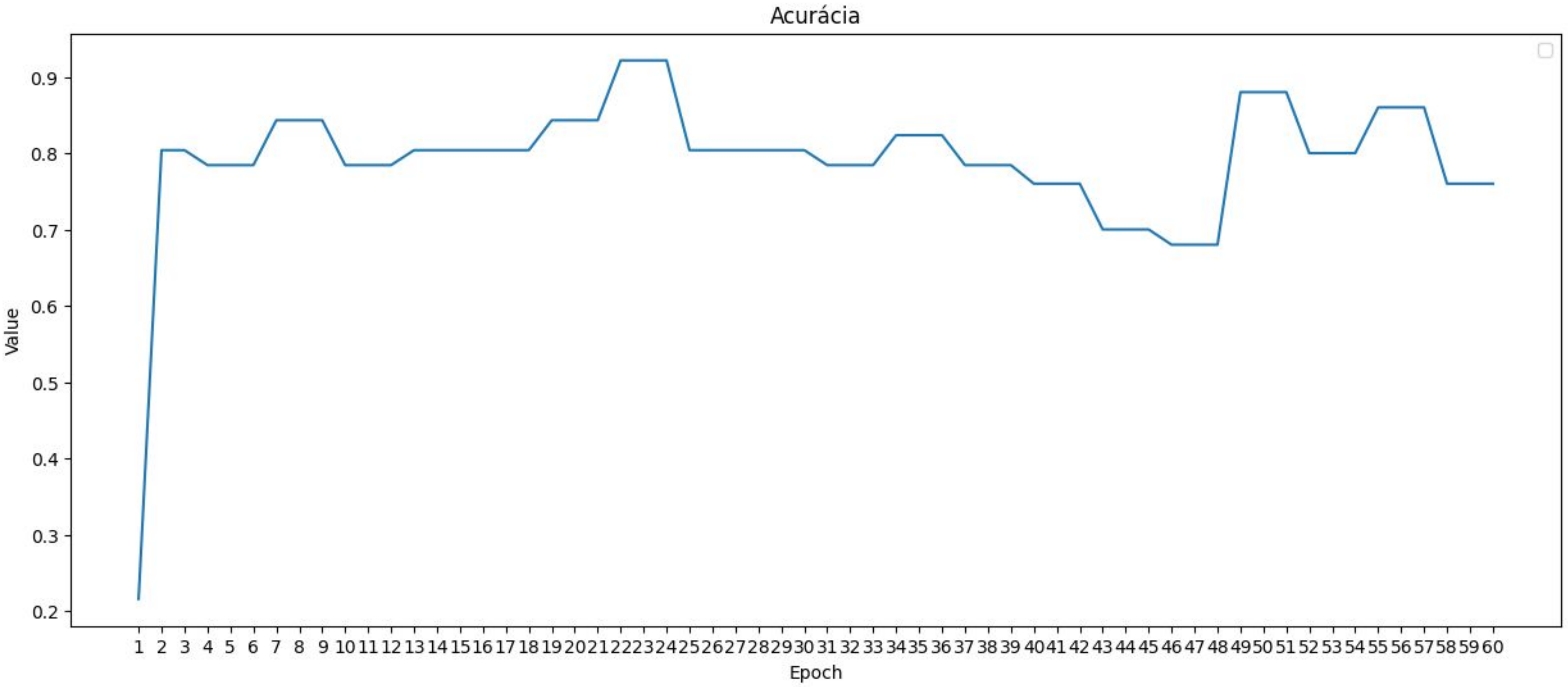


Iteration 1	Test	Train	Train	Train	Train
Iteration 2	Train	Test	Train	Train	Train
Iteration 3	Train	Train	Test	Train	Train
Iteration 4	Train	Train	Train	Test	Train
Iteration 5	Train	Train	Train	Train	Test

