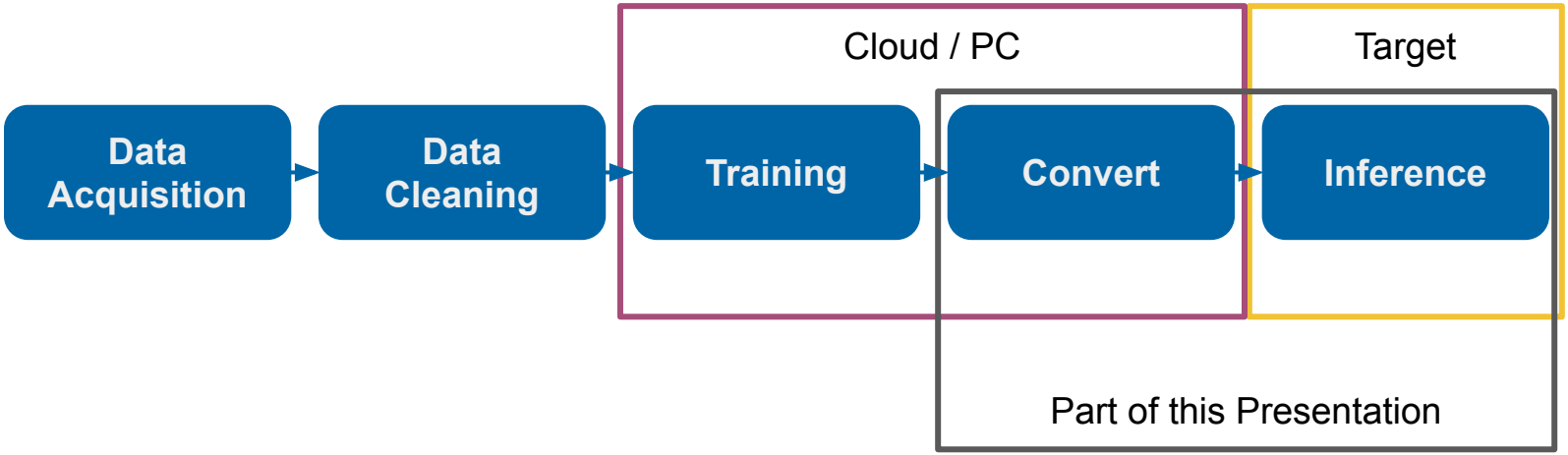
Benchmarking the Intelligent Edge

Table of Contents

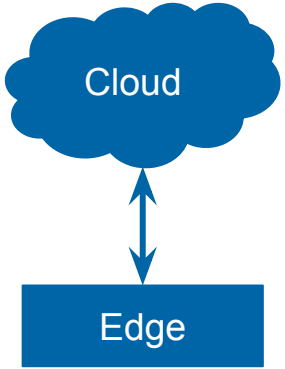
Neural Network - Workflow

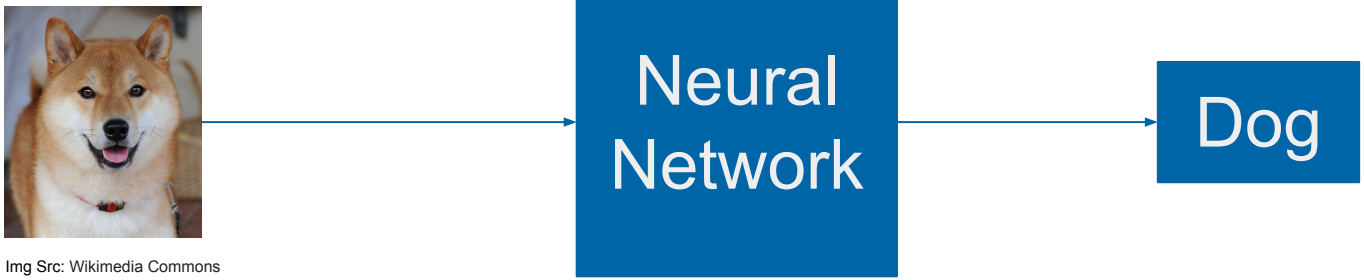
- First step is always about the data
- Frameworks for training include
  - MXNET
  - PyTorch
  - Caffe
  - **TensorFlow**
- Available conversion tools dependent on training framework chosen



