TRABALHO FINAL - IEST01

Caike De Souza Piza – 2016005404

Marcelo Tucci Maia - 2020032328

Projeto de Detecção de Mascara

ITAJUBA 2021

INTRODUÇÃO:

Com a pandemia de COVID-19, muitas restrições e práticas sanitárias tem sido empregadas pelo mundo todo, sendo uma das é o uso correto de máscaras faciais. Desse fato veio a inspiração para o desenvolvimento de um dispositivo embarcado em um microcontrolador (MCU) que possa identificar, através de *machine learning* se uma pessoa está usando máscara e, se sim, se o uso está sendo feito de modo correto.

Entretanto, um desafio deste projeto é a limitação de armazenamento e de processamento que os MCU apresentam. Ou seja, o modelo de *machine learning* não deve ocupar muito espaço de armazenamento, e nem necessitar de um processamento pesado para cumprir seu objetivo. Para isso, foram utilizadas técnicas de *TinyML* com o uso das ferramentas de *Tensorflow* e *Tensorflow lite.* Também foi empregada a ferramenta *Edge Impulse* que faz todo o processo de criação da estrutura do modelo, treinamento e *deployment* de forma semiautomática.

O objetivo inicial deste projeto era o desenvolvimento de um detector de utilização máscara, sendo ela incorreta ou não, e sua não utilização. Devido à dificuldade do modelo em reconhecer a utilização incorreta da máscara, foi decidido pelo grupo o desenvolvimento do modelo simplificado, onde possui a detecção de utilização de máscara ou não.

O projeto é um classificador de imagem, onde foi desenvolvido o modelo utilizando se da ferramenta *Edge Impulse*, para o desenvolvimento da arquitetura do modelo; da câmera do celular para captura e *live classification;* e da câmera do Arduino Nano 33 BLE Sense para o *deployment* e *inferência* do modelo. As classificações treinadas pelo modelo foram SEM_MSC e COM_MSC.

Link do Projeto no Edge Impulse: studio.edgeimpulse.com/public/44005/latest

DATA ACQUISITION

Os dados utilizados para treinamento e testes foram obtidos pelo *face-mask-lite-dataset* que está disponível no site Kaggle(https://www.kaggle.com/prasoonkottarathil/face-mask-lite-dataset), no qual foram utilizadas 1372 imagens para treinamento e 331 imagens para testes. Esta proporção de 80% para treinamento e 20% para testes foi selecionada no momento do *upload* das imagens. Lembrando que o *dataset* é formado por imagens artificiais de pessoas e com isso respeitando a privacidade e possíveis direitos autorais das imagens do banco de dados.

A este banco de dados também foram acrescentadas imagens de treino e de teste um dos membros da equipe com este usando máscara e sem máscara em diferentes posições. A intenção disso foi o de generalizar um pouco mais o *data set* visto que as imagens do *face-mask-lite* são de pessoas sem máscara, com o rosto centralizado e olhando diretamente para a câmera. Apesar das imagens adicionais não estarem centralizadas no rosto, o modelo final apresentou bons resultados.

Impulse Design

O passo inicial, depois da escolha e o tratamento dos dados é a criação da estrutura do modelo. Isso foi feito na parte de *Impulse Design* do *Edge Impulse*. No bloco de *Image Data*, que faz redução das imagens para 96x96, sendo esta a resolução de entrada do *MobileNet V1* (modelo de *transfer learning* que será usando para o treinamento deste modelo).

Em seguida o bloco de *Image*, seguido pelo *learning block*. Nesta etapa, como se quer classificar imagens de pessoas, foi utilizado o processo de *transfer learning* que é um método de *machine learning* onde o modelo desenvolvido para uma tarefa é reusado como o ponto de partida para o treinamento de um modelo para uma outra tarefa.

Será mostrado mais adiante que o modelo escolhido para este projeto é o *MobileNet V1.* Este modelo foi desenvolvido pela Google para que modelos de reconhecimento pudessem ser desenvolvidos para celulares, usando como base o banco de dados *ImageNet,* que contém mais de 14 milhões de imagens e mais de 20 mil categorias.

