DLBAIPEAI01

PROJECT: EDGE AI

TASK 2: AGE, GENDER, AND EXPRESSION RECOGNITION APPLICATION

STUDENT: HUGO ALBUQUERQUE COSME DA SILVA

ID:92126125

iu INTERNATIONALE
HOCHSCHULE

TABLE OF CONTENTS

1. INTRODUCTION

The goal of this project is to develop an application for Android mobile phones that can be used to detect the age, gender, and facial expressions of people based on photos from the gallery or pictures taken by the user through the phone's camera. The application will use Tensorflow lite models to perform the inference. I will use pre-trained models for each task and then convert them to tensorflow lite models. The Java code made available by the Tensorflow team for Image Classification will be used as a template for my Android application. As a final step, I will perform a demo of the project running on my smartphone and evaluate its performance. The first bibliographic reference in section 4 contains a link to a compressed file in my Google Drive with all the project's code, datasets, and models used throughout the project.

2. METHODOLOGY

2.1 TRANSFER LEARNING AND INITIAL PROJECT STRUCTURE

The system development started with the choice of models and application architecture used to develop the application. I have decided to use TensorFlow not only because of my previous professional experience but also because it offers a somewhat intuitive way of converting models to formats that are compatible with the deployment on mobile applications. Besides that, the TensorFlow team has shared a series of tutorials with some templates for creating your own ML application on the Age [here](#).

Since the focus of the project was not on spending most of the time training a model from scratch, and I have performed the development in a restricted environment in terms of computational resources (no GPU, limited CPU capacity, limited RAM), I have decided to apply transfer learning and download the weights from pre-trained models from open-source packages that have shown to perform well for each of the tasks involved in the project.

The model weights were then downloaded from the [deepface github page](#), which is a popular lightweight face recognition library for Python. For each model, there is a jupyter notebook where I download the model weights and test the performance of the models.

Fig.1 - Model setup notebooks in the project's folder structure

```python
def loadModel(
    url="https://github.com/serengil/deepface_models/releases/download/v1.0/age_model_weights.h5",
) -> Model:

    model = VGGFace.baseModel()

    # -------------------------

    classes = 101
    base_model_output = Sequential()
    base_model_output = Convolution2D(classes, (1, 1), name="predictions")(model.layers[-4].output)
    base_model_output = Flatten()(base_model_output)
    base_model_output = Activation("softmax")(base_model_output)

    # -------------------------

    age_model = Model(inputs=model.input, outputs=base_model_output)

    # -------------------------

    # load weights
    output = "/home/hualcosa/Documents/iu_project_edge_ai/project_files/artifacts/model_weights/age_model_weights.h5"
    if os.path.isfile(output) != True:
        gdown.download(url, output, quiet=False)

    age_model.load_weights(output)

    return age_model
```

Fig. 2 - load model utility function for the age model

4

```python
# testing the model on 20 random images
path = "/home/hualcosa/Documents/iu_project_edge_ai/project_files/UTKFace"

for _ in range(20):
    img = random.choice(os.listdir(path))
    # following UTK dataset filename convetion explained in the dataset website
    gender = 'Woman' if int(img.split("_")[1]) else 'Man'
    img = cv2.imread(os.path.join(path, img))

    img_copy = img.copy()
    # resizing image
    img = cv2.resize(img, (224, 224))
    # scaling image
    img = img / 255
    # adding batch dimension
    img = np.expand_dims(img, axis=0)
    # performing inference
    pred_gender = labels[np.argmax(model.predict(img, verbose=0))]

    # taking a glimpse at predicted age vs true age
    plt.imshow(img_copy[:,:,0], cmap='gray')
    plt.title(f"True gender: {gender}, predicted gender: {pred_gender}")
    plt.show()
```


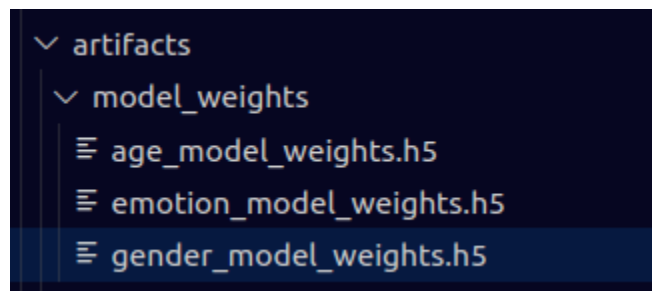Fig.3 - Testing the performance of the gender model