Neural Networks on the Edge

- Training best done on powerful system
- Inference is interesting on the edge
  - Inference is the name for running the neural network
  - Edge includes PCs, mobile phones, **Embedded Systems**
- Inference on the edge reduces raw data transmitted
  - Tasks of neural networks are mostly to classify or identify
    → a label is smaller than an image
- Inference on the edge can reduce latency
  - Computing in the cloud requires a standing internet connection

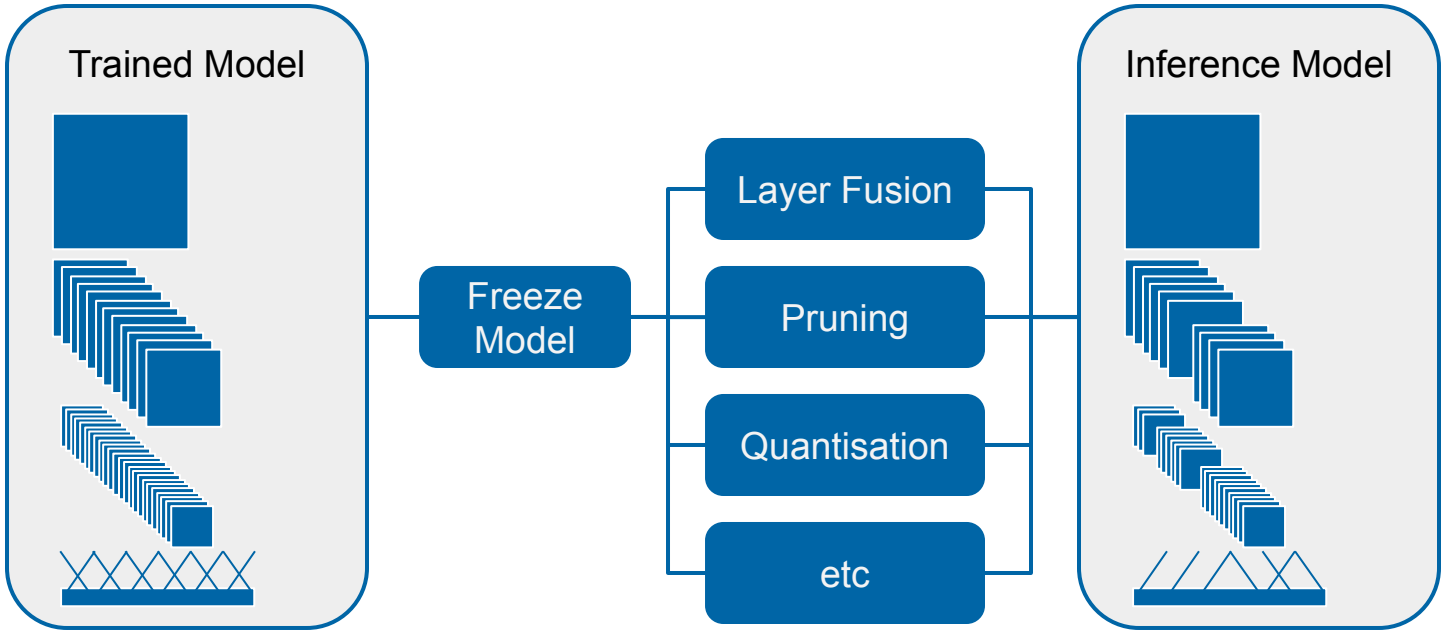




Img Src: Wikimedia Commons

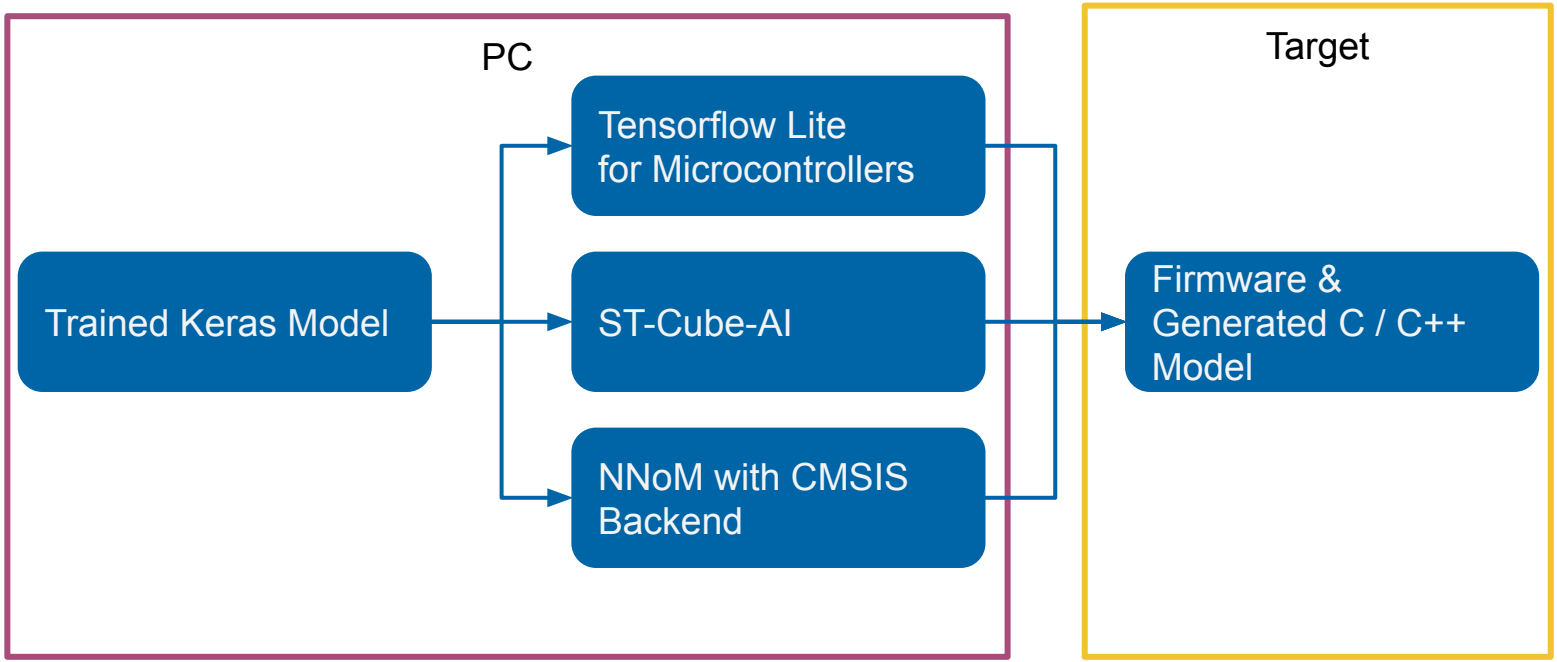
Convert a Model for Inference

- Optimize Model for Inference
  - Freeze Model → Remove training layers & elements
  - Layer Fusion → Pipeline computations instead of data
  - Quantisation → Change data types
  - Pruning → Remove unnecessary neural connections (computations)