No último bloco o *Output Features* nos fornece as saídas do modelo, sendo elas: *com_msc* e *sem_msc.* Na Figura 1 é apresentado as configurações de entrada no *Edge Impulse.*

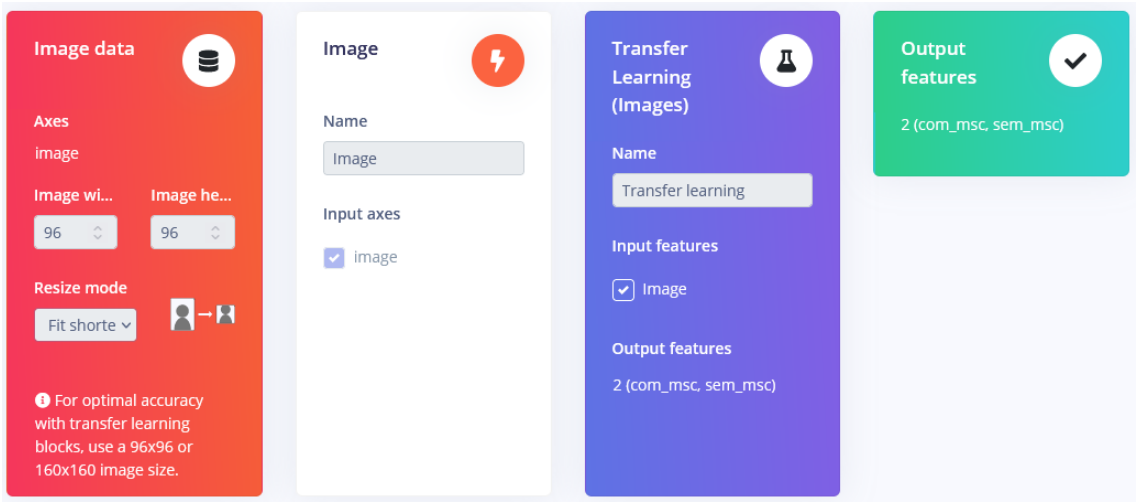

Figura 1: Configuração de *Impulse Design.*

Geração de *features*

A próxima etapa foi a geração de *features.* A Figura 2 mostra a disposição dos *features* de entrada. É possível ver que eles estão bem separados, o que facilitará o treinamento do modelo para sua classificação. Na Figura 3 é mostrado o uso de RAM e o tempo de pré-processamento das imagens.

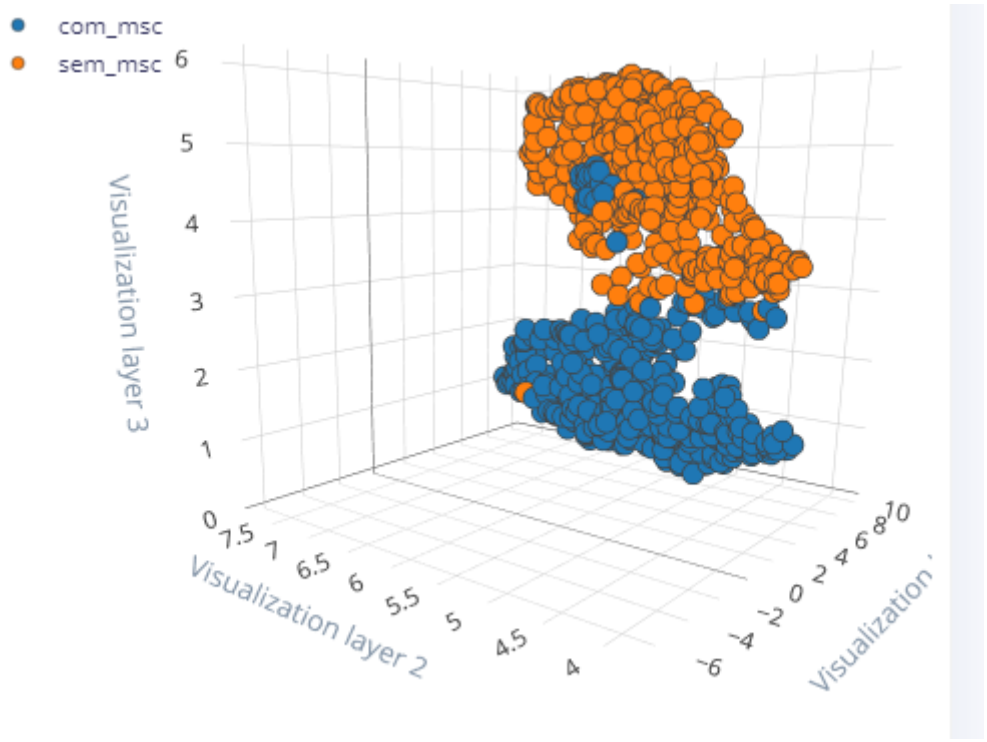

Figura 2: *Feature explorer*.