Fig. 4 - Folder where the downloaded model weights are saved

As requested by the project's task, I have downloaded 20 images from the web, 50-50 % adult-elderly people, and also for each group, 50-50% male-female and 50-50% happy-sad. These images were used to evaluate the performance of both the original models and the tf-lite models deployed to my Android phone.
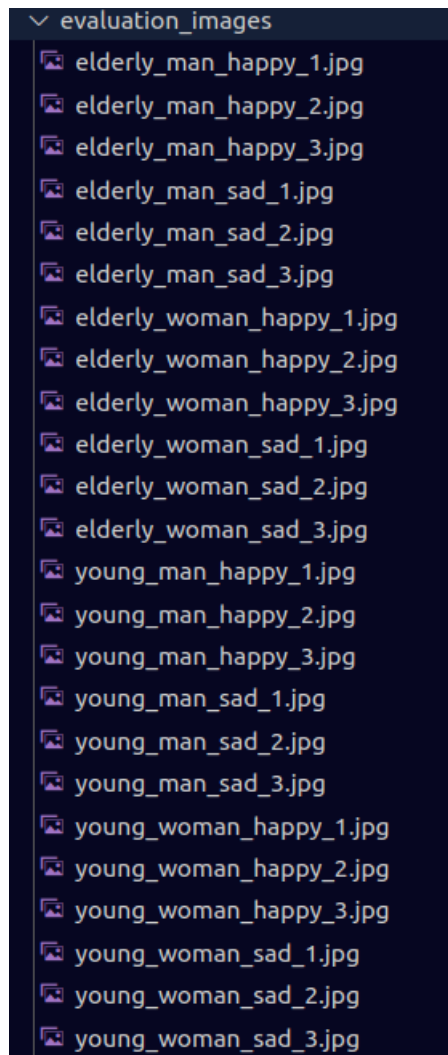
Fig. 5 - Final evaluation images in the project repository folder structure.

After being loaded and tested on image datasets available on Kaggle, each model was also evaluated on the 20 final evaluation images. This evaluation gives us some idea of what is to be expected of the original model's performance.

```python
img_paths = '/home/hualcosa/Documents/iu_project_edge_ai/evaluation_images'
img_paths = [os.path.join(img_paths, i) for i in os.listdir(img_paths)]
```
[7] ✓ 0.0s   Python

```python
# This list is going to store the comparison between true class vs predicted class.
evals = []
for path in img_paths:
    # retrieving the original names from the file names:
    age_label, gender_label, emotion_label, _ = path.split('/')[-1].split("_")
    # read image
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_orig = img.copy()
    # the model expects data in float32 format
    img = img.astype('float32')
    img_gender_age = cv2.resize(img, (224,224))
    img_gender_age = img_gender_age.reshape(1, 224, 224, 3) / 255 # normalizing the img

    predicted_label = labels[model.predict(img_gender_age, verbose=0).argmax()]

    evals.append(int(gender_label == predicted_label.lower()))
```
[9] ✓ 5.0s   Python

```python
print(f"The original model accuracy is: {100 * sum(evals)/ len(evals)} %")
```
[10] ✓ 0.0s   Python
··· The original model accuracy is: 87.5 %

Fig. 6 - evaluating the performance of the original gender model.

To summarize, there are three models, one for each task: Facial expression classification, gender classification, and age classification. The weights of each model were loaded from the web, and the accuracy on the final 20 images was computed to generate a benchmark. These same images are also used to evaluate the performance of the deployed models in the Android application, as I will show in the next sections.

2.2 - MODEL CONVERSION

After downloading and benchmarking the models, it is necessary to convert the models to a format that can be deployed to android. Using the TensorFlow API, it is fairly easy to do so. First, it is necessary to save the model in the savedModel format.

```
tf.saved_model.save(model, export_dir="/home/hualcosa/Documents/iu_project_edge_ai/project_files/artifacts/models/emotion")
```

Fig. 6 - Saving emotion model in savedModel format.

