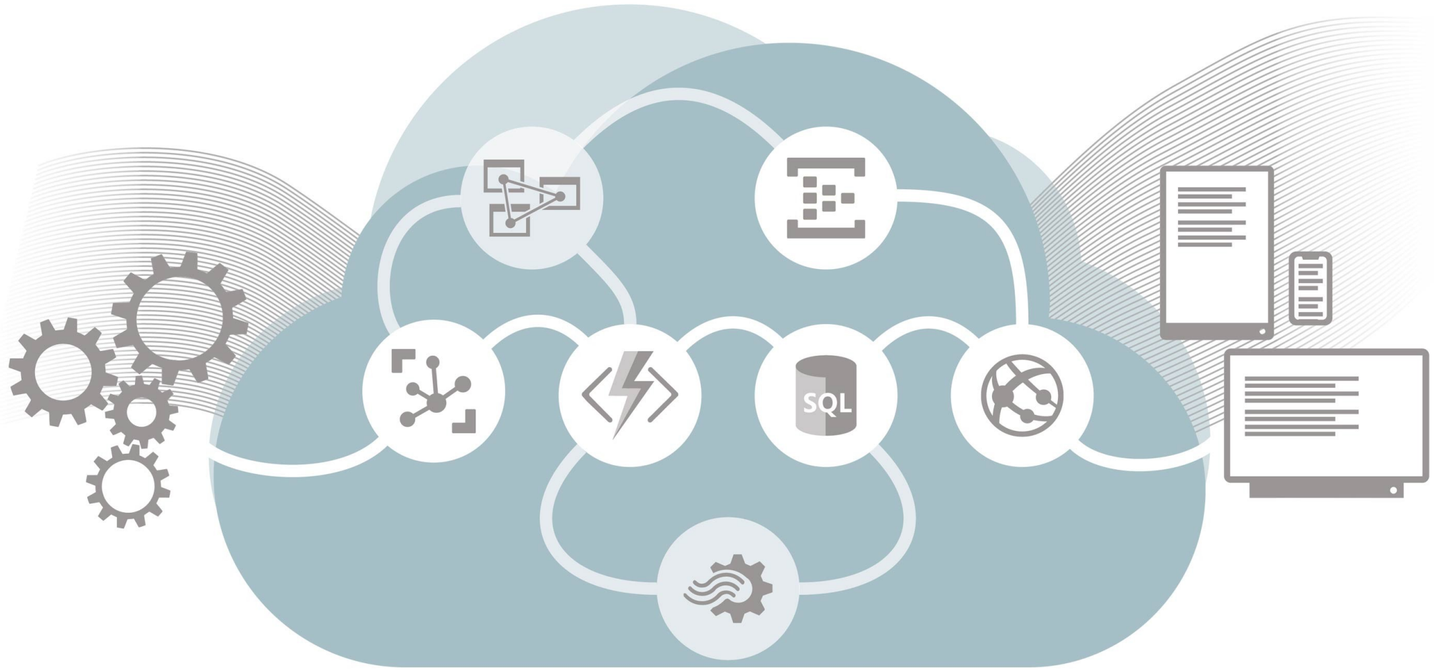
basysKom

# Build Deep Learning Applications with Qt and Tensorflow Lite



basyskom

# Outline

- About Us
- Introduction to TensorFlow and TFLite
- Building an example application with TFLite and Qt
  - Selecting a model
  - Loading and investigating the model
  - Fill the input tensors
  - Read the output tensors
- Results

basysKom

# About Us

Software Engineering Services

- Consulting, Training, Coaching, Development
- C++/Qt, Web/Angular, Cloud/Microsoft Azure
- Customers: Manufacturing, Measurement Systems, Industrial Automation, Software Development
- Open Source: Qt, Qt OPC UA & open62541