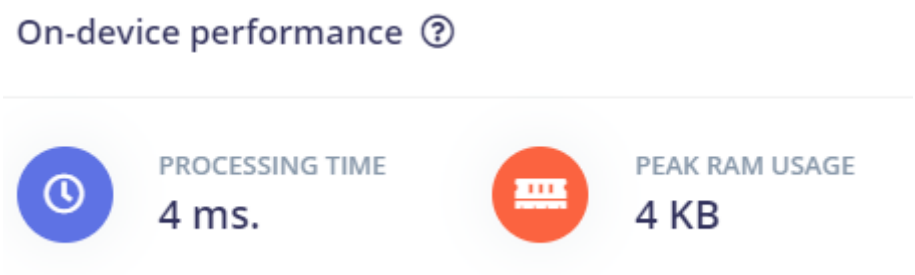

Figura 3: Performance do pré-processamento.

Treinamento do modelo

O treinamento do modelo, como dito anteriormente, foi feito usando o *MobileNet V1* com um alfa de 0.2 que criou um modelo com 202 ms de tempo de inferência; um pico de uso de RAM de 86 kbytes e um uso de memória *flash* de 226,8 kbytes.

O *MobileNet* é um modelo já treinado que possuí várias dezenas de camadas de *neural network* (foi tentado mostrar as camadas usando o Netron, mas são tantas que a imagem ficou indecifrado). Somente na última camada nós entramos com uma camada *Dense* com 16 neurônios com a função de ativação *softmax* (o *notebook* completo é mostrado no Anexo A).

Na Figura 4 é mostrado a configuração neural simplificada e na Figura 5 é apresentada a matrix de confusão dos dados treinados e sua acurácia.

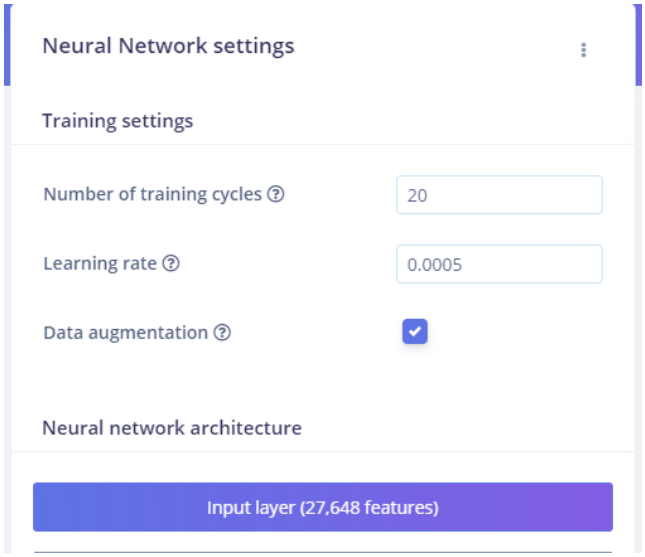

Figura 4: Configuração neural simplificada.

Figura 5: Matriz de confusão para os dados treinados.

Com o modelo treinado observa-se que ele teve uma precisão de 95,3% de modo geral. Para a classificação de pessoa com máscara teve uma precisão de 97,9% e 92,4% de precisão para pessoa sem máscara. E com desempenho de 200ms, 86,0k de RAM e 226,8K de Flash sendo adequado um microcontrolador.

MODEL TESTING

Nesta etapa foi feito o teste do modelo, onde novas aquisição de dados foram necessárias para o teste sendo feito por meio da câmera do celular. Foram feitas algumas novas aquisições para cada categoria do modelo, sendo essas categoriais, *com_msc* e *sem_msc* e também utilização das imagens deixadas para teste do *dataset*. Como é possível observar (Figura 6) no resultado tivemos uma precisão de 98,19% de modo geral.

