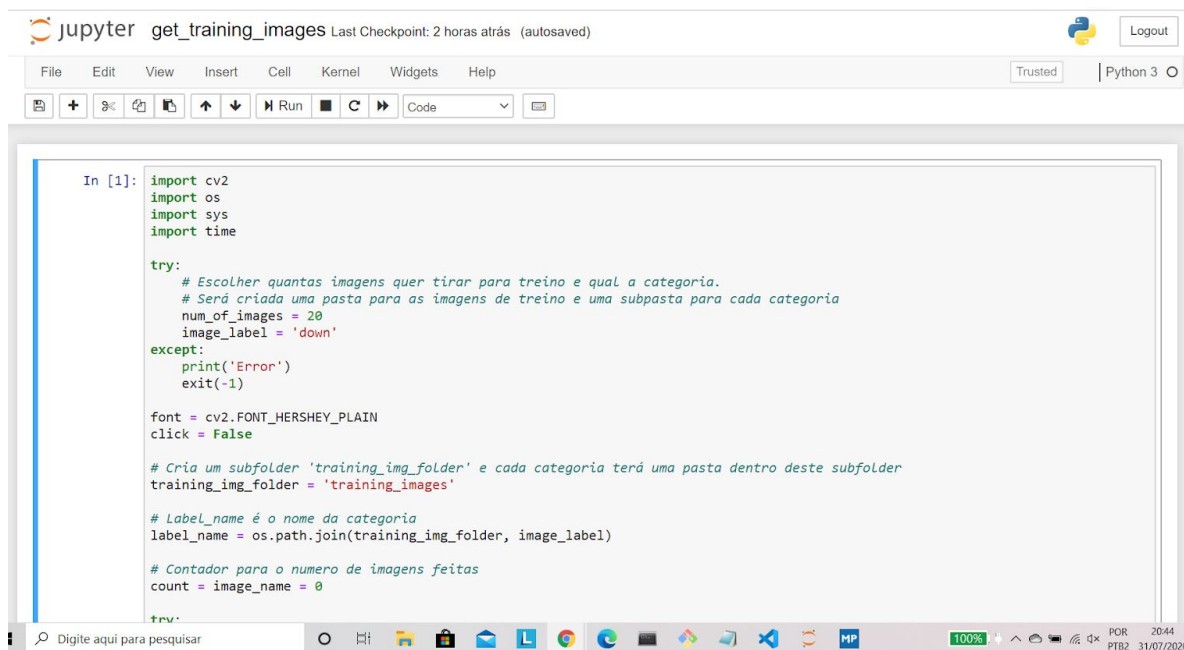


Funcionamento do programa

Existem três programas utilizados para gerar o modelo de testes. Get_training_images.ipynb, train_model.ipynb e test_model.ipynb. O programa que faz a classificação é o 'test_model.ipynb'. Mas vou explicar como funcionam os três programas e os métodos que utilizei para chegar no resultado.

