-- final draft --

Comparing crowd detection between traditional Neural Network and TinyML on embedded devices

Mini Master Project - 2021

Vrije Universiteit Amsterdam

Author: Asif Anwar

Thesis Supervisor:

-- final draft --

Table of Contents

Abstract

This study aims to determine whether there is a difference between the performance of tiny machine learning models on embedded devices and traditional convolutional neural network-based surveillance systems to identify crown (human).

Keywords: TinyML, Embedded devices, CNN, Quantization.

1. Introduction

Monitoring crown behaviour is very important for government and organizations to build public safety systems. Circuit televisions (CCTV) cameras are widely used in different public locations like airports, parks, stations, stadiums and other crowded locations.

In recent days, Machine learning algorithms like convolution networks became very useful for crowd density estimation, object detection, pose estimation etc. However, the available machine learning technologies are expensive to deploy. They are also very bandwidth-hungry and require a lot of power to operate. And most importantly, all the current machine learning techniques cannot provide proper public privacy solutions.

Tiny machine learning algorithms (TinyML) deployed over micro-controller based embedded devices have a huge potential to provide a low-cost crowd monitoring system. By using deep learning computer vision using TinyML (e.g. TensorFlow Lite), we can design low cost but effective solutions for tracking objects and monitoring behaviours. Due to its architecture, all tasks will be performed in real-time right on the device. So the system will not require sharing any images to downstream systems which will essentially protect user privacy.

2. Background:

The following section provides background on traditional convolution neural networks, a novel machine learning technique called TinyML, embedded devices and their applications, quantization and embedded devices.

a. Neural Network

A Neural Network is a machine learning technique that was first introduced in the mid-forties. The concept of artificial neural networks came from biological neurons. In the

neural network system, input data are processed using a collection of functions known as neurons which are connected in different ways in different layers. Each neuron has an activation function that creates non-linearity. A neural network is trained using the optimizer functions with different learning rates.

The first introduction of the neural networks showed a lot of promises, however, due to the high demand for data and processing power the application of neural networks kind of faded away until recent days. Currently, the neural network is one of the most powerful machine learning algorithms. Due to sufficient data and processing power availability, now experts can apply neural networks in different use cases.

There are several types of neural networks and there are plenty of use cases of them. For this paper, we will consider a simple convolutional neural network that has been proven to be very effective for image classification with a relatively simple architecture.

b. TinyML:

The term TinyML is fairly new. It is a fast-growing branch of low powered AI technologies running on cost-effective microcontroller-based embedded devices to perform computer vision, language processing, sensor data analysis and other machine learning models right at the data source.

Ideally, TinyML refers to running a neural network model at an energy cost of below 1mW (milliwatt). This means a full-stack solution can run on embedded devices at a very low energy footprint. This essentially gives the power to build machine learning-based solutions on low-cost devices that are small enough to run on a coin size battery for a year without any human interventions.

But of course, the term TinyML can also refer to getting machine learning into any IoT type devices like mobile phones, drones, where there are constraints over computation power, memory usage, limited space and energy usage.

The TinyML gives the power to implement neural network models at the very source, the embedded devices where the actual sensors data is being collected. In the context of the image, audio, video, biomedical data, the embedded sensor devices become handy to collect data in real-time and process them right there using TinyML.

In principle, these networks run on three major components; large datasets, huge memory and lots of processing power. None of these is readily available for mass populations, and most of the time these networks are built and controlled by large organizations. Even though many organizations are gradually giving access to everyone and making these technologies open source, it’s very costly to run them for a small startup or individuals.

However, the story for TinyML is a little different. The cost of devices with sensors are very low, hence anyone can access them. The simple model can be built using free cloud services like Google Colabs. Processing at the source of data gives lots of real-life applications.

c. Model Quantization:

The process of converting large floating-point values to the smallest integer value is known as quantization.

In recent days different types of neural networks have achieved unbelievable success in the field of computer vision, language processing etc. However, the application for neural networks remains out of reach for many due to its complex structure which requires high computation resources and energy to perform. Hence it's nearly impossible to apply already trained neural networks in the embedded systems where lots of possibilities remain untouched.

However with the power of model compression now it's getting more and more feasible to apply huge neural networks in a small device where both memory and processing power is limited. One of the most common methods of model compression is model quantization.

For model quantization, deep learning models are converted from floating-point architecture to low-bit architecture. In this approach, the model is training in a regular fashion where model training is done using a floating-point precision calculation to optimize the loss function and to set the network weights. However, before applying the model to embedded devices the floating-point weights are converted into low-bit integer values such as 8-bit or 16-bit. As a result, while performing the machine learning task in the device the model requires significantly low memory and energy to perform

d. Embedded devices

A low power cost effect microprocessor-based system with software that is designed to perform a particular function is generally known as an embedded device. An embedded system generally has the low processing power and very low memory, however, base on the application the devices are connected with different types of sensors. As a result price and size of the embedded device is significantly low compared to traditional computer system solution.

An embedded device can operate solo or can be part of a large integrated system. Due to low power consumption, the device can be powered by small batteries and due to the

advancement of microprocessors the device are capable of running powerful application. Hence in the recent past, we have seen all types of internet of things (IoT) devices in the market. These devices can perform dedicated tasks in real-time with their limited memory and power.

