ITMO UNIVERSITY

Saint Petersburg, Russia

Специализированные технологии машинного обучения / Advanced Machine learning Technologies

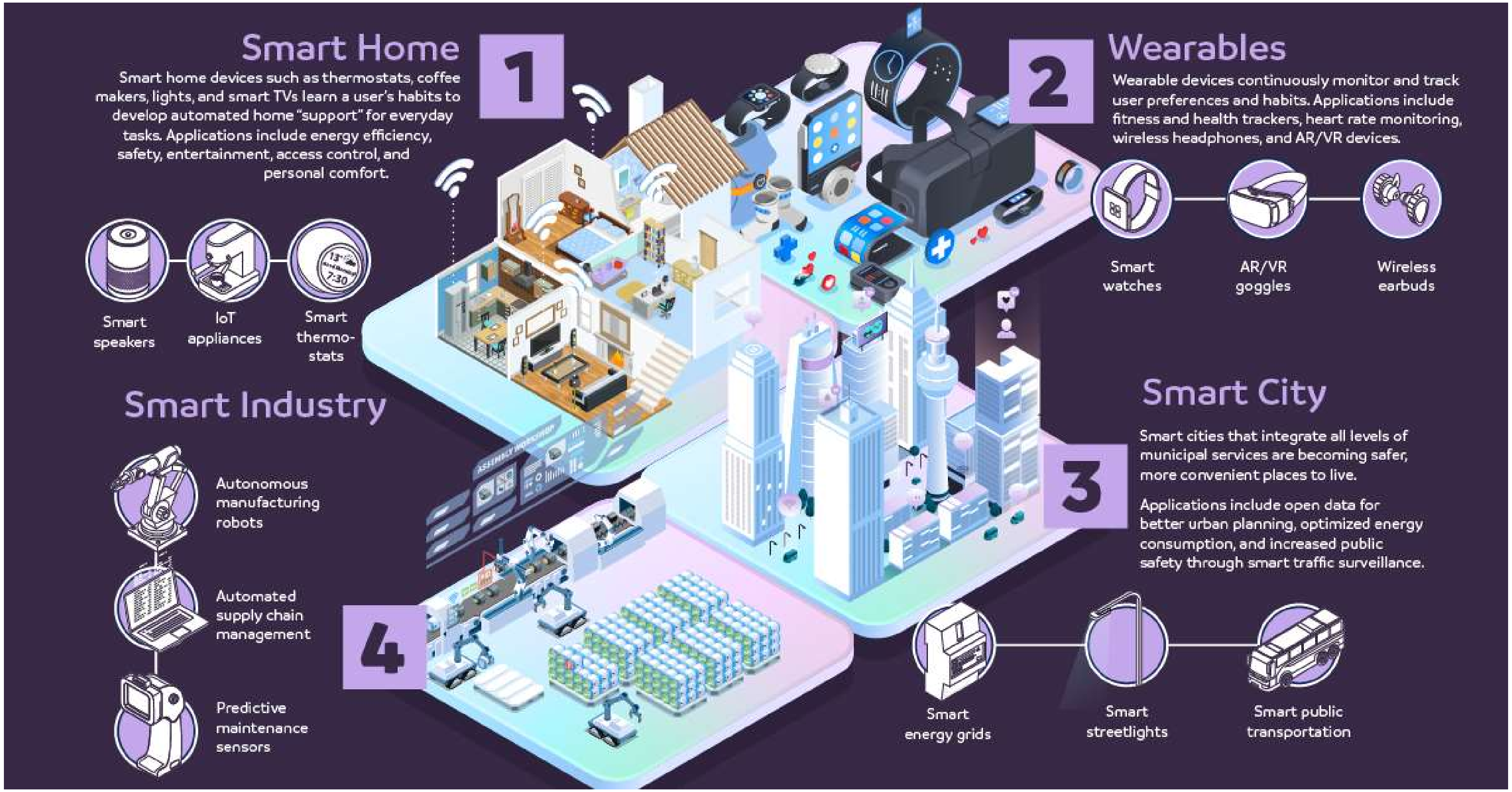
Lecture 8 – Tiny ML and Mobile AI

Outline

1. Overview Tiny ML and its goals
2. Real-world constraints for Inference of ML models
3. Quality vs. Complexity: models optimization techniques
4. Example: Once-for-all approach
5. Frameworks for TinyML
5. Conclusion

IoT, mobile AI and various edge devices

- IoT devices and embedded ML models are becoming increasingly ubiquitous in the modern world;
- There are more than 20 billion active devices by the end of 2020;
- During the last year (2021) Tiny ML technologies have become easier, more convenient, more powerful.



Training vs. Inference

Two stages in the world of neural networks:

- training
- inference

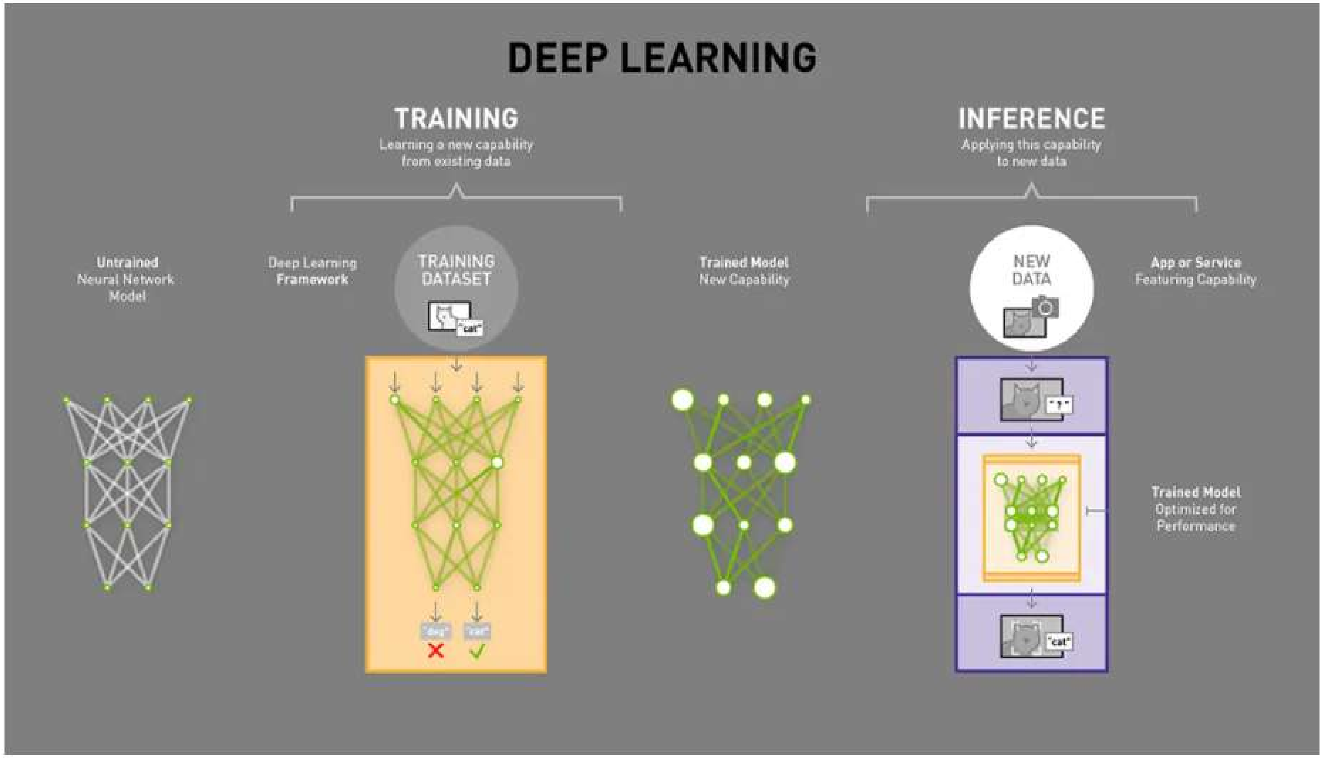
Training is more difficult. It requires more computational resources, the ability to monitor and analyze parameters, simple ways to integrate new layers and modify model.

There can exist training during inference (retraining) but it is not so often.

Actually, you can discard 90% of the math operations. This is **“inference”.**

Inference is faster and easier than training.
But: Inference can be on desktops, and on mobile phones, and on servers, and in the browser.
One need either special hardware and its support, or maximum utilization of the existing one.

Hardware is very different. This means that your module should be universal for the majority of existing.



https://habr.com/ru/company/recognitor/blog/524980/

Training vs. Inference

Ways of inference:

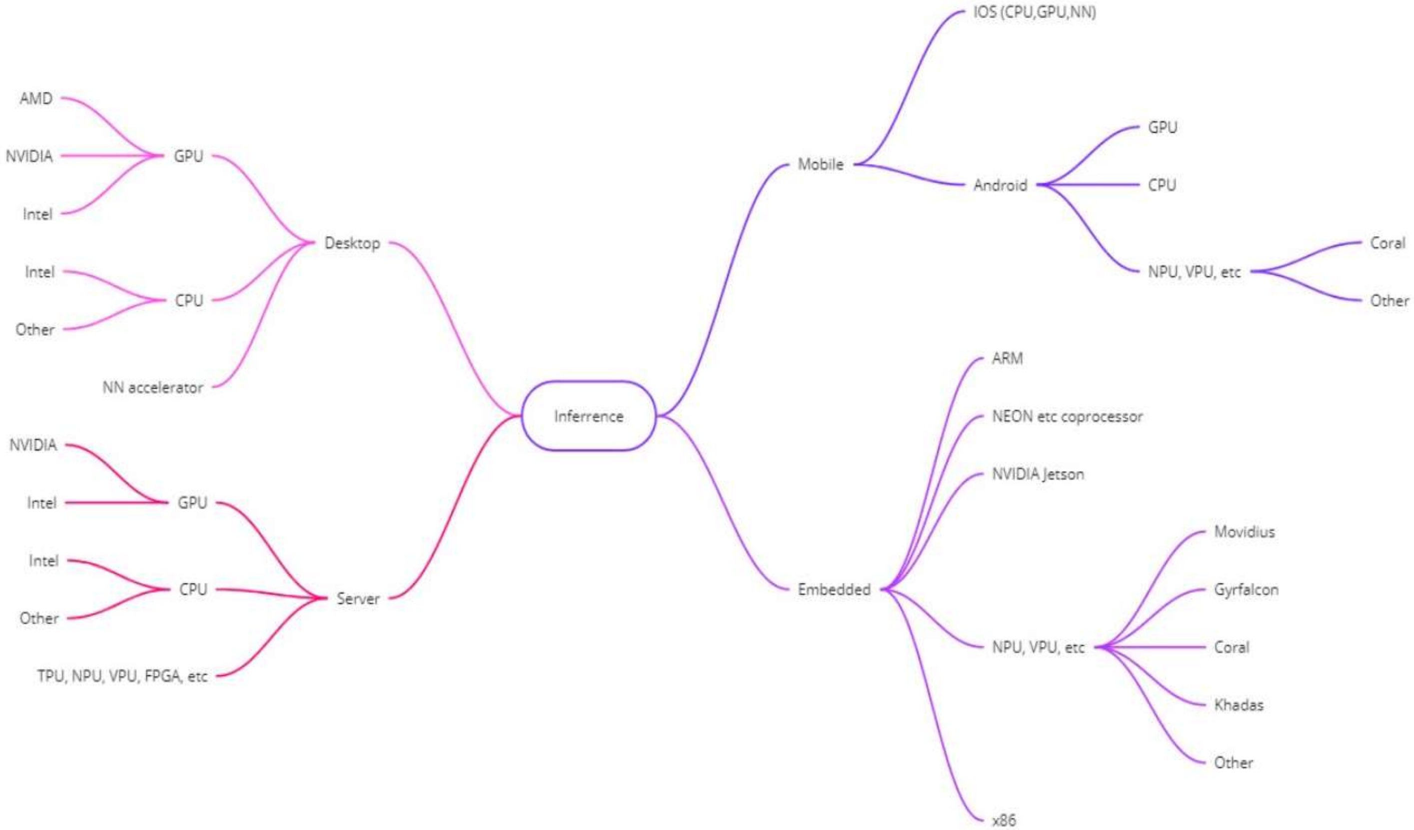
Server inference.
- executed on high-performance equipment.