Para a classificação de pessoa com máscara teve uma precisão de 98,3% e 98,1% de precisão para pessoa sem máscara. Também foram feitos testes usando o *live classification* onde os resultados foram bem coerentes.

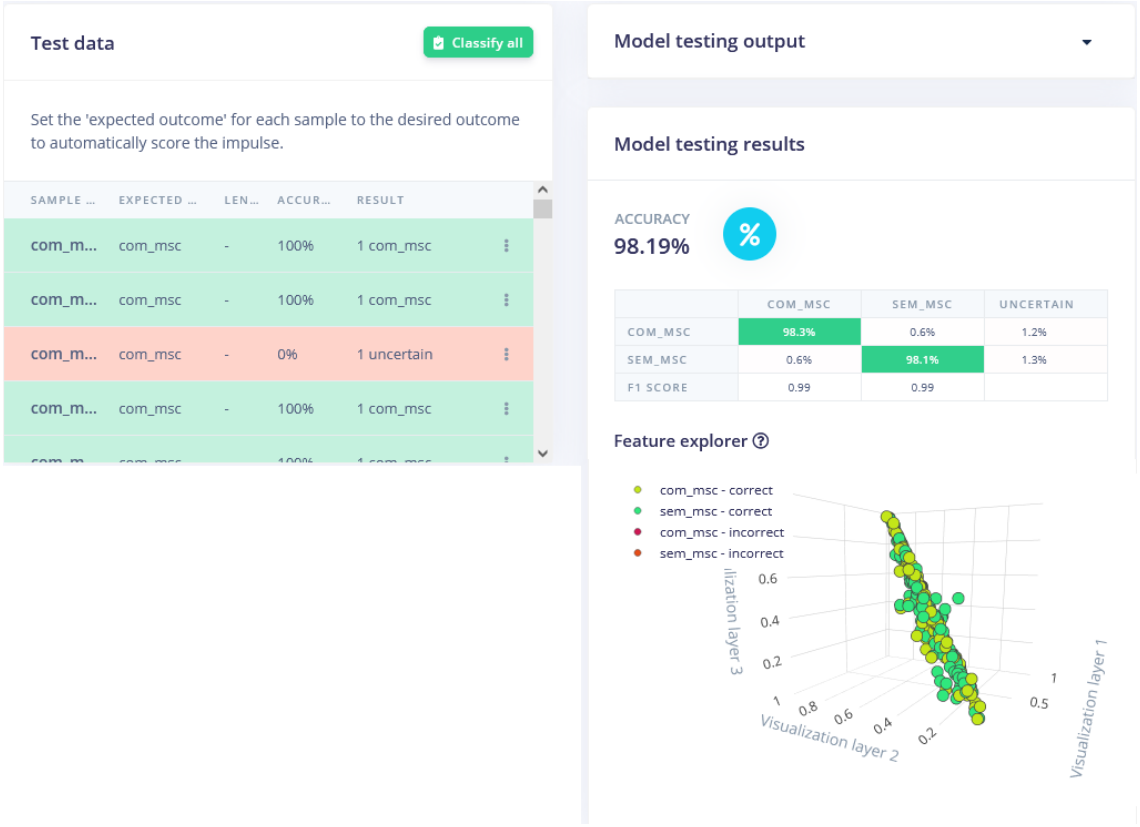

Figura 6: Teste do modelo.

Model Deployment

Para a implementação no Arduíno Nano 33, foi feito um teste inicial para saber se o MCU consegue trabalhar com o modelo criado e se ele consegue processar as imagens de entrada. Para isso, foi usado uma imagem *raw* em formato hexadecimal, como mostrado na Figura 7.

Figura 7: Exemplo de imagem em formato hexadecimal.

Como o modelo mostrou que estava conseguindo processar as imagens de entrada, o próximo passo foi verificar o funcionamento da câmera OV7675. Para isso, a câmera foi inicializada no código e algumas imagens foram obtidas (transformando a imagem em formato hexadecimal). A Figura 8 mostra um exemplo de captura de imagem.

Figura 8: Exemplo de imagem capturado na câmera OV 7675.