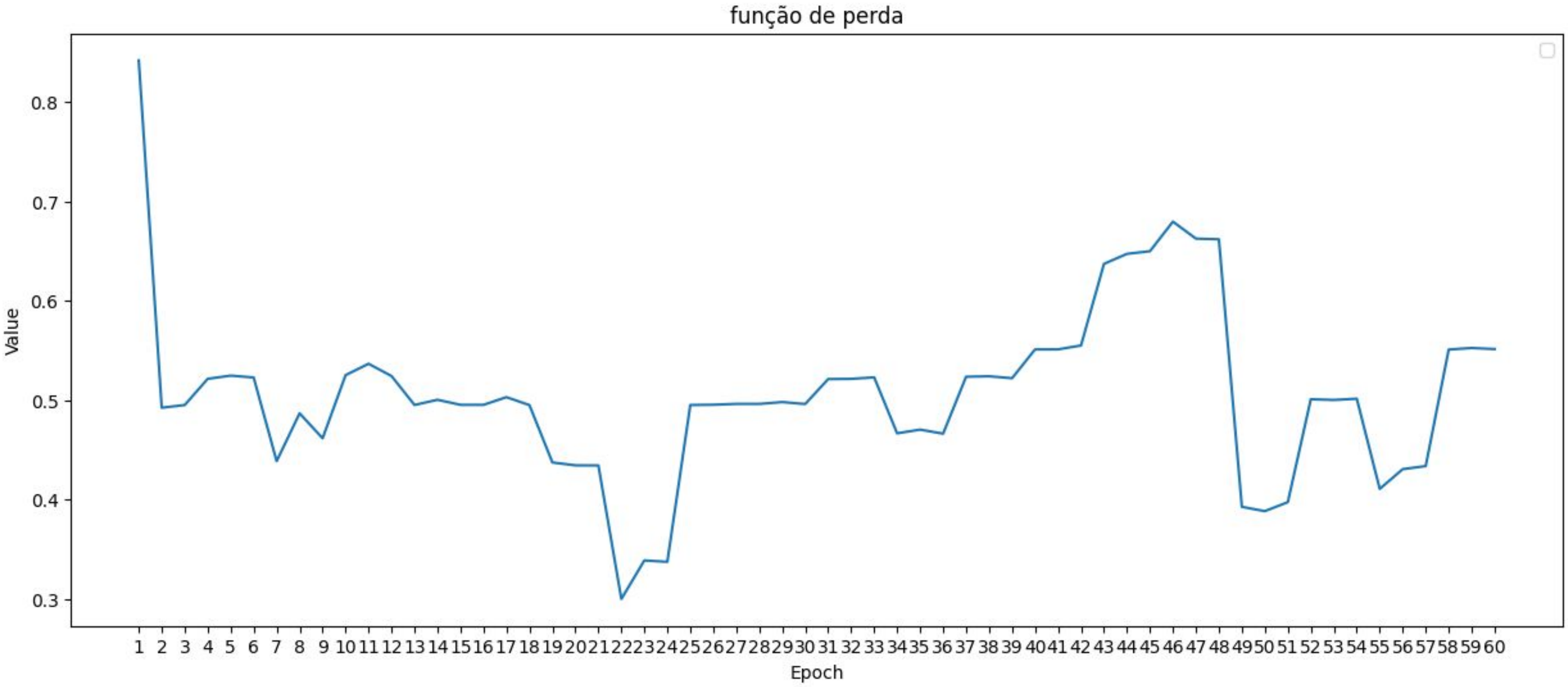


```python
# Cria o objeto KFold
kf = KFold(n_splits=num_splits, shuffle=True, random_state=42)

size = 256
mean = (0.485, 0.456, 0.406)
std = (0.229, 0.224, 0.225)

# Loop de validação cruzada
for fold, (train_idx, val_idx) in enumerate(kf.split(full_list)):
    print('-------------')
    print(f'Fold {fold+1}/{num_splits}')
    print('-------------')

    # Cria os subconjuntos de treinamento e validação usando Subset
    train_subset = Subset(full_list, train_idx)
    val_subset = Subset(full_list, val_idx)
```