Desktop inference.
- this is something that might work on your home computer.

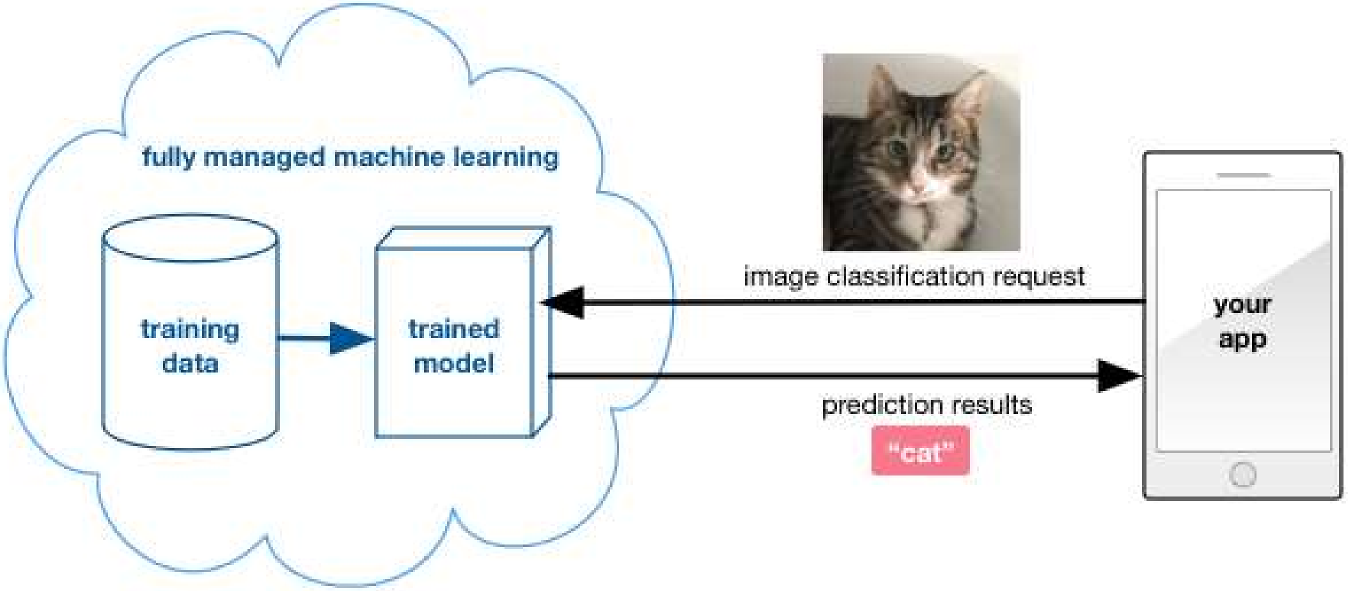
Mobile inference
- to be executed on mobile smartphones: Android and IOS with possibility of mobile GPUs, special accelerators, coprocessors, and so on.

Embedded.
- IoT with microcontrollers etc. Partly overlaps with desktops, and partly with mobile solutions, but some of the accelerators used are not like anything.

