BE.services

Exercise counter for healthier office lives

Face Recognition and Pose Estimation accelerated by Verdin iMX8M Plus NPU

Software.
Embedded.

Customer: NA
Project name: Workout tracking
Version: 0.1

CONTENT

1 Overview

The Verdin iMX8M Plus from Toradex is a System-on-Modules (SoM) that is built with NXP®'s i.MX 8M Plus application processor. It includes a co-processor specifically designed for execution of deep learning inference called the Neural Processing Unit (NPU). This enables the embedded device to perform vision tasks, such as image recognition, object detection or pose estimation, at high frame rates.

The aim of our demo application is to count workout exercises performed by employees, which can be used with incentives for certain milestones to promote a healthier office life. More precisely, the module is connected to a web camera and recognises when a person enters the frame. Then, face detection is performed to localise the face, which is used for facial recognition to determine which employee is currently standing in the frame. Then pose estimation is done to determine whether an exercise has been performed. These are counted and can be persisted in a database for accumulating scores over a chosen period, such as weekly or monthly.

1.1 System architecture

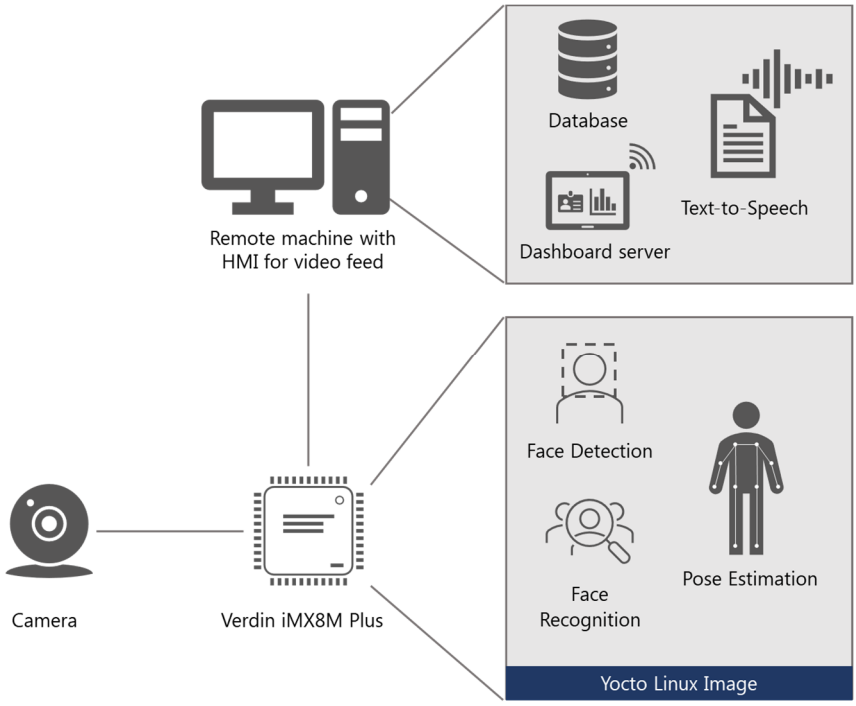

Figure 1: System architecture

The Verdin is connected to a USB webcam whose images are captured and analysed on the SoM. Various information, for instance detected employee, repetition counter etc, are added to the video feed. The feed is then streamed via network to machine that has a display attached. Optionally the display can also be attached to the Verdin. Additionally, a database (MongoDB 4.4) is setup on the remote machine, which is used to save the counted exercises. The daily and weekly statistics are displayed on a dashboard that is also served from this machine. Lastly, it also runs a Text-to-Speech (TTS) model for greeting recognized people and to give an audio feedback about successful saving of the respective count.

The module's OS is a Yocto Linux Image, which is based on Toradex's reference images and includes the recipes for eIQ software stack provided by NXP. It includes a few Machine Learning inference engines, for instance Tensorflow Lite, which is used for the application, as well as multimedia software Gstreamer for recording and sending video streams.