O primeiro programa get_training_images faz a captura de imagens pela webcam. Ele funciona da seguinte maneira:



```
In [1]: import cv2
import os
import sys
import time

try:
    # Escolher quantas imagens quer tirar para treino e qual a categoria.
    # Será criada uma pasta para as imagens de treino e uma subpasta para cada categoria
    num_of_images = 20
    image_label = 'down'
except:
    print('Error')
    exit(-1)

font = cv2.FONT_HERSHEY_PLAIN
click = False

# Cria um subfolder 'training_img_folder' e cada categoria terá uma pasta dentro deste subfolder
training_img_folder = 'training_images'

# Label_name é o nome da categoria
label_name = os.path.join(training_img_folder, image_label)

# Contador para o numero de imagens feitas
count = image_name = 0

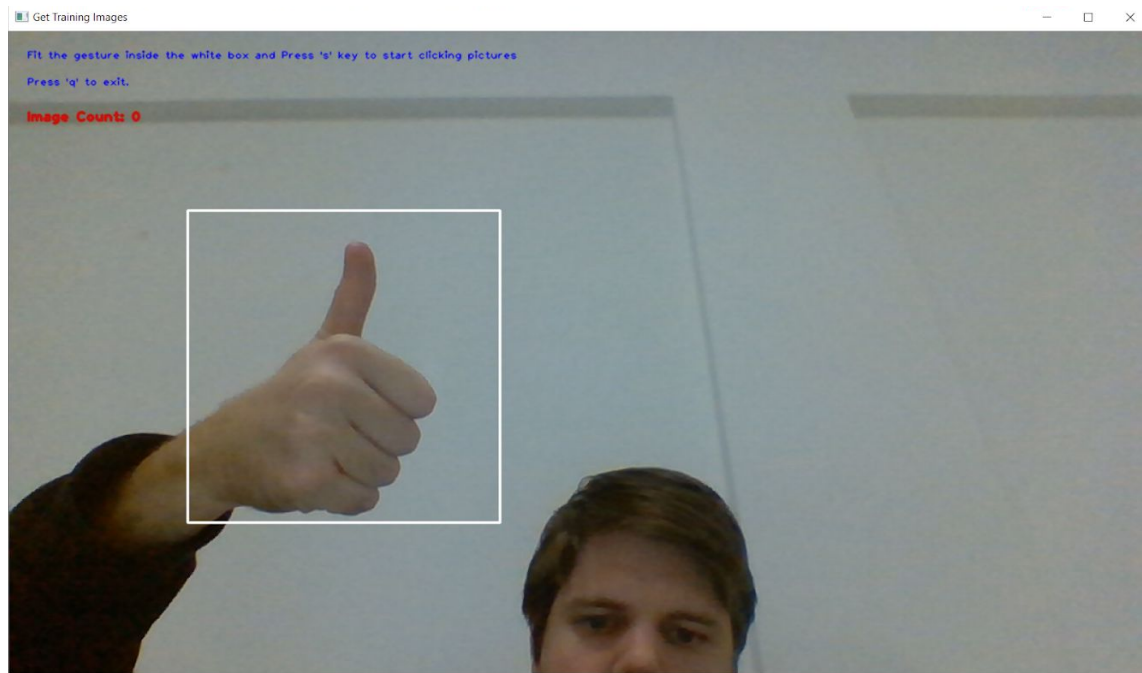
try:
```

Em 'num_of_images' deve-se escolher quantas imagens quer registrar para o treinamento e em 'image_label' deve-se escrever qual a categoria das imagens. 'Down' para sinalização negativa e 'up' para sinalização positiva.

Uma janela irá abrir com o vídeo da webcam. Para começar a salvar as fotos é necessário apertar a tecla 's'. Se desejar sair do programa aperte 'q'.

Quando quiser trocar a classe basta mudar 'image_label' no programa e rodar novamente.