Now with the advancement of quantization and other memory utilization approach; a full size trained network can be implemented into these embedded systems. Which gives us the power to apply already implemented solutions to embedded devices.

e. Surveillance system

The use case for this research is to compare the performance between the traditional neural network and TinyML system for surveillance. The system architectures of two different systems are given below.

For neural network-based surveillance systems already have their groundbreaking achievements. Traditionally for a crowd monitoring system, surveillance cameras are used which records everything in a local system. A train operator then has to keep an eye on the computer screens to observe and determine the activities. Either in real-time or post-event situations, manually video clips were extracted from the surveillance system.

NN Based Surveillance: With the success of neural network object identification; the manual tasks for the human operators have been reduced significantly. Below are the major components of a neural network-based surveillance system.

- **Surveillance Camera:** There are different types of surveillance cameras are used for capturing images and videos for surveillance. Base on the location, types of images are taking, lighting conditioning, the camera can features like low to high definition, 360 degrees vision, low light sensor, and heat-sensitive capture etc. In most cases, these cameras require AC power to operate, a good network connection for transmitting images and videos and hefty price tags.

- **Source System:** Usually the surveillance cameras are connected with a local system that is assigned to capture the images, do initial processing and be kept in local storage. As the surveillance cameras generally do not have any smart capabilities, this source system will run a specific application that will capture the based on the end application required and transfer the captures images and videos to the upstream system.

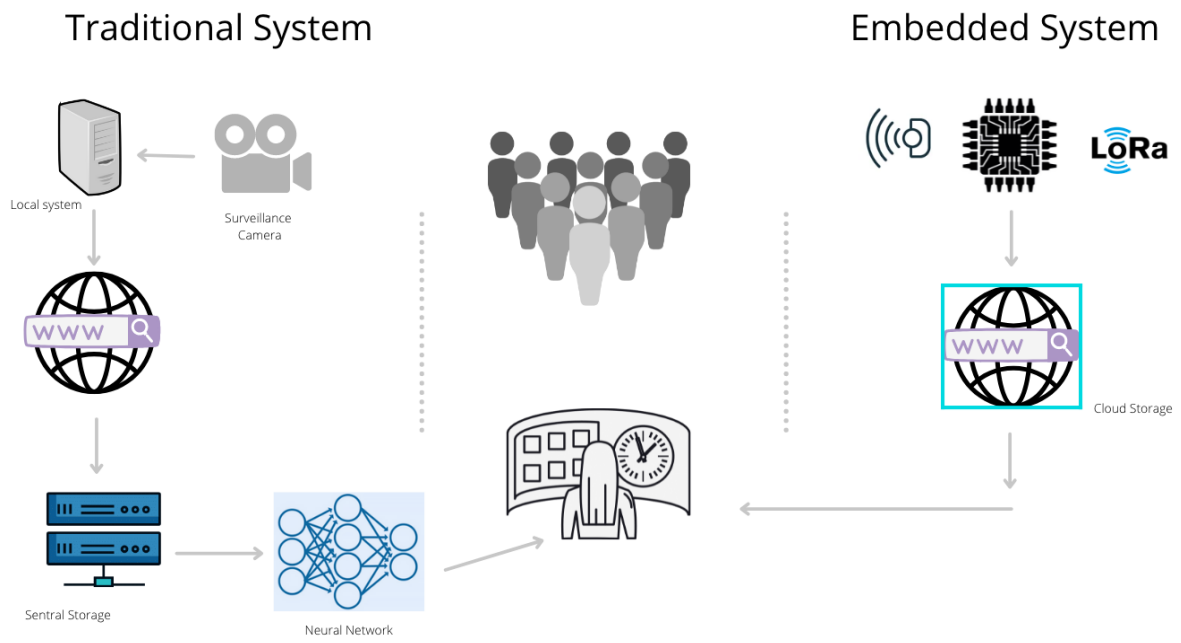
- **High-speed Internet/Local Network:** The unprocessed images from the surveillance camera are very large, hence the source system and surveillance

camera are connected with either a local network or high speed for instant transmission.

- **Cloud or central storage:** All the images from different locations surveillance areas and all historical data is kept in a central storage system or cloud facilities. Generally, some preprocessing are done with the images to reduce the size of the content before storing, like compression, and deletion of unnecessary data. However, based on the application requirements the size of this storage is generally very high which comes with a huge cost.

- **Neural Network:** There are different types of neural networks are trained and used based on the captures data. As mentioned earlier this part of the system has reduced human operation, as a result, the processing is very fast and saves manhours significantly. However, processing all captured images using a neural network requires lots of energy, server memory and cost.

- **Business Application:** Last but not least, a business application is used for presenting output data to the end-user. Based on the requirement the system can provide real-time and scheduled based alerts to the users. Users can find different types of reports coming out of the network. Sometimes users are also providing feedback which is used to improve the network.



Traditional Neural Network Vs TinyML for crowd surveillance

TinyML Architecture: The architecture for the surveillance system using tiny machine learning model using embedded devices are different from surveillance system using neural network. Below are the major components of the system:

- **Integrated System:** The main component of the TinyML architecture is an integrated device that covers most of the components together. There are five major parts of this integrated system.

    - **ARM Processor:** A very tiny microprocessor is the main component of the device. This processor is capable to run converted neural networks using low memory and power.

    - **Camera Sensor:** Low power camera sensor capable enough to capture images or videos based on given triggers or schedule based operation.