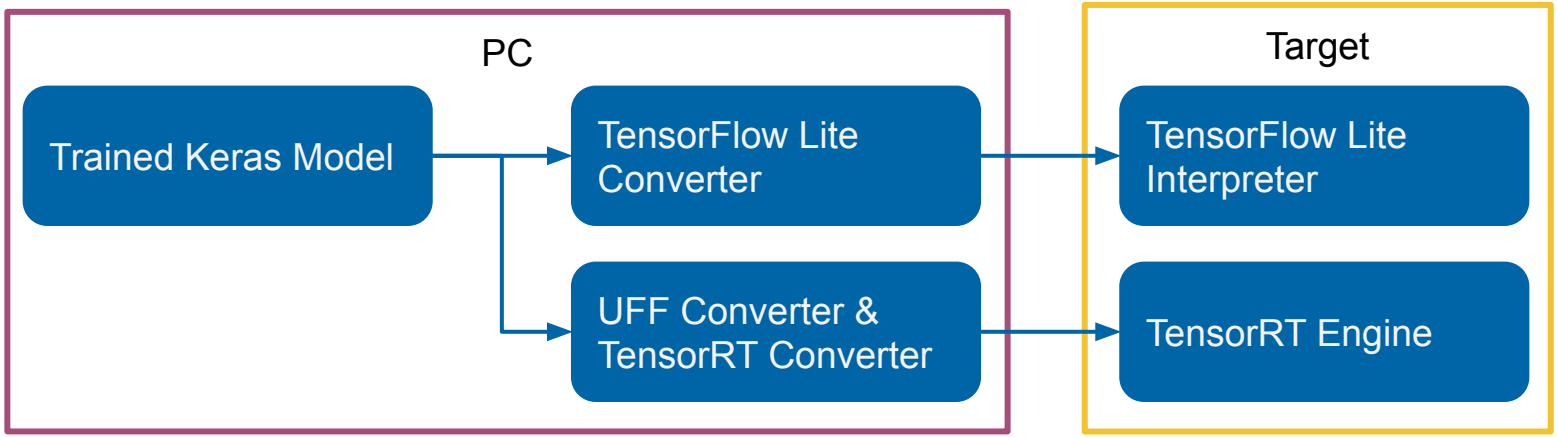
Running Inference - Microcontroller

- Multiple frameworks tested
- Generated C Code Model is used with Testing Firmware
- Time measurements using GPIO
  - Median of 10’000 samples



Running Inference - NVIDIA

- Tensorflow Lite used to convert to inference models
  - TensorFlow integration for TensorRT is in development (tensorflow.tensorrt)
- Time measurement using Linux-Time function
  - Median of 10’000 samples
  - Measurements done using TensorFlow Lite
- Target must have appropriate libraries



Hardware

- STM32F4DISCOVERY
  - Arm Cortex M4
  - 192 Kbyte RAM
  - 2Mbyte Flash
- NVIDIA Nano
  - GPU - 128-core Maxwell - 10W Max. Power consumption (Whole System)
- NVIDIA Jetson TX2
  - GPU - 256-core Pascal - 15W Max. Power consumption (Whole System)
- NVIDIA Xavier
  - GPU - 512-core Volta - 30W Max. Power consumption (Whole System)
- NVIDIA Quadro K620
  - GPU - 384-core Maxwell Architecture - 45W Max. Power consumption (GPU Alone)
  - 2GB DDR3 VRAM
  - 16GB RAM
  - Intel Xeon E3-1270V5

Example 1: Dense1 - Setup

- MNIST Data
- Easiest possible network
  - 1 Fully connected layer
  - 7’850 parameters
  - ~92% Accuracy

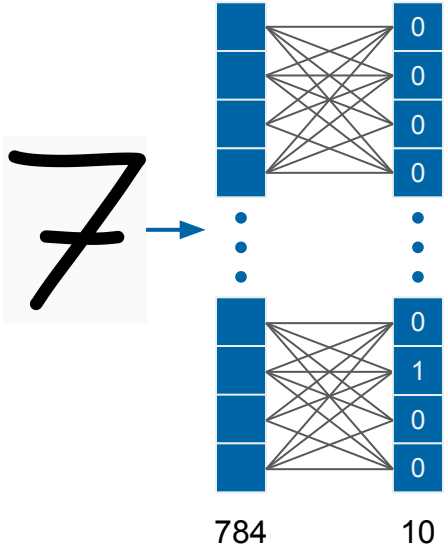
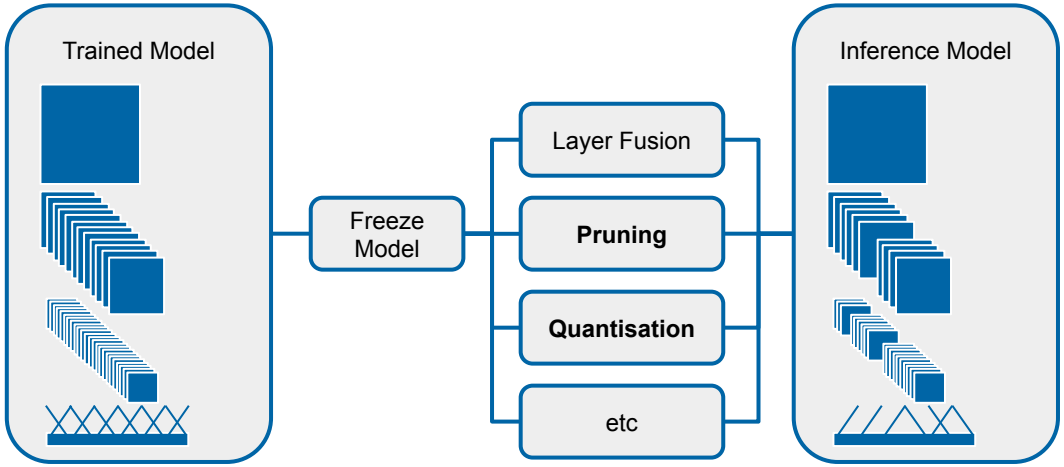


