Chair of Real-Time Computer Systems
Department of Electrical and Computer Engineering
Technical University of Munich

Research Internship with RCS@TUM

Implementation of tiny machine learning models on microcontrollers

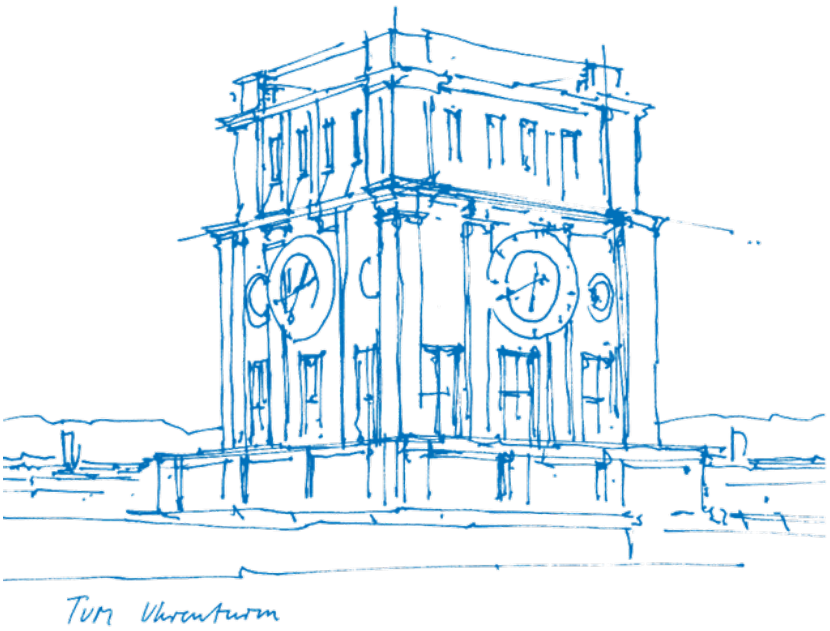
Philipp van Kempen
Chair of Real-Time Computer Systems
Department of Electrical and Computer Engineering

Research Internship Report
24.08.2020 - 25.10.2020

Supervised by
M.E, M.Sc Alex Hoffman
Chair of Real-Time Computer Systems

Author
Philipp van Kempen
Sepp-Manger-Str. 3b
85375 Neufahrn b. Freising

Submitted
03.12.2020



1 Introduction

Neural networks and artificial intelligence are omnipresent topics in today's society and economy. Examples can be found as computer vision implementations for the automotive industry and spam detection filters used by email providers.

1.1 Motivation

Machine Learning focuses on the development of data-dependent computer programs which are able to adapt themselves according to an optimization problem. While just a few decades ago only a small subset of electronic devices was able to run those compute-intensive algorithms, they became more and more relevant for data centers, personal computers and lastly portable devices like smartphones from time to time. Currently the exploitation of artificial models on edge devices like microcontrollers with memory capacities as low as just a couple of 100 kilobytes is a huge research topic. This trend can be also seen in the Gartner Hype Cycles, which are a graphical depiction of a common pattern that arises with each new technology or other innovation. In the release for 2020, shown in figure 1.1, AI-related topics and especially 'Embedded AI' are dominating the field of technologies.