    - **Tiny Neural Network:** a converted version of a neural network is deployed as the main system of this integrated device. Theoretically, this network should be able to perform the same as a full network or at least very close. The research question of this paper tries to cover this.

    - **LoRa:** LoRa is a new radio technology for transmitting information. It runs with low power and is capable of transmitting information to the cloud when as required. As for the tinyML architecture only share the results, the system never has to send large images or data using LoRa.

    - **Power:** The whole device can run using a small tablet size battery for a month. So the batter requirements depend on the application and frequency of the use. Solar power cells can be easily integrated to power the system for an indefinite time.

-

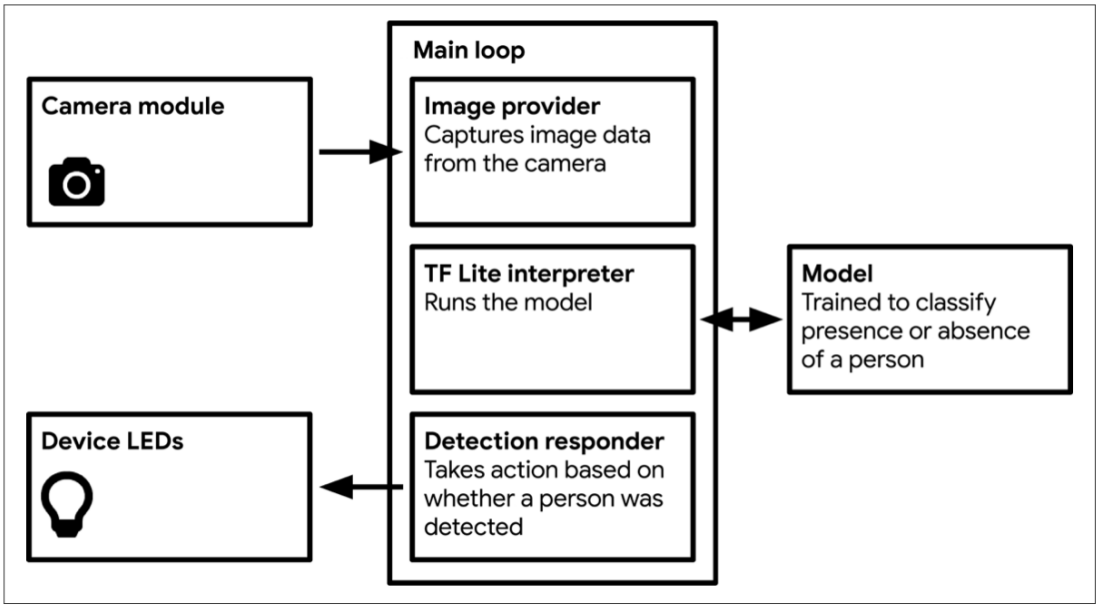
3. Method

3.1 System Architecture:

To implement the person detection system using tiny machine learning on an embedded device we need to below the sequence of tasks.

- **Obtain input data:** The input for the person detection system will be image data. Image data is an array of pixel values. The basic camera module of an embedded device should be capable enough to provide us with pixel arrays. As image data can be easily processed, we don't necessarily need images from the camera model to train, test and validate the model. For this case, we will use images from the COCO dataset (details of the data set is mentioned in the dataset section of the paper).

- **Process and feature extraction:** Image data from our selected database can be pre-processed with standard image processing techniques based on the input requirements on the model. In the embedded device this pre-processing can be done easily by cropping and reducing the pixel size based on the given input requirements.



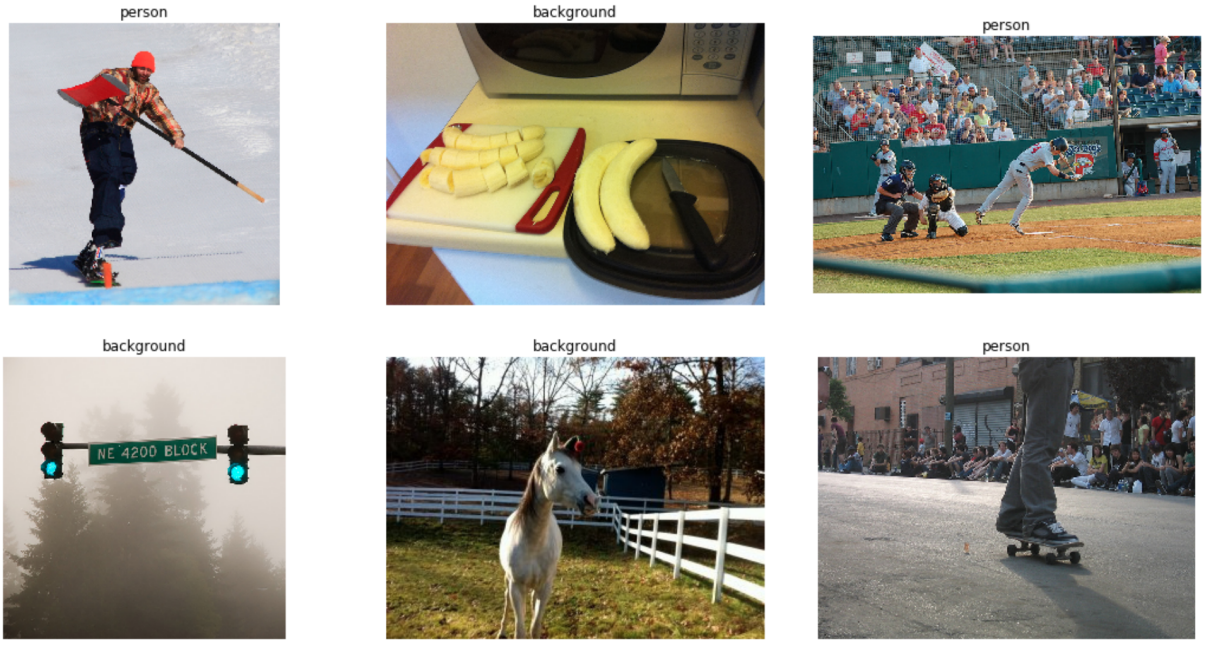
- **Run inference on the processed input:** Based on the input data model will classify the image in the device. This can be done based on time intervals or by an additional device triggering method like a motion sensor. For the testing purpose of the system, this part will be done in a converted model which runs in a similar environment by using the emulator.