Located in Darmstadt and Nuremberg

Division of UX Group

Berthold Krevert

Software Engineer @ basysKom

Qt, QML, Python, Cloud Development

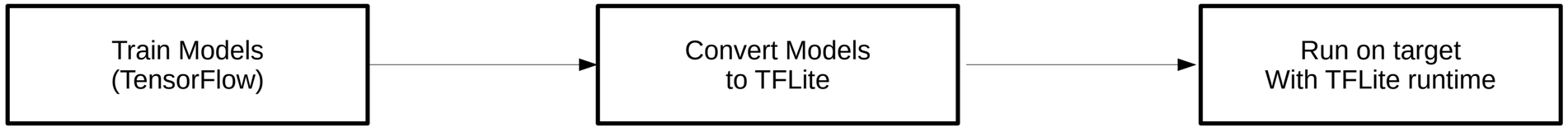
# Introduction to TensorFlow and TFLite

basysKom

# TensorFlow and TFLite

- TensorFlow
  - Deep Learning Framework released by Google
  - Huge and Complex
  - Used to train **and** run models
- TensorFlow Lite (TFLite)
  - Developed for Embedded, Mobile, Edge Devices
  - Used to run models
    - Runtime to execute models on target hardware
  - Can be accelerated by hardware (like Google Coral)