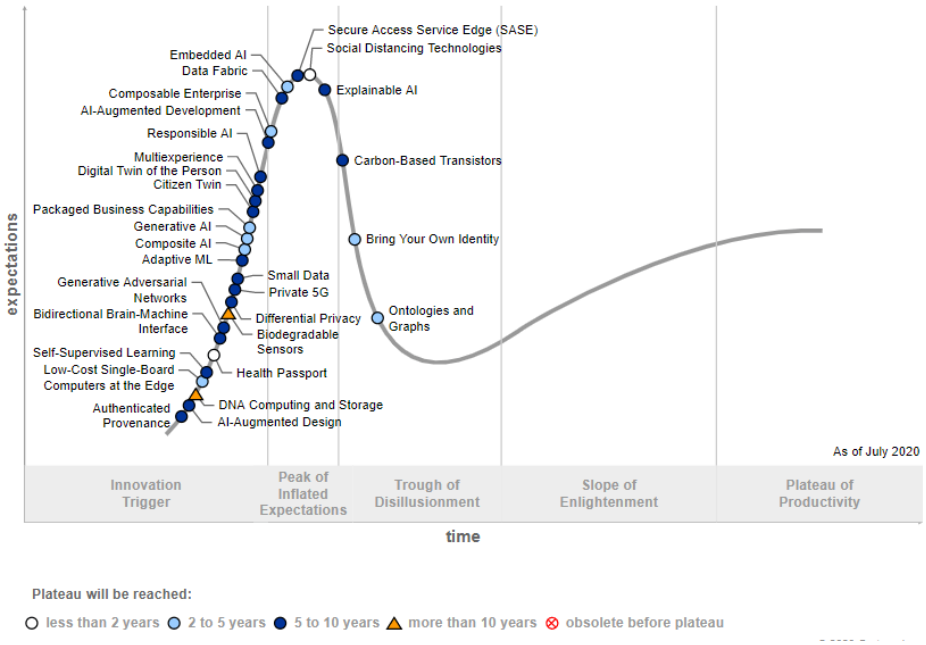


Figure 1.1 Gartner Hype Cycle for 2020 [1]

1.2 Goals

In the 9 weeks long research internship at the research group for Electronic System Level (ESL) applications founded by the chair of Electronic Design Automation (EDA) and the chair of Real-time Computer Systems (RCS), reference implementations for running TinyML[1]-Models on STM32 Evaluation Boards had to be designed. While an initial open-source toolchain for the project already existed, it had to be extended with essential features and extensive documentation.

[1] https://www.tinyml.org/summit/

2 Steps

About a third of the time during the research internship was spent studying literature on the topic. The TinyML book [2] was the most helpful but web research was also omnipresent.

Before getting the actual examples running, the outdated toolchain had to be modified as it was written for comparatively old versions of Tensorflow. To make this step a little bit easier, the official ARM Mbed [1] was regularly used as an reference.

Another important part of the process was understanding the provided training scripts. This also involved running slightly modified versions of those scripts to tune the resulting accuracy and model sizes.

The implementation of the hello_world example was trivial as it wokrd out of the box on the STM32 Hardware. The more complex micro_speech demo was not working at all. While mnist had similar problems it was done way faster because the findings of the previous step. Time was mostly spent for improving the usability by adding special features.

The last essential part of the project was writing the documentation. With the help of another student, it was done in approximately 3 weeks at the end of the FP period.

The time needed for completing the tasks over the duration of the internship is depicted in figure 2.1.