- p**ost-process the model's output**: For a successful event, a trigger system is generally used so that only the required information is passed to the downstream system. Considering the scope of the study this part will be out of scope.

3.2. The Data set:

To train the person detection model, a large image collection is needed which is annotated, preferably with boundary boxes. For this, a version of the popular COCO dataset has been used for this research. The main COCO dataset does not come with the person and non-person labels which is crucial for training this model. To overcome this a version of the dataset is used known as the Visual Wake Words dataset [ ]. The dataset is converted by Aakanksha Chowdhury. The size of the dataset is about 40 GB.

The version of the dataset can be found with the TensorFlow dataset where the labels are done as per the requirements of the target model. If an image contains any person the image is labelled as a person and if the image doesn't contain any person is marked as background. Below are some sample images from the Visual Wake Words dataset.



The used dataset is prepared based on the targeted use for Internet of Things (IoT) applications. The dataset represents a microcontroller vision use-case of identifying whether a person is in the image or not. The database is known to be a realistic benchmark for Tiny Vision models.

3.4 Model selection:

The main evaluation criteria for the model selection is using the matric which helps us identify which models identify a person from the given image the best. Considering the use case the accuracy matrics is suitable for this image classification task. While training, the train and the validating split of the dataset are being used. For model evaluation, the test dataset is used. This split dataset is not used during the training and is only being used in the test dataset.

However, the main object of the model selection is to find the model which performs the best after the model conversion for the embedded device. For this top models were selected based on the performance on the test dataset. Then the models are converted (details is mentioned in the next section) and retested with the test dataset. Statistical methods were used to find the best model which performs the best after the conversion.

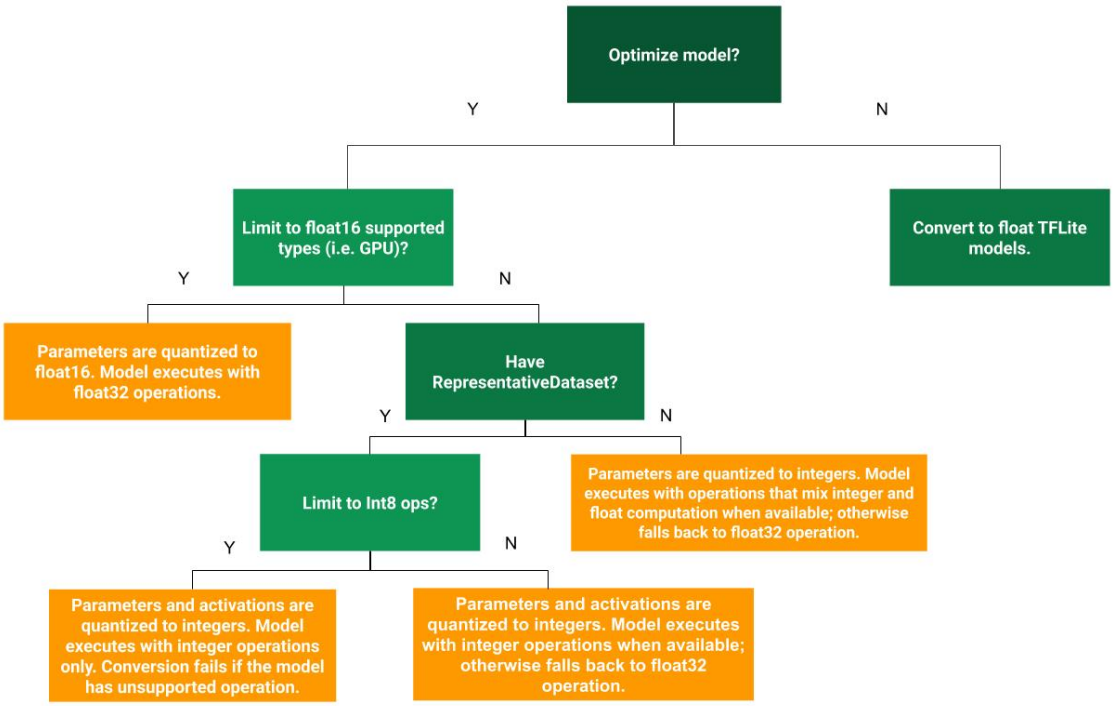
The target of the research as mentioned in the previous section is the compare the performance of the traditional neural network and TinyML. Hence for this paper performance, variable detection is more important than the actual performance of each model. The below model was picked up for the experiment. The selected model is a 9 layer convolution model which performed relatively well on the selected data set.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 248, 248, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 124, 124, 16)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 122, 122, 32)      4640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 61, 61, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 59, 59, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 29, 29, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 53824)             0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               27558400
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 1026
=================================================================
Total params: 27,583,010
Trainable params: 27,583,010
Non-trainable params: 0
```

3.5 Model conversion

There are several ways a model can be converted to a model suitable for microcontrollers. The two major approaches are Pre and Post-training quantization. As our target for this paper is to compare the performance of the traditional neural networks with TinyML neural networks, we first train the model in a regular method and then convert the model into embedded device suitable models. To convert our selected model we will follow below steps illustrated in the below image. The model conversion will be done on each step until

we get our desired model.



TensorFlow Lite Model: At first we can convert our trained model to TensorFlow Lite Model using TFLite Converter API and review the performance and size of the model. The TensorFlow Lite Model is simply a lither version of the model without any quantization. These types of models are suitable for handheld devices where memory, processing power are still significant. The outcome model will still be able to use 32-bit float values for all parameter data.

Optimized TFLite Float: The previous conversion doesn't do any quantization that means there is no model optimization. To to make the model smaller with quantization we can enable optimize features of TFLite. This feature allows the converter to estimate a range for all the variables. This optimization further reduce the size but the variable data is still in float format.