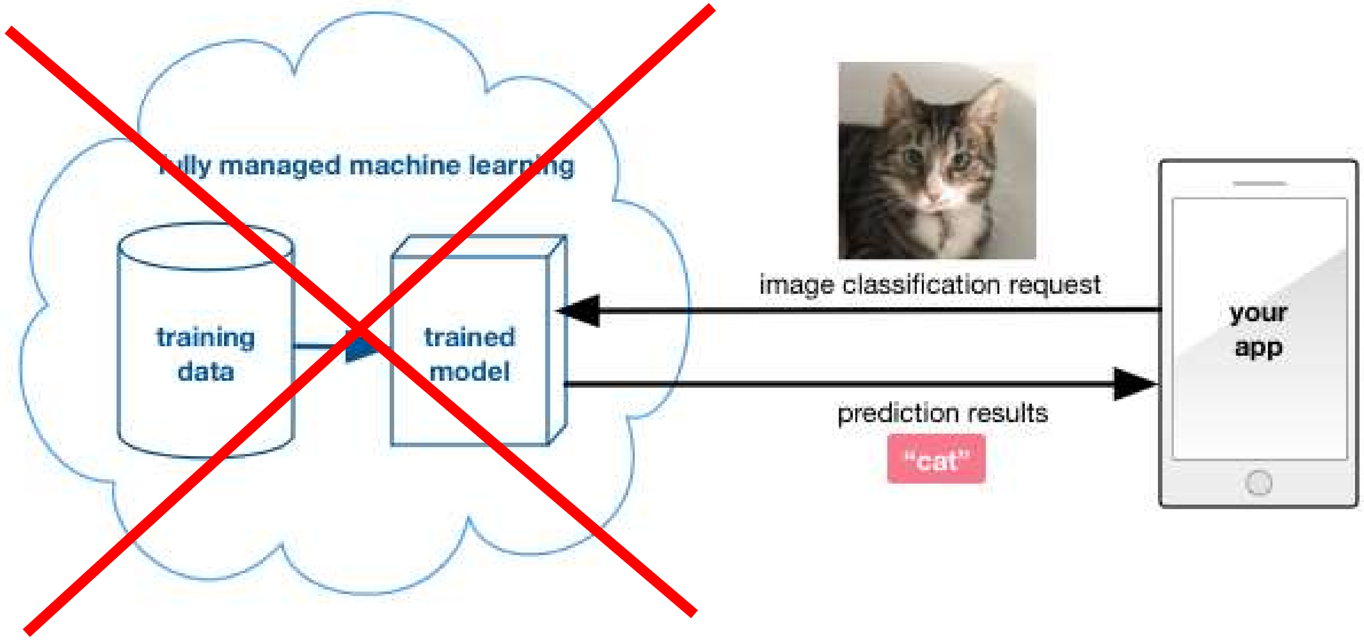
https://habr.com/ru/company/recognitor/blog/524980/



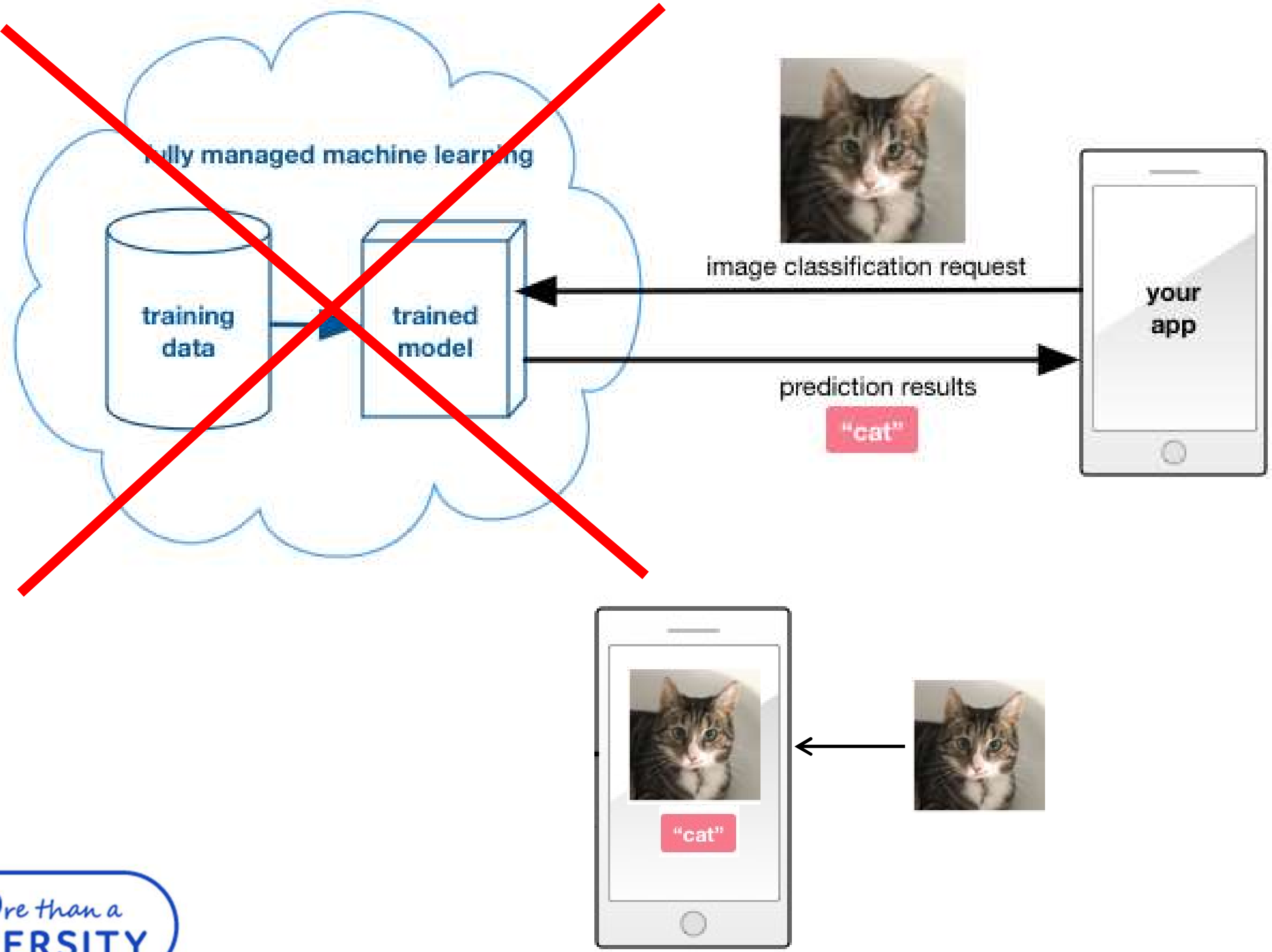
Model’s inference with Mobile Application



Model’s inference with Mobile Application



Model's inference with Mobile Application



Challenges in DL: quality vs. complexity

ITMO UNIVERSITY

➢ Major challenges in deploying deep learning models in mobile and embedded devices can be summarized as follows:

- – Diverse computing environment;
- – Limited resources;
- – Long delay;
- – Data privacy.