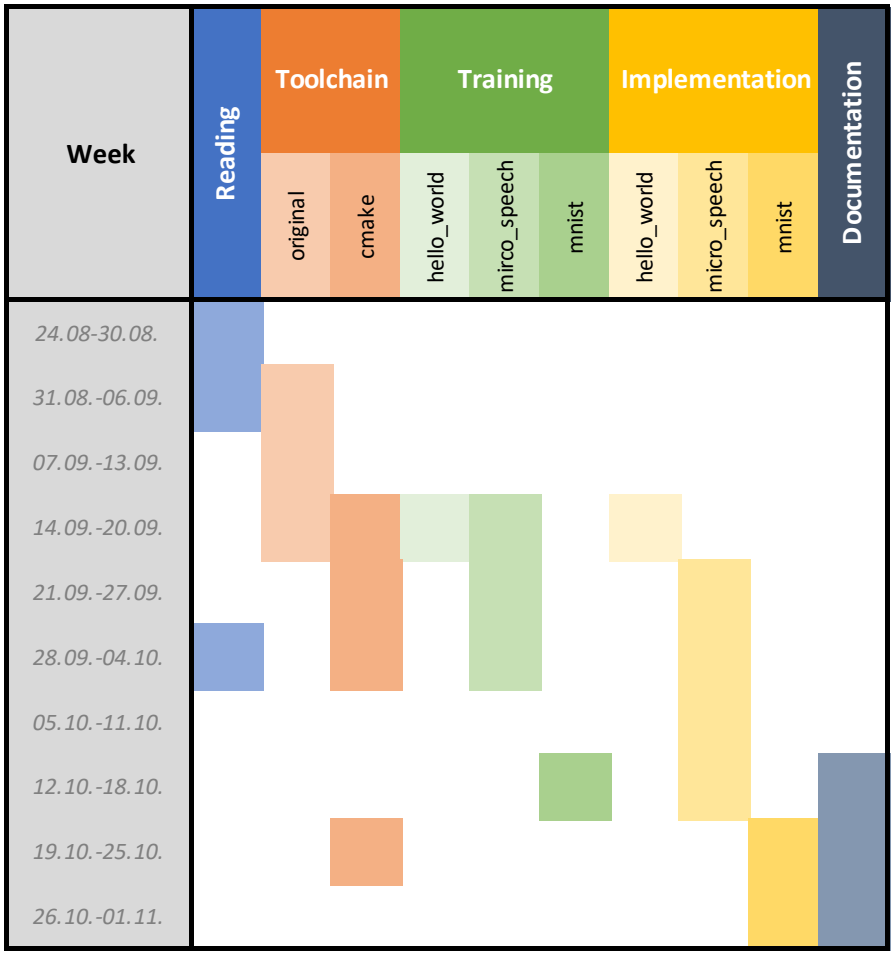


Figure 2.1 Schedule

[1] `https://os.mbed.com`

2.1 Reading

The book by Pete Warden and Daniel Situnayake introduces the usage of the Tensorflow Lite for Microcontrollers (TFLM) framework in a very detailed way. Two of the practical applications, which are all referencing examples located in the Tensorflow source tree, are especially interesting as they are supported a a large number of platforms:

- **Hello World:** A very simple model designed to approximate the value in the trigonometric sine function for a given x-value. As the memory and performance requirements for running this program are very low, it is often used to teach the basics of machine learning.
- **Micro Speech:** Inspired by voice assistants found in devices like smartphones, TVs,... which are able to detect full sentences of human speech via a microphone, this model can classify the voice commands 'yes' and 'no'. Those models, which can only detect single words are often referred as Keyword-Spotting (KWS) models. Running those models on low-power MCUs in embedded devices allows to wake up a larger processor if certain commands are detected as there accuracy alone is often not good enough for reliable results.

In addition to the book, the official documentation of Tensorflow was a great resource to git started with the topic. It was especially helpful as it contains information for old the latest versions 2.3 of the framework as well as for Tensorflow 1.5 which is still used often in many older models.

2.2 Toolchain

The target devices for the implementations are STM32 Evaluation Boards like the following:

(a) STM32F413H-DISCOVERY[a]

(b) STM32F769I-DISCOVERY[b]

[a] https://www.st.com/en/evaluation-tools/32f413hdiscovery.html

[b] https://www.st.com/en/evaluation-tools/32f769idiscovery.html

Figure 2.2 Target Boards

The CMake-based toolchain for those boards was initially developed by Konstantin Oblaukhov [2] and further improved by Alex Hoffman in a reference application [3].

First, the CMake Modules had to be patched to support the components on the mentioned boards (LCD, Touchscreen, Microphone,...). With the example demonstrations from the official manufacturer firmware [4] it was possible to create demo applications making use of these peripherals.

To integrate The TFLite Micro library into those embedded applications several changes and workarounds have been required, especially to the the optimized CMSIS-NN kernels running.

The fact that the contents of the TF repository change in a very frequent manner, turned out to be an huge issue as to broke the build system from time to time. Many adaptions were required to fix these issues when they came up.