```
input:  <class 'numpy.float32'>
output:  <class 'numpy.float32'>
```

Optimized TFLite Int Model: So far all the variables (input and output) and all weights are quantized and which would make the model significantly smaller than the original one. However, because we want to implement the model in the embedded devices, the best way to quantize the model into integer values for all types of parameters. This conversion will not further reduce the size of the model, but the model can be used in microcontroller devices.

```
input:  <class 'numpy.uint8'>
output:  <class 'numpy.uint8'>
```

Model Saving & Emulator:
After the conversion of the model in each stage, the converted models are saved to a .tflite file so that this can be deployed to other devices and run in an interpreter to verify emulate the performance of desired devices. For this experiment, all the converted models are firsts saved and then run through the same emulator so that the performance testing can be done. Below is the input and output of the emulator also known as the TensorFlow interpreter.

```
<tensorflow.lite.python.interpreter.Interpreter at 0x7f4eed773a10>
```

Input Details:

```
{'dtype': numpy.uint8,
 'index': 0,
 'name': 'conv2d_6_input',
 'quantization': (0.003921568859368563, 0),
 'quantization_parameters': {'quantized_dimension': 0,
  'scales': array([0.00392157], dtype=float32),
  'zero_points': array([0], dtype=int32)},
 'shape': array([  1, 250, 250,   3], dtype=int32),
 'shape_signature': array([ -1, 250, 250,   3], dtype=int32),
 'sparsity_parameters': {}}
```

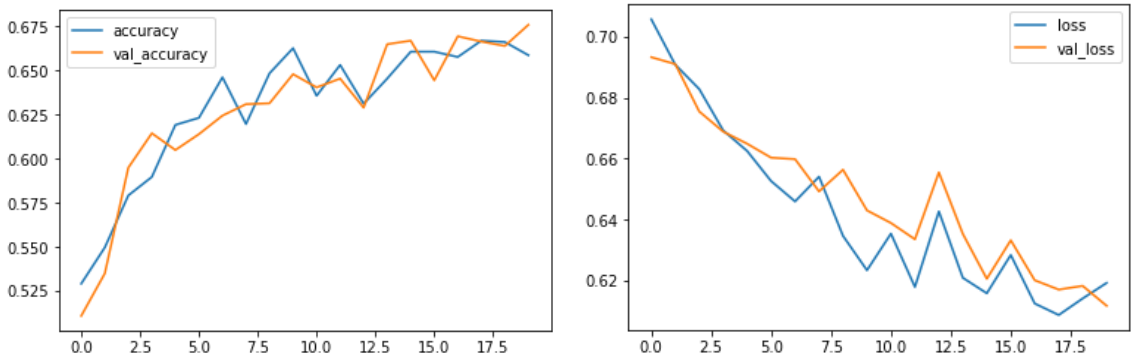
Output Details:

```
{'dtype': numpy.uint8,
 'index': 23,
 'name': 'Identity',
 'quantization': (0.00390625, 0),
 'quantization_parameters': {'quantized_dimension': 0,
  'scales': array([0.00390625], dtype=float32),
  'zero_points': array([0], dtype=int32)},
 'shape': array([1, 2], dtype=int32),
 'shape_signature': array([-1,  2], dtype=int32),
 'sparsity_parameters': {}}\
```

4. Results

a. Full Model Performance:

The selected convolution neural network was trained for 20 epochs for over 82.7K labelled images from the coco data set. The network performance of the network over the test data set is 68.00%.



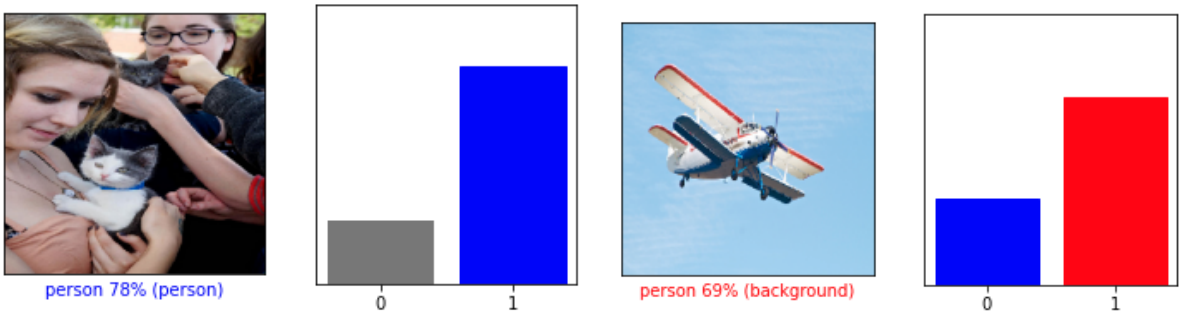
The above figures show the accuracy and loss during the training for the train and the validation dataset. The below is the final outcome of the model.

loss: 0.6193 - accuracy: 0.6587 - val_loss: 0.6117 - val_accuracy: 0.6760

The performances of the model over the test dataset for different sample size are given below.