On the module several Deep Learning components are used: On the one hand there is a face recognition pipeline, consisting of detection and the recognition, on the other hand there is the pose estimation. The former

is required to associate the exercise count with the correct person, the latter is used for the actual counting of the exercises by tracking the movement of detected key points.

2 Implementation of Deep Learning models

2.1 8-bit Quantization

In contrast to other inference accelerators, most notably GPU's, the NPU only supports only integer-based calculations. Consequently, Neural Networks, which after training typically use weights and activations in floating-point format, must be fully quantized to 8-bit integers to be executed on the NPU.

Mathematically, quantization means mapping a floating-point number $x \in [\alpha, \beta]$ to a fixed-point number $x_q \in [\alpha_q, \beta_q]$.

Quantization and Dequantization are defined as $x_q = round(\frac{1}{s}x + z)$ and $x = s(x_q - z)$, respectively.

Scale s and zero-point z used in these formulas are calculated as $s = \frac{\beta - \alpha}{\beta_q - \alpha_q}$ and $z = round(\frac{\alpha_q \beta - \alpha \beta_q}{\beta - \alpha})$.

Additionally, values must be clipped if the exceed the maximum or minimum value of the respective data type. For a more detailed discussion, see Lei Mao's blog entry.

In general, there are two methods for quantization: post-training quantization, which is less complex, and quantization-aware training, which is better for maintaining model accuracy. Post-training quantization is applied after the training was completed and converts the weights and activations into the nearest 8-bit fixed-point numbers. As the activations depend on the input of the model, a representative dataset is required to estimate the range of possible activations.

We are using the Python API of the Tensorflow Lite converter for quantization and conversion of the models. The TFLite conversion with full integer quantization is performed as follows, see documentation:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data_gen
# Ensure that if any ops can't be quantized, the converter throws an error
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
# Set the input and output tensors to uint8 (APIs added in r2.3)
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
tflite_model_quant = converter.convert()
```

In Tensorflow 2.x the model can be supplied either in SavedModel format or as a Keras model. Additionally, it is possible to convert concrete functions to TFLite models. The quantization of input and output nodes of the graph ensure that the whole model graph is integer quantized. As a result, input data must be quantized prior to feeding it into the model. Likewise, the output of the model might have to be dequantized to perform subsequent calculations. For instance, the bounding box output in UINT8 ranges from 0 to 255, which must be dequantized back to a floating-point number and rescaled to the input picture's dimensions to accurately draw a bounding box.

Since the face detection model is provided as an ONNX model, conversion to TF is required before converting to TFLite. This is done using this ONNX to TF converter. The respective API is very straightforward:

```
import onnx
from onnx_tf.backend import prepare


onnx_model = onnx.load("input_path")  # load onnx model
```

```
tf_rep = prepare(onnx_model)  # prepare tf representation
tf_rep.export_graph("output_path")  # export the model
```

As a result, all operations that are part of a model must have a quantized implementation in the respective inference runtime you are using, otherwise the operation will fall back to execution on the CPU, which reduces inference speed.

2.2 Face Detection

Figure 1 Face recognition workflow

The first step when a new person enters the frame is to associate the person with a known employee, which is realized via face recognition. Figure 1 shows the workflow of modern facial recognition pipelines. Generally, a face alignment step is part of this process, however in our application this step is omitted.

For detection the ultra-lightweight face detection model in the slim variant by Linzaer is used, which is provided under MIT license. It is an object detection model that is pre-trained on the WIDERFACE dataset.

A qualitative assessment of the output of the quantized model is given in Figure 2.

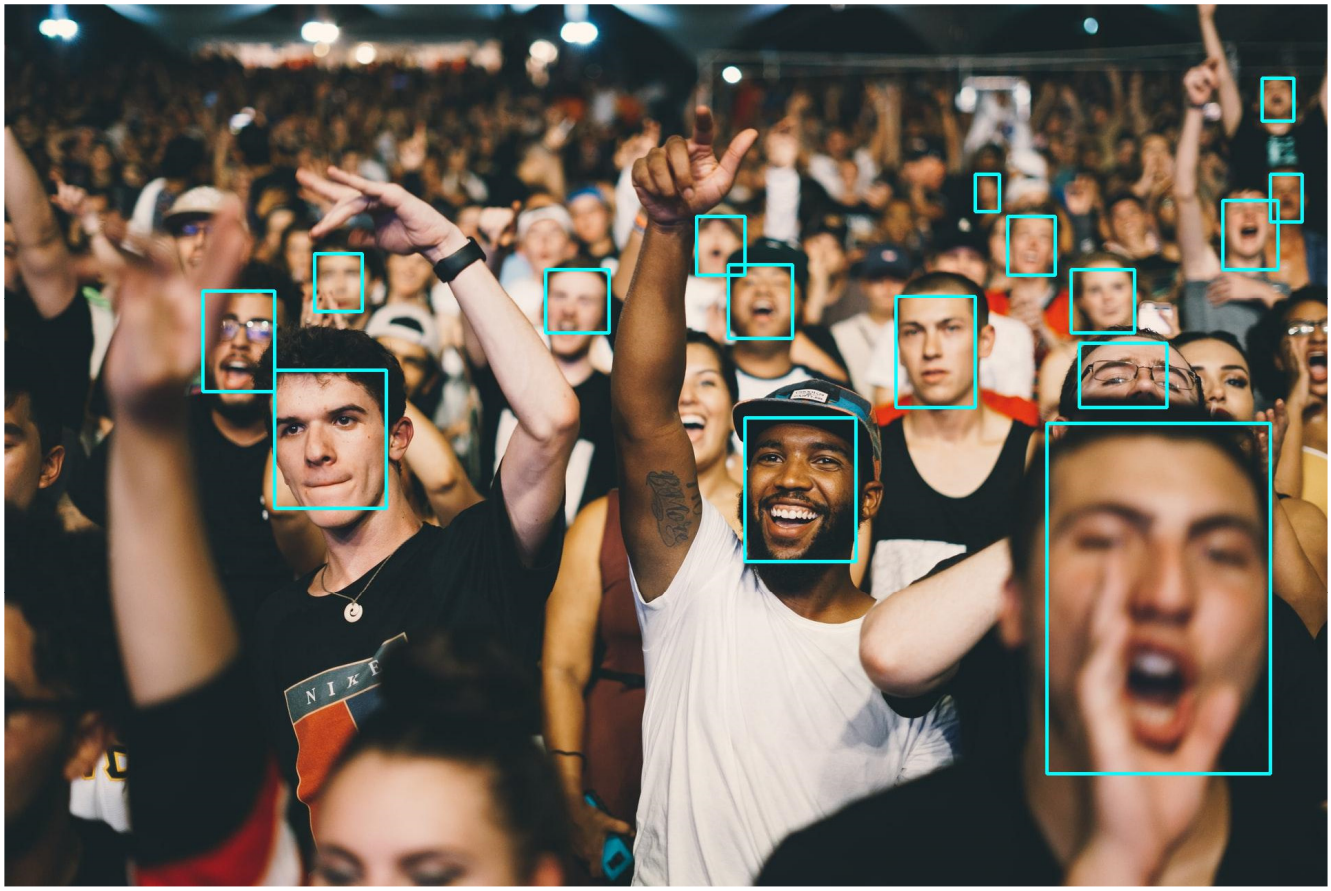

Figure 2: Face detection example, Photo by Nicholas Green on Unsplash

It works well on faces on different scales, faces that are partially occluded and with varying complexion. The model is provided in ONNX format and converted to Tensorflow 2.0 SavedModel before being quantized and converted to a Tensorflow Lite model which can be deployed on the Verdin.

For this a representative dataset is required, as explained in section 2.1. For this, a subset of the *Labeled Faces in the Wild* (LFW) dataset is used. Images are loaded from storage and used during quantization. After inference the proposed boxes have to be filtered by Non-Maximum suppression and therefore the output of the model has to be dequantized beforehand. The remaining boxes indicate the location of faces in the image and are used to crop out said region as input for the subsequent feature extraction by the face recognition model.

2.3 Face Recognition

The second part of the face recognition pipeline is feature extraction and comparing these features, to features of faces of known employees. This is done by using a CNN to calculate a 4096-dimensional embedding vector for every detected face and comparing the distance to embedding vectors of known employees. Frequently, the Euclidean distance is used for comparison, however, other distance measures such as the cosine distance are possible. If the distance is larger than a given threshold, it can be assumed that a person is unknown and not one of the employees.