[2] https://github.com/ObKo/stm32-cmake

[3] https://github.com/alxhoff/STM3240G-EVAL-TensorFlow-MNIST

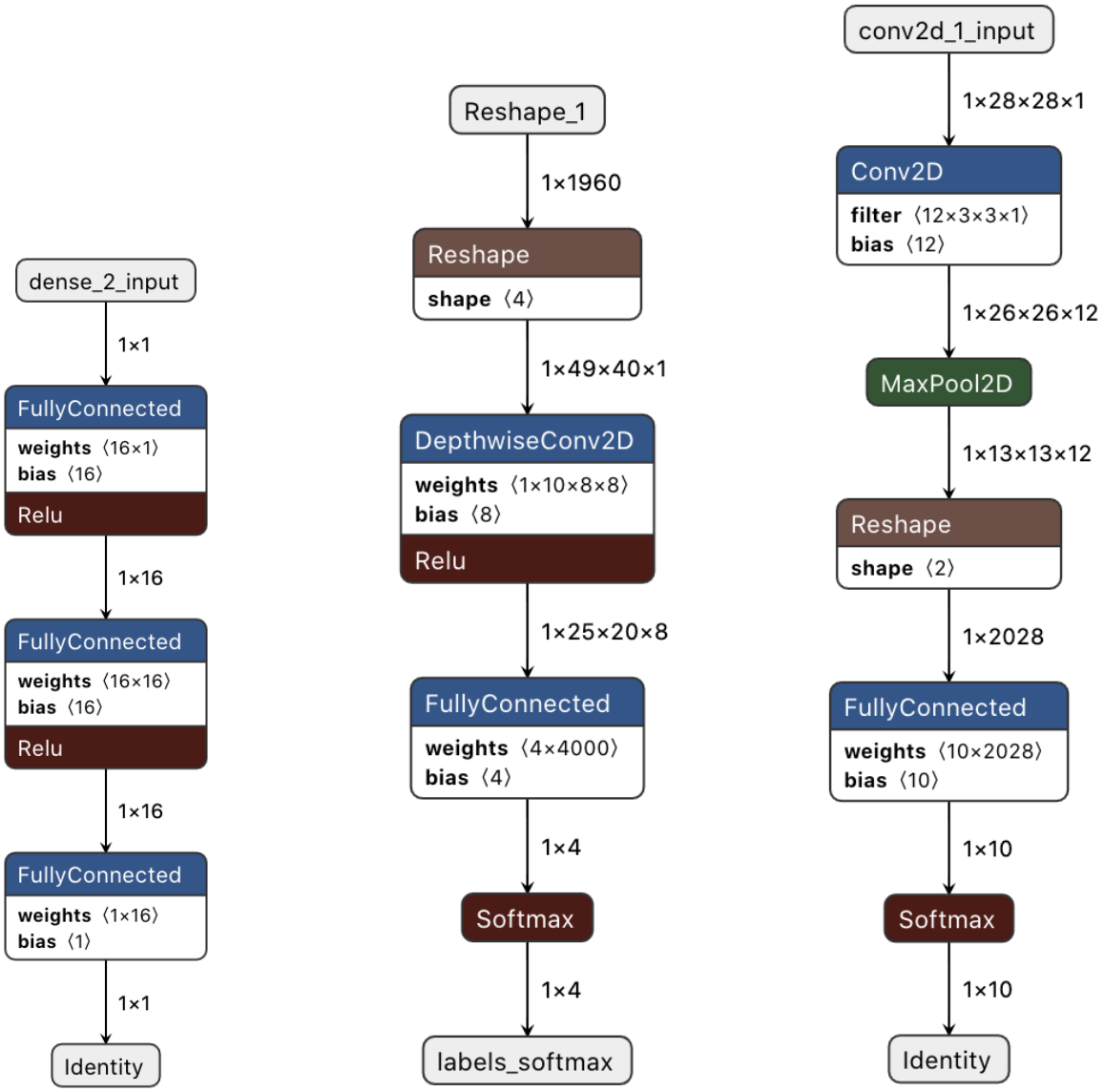
[4] https://github.com/STMicroelectronics/STM32CubeF7

2.3 Training

Fortunately all the required training steps can be performed in an interactive notebook hosted on Google Colaboratory [5]. Even GPU-intense workload can run for free on machines provided by Google to speed up the training time.

A very important step to deploy the trained networks on a microcontroller is the model size reduction by quantization and other optimization. The conversion step also ensures that only TFLite Micro compatible operations can be used. The list of kernels to run those operations is quite short at the moment but it gets new elements from time to time.

In figure 2.3 the network graphs for all the examples after the conversion to TFLite is shown.



(a) Hello World **(b)** Micro Speech **(c)** MNIST

Figure 2.3 TFLite Model Graphs per Example

[5] https://colab.research.google.com

2.4 Implementation

2.4.1 Hello World

Most of the Hello World Demo was running out of the box after the toolchain was updated to be compatible with the latest tensor flow version.

Shortly the first example was running an initial version of a custom benchmarking module was added to the source code which allows to extract execution times of several parts of the main program loop.

2.4.2 Micro Speech

The hardest challenge of the Micro Speech example was streaming the input samples for the network in real-time which requires a sufficient execution speed of the four main processes:

- **Feature Generation:** Uses quite large ring-buffers to fit multiple time slots. If the main loop takes to long for one iteration, the buffer size has to be increased to prevent an overflow. Spectrogram-slices are generated with the new samples.
- **Invoke Network:** A set of spectrogram slices is feed into the the network input. The quantized output is an array of signed chars representing a probability of the four classes `yes,no,unknown,silence`
- **Post processing Results:** To reduce false positives and handle fluctuations a moving average has to be applied to the history of network outputs. Combined with thresholds and other parameters the detected class of command with the respective probability is returned
- **Command Responding:** If there is a new command detected, the result is displayed on the screen and/or printed on the serial terminal.