```
Test accuracy: 0.6800000071525574 (50 samples)
Test accuracy: 0.6299999952316284 (100 samples)
Test accuracy: 0.6499999761581421 (1000 samples)
```

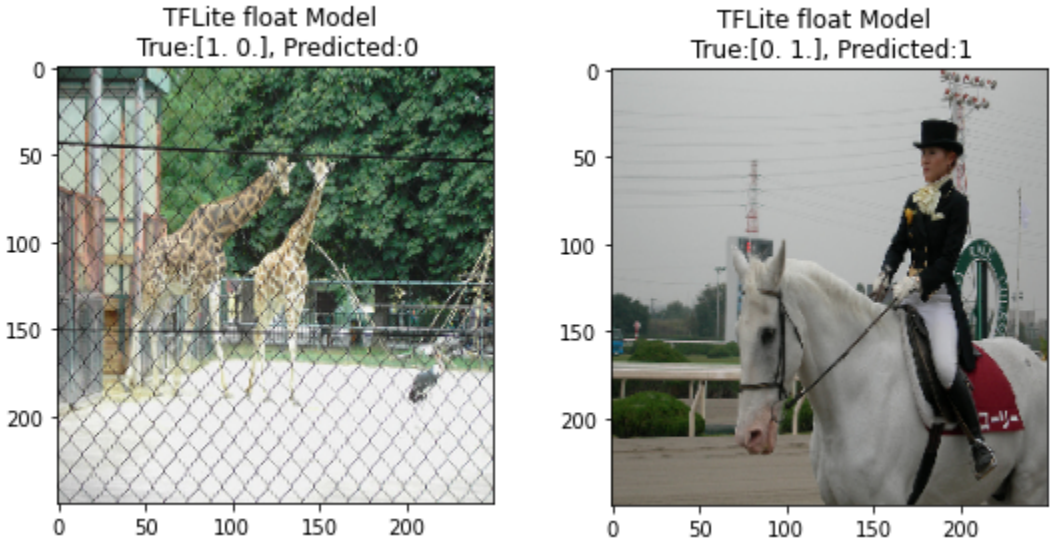
The below figure shows an example of the successful prediction and an example of unsuccessful prediction by the model. A sigmoid function was used in the last layer to get the probability. Hence the bars beside the images are showing the probability distribution of each label. Point to be noted that for integer converted model the final output would be simple 0 or 1 instead of the probability distribution.



b. Model Conversion Results:

TensorFlow Lite Model: After the Conversion of the model, we can see that the size of the model reduced from 320 MB to 105.22 MB. Which is less than one-third of the actual

model. And there is no performance reduction as the model is still using 32-bit float values for all parameter data. However, this is still very large for embedded devices.
The sample predictions of the model are shown below.

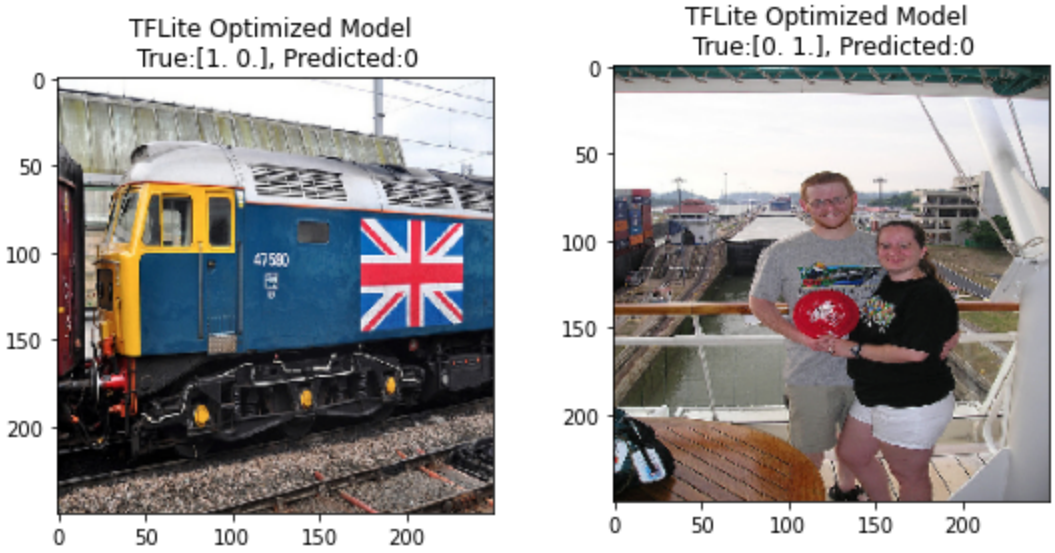


The performance of the TFLite model for different sample sizes is given below. As we can see the performance of the TFLite is almost equal to the main model.