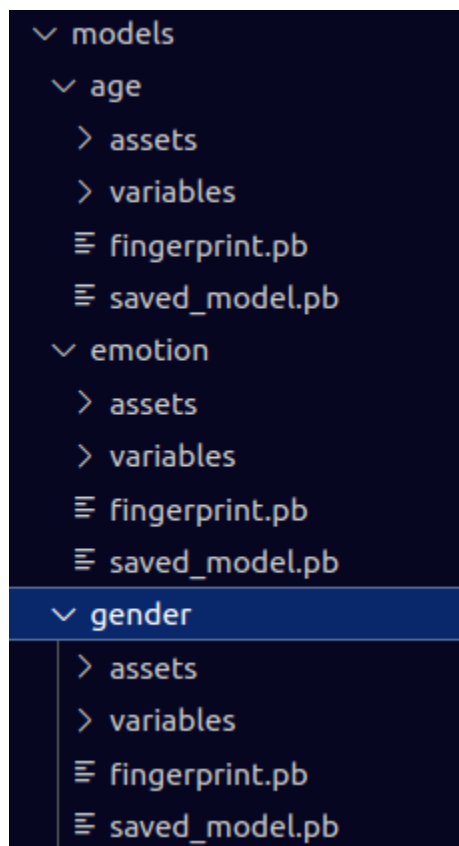Hence each model was saved in this format in the artifacts/models folder:

Fig.7 - Saved models folder structure.

The **tf-lite_convertion.ipynb** notebook contains the code that converts the models to the tf-lite format. These models are also saved in the artifacts/models folder.

```python
import tensorflow as tf


age_path = '/home/hualcosa/Documents/iu_project_edge_ai/project_files/artifacts/models/age'
emotion_path = '/home/hualcosa/Documents/iu_project_edge_ai/project_files/artifacts/models/emotion'
gender_path = '/home/hualcosa/Documents/iu_project_edge_ai/project_files/artifacts/models/gender'

for model_name, saved_model_dir in zip(["age", "emotion", "gender"],[age_path, emotion_path, gender_path]):
    # Convert the model
    converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir) # path to the SavedModel directory
    # age and gender models needs to go through a post-training quantization process, because the original models are too big to be in
    # see more details about post training quantization here: https://www.tensorflow.org/lite/performance/post_training_quant#convert
    if model_name in {"age", "gender"}:
        converter.optimizations = [tf.lite.Optimize.DEFAULT]
        # converter.target_spec.supported_types = [tf.float16]
    tflite_model = converter.convert()

    # Save the model.
    with open(f'/home/hualcosa/Documents/iu_project_edge_ai/project_files/artifacts/models/{model_name}_model.tflite', 'wb') as f:
        f.write(tflite_model)
```
Python

Fig. 8 - Converting models to tf-lite format.

```
~/Documents/iu_project_edge_ai/project_files/artifacts/models
❭ ls -lh
total 263M
drwxrwxr-x 4 hualcosa hualcosa 4,0K dez 26 16:56 age
-rw-rw-r-- 1 hualcosa hualcosa 129M dez 28 22:08 age_model.tflite
drwxrwxr-x 4 hualcosa hualcosa 4,0K dez 26 16:58 emotion
-rw-rw-r-- 1 hualcosa hualcosa 5,7M dez 28 22:08 emotion_model.tflite
drwxrwxr-x 4 hualcosa hualcosa 4,0K dez 26 16:55 gender
-rw-rw-r-- 1 hualcosa hualcosa 129M dez 28 22:09 gender_model.tflite
```

Fig.9 Model conversion result

2.3 - POST TRAINING QUANTIZATION

As can be seen in the previous screenshots, the gender and age models have 129 MB each. The original models nonetheless had more than 500MB. These models use the VGGFace network as their backbone. That is why their size is significantly bigger than the emotion model. Android Studio, the IDE used in this project to develop the Android app, allows the upload of model files that are at most 200MB in size. Because of that, as can be seen in Figure 8, age and gender models needed to pass through a post-training quantization process. This is done by setting the **optimizations** property of the TFLiteConverter object.

2.4 - ANDROID APP DEVELOPMENT

The Android application was developed using the image classification tf-lite example as a template. All the app-related files are in the **android_app_files** folder. The most important piece of code resides in the MainActivity class. This class contains the functionality that will prompt the user to choose whether to take a picture using the smartphone's camera or to select a picture from the Gallery. This is implemented in the onCreate and onActivity methods. The classifyImage method is the one responsible for loading the models, reshaping the input image to the proper input size, and performing the classification. Notice that the image and gender models expect images of size (224, 224, 3) while the emotion model expects images of size (48, 48, 1). That's why I created two separate buffers for the two different models.