➢ These constraints limit the scope of use for the most powerful SOTA models directly on digital unplugged devices.

Table 1. Left: Microcontrollers have 3 orders of magnitude *less* memory and storage compared to mobile phones, and 5-6 orders of magnitude less than cloud GPUs. The extremely limited memory makes deep learning deployment difficult. **Right**: The peak memory and storage usage of widely used deep learning models. ResNet-50 exceeds the resource limit on microcontrollers by 100×, MobileNet-V2 exceeds by 20×. Even the int8 quantized MobileNetV2 requires 5.3× larger memory and can't fit a microcontroller.

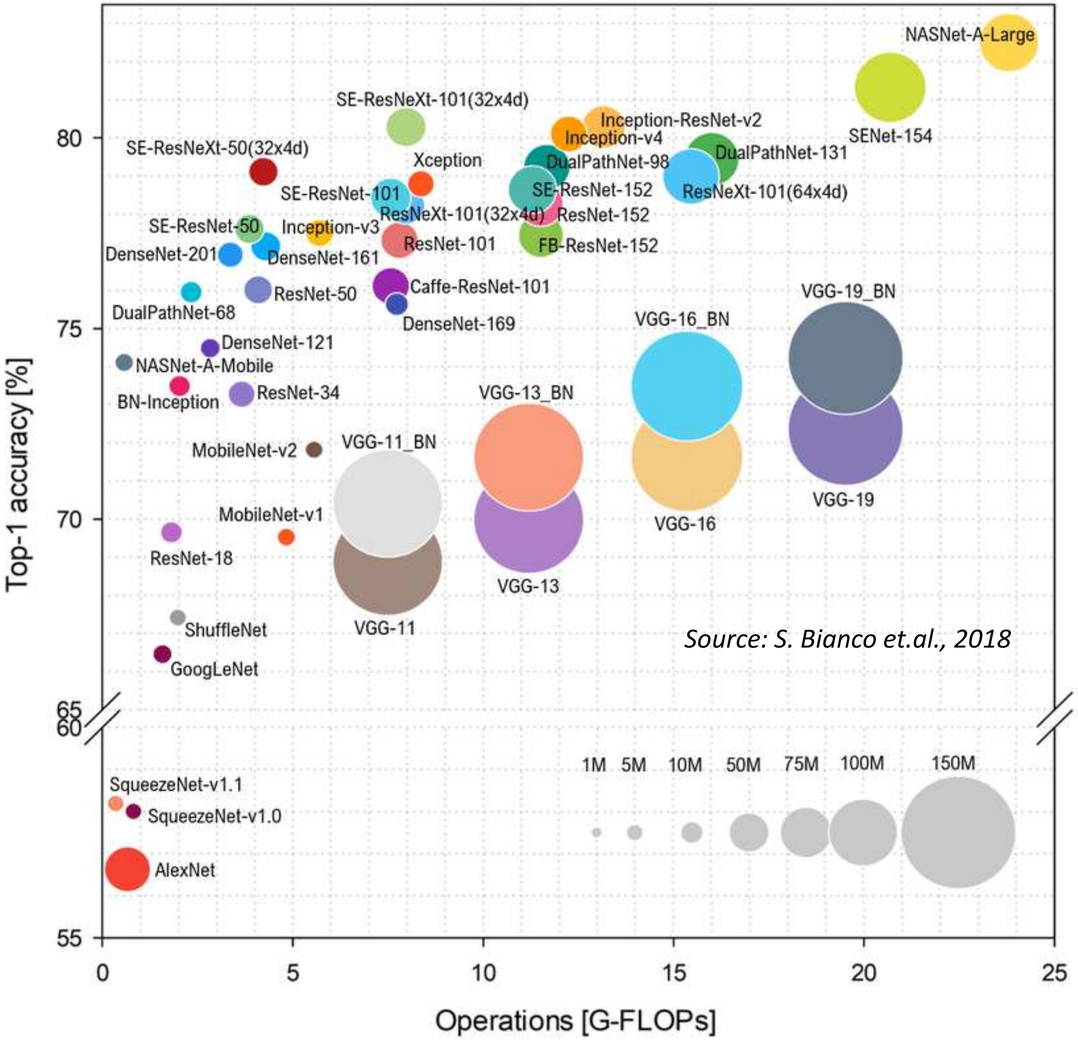
	Cloud AI (NVIDIA V100)	➔	Mobile AI (iPhone 11)	➔	Tiny AI (STM32F746)		ResNet-50	MobileNetV2	MobileNetV2 (int8)
Memory	16 GB	$\xrightarrow{4\times}$	4 GB	$\xrightarrow{3100\times}$	**320 kB**	←gap→	7.2 MB	6.8 MB	1.7 MB
Storage	TB~PB	$\xrightarrow{1000\times}$	>64 GB	$\xrightarrow{64000\times}$	**1 MB**	←gap→	102MB	13.6 MB	3.4 MB

ITsMOre than a UNIVERSITY

Lin et.al., MCUNet: Tiny Deep Learning on IoT Devices, 2020

Challenges in DL: quality vs. complexity

- Significant advances in deep learning imply significant rise of model's complexity today;
- With increasing performance of modern models one can note:
  - increasing power consumption,
  - the number of MACs for model's inference
  - memory consumption