```
TFLite model accuracy is 68.0000% (Number of test samples=50)
TFLite model accuracy is 64.0000% (Number of test samples=100)
TFLite model accuracy is 53.4000% (Number of test samples=1000)
```

Optimized TFLite Float: The TFLite converted model is not optimized. Hence even the size has reduced around one-third of the main model, this can be reduced further using optimization. With the TensorFlow default optimization function, we could further reduce the size of the model from 315.75 MB to 26.31 MB.

The sample predictions of the model are shown below.

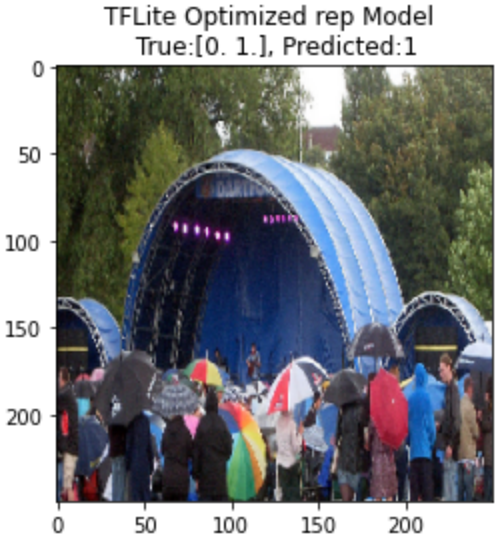


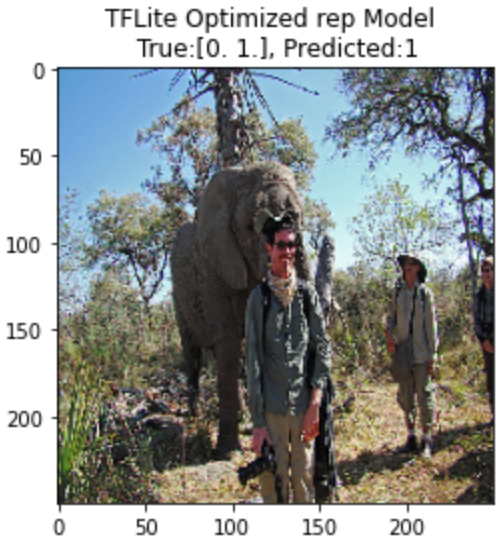
The performance of the TFLite optimized float model for different sample sizes is given below. As we can see the performance of this model is almost equal to the main model.

```
TFLite float model accuracy is 68.0000% (Number of test samples=50)
TFLite float model accuracy is 64.0000% (Number of test samples=100)
TFLite float model accuracy is 53.4000% (Number of test samples=1000)
```

Optimized TFLite Int Model: After optimization of the input and weights the size of the model does not reduce. However, as mentioned earlier in the experiment section, to test the model similar to the embedded device the model can be converted to the integer value.

The sample predictions of the model are shown below.





The performance of the TFLite optimized float model for different sample sizes is given below. As we can see the performance of this model is almost equal to the main model.

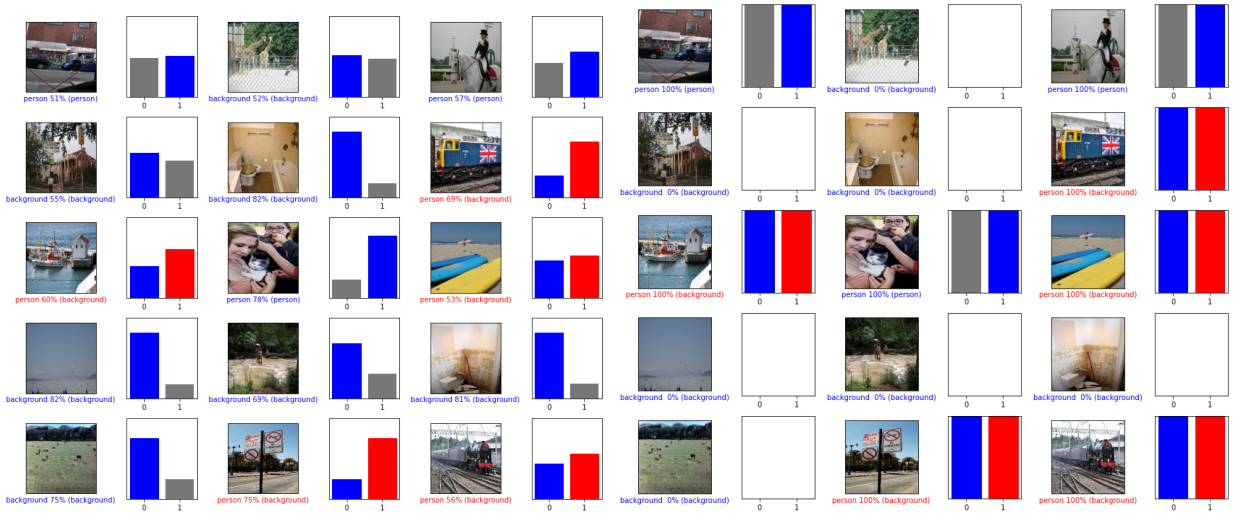
```
TFLite int model accuracy is 68.0000% (Number of test samples=50)
TFLite int model accuracy is 63.0000% (Number of test samples=100)
TFLite int model accuracy is 65.1000% (Number of test samples=1000)
```

c. Final Converted Model vs Actual model

To compare the performance of the main convolutional model with the optimized integer model suitable for the embedded devices are done over randomly selected samples. For the below verification to 50 samples randomly been taken and we found that the accuracy of the actual model and the Quantized model is the same.