It is also important to note that this part of the development was done using Android Studio. In order to use the tf-lite models from the previous stages in the app, it is necessary to import them to Android Studio so they are put in the adequate folder and the application can make use of them. After you import the models, Android Studio automatically creates a ml folder for you. this allows you to import the model as a package in the mainActivity class.
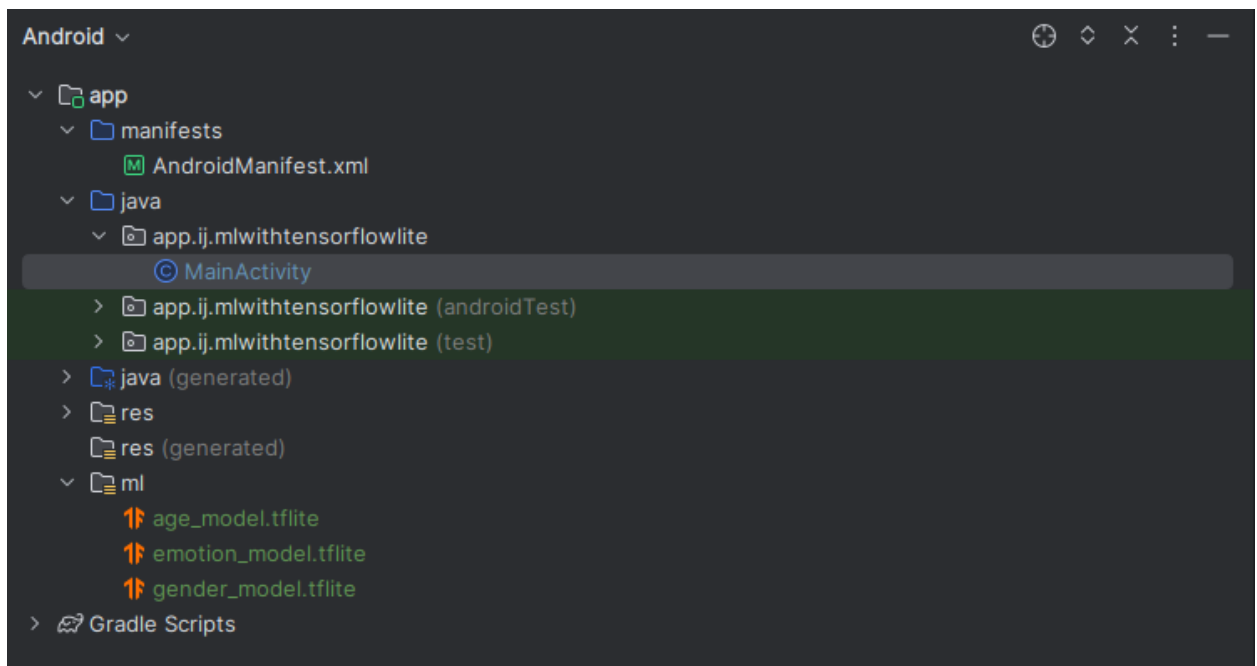

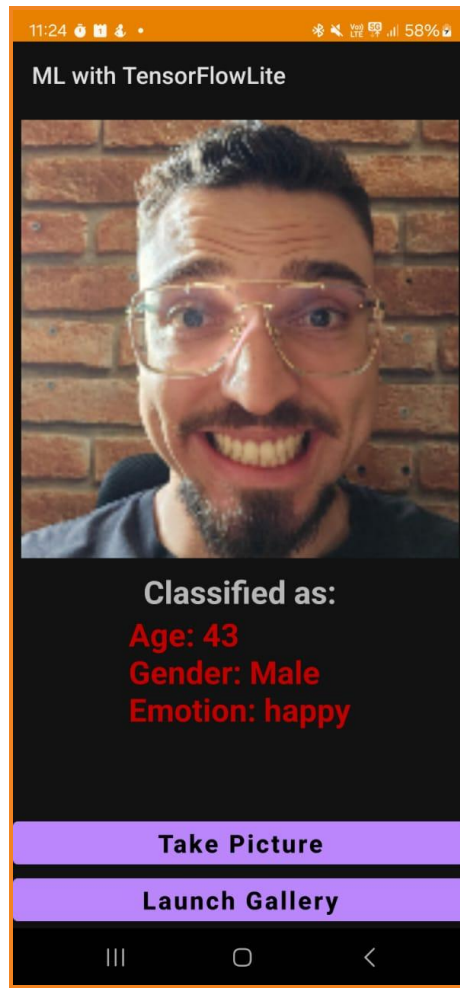Fig.10 - Android app project structure.