Source: S. Bianco et.al., 2018

Relative complexity of performance of different ML areas

ITMO UNIVERSITY

Machine Learning Use Cases

0 128 512 1 2 3

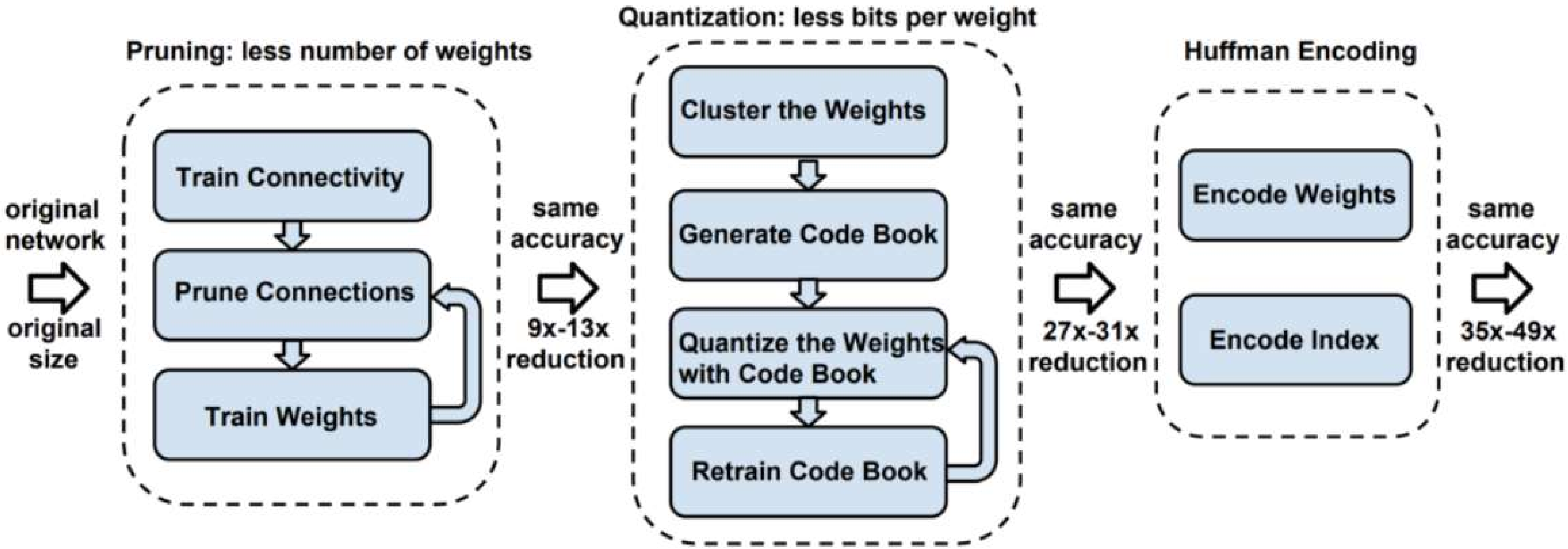
	Performance (GOPS)		Performance (TOPS)	
Computer Vision	Face and still image recognition, person detection (images)	Multi-face recognition, object detection (video)	Live video face, object recognition, object interaction (robotics)	Multi-object surveillance (people, cars, animals)
Speech Analysis	Wake word, 10 Word speech, speaker recognition	Automatic speech recognition (basic command phrases)	40,000 Word speech, multiple speaker recognition, affect/emotion recognition	High-level Speech accents interpretation
Image/Video Processing	Basic segmentation, super resolutionupscaling, denoising	Live video upscaling, denoising		Scene segmentation, single and multi-camera scene reconstruction
Sequence Analysis	Anomaly detection (environmental sensors)	Hand gesture recognition	Pose estimation	Complex real-time motion analysis

From ML to Tiny ML

- Tiny ML explores the techniques and types of ML and DL models to be run directly on low-powered devices;
- There are following model compression and acceleration techniques for this task:
  - **Pruning** (removing unimportant parameters);
  - **Quantization** (decrease the number of bits for the weights representation);
  - **Models distillation** (training a distilled “student” network that mimics the baseline “teacher” model);
  - **Network design** (designing and searching for the low-weight & efficient architectures)
  - **Low-rank factorization** (using approximate low-rank tensors, using different decomposition techniques).
- Tiny ML provides the following advantages: low latency, low power consumption, low internet bandwidth, data privacy;
- For the further acceleration of the on-device ML/DL models one may use model/data parallelism, specific dataflow design, preprocess the data using hardware (in analog domain).