```
actual model accuracy is 68.0000% (Number of test samples=50)
Quantized model accuracy is 68.0000% (Number of test samples=50)
```

From the below figures we can see that both the actual and TinyML model has performed identically for the randomly selected image samples. In the figure, the blue title means the model prediction is correct and for the red title, the model predicted wrong. We can observe that both the models have done the same wrong and right prediction over the dataset.

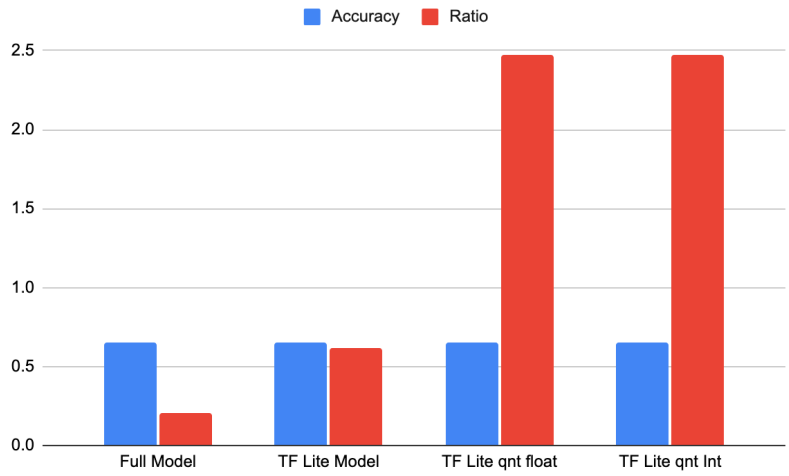


The below tables shows the performance of the different converted model in terms of size, accuracy and the ratio of success over the size of the model file. The comparison has been done for 50, 100 and 1000 samples.

Average Accuracy Over-Size Ratio 50 samples				
No	Model Name	Accuracy	Size (MB)	Ratio
1	Full Model	68.00%	315.71 MB	0.215388
2	TF Lite Model	68.00%	105.22 MB	0.646265
3	TF Lite qnt float	68.00%	26.31 MB	2.584569
4	TF Lite qnt Int	68.00%	26.31 MB	2.584569

Average Accuracy Over-Size Ratio - 100 Samples				
No	Model Name	Accuracy	Size (MB)	Ratio
1	Full Model	63.00%	315.71 MB	0.199550
2	TF Lite Model	63.00%	105.22 MB	0.598745
3	TF Lite qnt float	63.00%	26.31 MB	2.394527
4	TF Lite qnt Int	63.00%	26.31 MB	2.394527

Average Accuracy Over-Size Ratio -1000 Samples				
No	Model Name	Accuracy	Size (MB)	Ratio
1	Full Model	65.00%	315.71 MB	0.205885
2	TF Lite Model	65.00%	105.22 MB	0.617753
3	TF Lite qnt float	65.00%	26.31 MB	2.470544
4	TF Lite qnt Int	65.00%	26.31 MB	2.470544



From the below table we can see that for 1000 sample sizes only 5 mismatches between the models can be notified. That means the performance difference or error rate of the two models are only 0.005 or 0.5%.

Title	Value
No of Samples	1000
No of Miss Match	5
Error Ratio	0.005

d. Statistical Comperision

To further verify the performance difference a statistical model has been used. Given our outcome, a two paired t-testing would let us compare the two models.

Null Hypothesis, H0: The accuracy of the actual model would be better than the quantized tinyML model. That means p1 > p2,

Alternative Hypothesis, HA: The performance of the two models are the same. p1=p2

where, p1 = performance accuracy, p2 = performance accuracy of the TinyML model

```
        Paired t-test

data:  outcome$full and outcome$tflite_int
t = 0.44703, df = 999, p-value = 0.6549
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.003389685  0.005389685
sample estimates:
mean of the differences
                  0.001
```

From the above paired t-test result, we can see that the p-value = 0.6549, which means the p-value is with the significant acceptance level. So we can reject the null hypothesis and confirm that the performance of the two models is the same.

5. Conclusion

The Tiny Machine Learning models over embedded devices have lots of real-life applications where the new solution can develop over this technology which will be cost-effective, secure and less power-hungry. However, the main question remains if the technology is up to the mark to compete with the current solution. The objective of this paper is to compare the technology where person detection using TinyML is tested.

From the results, this is quite evident that the performance of the tinyML model is equally good for the particular task. The result shows that for randomly selected 1000 samples only 0.5% miss-match between the actual model and the TinyML model.

6. Future Work

References

1. WARDEN, P. (2020). TINYML: Machine learning with TensorFlow on Arduino, and ultra-low Power micro-controllers. In TINYML: Machine learning with TensorFlow on Arduino, and ultra-low power micro-controllers. O'REILLY MEDIA.
2.