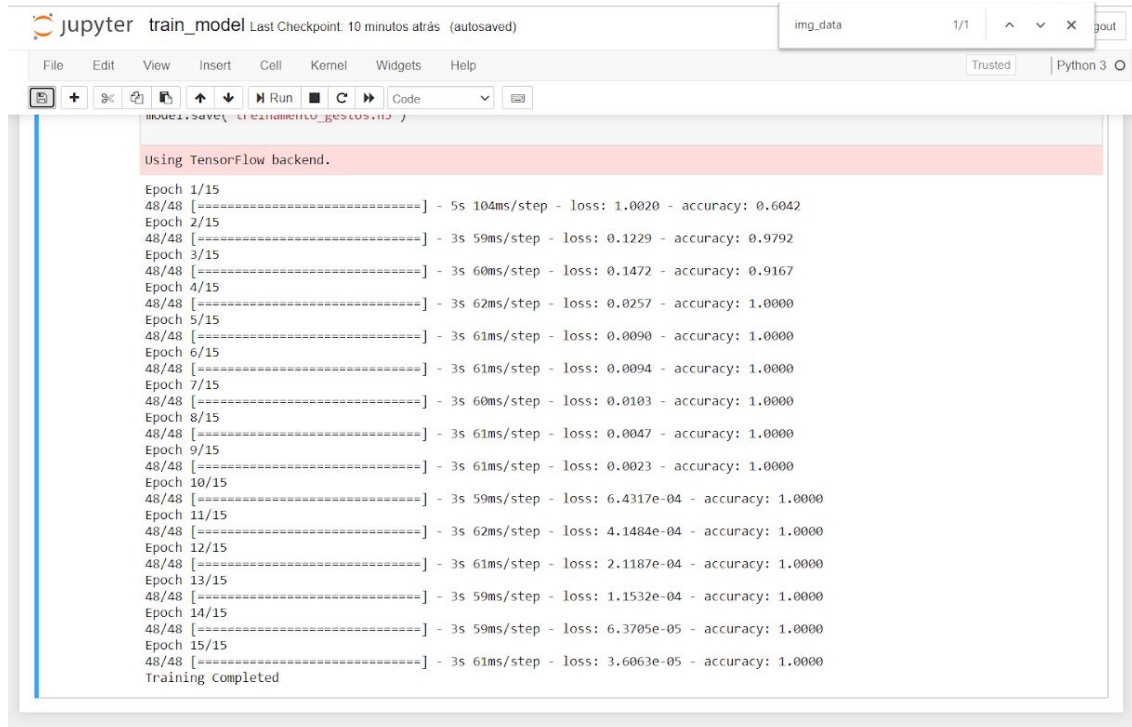
Abaixo está um printscreen da janela de captura de imagens.



O segundo programa 'train_model.ipynb' é o de treinamento. Nele foi utilizado tensorflow com o algoritmo de classificação SqueezeNet. SqueezeNet é um modelo pré treinado para a classificação de imagens. Foi usada a biblioteca Keras para fazer o treino das imagens.

Os principais parâmetros usados no treino foram: 'Sequential' foi usado um modelo sequencial, onde as camadas estão ordenadas em sequência; 'Adam' foi usado como *optimizer*; 'number of epochs' foi igual a 15; 'ReLU' foi usado como função de ativação.

Depois de treinado o modelo de treino é salvo com o nome de 'treinamento_gestos.h5'. Ele será chamado no programa de testes.