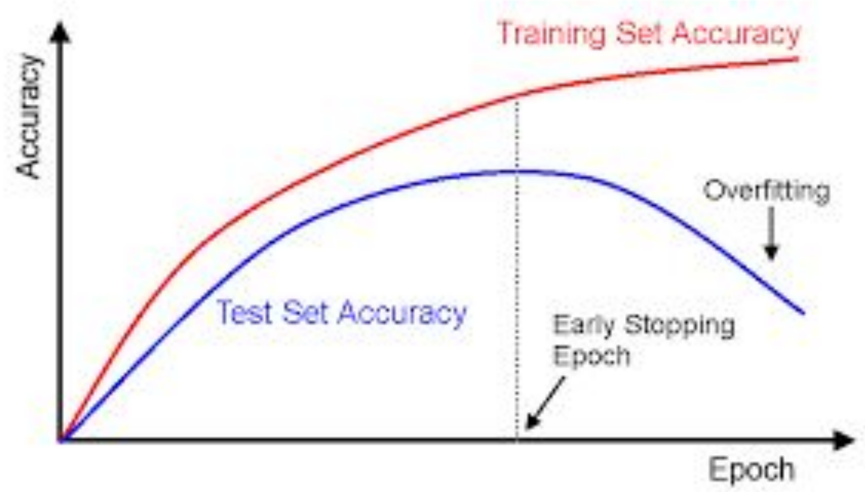
Acurácia

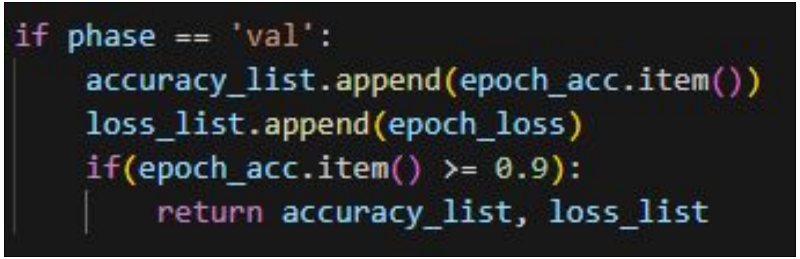

Acurácia
Value
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Epoch
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