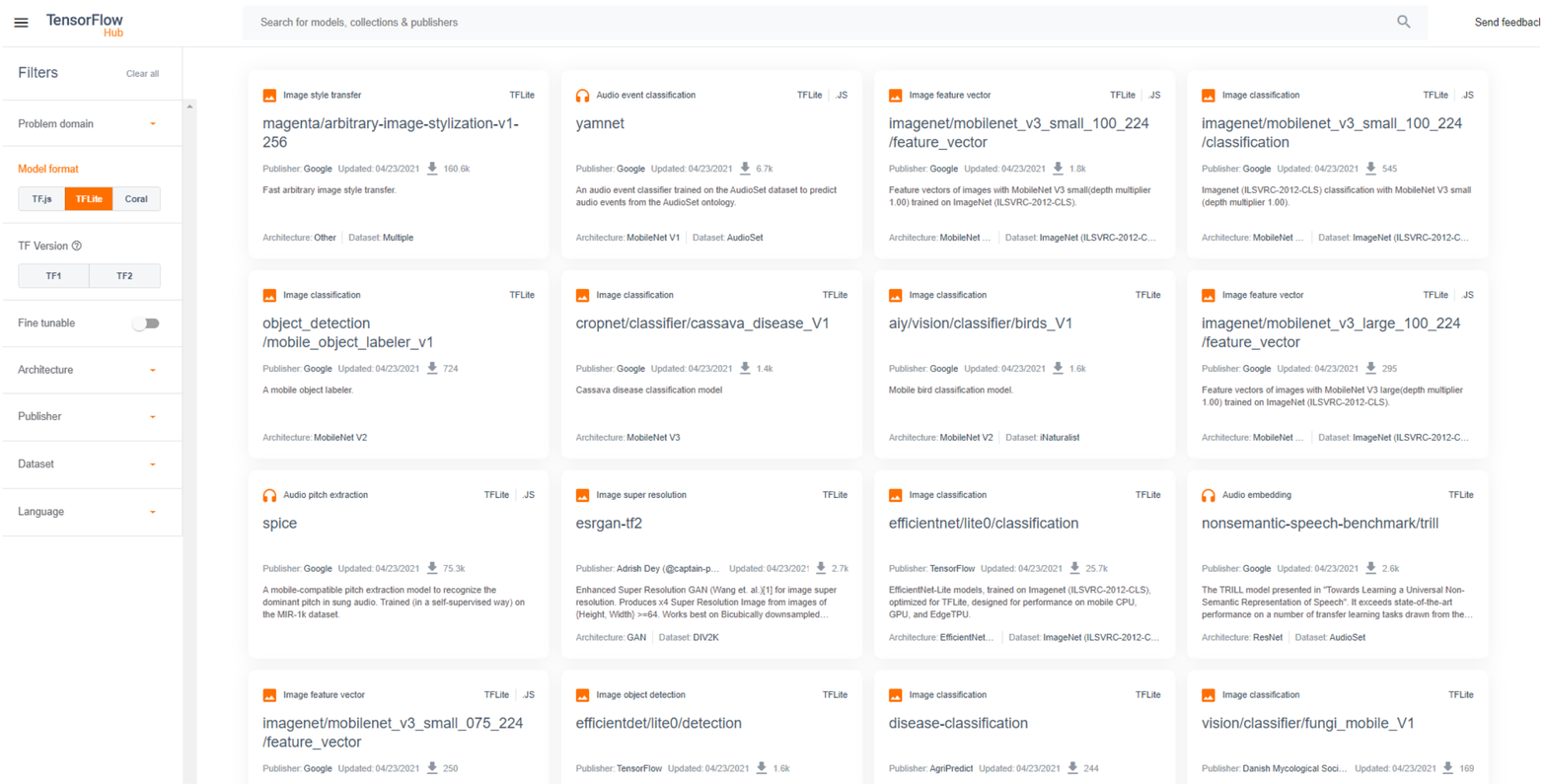
TensorFlow



basysKom

# TFLite Models

- Easy to start with
  - Google offers pre-trained TFLite models
  - Can be downloaded on TensorFlow Hub ( https://tfhub.dev/)
  - Mainly Models for Image Processing
    - Classification
    - Object Detection
    - Segmentation
    - OCR