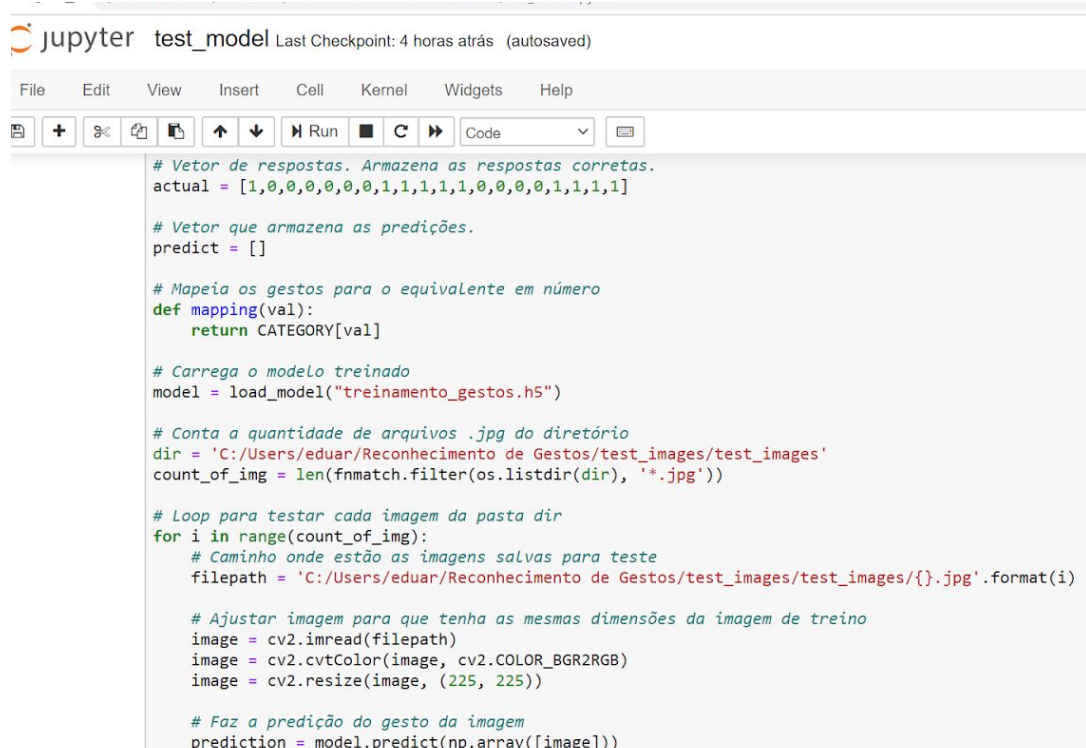
```
model.save('treinamento_gestos.h5')

Using TensorFlow backend.

Epoch 1/15
48/48 [=====] - 5s 104ms/step - loss: 1.0020 - accuracy: 0.6042
Epoch 2/15
48/48 [=====] - 3s 59ms/step - loss: 0.1229 - accuracy: 0.9792
Epoch 3/15
48/48 [=====] - 3s 60ms/step - loss: 0.1472 - accuracy: 0.9167
Epoch 4/15
48/48 [=====] - 3s 62ms/step - loss: 0.0257 - accuracy: 1.0000
Epoch 5/15
48/48 [=====] - 3s 61ms/step - loss: 0.0090 - accuracy: 1.0000
Epoch 6/15
48/48 [=====] - 3s 61ms/step - loss: 0.0094 - accuracy: 1.0000
Epoch 7/15
48/48 [=====] - 3s 60ms/step - loss: 0.0103 - accuracy: 1.0000
Epoch 8/15
48/48 [=====] - 3s 61ms/step - loss: 0.0047 - accuracy: 1.0000
Epoch 9/15
48/48 [=====] - 3s 61ms/step - loss: 0.0023 - accuracy: 1.0000
Epoch 10/15
48/48 [=====] - 3s 59ms/step - loss: 6.4317e-04 - accuracy: 1.0000
Epoch 11/15
48/48 [=====] - 3s 62ms/step - loss: 4.1484e-04 - accuracy: 1.0000
Epoch 12/15
48/48 [=====] - 3s 61ms/step - loss: 2.1187e-04 - accuracy: 1.0000
Epoch 13/15
48/48 [=====] - 3s 59ms/step - loss: 1.1532e-04 - accuracy: 1.0000
Epoch 14/15
48/48 [=====] - 3s 59ms/step - loss: 6.3705e-05 - accuracy: 1.0000
Epoch 15/15
48/48 [=====] - 3s 61ms/step - loss: 3.6063e-05 - accuracy: 1.0000
Training Completed
```

O programa de teste é o 'test_model.ipynb'. Ele é o responsável pela classificação das imagens. Inicialmente é importado o modelo de treino "treinamento_gestos.h5".