Fig.12 - App User Interface

2.5 - ANDROID APP SETUP

The demonstration of the final application running on my Android phone can be found here. If you wish to run the app on your mobile phone as well, after you download the project files to your local machine, you should apply the following steps:

1. Install Android Studio on your local machine
2. Inside Android Studio, download the SDK for the Android version of your smartphone
3. On your smartphone, enable the Developer Options and then enable USB Debugging.
4. Open the **android_app_files** folder on Android Studio.

If the previous steps were performed correctly, Android Studio will be able to recognize your phone, and a green run button will appear in the top right corner of the UI:
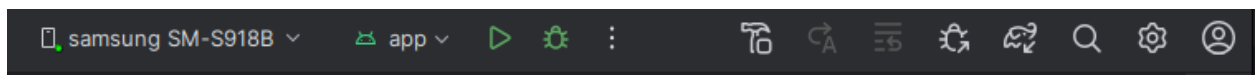

Fig. 13 - Android Studio recognizing my Samsung Galaxy phone

Now to run the application, just press the play button, and the app will be launched in your mobile device.

2.6 - APP PERFORMANCE BENCHMARK

To make the models' benchmark more reproducible and automated, I have developed a jupyter notebook called **test_tflite_models.ipynb.** In it, I load the same three models that are deployed to the app and use the tf.lite.Interpreter class to perform inference on this model

```python
for path in img_paths:
    # read image
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_orig = img.copy()
    # the model expects data in float32 format
    img = img.astype('float32')
    # creating two separate image versions: One for the emotion model and one for the age and gender model
    img_gender_age = cv2.resize(img, (224,224))
    img_gender_age = img_gender_age.reshape(1, 224, 224, 3) / 255 # normalizing the img
    img_emotion = cv2.resize(img, (48,48))
    img_emotion = cv2.cvtColor(img_emotion, cv2.COLOR_RGB2GRAY)
    img_emotion = img_emotion.reshape(1, 48,48, 1) / 255 # normalizing the img

    # Set the input tensor
    emotion_interpreter.set_tensor(emotion_input_details[0]['index'], img_emotion)
    age_interpreter.set_tensor(age_input_details[0]['index'], img_gender_age)
    gender_interpreter.set_tensor(gender_input_details[0]['index'], img_gender_age)

    # Run the model
    emotion_interpreter.invoke()
    age_interpreter.invoke()
    gender_interpreter.invoke()

    # Get the output tensor
    emotion_output_data = emotion_interpreter.get_tensor(emotion_output_details[0]['index'])
    age_output_data = age_interpreter.get_tensor(age_output_details[0]['index'])
    gender_output_data = gender_interpreter.get_tensor(gender_output_details[0]['index'])

    # retrieving the original names from the file names:
    age_label, gender_label, emotion_label, _ = path.split('/')[-1].split("_")

    # appending the evaluations to the lists, so we can later compute the accuaracies of each deployed model
    age_evals.append(int(age_label == ("young" if age_output_data.argmax() < 65 else "elderly")))
    gender_evals.append(int(gender_label == gender_labels[gender_output_data.argmax()]))
    emotion_evals.append(int(emotion_label == emotion_labels[emotion_output_data.argmax()]))
```

Fig. 13 - Classifying the images using the tf-lite models
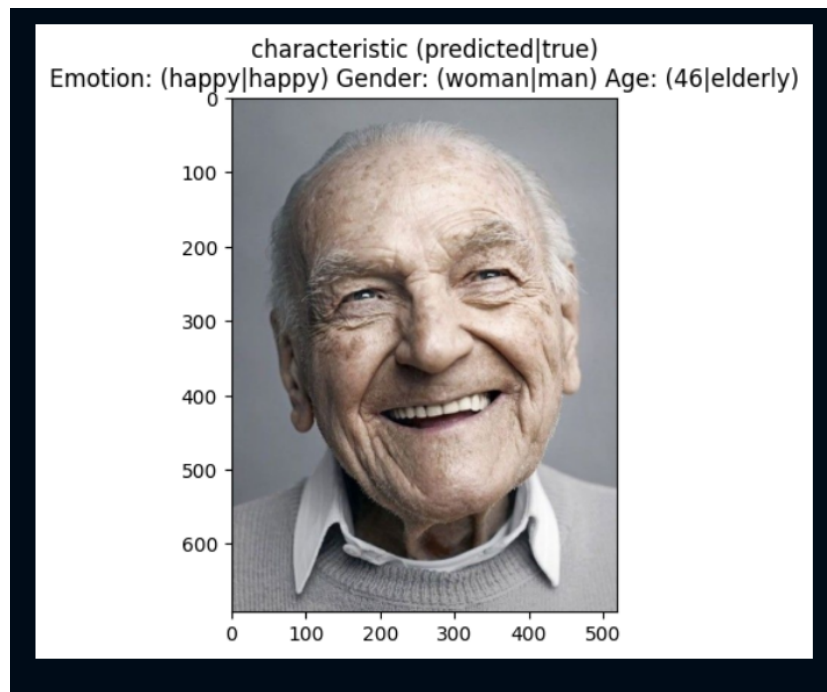
iu INTERNATIONALE
HOCHSCHULE