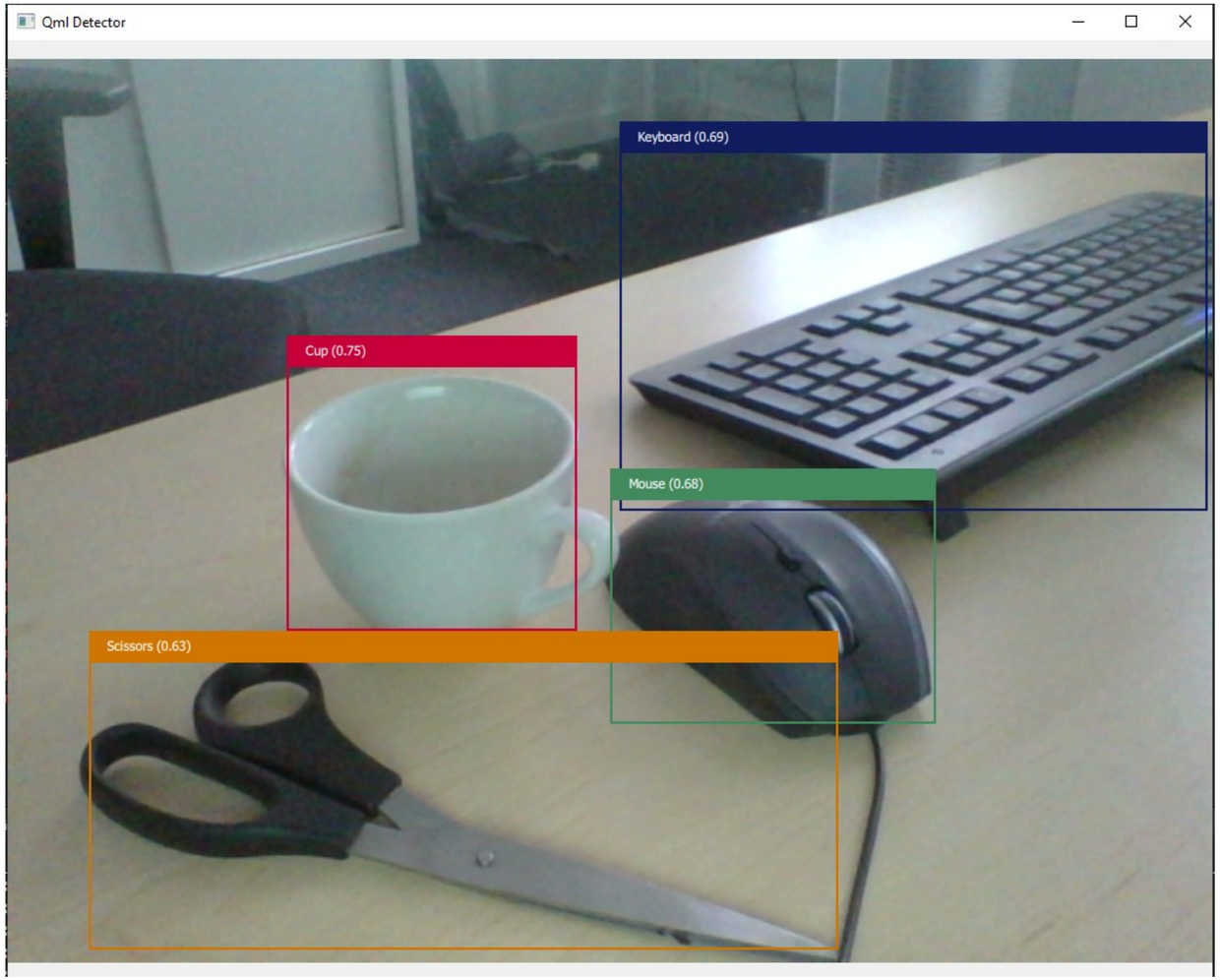


# Building an example application with TFLite and Qt

basysKom

# Example Application

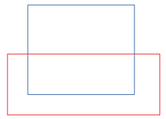
- Object Detection
  - Model pretrained by Google (SSD Mobilenet)
- The papers for the details:
  - SSD: Single Shot MultiBox Detector: https://arxiv.org/pdf/1512.02325.pdf
  - MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications: https://arxiv.org/pdf/1704.04861.pdf



# Example Application

Input: Image 300x300x3
(UINT8)

TFLite Model



Box Positions (Top, Left, Bottom, Right)

(46, 51, …) Classes as indexes

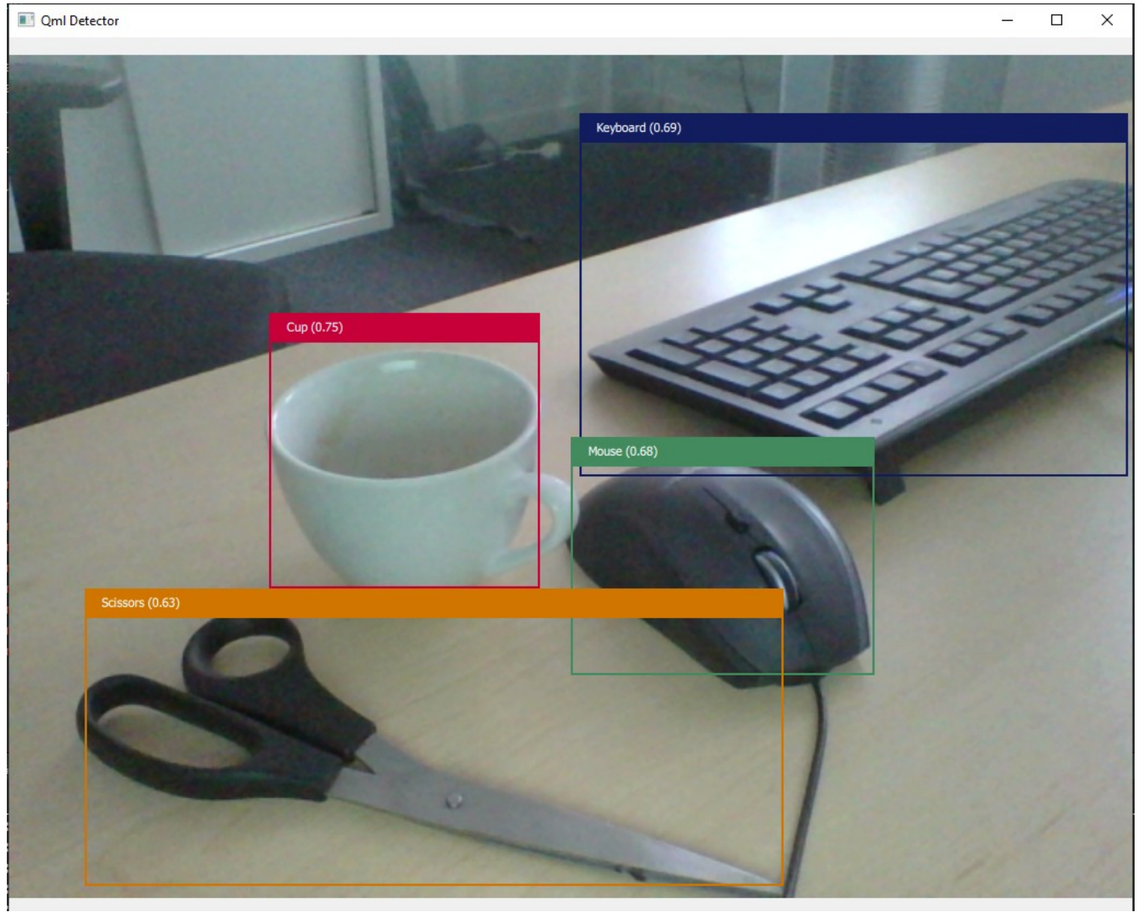
(0.9, 0.75, …) Probabilities

2 (Number of found objects)