Feito isso, o código foi configurado para mostrar no *Serial Monitor* a previsão (*com_msc* ou *sem_msc*) e a probabilidade associada a ela, como mostra a Figura 9. O código também foi modificado para acender o led verde quando detectar o uso de máscara e para acender o led vermelho em caso negativo. O código completo da implementação deste modelo no arduino é mostrado no Anexo B.

```
IESTI01 - Edge Impulse - Image InferencingInferencing settings:
        NN_INPUT_FRAME_SIZE: 27648
        INPUT_WIDTH_COLS: 96
        INPUT_HEIGHT_ROWS: 96
        DSP_INPUT_FRAME_SIZE: 9216
        TFLITE_ARENA_SIZE: 100153
        Interval: 0.00 ms.
        Frame size: 9216
        Sample length: 576 ms.
        No. of classes: 2
Edge Impulse standalone inferencing (Arduino)
run_classifier returned: 0
Predictions (DSP: 15 ms., Classification: 714 ms., Anomaly: 0 ms.):
:
  PREDICTION: ==> sem_msc with probability 0.89
```

Figura 9: Serial monitor.

Finalmente, foram feitos testes e inferências com o modelo embarcado. Um vídeo com o teste pode ser encontrado no seguinte link:

https://youtu.be/VWDlHWKz8yk

CONCLUSÃO

Com o projeto concluído podemos dizer que o projeto inicialmente proposto de classificação de pessoas sem máscara, com máscara e mascara incorreta, foi mudado para pessoas sem máscara e com máscara, uma vez que o modelo possui a dificuldade em reconhecer que a máscara está de modo incorreto pois ele analisa para pessoas com máscara se ele não possui boca e nariz na determinada imagem. Com essa mudança o modelo apresentou alta precisão para imagens focadas no rosto, sendo elas as fotos 3x4. E uma dificuldade em classificar imagens em que a pessoa está em um ambiente ou com muita luminosidade.

Em relação a TinyML, o modelo desenvolvido foi apto a rodar no microcontrolador, sendo utilizado neste projeto a placa de desenvolvimento Arduino Nano 33 BLE Sense. No qual foram feitos testes do modelo, com adaptações dos leds para indicar se detectou pessoa com máscara (led verde) e pessoa sem máscara (led vermelho).

ANEXO A

Download the data - after extracting features through a processing block - so we can train a machine learning model.

```
import numpy as np
import requests

API_KEY = 'ei_23dc81a27a06139420d397e923d432ba5f4aecb10d81f98b3d0
a02af96bb02e6'

def download_data(url):
    response = requests.get(url, headers={'x-api-key': API_KEY})
    if response.status_code == 200:
        return response.content
    else:
        print(response.content)
        raise ConnectionError('Could not download data file')

X = download_data('https://studio.edgeimpulse.com/v1/api/44450/tr
aining/6/x')
Y = download_data('https://studio.edgeimpulse.com/v1/api/44450/tr
aining/6/y')
```

Store the data in a temporary file, and load it back through Numpy.

```
with open('x_train.npy', 'wb') as file:
    file.write(X)
with open('y_train.npy', 'wb') as file:
    file.write(Y)
X = np.load('x_train.npy')
Y = np.load('y_train.npy')[:,0]
```

Define our labels and split the data up in a test and training set:

```
import sys, os, random
import tensorflow as tf
from sklearn.model_selection import train_test_split

import logging
tf.get_logger().setLevel(logging.ERROR)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# Set random seeds for repeatable results
RANDOM_SEED = 3
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)

classes_values = [ "com_msc", "sem_msc" ]
classes = len(classes_values)

Y = tf.keras.utils.to_categorical(Y - 1, classes)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_si
ze=0.2, random_state=1)
```

```
input_length = X_train[0].shape[0]

train_dataset = tf.data.Dataset.from_tensor_slices((X_train, Y_tr
ain))
validation_dataset = tf.data.Dataset.from_tensor_slices((X_test,
Y_test))

def get_reshape_function(reshape_to):
    def reshape(image, label):
        return tf.reshape(image, reshape_to), label
    return reshape

callbacks = []
```