Fig. 14 - Example of classification

Looking at this notebook's results, it is possible to evaluate the impacts of quantization on the model's performance. The original gender model had an accuracy of 87.5%, while the equivalent tf-lite model had an accuracy of 50% on the same images. for the age model, I used the age of 65 to be the threshold between young and elderly people. That being said, the accuracy of the original model was 66,7%, while the tf-lite model had an accuracy of 50% on the same images. The emotion model was the only one to have the same accuracy(29,2%) in both cases.

The downgrade in the model performance makes sense since it is only observed in the two models that have gone through quantization. These models had their size reduced by approximately 75%, so it is expected that some accuracy loss would happen. In a more realistic scenario, one would have to weigh the trade-off between efficiency X accuracy and evaluate whether the model could make it to production or whether further optimizations are necessary.

3 - CONCLUSION

During this project, I developed an Android application that runs three Neural Networks on the edge, i.e., using the hardware of my mobile phone. Using pre-trained neural networks, I have evaluated the original model's performance in the final evaluation set of 20 images. Each of the models needed to be converted to the tf-lite format so that they could be deployed to my smartphone. For the gender and age models, given their original size, post-training quantization was necessary during the conversion process. A simple Android application was developed, and a demo of the working system was made.

using the Jupyter notebooks built throughout the project, It was possible to benchmark the performance of each model on different scenarios (original vs. edge) and assess how post-training quantization impacts the models' accuracy when necessary.

As possible next steps, I would improve the model's accuracy, mainly the emotion model, which showed an accuracy of only 29,2%. Maybe training the models for some additional epochs helps it to differentiate between the classes. Another possible improvement avenue is trying to minimize the impact of quantization in the deployed models. The gender model, for example, was performing really well in the original scenario but seemed to make more incorrect predictions when deployed to the phone.

REFERENCES

1. Project files:
   https://drive.google.com/file/d/1GBYADstZUy6HaYidOhYKbRbUE6ebKJz
   D/view?usp=sharing
2. hualcosa. (n.d.). DLBAIPEAI01_edge_ai. GitHub.
   https://github.com/hualcosa/DLBAIPEAI01_edge_ai/tree/main
3. Albuquerque, H. (n.d.). Edge AI - TensorFlow Lite for Microcontrollers.
   YouTube.https://www.youtube.com/watch?v=W6xsKZ0PbUg&ab_channel
   =HugoAlbuquerque
4. TensorFlow. (n.d.). Examples. TensorFlow.
   https://www.tensorflow.org/lite/examples
5. Serengil, S. I. (n.d.). deepface. GitHub.
   https://github.com/serengil/deepface/tree/master
6. TensorFlow. (n.d.). SavedModel format. TensorFlow.
   https://www.tensorflow.org/guide/saved_model
7. TensorFlow. (n.d.). Post-training quantization. TensorFlow.
   https://www.tensorflow.org/lite/performance/post_training_quant#convert_t
   o_a_tensorflow_lite_model
8. TensorFlow. (n.d.). Image classification. TensorFlow.
   https://www.tensorflow.org/lite/examples/image_classification/overview
9. Kumar, S., Rani, S., Jain, A., Verma, C., Raboaca, M. S., Illés, Z., &
   Neagu, B. C. (2022). Face spoofing, age, gender and facial expression
   recognition using advance neural network architecture-based biometric
   system. Sensors, 22(14), 5160.
10. Mikhailiuk. A. (2022). On the edge — deploying deep learning applications
    on mobile. Retrieved 22 November 2023 from
    https://towardsdatascience.com/on-the-edge-deploying-deep-applications-
    on-constrained-devices-f2dac997dd4d