Image Source: Quora,
https://www.quora.com/What-is-MNIST, 17.04.2019

Dense 1 - Why use Inference Models

Hardware	Compiler	DataType	Median Time for 1 Sample in µs	Median Time for 1 Sample when using 32-Batch in µs
NVIDIA Quadro K620	None	float32	**13'909**	430
	TFLite	float32	**6**	3

- Performance gain from using converted model
- Smaller Memory footprint
  - Keras Model is 112KB
  - TFLite Model is 8.7KB



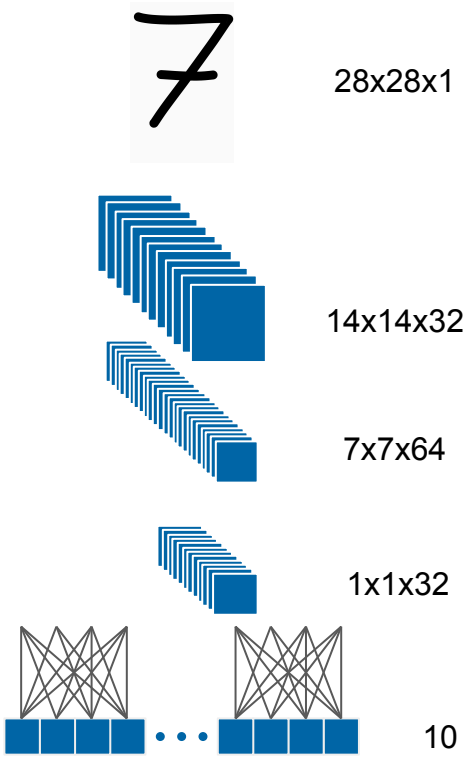
Dense 1 - Results

Hardware	Compiler	DataType	Memory of Firmware	Median Time for 1 Sample in µs	Median Time for 1 Sample when using 32-Batch in µs
STM32F4 Cortex M4	TFLite	int8	90Kbyte	1120	-
	ST-Cube-AI	float16	51Kbyte	**3950**	-
	nnom with cmsis	int8	26Kbyte	**207**	-
NVIDIA Nano	TFLite	float32		32	**10**
NVIDIA TX2	TFLite	float32		16	7
NVIDIA Xavier	TFLite	float32		10	**4**
NVIDIA Quadro K620	TFLite	float32		6	4
	TFLite	float16		5	1

- float16 for TFLite only in TF 1.15 (pre-release Nightly Builds)
  - NVIDIA Jetson boards are using TF 1.12
- int8 showed the same speed as float32 on NVIDIA devices
  - NVIDIA measured better results when using int8 with TensorRT
    src: NVIDIA AI Workshop at NeurIPS Expo 2018
- Accuracy change from int8 conversion < ± 1%

Example 2: Conv3Dense1 - Setup

- MNIST Data
- 3 Convolutions and 1 Dense
  - 119’530 Parameters
  - ~98% Accuracy