After the microphone functionality was added and the buffer size tuned the model was not responsive at all. It was unclear if the reason for this issue was a to slow processor but at this point the development was shifted to the faster F7 Board. In the meantime, by enabling the support of CMSIS-NN the execution time of the Invoke step was drastically reduced resulting in a higher sample throughput.

The final step which was required to achieve meaningful results with the application was reducing thresholds and tuning timing constants an the post processing step. In fact, this lead to more wrong detections but at least allowed to detect some correct samples in the end.

To find out if the rather bad performance of the network was cause by the microphone samples being too different compared to the original data set much time was invested adding the ability to provide samples from WAV-files via the on-board SD-card slot. As the manufacturer libraries for the audio codecs did not support playing mono audio. The original 1 second WAV files had to be converted to stereo files beforehand with a custom script.

2.4.3 MNIST

The well-known MNIST dataset contains ten thousands of hand written digits. They have a quite small resolution and only feature grayscale values to reduce the size and therefore the complexity of the samples.

To use feed digits written via a touchscreen into the network some prepossessing is needed:

- Find a suitable brush size (circle radius) for the input
- Interpolate between inputs to get a smoother line
- Downscale the input image to 28x28 pixels with a proper filter
- Convert the color space of the display to 8 bpp grayscale

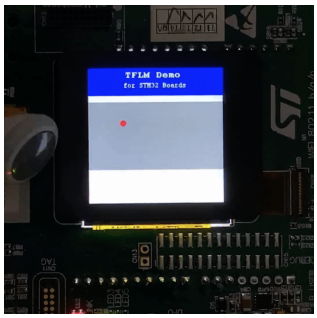
- Invert the color spectrum (whit on black instead of black on white)

Similar to the MNIST implementation SD-Card support for feeding original samples instead of touchscreen data was also added. Scripts to generate board-compatible BMP files are provides with the source code.

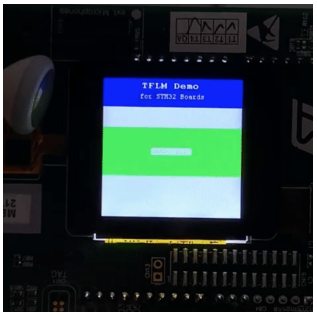
While some of the predictions have been very accurate the network as not capable some digits neither with original samples nor with touchscreen inputs. This leads to the conclusion that the model architecture needs some further tuning.

As the TFLite Micro currently does not support unsigned 8 bit quantization the training script had to be modified to use signed values between -128 and 127. This fact also had to be considered when preparing the input tensor for the network.

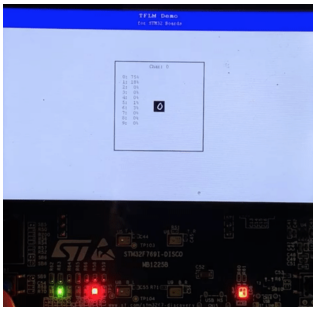
Figure 2.4 shows the examples in running on the actual hardware. Animated versions of these graphics can be found in the respective repository!



(a) Hello World



(b) Micro Speech



(c) MNIST

Figure 2.4 Examples running on the STM32 Boards

2.5 Documentation

To make the code easier to maintain, the common toolchain parts have been outsourced to submodules. A wrapper repository was added to provide links to all actual examples and to provide overarching documentation. This reduced the amount of redundant documentation by only providing example specific explanations together with the submodules. The resulting project structure is easy extendable and can be found on GitHub[6].

[6] `https://github.com/PhilippvK/stm32-tflm-demos`

3 Evaluation

3.1 Board Metrics

Before reviewing the collected metrics of the models running on the hardware platform, the main properties of both evaluation boards are shown in the following table. Its not hard to tell that the F7 board features a much faster MCU than the F4 counterpart while both of them are quite good in terms of available memory resources.