Train the model:

```
import math
from pathlib import Path
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer, Dropout, C
onv1D, Flatten, Reshape, MaxPooling1D, BatchNormalization, Conv2D
, GlobalMaxPooling2D, Lambda
from tensorflow.keras.optimizers import Adam, Adadelta
from tensorflow.keras.losses import categorical_crossentropy


WEIGHTS_PATH = './transfer-learning-
weights/edgeimpulse/MobileNetV1.0_2.96x96.color.bsize_96.lr_0_05.
epoch_170.val_loss_3.61.val_accuracy_0.27.hdf5'
# Download the model weights
root_url = 'http://cdn.edgeimpulse.com/'
p = Path(WEIGHTS_PATH)
if not p.exists():
    if not p.parent.exists():
        p.parent.mkdir(parents=True)
    weights = requests.get(root_url + WEIGHTS_PATH[2:]).content
    with open(WEIGHTS_PATH, 'wb') as f:
        f.write(weights)

INPUT_SHAPE = (96, 96, 3)

base_model = tf.keras.applications.MobileNet(
    input_shape = INPUT_SHAPE,
    weights = WEIGHTS_PATH,
    alpha = 0.2
)

base_model.trainable = False

model = Sequential()
model.add(InputLayer(input_shape=INPUT_SHAPE, name='x_input'))
# Don't include the base model's top layers
last_layer_index = -6
```

```
model.add(Model(inputs=base_model.inputs, outputs=base_model.laye
rs[last_layer_index].output))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.1))

model.add(Dense(classes, activation='softmax'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.
0005),
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# Set the data to the expected input shape
train_dataset = train_dataset.map(get_reshape_function(INPUT_SHAP
E), tf.data.experimental.AUTOTUNE)
validation_dataset = validation_dataset.map(get_reshape_function(
INPUT_SHAPE), tf.data.experimental.AUTOTUNE)

# Implements the data augmentation policy
def augment_image(image, label):
    # Flips the image randomly
    image = tf.image.random_flip_left_right(image)

    # Increase the image size, then randomly crop it down to
    # the original dimensions
    resize_factor = random.uniform(1, 1.2)
    new_height = math.floor(resize_factor * INPUT_SHAPE[0])
    new_width = math.floor(resize_factor * INPUT_SHAPE[1])
    image = tf.image.resize_with_crop_or_pad(image, new_height, n
ew_width)
    image = tf.image.random_crop(image, size=INPUT_SHAPE)

    # Vary the brightness of the image
    image = tf.image.random_brightness(image, max_delta=0.2)

    return image, label

train_dataset = train_dataset.map(augment_image, tf.data.experime
ntal.AUTOTUNE)

BATCH_SIZE = 32
train_dataset = train_dataset.batch(BATCH_SIZE, drop_remainder=Fa
lse)
validation_dataset = validation_dataset.batch(BATCH_SIZE, drop_re
mainder=False)

model.fit(train_dataset, validation_data=validation_dataset, epoc
hs=20, verbose=2, callbacks=callbacks)

print('')
print('Initial training done.', flush=True)

# How many epochs we will fine tune the model
FINE_TUNE_EPOCHS = 10
# What percentage of the base model's layers we will fine tune
FINE_TUNE_PERCENTAGE = 65
```

```
print('Fine-
tuning model for {} epochs...'.format(FINE_TUNE_EPOCHS), flush=Tr
ue)

# Determine which layer to begin fine tuning at
model_layer_count = len(model.layers)
fine_tune_from = math.ceil(model_layer_count * ((100 - FINE_TUNE_
PERCENTAGE) / 100))

# Allow the entire base model to be trained
model.trainable = True
# Freeze all the layers before the 'fine_tune_from' layer
for layer in model.layers[:fine_tune_from]:
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.
000045),
                loss='categorical_crossentropy',
                metrics=['accuracy'])

model.fit(train_dataset,
                epochs=FINE_TUNE_EPOCHS,
                verbose=2,
                validation_data=validation_dataset,
                callbacks=callbacks)
```