The representative dataset for quantization of this model is generated similarly to section 2.2. Instead of using the unprocessed images from the LFW dataset, the images are loaded, and face detection is performed to crop out only the faces in the image. Next, the pre-processing function specific to the Resnet50 version of the model is applied and the result is used as an input.

Comparing the people entering the frame to known employees requires a collection of features generated by running the feature extraction on images of the employees faces. This means, the whole pipeline in Figure 1 must be executed and the results must be saved prior to running the application.

For recognition, a Keras implementation of the VGG Face descriptor Caffe model originally developed by researchers from Oxford university is used. The conversion process from Keras model to Tensorflow Lite model is almost identical to the one described in the previous section, however, the conversion from ONNX to Tensorflow is not required this time. In our project the Resnet50 variant is used.

2.4 Pose Estimation

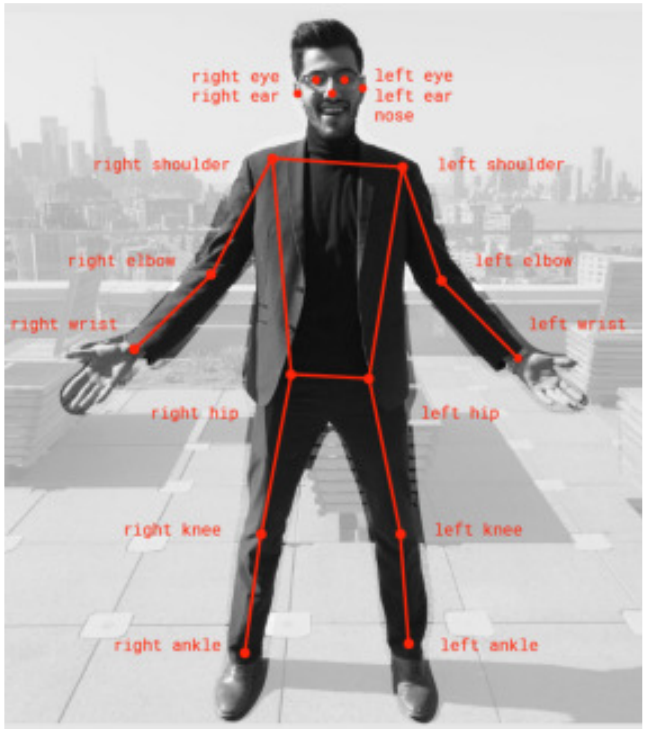

Figure 3: PoseNet keypoints, Source: Google Coral

Pose estimation is the process of estimating the locations of key body joints of a person from a video or image, with respect to the image. These locations can be used to calculate angles and distances between one keypoint to another, which in turn can be used to track motions of the human body, see Figure 3.

In this application it is used to count squats done by a person in front of the camera by calculating the angle of the legs and the distance of hip and knee keypoints.

Here the PoseNet model pre-trained by the Google Coral team and provided under Apache 2.0 license is used. As it is intended for use on Google EdgeTPUs, the model is already available in quantized format, with the caveat that there is a custom operation at the end of the network graph, which decodes the heatmaps and offset vectors into a list of actual keypoints and which cannot be executed by the NPU. However, the repository also provides a custom delegate for this operation, which makes the decoding of the outputs very convenient. Instead of working with the heatmaps and the offset vectors directly, the custom operation

provided in the repository performs the decoding. As this custom operation is not included in the Tensorflow Lite runtime, it can only be executed via the delegate API in TFLite. The delegate is a shared library that contains the actual implementation of the custom operation. In our case it is loaded as an external delegate via the Python API:

```
from tflite_runtime.interpreter import load_delegate
POSENET_SHARED_LIB = 'posenet_decoder.so'
posenet_decoder_delegate = load_delegate(POSENET_SHARED_LIB)
interpreter = Interpreter(model_path,
experimental_delegates=[posenet_decoder_delegate])
```

Since this decoding operation is executed via its own delegate, the NNAPI delegate, which is used for execution of calculations on the NPU, cannot be used for this operation, instead it is executed on the CPU. Also, the output of the main PoseNet model must be dequantized before using it as input for the decoding operation, as the latter uses FP32 as the data type.

An example of the output of the complete PoseNet model is given in Figure 4.

Figure 4: Pose estimation example, Photo by Eliott Reyna on Unsplash

2.5 TTS

Coqui TTS is used on for giving audio feedbacks to users of the system. More specifically, the default Tacotron 2 model for generating the spectrograms from the input character sequence and a Multiband-MelGAN to transform those spectrograms to waveform audio files were used. These models are pretrained for English language on the *LJ Speech Dataset*. It is important to note that TTS is not executed by the Verdin, but rather on the machine that is running the database as a means of informing the user whether his exercises were saved successfully in the database.

3 Squat Counter

Our application is custom-built to detect and count squats. Face recognition and pose estimation are performed independently from each other. The logic is as follows:

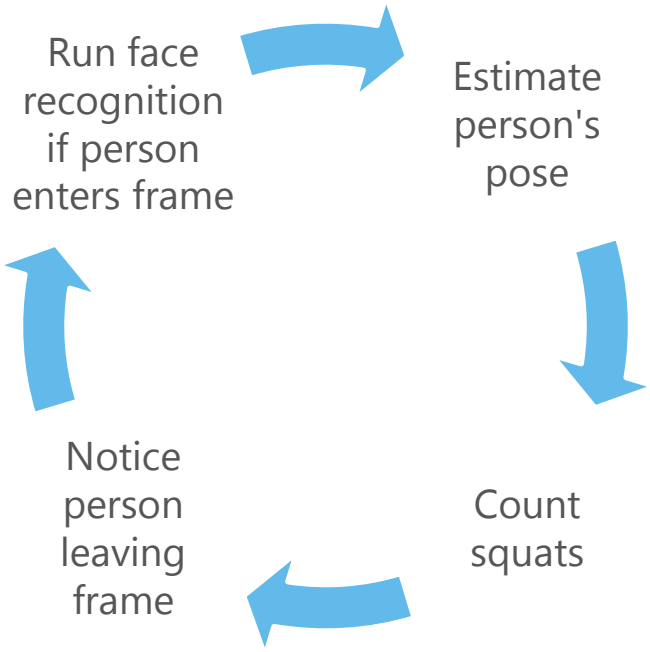

Figure 5: Process of squat counter

The application has essentially two states: Face recognition mode and pose estimation mode. If no person is in the frame, pose estimation is running continuously. A person entering the frame is detected once a pose has been detected with sufficient certainty by the pose estimation model. Then pose estimation is paused and face recognition is performed for several frames, while keeping track of the distances and the most likely employee. After this, the distances are evaluated, and the respective employee is selected in case the distances are below the cut-off. Currently the application only supports individual persons in the frame, therefore the application is paused, if more than one person's pose is detected. Once only one person is in the frame again, the face recognition is run before entering pose estimation mode.

Whenever the person is detected the Verdin sends an HTTP request to the remote machine that is running the TTS and database, so that a greeting is played. The remote machine has a simple RESTful API that performs the TTS calls and the calls to save data in the database. Additionally, when squats were done and the respective person leaves the frame, the number is sent with a POST request to be saved in the DB, of which the person in front of the camera is informed of by a TTS message.