Função de perda



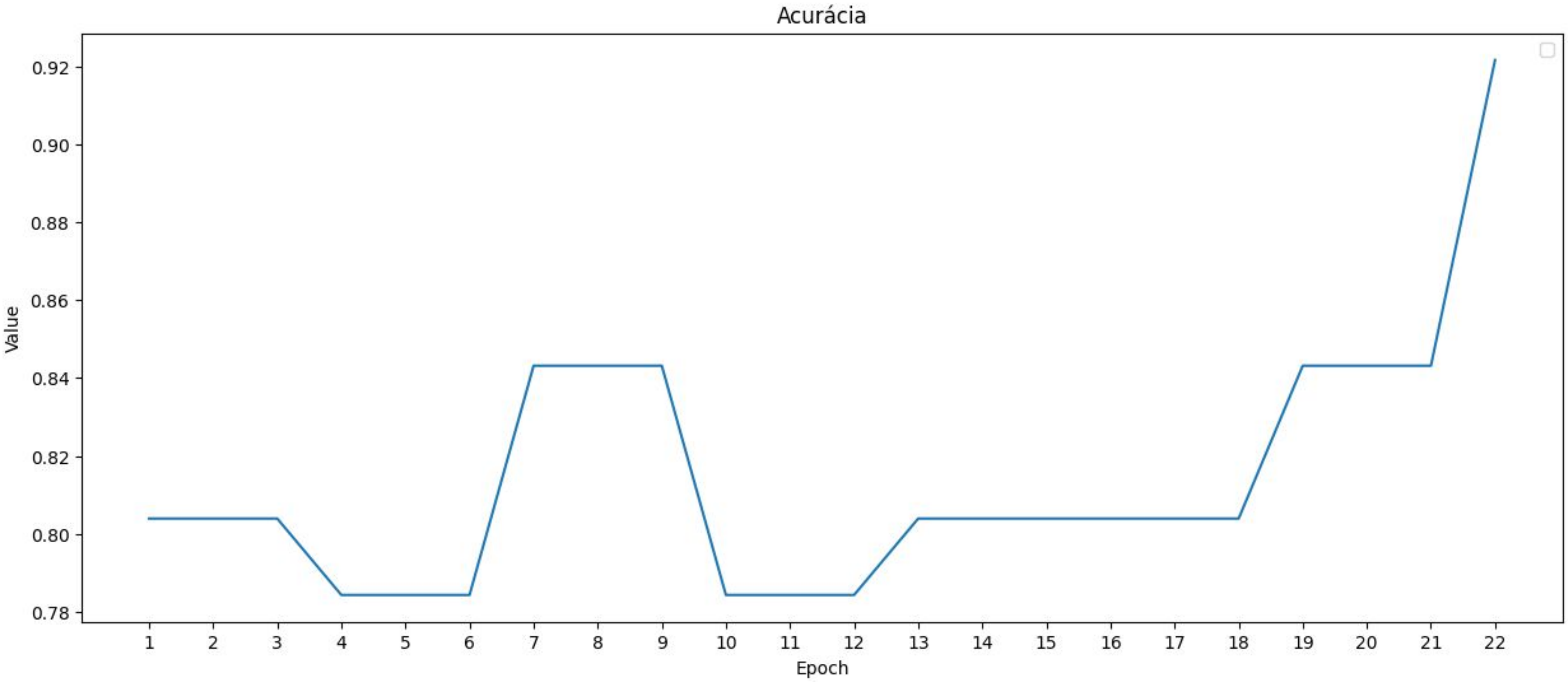
Early stopping



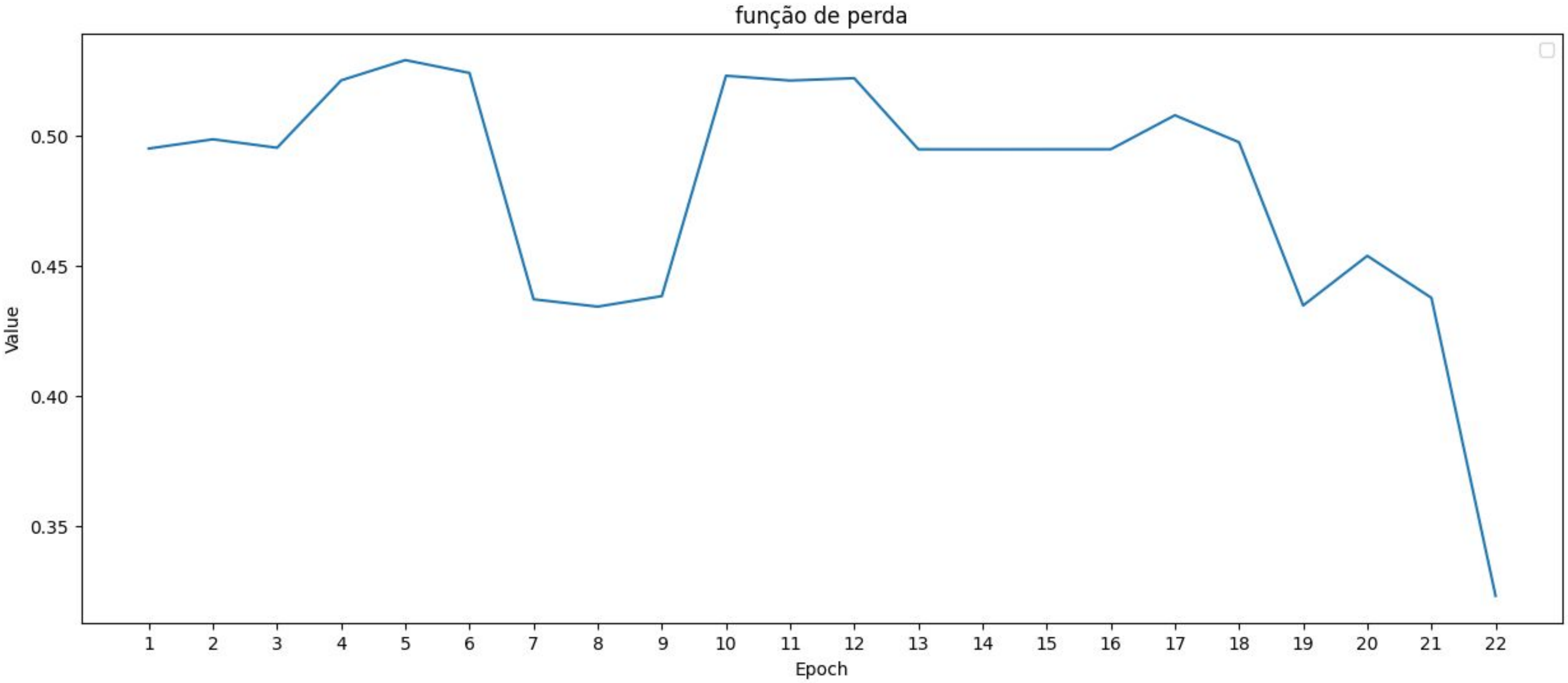


```python
if phase == 'val':
    accuracy_list.append(epoch_acc.item())
    loss_list.append(epoch_loss)
    if(epoch_acc.item() >= 0.9):
        return accuracy_list, loss_list
```

Acurácia


Acurácia
Value
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
Epoch
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

Função de perda


função de perda
Value
Epoch