	Boards		
Metrics	STM32F413HDISCOVERY	STM32F769IDISCOVERY	**Units**
Clock Frequency	100	216	*MHz*
Special Features	-	Double Issue, I/D-Cache	-
Flash Memory	1.5	2	*MB*
SRAM Memory	256	512	*kB*

3.2 Memory Usage

See the following table for approximate Flash and RAM usage

	Examples			
Type	hello_world	mirco_speech	mnist	**Units**
Model Size (FLASH)	2	18	23	*kB*
TensorArena Size (SRAM)	1	7	11	*kB*

According to a paper[3], the approximated Memory requirements for TFLite Micro are the following:

- SRAM: 4 kB
- FLASH: 37 kB

3.3 Number of Ops

The FLOPS (not FLOP/s) of the model graph should give an idea on how many additions and multiplications are needed for a single invocation on the network. The stated values are highly approximatively as they do not take every TF operation into account and are generated before quantization on the eras models itself.

Examples		
hello_world	mirco_speech	mnist
41 *FLOPS*	689980 *FLOPS*	202810 *FLOPS*

3.4 Runtime Measurements

The time spend in different sections on the main program `loop()` is analyzed quantitatively in the following section. The results are heavily influenced by the implementation of the network, as interrupts, serial communication and I/O-activity may be present during the measurements. Therefore and because of the reasons mentioned above, the relationship between the FLOPS and the invocation time is not linear.

All values have been collected by averaging over 1 minute of runtime with the CMSIS-NN kernels enabled.

The program loop was splitted in the following intervals for all 3 examples: `POPULATE/INVOKE/RESPOND`

		Examples			
Section	**CPU**	hello_world	mirco_speech	mnist	**Units**
Populate	F4	∼0	38	132	*ms*
	F7	∼0	11	88	
Invoke	F4	∼0	49	34	*ms*
	F7	∼0	52	13	
Respond	F4	∼0	∼0	125	*ms*
	F7	∼0	∼0	93	

3.5 CMSIS-NN Optimizations

In the following, the differences in execution time for running the model with the cmsis-nn kernels turned ON/OFF are compared briefly. The extend of the improvement depends on the used operations in the graph because only the following kernels are currently supported:

- `add`
- `conv`
- `depthwise_conv`
- `fully_connected`
- `mul`
- `pooling`
- `softmax`
- `svdf`

Notice: The values for the STM32F413HDISCOVERY are omitted here as they relatively similar compared to the F7 values.

		Examples			
Settings		hello_world	mirco_speech	mnist	**Units**
CMSIS_NN	OFF	∼1	413[1]	52	*ms*
	ON	∼0	52	13	*ms*
Difference		(?)	(-87)	(-75)	*%*

3.6 Details on Results

The previously presented results can also be found in the repos documentation alongside with a short manual on how to obtain those numbers. See: `https://github.com/PhilippvK/stm32-tflm-demos/blob/main/docs/Metrics.md`

4 Conclusions

4.1 Summary

While the goal of the research internship was to implement two of the examples from the TinyML book on real hardware, it was possible to get the MNIST example (which is not covered in the book) running as well. In terms of model performance, further work is required but the current implementation suffices as a demonstration to show the capabilities of TinyML on respective microcontroller platforms. As the work on this topic is definitely not done, a short outlook is presented in the following section.

4.2 Outlook

Here is a list of task or extensions which could improve the project even more in the future:

- Support USB-Storage as alternative to SD-Card
- Add TinyFace Model[4]
- Merge Deployment and Evaluation Flow with EDA RISCV-toolchain
- Enable usage of FreeRTOS instead of baremetal
- Add possibility to parse results via UART
- Feed samples via UART for automated testing

  $\Rightarrow$ Extract data on the accuracy of the models

Bibliography

[1] G. Inc., “Gartner hype cycle,” 2020. [Online]. Available: https://www.gartner.com/smarterwithgartner/5-trends-drive-the-gartner-hype-cycle-for-emerging-technologies-2020/

[2] P. Warden and D. Situnayake, *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O’Reilly Media, Incorporated, 2020.

[3] C. Banbury, C. Zhou, I. Fedorov, R. M. Navarro, U. Thakkar, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough, “Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers,” *arXiv preprint arXiv:2010.11267*, 2020.

[4] T. Fratiloiu, “Tinyface: Extreme edge face detection on embedded devices,” 2020, https://github.com/munober/thesis/blob/master/digital_edition.pdf.