During pose estimation mode, the bending of the legs during the squats are measured by calculating angles and distances between hip and knee keypoints. One successful repetition consists of bending the legs sufficiently, so the angle is larger than a given threshold, and returning to a position with straight legs again. Whenever the number of detected people in the frame changes, the number of squats is persisted, if there are more than zero and the respective person is a known employee.

Images of the states described in Figure 5 can be seen in Figure 6.

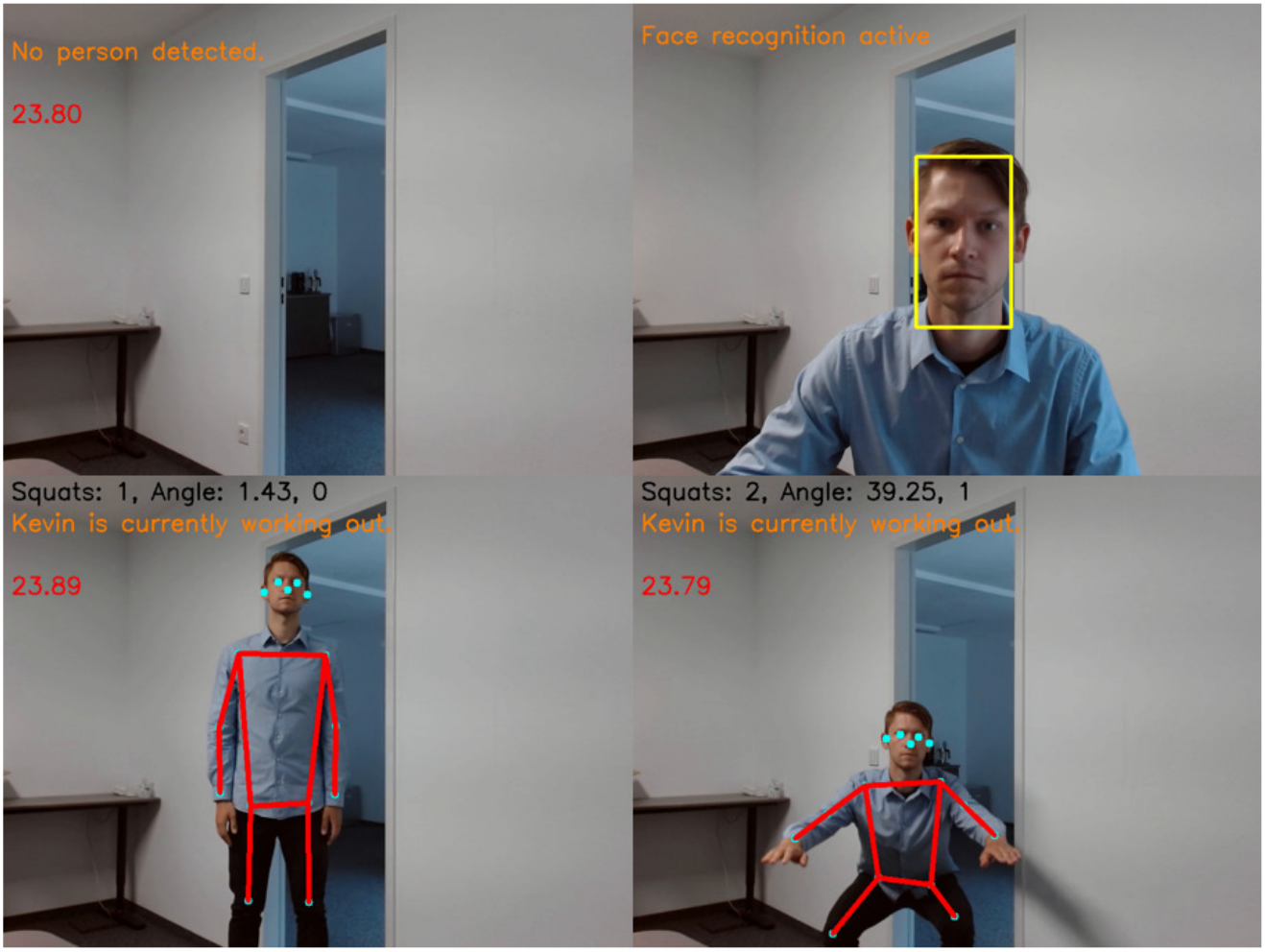

Figure 6: Squat counter live images

4 Performance Overview

The latency times for the used models in different versions are given in Figure 7. For the pose estimation models the latency times are split into the main model and the decoding operation, as the former is executed by the NPU and the latter by the CPU. The total times include the feed forward pass through the network as well as potential post-processing operations, such as Non-Max suppression or dequantization of outputs. They can be seen as the end-to-end from input of the Neural Network to usable results for other parts of the program. The webcam is recording with 24fps at a resolution of 800x600 pixels. The pose estimation models are available with two different architectures, Mobilenet for higher throughput and Resnet50 for better accuracy. At the lowest input dimensions of 481x353, the pose estimation requires roughly 20ms per frame; a framerate of around 50fps. Using the lowest resolution Resnet50 backbone takes 76ms per iteration, yielding roughly 13fps. The face detection has a latency of 37ms and the feature extraction of around 20ms, which brings the facial recognition pipeline to around 20fps.