Salvando o modelo:

```
# Save the model to disk
model.save('saved_model')
```

ANEXO B

Projeto_mascara.ino

```
/* Includes ---------------------------------------------------------------- */
#include <projeto_mascara2_inferencing.h>
#include <TinyMLShield.h> // Includes the Arduino_OV767X.h library


/* Image Definitions  */

// raw frame buffer from the camera (QCIF grayscale from camera = 176 * 144 * 1)
#define FRAME_BUFFER_COLS          176
#define FRAME_BUFFER_ROWS          144
uint16_t frame_buffer[FRAME_BUFFER_COLS * FRAME_BUFFER_ROWS] = { 0 };

// Resize image cutting out the edges (it is not a true resize)
#define CUTOUT_COLS                EI_CLASSIFIER_INPUT_WIDTH
#define CUTOUT_ROWS                EI_CLASSIFIER_INPUT_HEIGHT
const int cutout_row_start = (FRAME_BUFFER_ROWS - CUTOUT_ROWS) / 2;
const int cutout_col_start = (FRAME_BUFFER_COLS - CUTOUT_COLS) / 2;

/*
 *  Functions to help convert color data
 */

// helper methods to convert from rgb -> 565 and vice versa this one not used
uint16_t rgb_to_565(uint8_t r, uint8_t g, uint8_t b) {
   return ((r >> 3) << 11) | ((g >> 2) << 5) | (b >> 3);
}

// funciton converts from RGB565b to RGB888 and is used.
void r565_to_rgb(uint16_t color, uint8_t *r, uint8_t *g, uint8_t *b) {
   *r = (color & 0xF800) >> 8;
   *g = (color & 0x07E0) >> 3;
   *b = (color & 0x1F) << 3;
}

/**
 * This function is called by the classifier to get data
 * We don't want to have a separate copy of the cutout here, so we'll read from the frame
buffer dynamically
 */

int cutout_get_data(size_t offset, size_t length, float *out_ptr) {
   // offset and length naturally operate on the *cutout*,
   // so we need to cut it out from the real framebuffer
   size_t bytes_left = length;
   size_t out_ptr_ix = 0;

   // read byte for byte
   while (bytes_left != 0) {
      // find location of the byte in the cutout
      size_t cutout_row = floor(offset / CUTOUT_COLS);
      size_t cutout_col = offset - (cutout_row * CUTOUT_COLS);
```

```
      // then read the value from the real frame buffer
      size_t frame_buffer_row = cutout_row + cutout_row_start;
      size_t frame_buffer_col = cutout_col + cutout_col_start;

      // grab the value and convert to r/g/b
      uint16_t pixelTemp = frame_buffer[(frame_buffer_row * FRAME_BUFFER_COLS) +
frame_buffer_col];

      // This line needed to switch big and little endians
      uint16_t pixel = (pixelTemp>>8) | (pixelTemp<<8);

      uint8_t r, g, b;
      r565_to_rgb(pixel, &r, &g, &b);

      // then convert to out_ptr format
      float pixel_f = (r << 16) + (g << 8) + b;
      //float pixel_f = (r << 16) | (g << 8) | b;
      out_ptr[out_ptr_ix] = pixel_f;

      // and go to the next pixel
      out_ptr_ix++;
      offset++;
      bytes_left--;
    }
    return 0;
}

/**
 * @brief      Arduino setup function
 */

void setup()
{
    Serial.begin(115200);
    while (!Serial);

    //Pins para o LED RGB
    pinMode(LEDR, OUTPUT);
    pinMode(LEDG, OUTPUT);
    pinMode(LEDB, OUTPUT);

    //Pin para o led embutido
    pinMode(LED_BUILTIN, OUTPUT);

    //Para ter certeza que os LEDS estão desligados
    //Desligado: HIGH
    //Ligado: LOW

    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, HIGH);

    digitalWrite(LED_BUILTIN, LOW);

    ei_printf("IESTI01 - Edge Impulse - Image Inferencing");
    // summary of inferencing settings (from model_metadata.h)
```

```
  ei_printf("Inferencing settings:\n");
  ei_printf("\tNN_INPUT_FRAME_SIZE: %d\n", EI_CLASSIFIER_NN_INPUT_FRAME_SIZE);
  ei_printf("\tINPUT_WIDTH_COLS: %d\n", EI_CLASSIFIER_INPUT_WIDTH);
  ei_printf("\tINPUT_HEIGHT_ROWS: %d\n", EI_CLASSIFIER_INPUT_HEIGHT);
  ei_printf("\tDSP_INPUT_FRAME_SIZE: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
  ei_printf("\tTFLITE_ARENA_SIZE: %d\n", EI_CLASSIFIER_TFLITE_ARENA_SIZE);
  ei_printf("\tInterval: %.2f ms.\n", (float)EI_CLASSIFIER_INTERVAL_MS);
  ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
  ei_printf("\tSample length: %d ms.\n", EI_CLASSIFIER_RAW_SAMPLE_COUNT / 16);
  ei_printf("\tNo. of classes: %d\n", sizeof(ei_classifier_inferencing_categories) /
                                    sizeof(ei_classifier_inferencing_categories[0]));

  // Initialize TinyML Kit
  initializeShield();
/*
  VGA – 640 x 480
  CIF – 352 x 240
  QVGA – 320 x 240
  QCIF – 176 x 144
*/
  // Initialize the OV7675 camera
  if (!Camera.begin(QCIF, RGB565, 1, OV7675)) {
    Serial.println("Failed to initialize camera");
    while (1);
  }
}

/**
 * @brief     Arduino main function
 */

/**
* @brief     Turn off leds
*
* @param[in]  format    Variable argument list
*/

void turn_off_leds(void){
 digitalWrite(LEDR, HIGH);
 digitalWrite(LEDG, HIGH);
 digitalWrite(LEDB, HIGH);
}

 void turn_on_leds(int pred_index){
 switch (pred_index)
 {
  case 0:
  turn_off_leds();
  digitalWrite(LEDG, LOW);
  break;

  case 1:
  turn_off_leds();
  digitalWrite(LEDR, LOW);
  break;
  }
```

```
 }

void loop()
{

   ei_printf("Edge Impulse standalone inferencing (Arduino)\n");

   ei_impulse_result_t result = { 0 };

   // Get image from Camera
   Camera.readFrame((uint8_t*)frame_buffer);

   // Set up pointer to look after data, crop it and convert it to RGB888
   signal_t signal;
   signal.total_length = CUTOUT_COLS * CUTOUT_ROWS;
   signal.get_data = &cutout_get_data;

   // invoke the impulse
   EI_IMPULSE_ERROR res = run_classifier(&signal, &result, false /* debug */);
   ei_printf("run_classifier returned: %d\n", res);

   if (res != 0) return;

   // print the predictions
   ei_printf("Predictions ");
   ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
      result.timing.dsp, result.timing.classification, result.timing.anomaly);
   ei_printf(": \n");
   //ei_printf("[");

   int pred_index = 0;     // Initialize pred_index
   float pred_value = 0;   // Initialize pred_value

   for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
      //ei_printf("%.5f", result.classification[ix].value);
      if (result.classification[ix].value > pred_value){
         pred_index = ix;
         pred_value = result.classification[ix].value;
      }
   }

   ei_printf(": \n");
   ei_printf("  PREDICTION: ==> %s with probability %.2f\n",
   result.classification[pred_index].label, pred_value);
   ei_printf(": \n");
   turn_on_leds (pred_index);



#if EI_CLASSIFIER_HAS_ANOMALY == 1
      ei_printf(", ");
//#else
      //if (ix != EI_CLASSIFIER_LABEL_COUNT - 1) {
         //ei_printf(", ");
      //}
```

```
#endif
    }
//#if EI_CLASSIFIER_HAS_ANOMALY == 1
//    ei_printf("%.3f", result.anomaly);
//#endif
    //ei_printf("]\n");

    // human-readable predictions
    //for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
    //    ei_printf("    %s: %.5f\n", result.classification[ix].label,
result.classification[ix].value);
    //}
#if EI_CLASSIFIER_HAS_ANOMALY == 1
    ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif

 /*
 * The code portion bolow is used to show what image the camera is seem.
 * Use the pgm : IEST01_OV7675_Image_Viewer
 * (for test only)

     Serial.println();
     for (size_t ix = 0; ix < signal.total_length; ix++) {
        float value[1];
        signal.get_data(ix, 1, value);
        ei_printf("0x%06x", (int)value[0]);
        if (ix != signal.total_length - 1) {
           ei_printf(", ");
        }
     }

    Serial.println();
    delay(1000);
}
 */
/**
 * @brief      Printf function uses vsnprintf and output using Arduino Serial
 *
 * @param[in]  format     Variable argument list
 */
void ei_printf(const char *format, ...) {
    static char print_buf[1024] = { 0 };

    va_list args;
    va_start(args, format);
    int r = vsnprintf(print_buf, sizeof(print_buf), format, args);
    va_end(args);

    if (r > 0) {
        Serial.write(print_buf);
    }
}
```