DL models optimization techniques

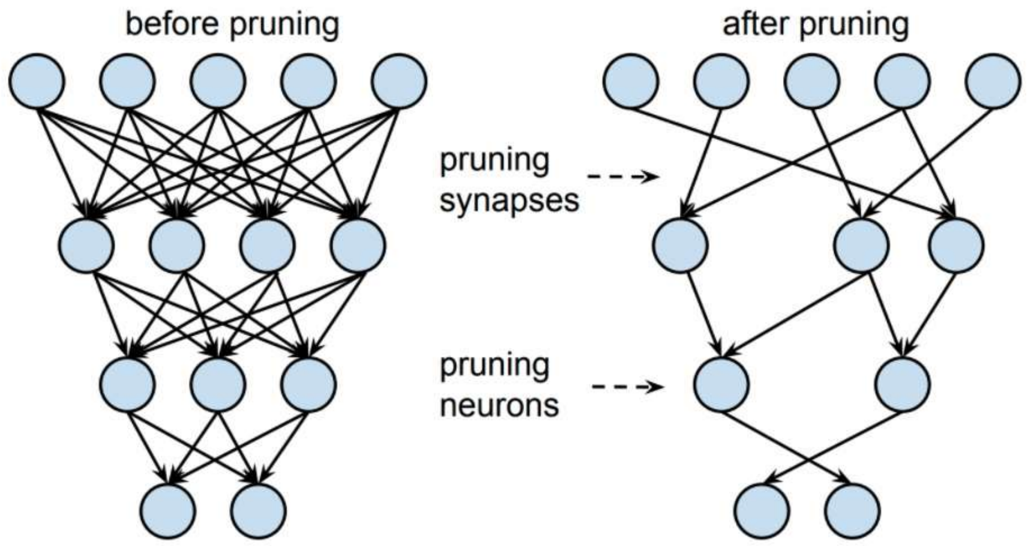
Deep model's compression process



https://towardsdatascience.com/tiny-machine-learning-the-next-ai-revolution-495c26463868

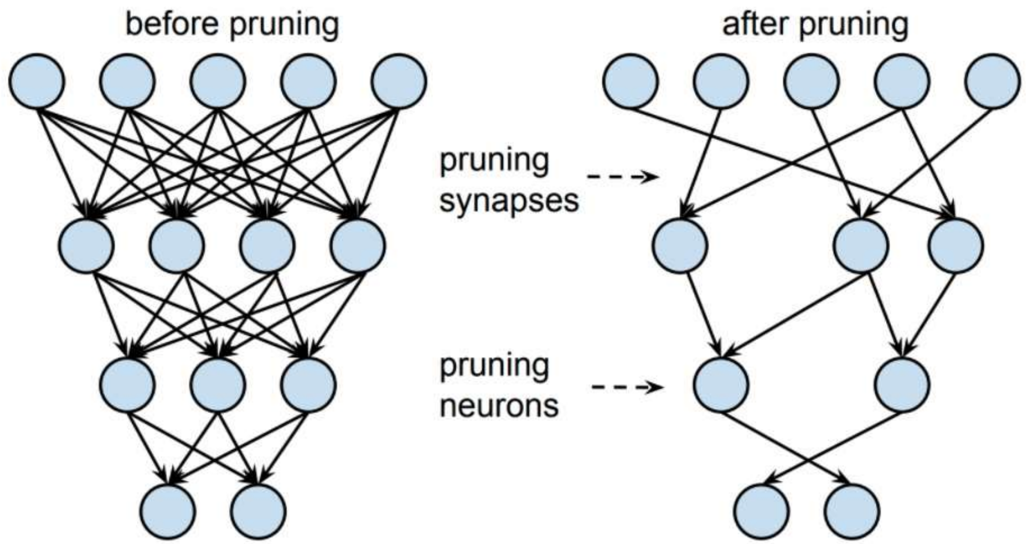
Pruning

- **pruning** can help to make the model's representation more compact;
- attempts to remove **neurons that provide small impact** to the output prediction;
- not refer to pruning operation for decision trees: we don't avoid overfitting (the model is already good), we want to delete unnecessary connections;