basysKom

# Example Application

- Based on **QML/QtQuick**
- Video-Stream captured and rendered by **QtMultimedia**
- Processed by a **QVideoFilterRunnable**
  - Stores the frame in an **QImage** with format **QImage::Format_RGB888**
  - Flips the image in y-direction if its stored bottom to top
- Detection is executed in separate thread
  - Invokes the TFlite model
  - Stores results in QAbstractItemModel
- QAbstractItemModel is used to ...
  - … map class indexes to label names
  - … render Rectangles and Text items in QML for probabilities and labels



basysKom

## Initialisation

- Load model
- Initialize TFLite
- Try to find out more about the model architecture (i.e. input ar output sizes)
- There is one input tensor:
  - **Input dimensions: (1, 300, 300, 3)**
- There are four output tensors:
  - **Output dimensions: (1, 10, 4)**
  - **Output dimensions: (1, 10)**
  - **Output dimensions: (1, 10)**
  - **Output dimensions: (1)**

```
m_model = tflite::FlatBufferModel::BuildFromFile(filename.toLocal8Bit());
// ... init TFLite
int counter = 0;
for (auto tensorInputs : m_interpreter->inputs()) {
    TfLiteIntArray* dims = m_interpreter->tensor(tensorInputs)->dims;
    if (dims) {
        QVector<int> inputDims;
        for (int j = 0; j < dims->size; j++) {
            inputDims << dims->data[j];
        }
        qCInfo(objectworker) << ": Input dimensions: " << inputDims;
    }
    counter++;
}
```

basyskom

## Input

- Ensure that image and input tensor have the same format
  - Here RGB888
  - Size: 300x300
  - 3 channels
- Copy the raw image data to the tensor input

```
auto scaledImage = image.scaled(m_requestedInputWidth,
                                m_requestedInputHeight,
                                Qt::IgnoreAspectRatio);

copyImageToTensor(m_interpreter->typed_tensor<uint8_t>(imageInput),
                  scaledImage.constBits(), scaledImage.width(),
                  scaledImage.height(),
                  channels);
```

basysKom

## Inference

- Execute the mode

```
TfLiteStatus status = m_interpreter->Invoke();

if (status != kTfLiteOk) {
    qCWarning(objectworker) << "Failed to inference the image" << status;
    return;
}
```

basysKom

## Output

- After executing the model, the results can be copied from the output tensors
- For this model, outputs are stored as floats
- Four output tensors
  - Locations (Top, Left, Bottom, Right) of Bounding Boxes
  - Classes as indexes
  - Probabilities (Confidence scores)
  - Number of found detections in this frames
- Iterate over the floats and fetch data

```
float* CocoDetectionWorker::extractOutputAsFloats(int tensorIndex) const
{
    TfLiteTensor* tensor = m_interpreter->tensor(tensorIndex);
    Q_ASSERT(tensor && tensor->type == kTfLiteFloat32);
    return tensor->data.f;
}
```

```
const auto outputTensors = m_interpreter->outputs();
const float* locations          = extractOutputAsFloats(outputTensors[0]);
const float* outputClasses      = extractOutputAsFloats(outputTensors[1]);
const float* outputScores       = extractOutputAsFloats(outputTensors[2]);
const float* foundDetections    = extractOutputAsFloats(outputTensors[3]);
```

# Results

basysKom

# Result

Test Environment

- Detection on a Raspberry Pi 4
- Attached camera module
- A Google Coral accelerator connected to USB

Performance

- Without Google Coral: 300ms - 500ms
- With Google Coral: 30ms – 65ms
- With Google Coral (Max Setup): 15ms - 30ms

basysKom

# Questions?

Contact: **berthold.krevert@basyskom.com**

Please visit **http://www.basyskom.com** and **http://blog.basyskom.com**