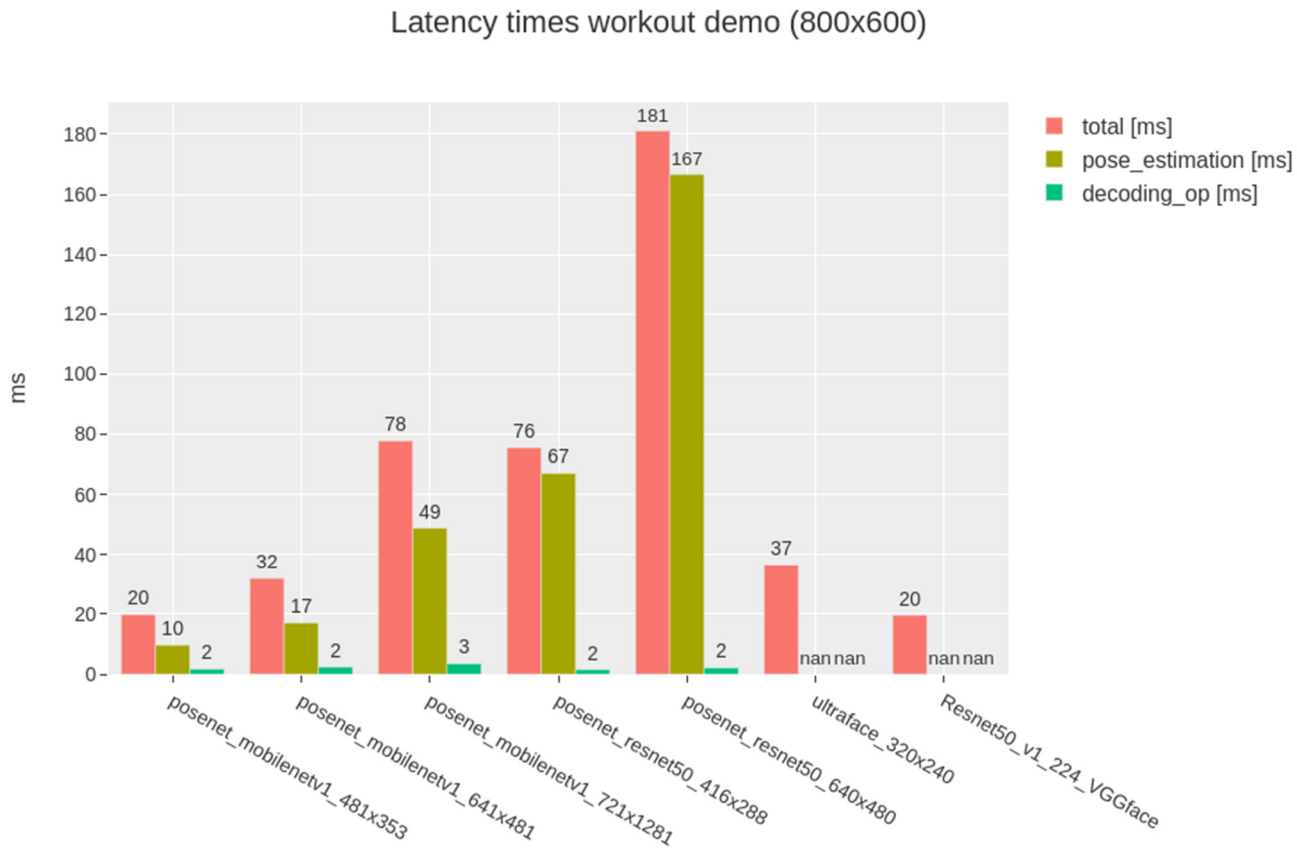

Figure 7: Latency times for different models

5 Dashboard Overview

The dashboard contains mainly a daily and a weekly overview. The daily overview shows a donut chart with the total amount of the daily exercises and which fraction was done by which employee. Additionally, also the absolute number of exercises is shown in form of a bar chart that is sorted in descending order. A horizontal bar chart shows the amount that was accumulated in the current week. In this weekly overview, the chosen thresholds of the incentives can be seen and whether they were surpassed. For additional motivation, the three people with the highest count this week are highlighted. Selection of the date and the corresponding week to display is done via the date picker at the top. A view of the dashboard is given in Figure 8.

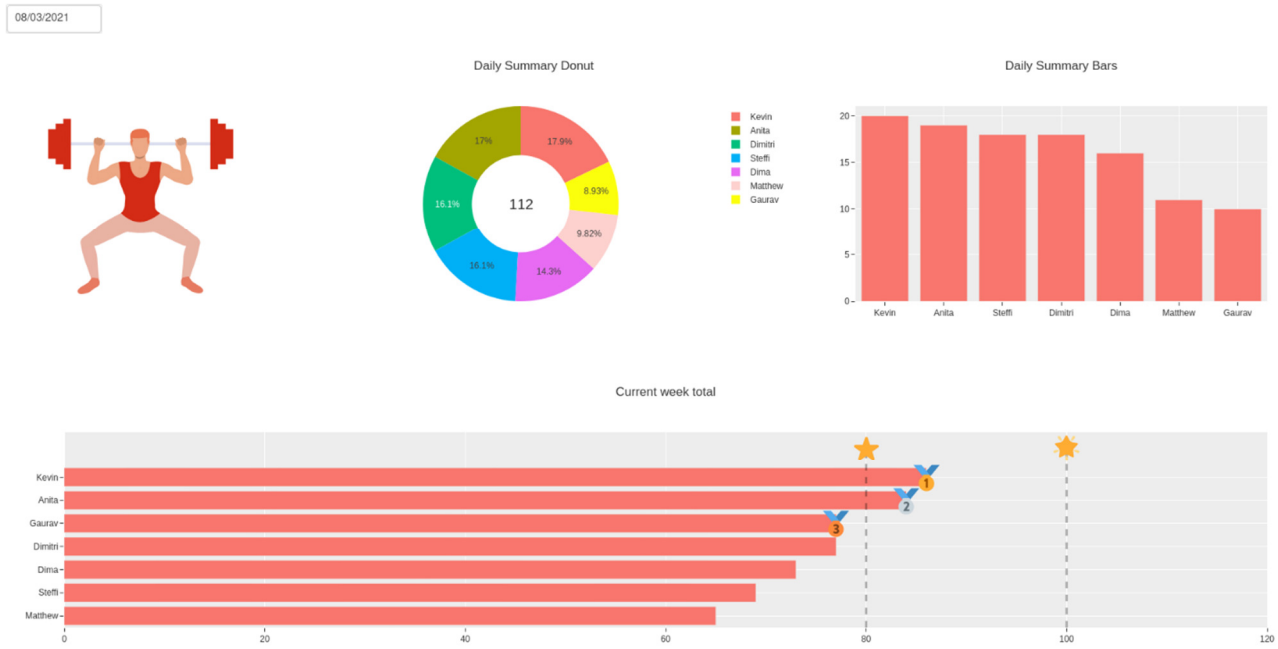

Figure 8: Dashboard

6 Glossary

API Application Programming Interface

CNN Convolutional Neural Network

CPU Central Processing Unit

GPU Graphics Processing Unit

LFW Labelled Faces in the Wild, image dataset

NPU Neural Processing Unit

NXP Semiconductor Manufacturer

ONNX Open Neural Network Exchange

OS Operating System

SoM System-on-Module

TF Tensorflow

TTS Text-to-Speech

USB Universal Serial Bus