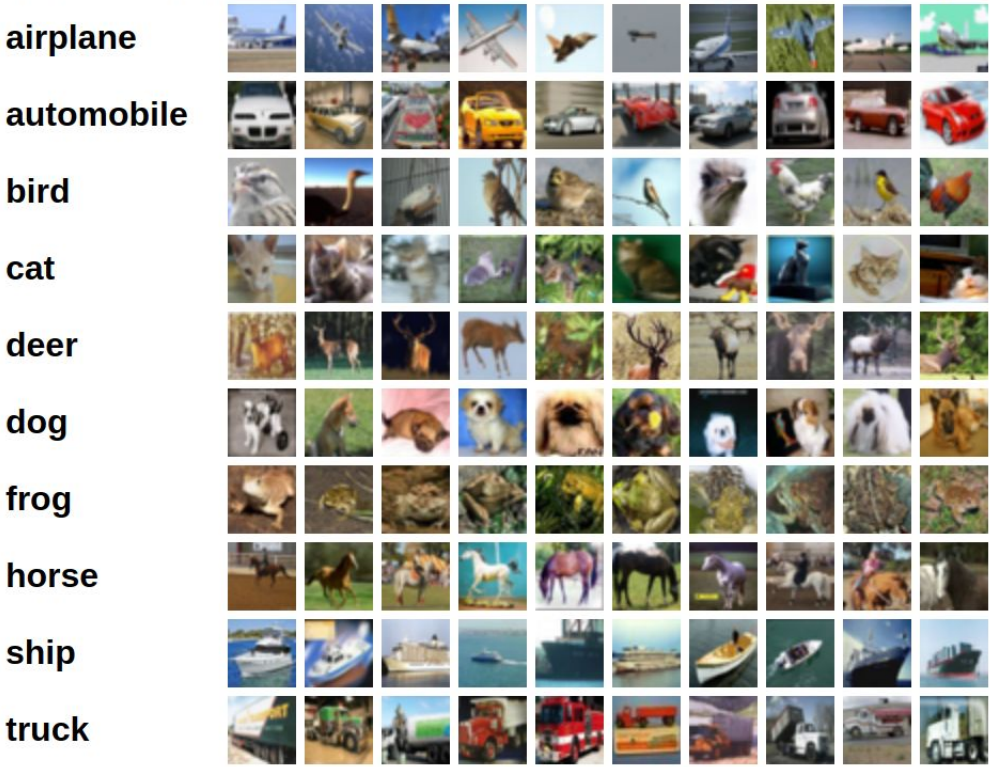
Conv3Dense1 - Results

Hardware	Compiler	DataType	Memory of Firmware	Median Time for 1 Sample in µs	Median Time for 1 Sample when using 32-Batch in µs
STM32F4 Cortex M4	TFLite	int8	203Kbyte	331'600	-
	ST-Cube-AI	float16	488Kbyte	**300'340**	-
	nnom with cmsis	int8	143Kbyte	**104'600**	-
NVIDIA Nano	TFLite	float32		564	**535**
NVIDIA TX2	TFLite	float32		404	388
NVIDIA Xavier	TFLite	float32		260	**259**
NVIDIA Quadro K620	TFLite	float32		**182**	**184**
	TFLite	float16		**80**	**30**

- Accuracy change from int8 conversion < ± 3%
  - nnom has highest accuracy loss due to fixed point arithmetics
- Massive Performance gain from using batch processing with float16
- NVIDIA Xavier again twice as fast as Nano

Example 3: MobileNet - Setup

- CIFAR10 Data
- 2’270’794 Parameters
- 157 Layers
- Keras Model Size 27MB
- TFLite file size 2.3MB



Screenshot of the CIFAR10 Dataset
image src: https://www.cs.toronto.edu/~kriz/cifar.html

MobileNet - Results

Hardware	Compiler	DataType	Median Time for 1 Sample in µs	Median Time for 1 Sample when using 32-Batch in µs
NVIDIA Nano	TFLite	float32	4’916	**4453**
NVIDIA TX2	TFLite	float32	3’600	3210
NVIDIA Xavier	TFLite	float32	2’205	**2079**
NVIDIA Quadro K620	TFLite	float32	1’200	1210
	TFLite	float16	1’100	**314**