Para executar o programa é necessário mudar a variável 'dir' com o caminho para a pasta onde estão salvas as imagens de teste. Também é necessário alterar a variável 'filepath' para esse mesmo caminho, porém deve-se adicionar uma barra / no final para o programa ser capaz de selecionar item por item dentro da pasta.



```
# Vetor de respostas. Armazena as respostas corretas.
actual = [1,0,0,0,0,0,0,1,1,1,1,0,0,0,1,1,1,1]

# Vetor que armazena as predições.
predict = []

# Mapeia os gestos para o equivalente em número
def mapping(val):
    return CATEGORY[val]

# Carrega o modelo treinado
model = load_model("treinamento_gestos.h5")

# Conta a quantidade de arquivos .jpg do diretório
dir = 'C:/Users/eduar/Reconhecimento de Gestos/test_images/test_images'
count_of_img = len(fnmatch.filter(os.listdir(dir), '*.jpg'))

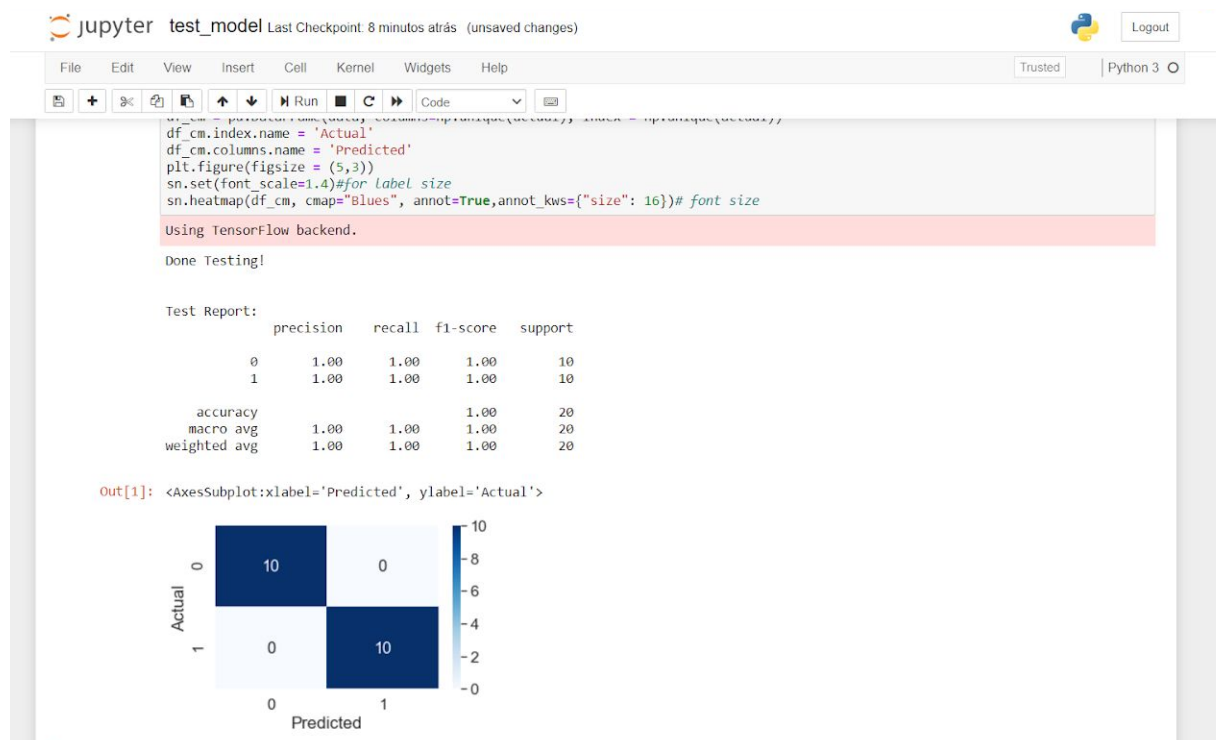
# Loop para testar cada imagem da pasta dir
for i in range(count_of_img):
    # Caminho onde estão as imagens salvas para teste
    filepath = 'C:/Users/eduar/Reconhecimento de Gestos/test_images/test_images/{}.jpg'.format(i)

    # Ajustar imagem para que tenha as mesmas dimensões da imagem de treino
    image = cv2.imread(filepath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (225, 225))

    # Faz a predição do gesto da imagem
    prediction = model.predict(np.array([image]))
```

Para o teste foi usado o mesmo programa que capturou as imagens do treino para capturar as imagens do teste, mudando apenas o diretório onde as imagens foram salvas. Foi criado um vetor 'answer' contendo as respostas das imagens a serem testadas para futura avaliação do método.

O programa carrega as imagens salvas na pasta 'test_images' que estão em formato '.jpg' e garante que elas estão no mesmo tamanho que as imagens que foram treinadas. Então, o modelo faz a predição das imagens e salva elas em um vetor 'predict'. O vetor 'predict' é então comparado com o vetor 'answer' e são feitos o Classification Report e a Matriz Confusão, como mostra a imagem abaixo.



Como resultado foi obtido 100% de precisão. Acredito que esse valor aconteceu pois as imagens de teste foram tiradas em um ambiente com pouco ruído e sem objetos para atrapalhar a visão.