- Too large for STM32F4DISCOVERY (flash size of 2MB)
- Confirms tests done with smaller networks
  - Xavier is always a bit more than twice as good as NVIDIA Nano

Available Benchmarks

- https://mlperf.org/
  - Performance of Hardware for Training and Inference
- http://ai-benchmark.com/
  - Benchmark for AI on Phones
- https://mlbench.github.io/
  - Benchmark for Machine Learning Implementations

Summary & Questions

- Microcontrollers
  - Good results for classification of small images or signals
  - Greatest limitation is available memory & RAM
  - Little accuracy loss when quantizing to int8
  - Accuracy loss can depend on trained weights
    - Weights influenced by random seed, data and model
- Embedded GPUs
  - Excellent performance in relation with power consumption
  - TensorFlow Lite easy to use
  - NVIDIA did show TensorRT increases performance even further than TensorFlow Lite

Any Questions?

Contacts

School of Engineering
InES Institute of Embedded Systems

- AI on Arm
  Raphael Zingg
  Institute of Embedded Systems, ZHAW
  High Performance Multimedia Group
  zing@zhaw.ch

- AI on NVIDIA
  Bruno Zimmermann
  Institute of Embedded Systems, ZHAW
  High Performance Multimedia Group
  zimb@zhaw.ch

Appendix - Preparing a Keras Model

School of Engineering
InES Institute of Embedded Systems

```
from tensorflow.keras.layers import Input, Conv2D, Flatten, Dense
x = Input(shape=inputShape)
conv1 = Conv2D(32, kernel_size=(3, 3), strides=(2, 2), padding='same', activation='relu')(x)
conv2 = Conv2D(64, kernel_size=(3, 3), strides=(2, 2), padding='same', activation='relu')(conv1)
conv3 = Conv2D(32, kernel_size=(int(inputShape[0]/4), int(inputShape[1]/4)), activation='relu')(conv2)
flat = Flatten()(conv3)
y = Dense(num_classes, activation='softmax')(flat)
model = tf.keras.Model(inputs=x, outputs=y)
model.compile()
# Training Dataset using tensorflow.data.Dataset
model.fit(trainingDataset,epochs=5,steps_per_epoch=10000)
model.save(keras_model_file)
```

Appendix - Convert Keras to TFLite

School of Engineering
InES Institute of Embedded Systems

```
import tensorflow as tf
converter = tf.lite.TFLiteConverter.from_keras_model_file(keras_model_file)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.lite.constants.FLOAT16]
tflite_model = converter.convert()
```

Appendix - Inference TFLite

School of Engineering
InES Institute of Embedded Systems

```
interpreter = tf.lite.Interpreter(modelFile)
interpreter.resize_tensor_input(
     interpreter.get_input_details()[0]['index'], inputTensorShape
)
interpreter.allocate_tensors()
for img in datasource:
    interpreter.set_tensor(interpreter.get_input_details()[0]['index'],img)
    t1 = time.time()
    interpreter.invoke()
    t2 = time.time()
    times.append(t2-t1)
    pred = interpreter.get_tensor(interpreter.get_output_details()[0]['index'])
```

Appendix - C Runtime measurement


```c
/* start time measurement */
HAL_GPIO_WritePin(GPIOB, ai_timing_measurement_Pin, GPIO_PIN_SET);

/* scale into float range [0, 1] */
for(i = 0; i < IMAGE_SIZE; i++)
  mnist[i] = ((float)inputPicture[i]) / 255.0;

/* run neural network to get a prediction of inputPicture */
pred = MX_X_CUBE_AI_Process(mnist);

/* stop time measurement */
HAL_GPIO_WritePin(GPIOB, ai_timing_measurement_Pin, GPIO_PIN_RESET);

/* return the prediction */
HAL_UART_Transmit(&huart4, &pred, 1U, 1000);
```

Appendix - C Runtime Measurement Scope

School of Engineering
InES Institute of Embedded Systems