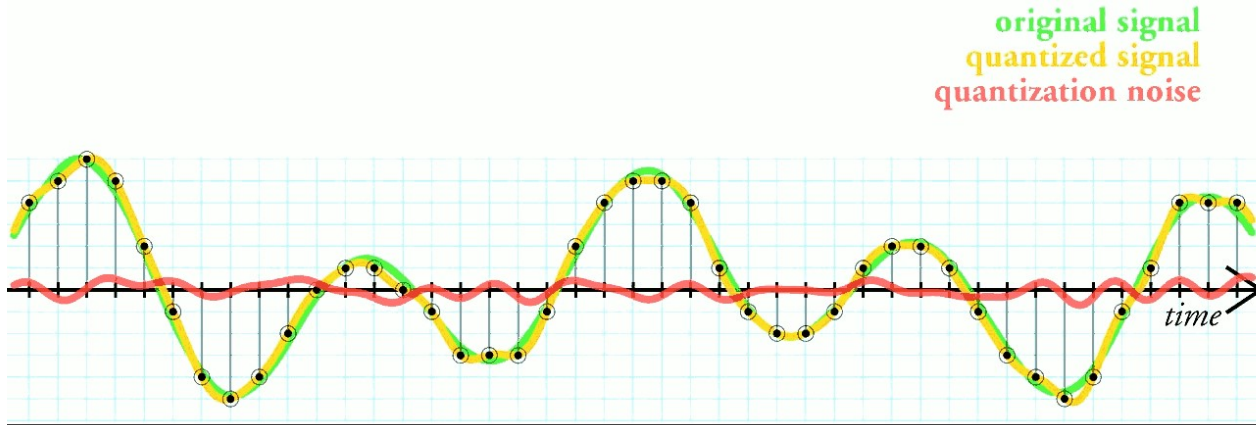
Pruning

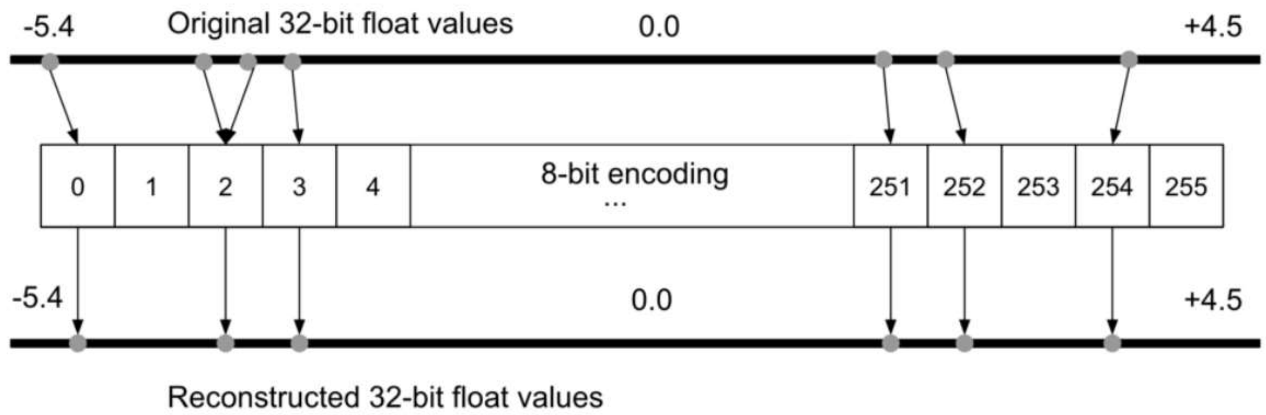
- **pruning** can help to make the model’s representation more compact;
- attempts to remove **neurons that provide small impact** to the output prediction;
- not refer to pruning operation for decision trees: we don’t avoid overfitting (the model is already good), we want to delete unnecessary connections;
- often associated with small neural weights, whereas larger weights are kept due to their greater importance during inference (e.g., on real data, activations for some layers/channels in your network may be always zeros – it is not rare case – so that it can be easily removed);
- the network is then **retrained on the pruned architecture** to fine-tune the output and test the quality decreasing.



Quantization

To run a model on the low-powered MCU, the model weights would ideally have to be stored as **8-bit integer values** (whereas many desktop computers and laptops use 32-bit or 64-bit floating-point representation).





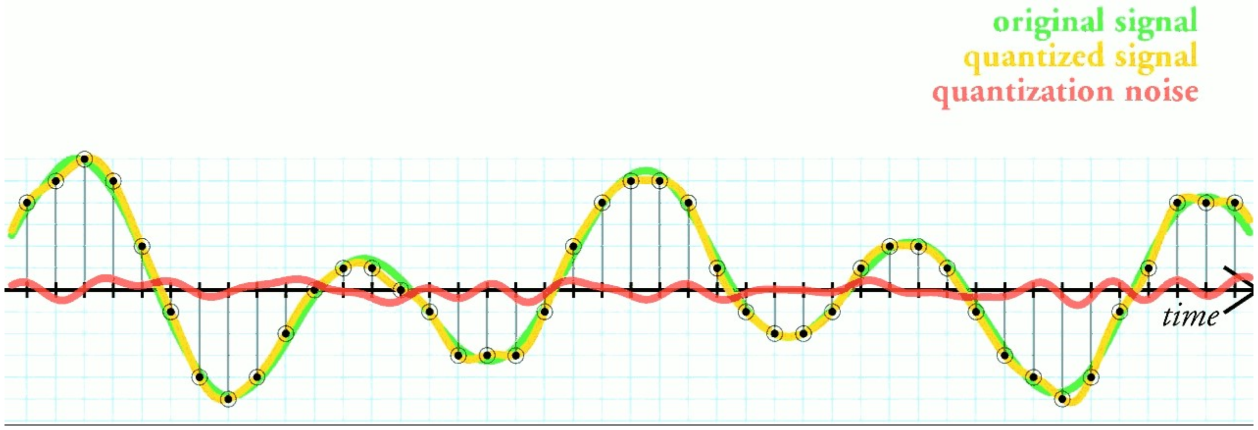
*Data encoding is an optional step that is sometimes taken to further reduce the model size by storing the data in a maximally efficient way: often via the famed **Huffman encoding.**

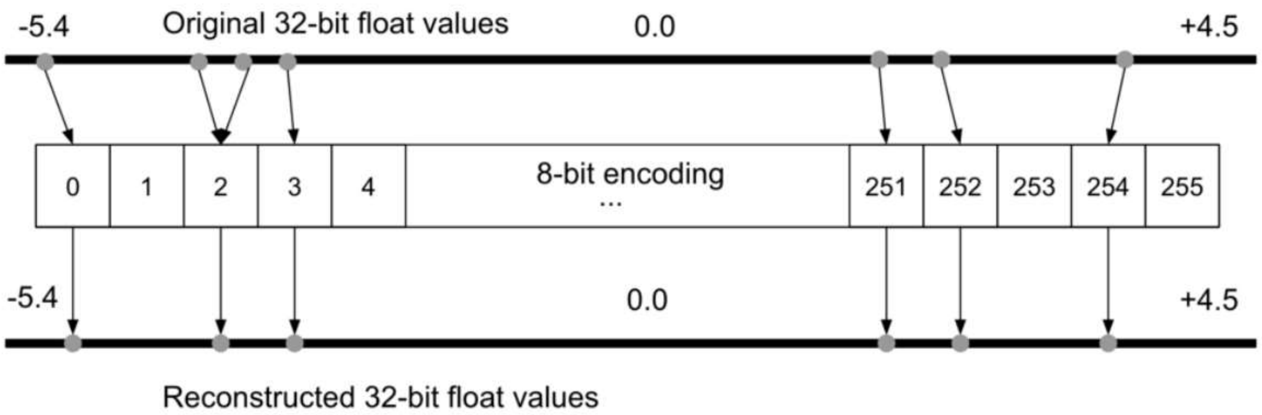
https://towardsdatascience.com/tiny-machine-learning-the-next-ai-revolution-495c26463868

Quantization

To run a model on the low-powered MCU, the model weights would ideally have to be stored as **8-bit integer values** (whereas many desktop computers and laptops use 32-bit or 64-bit floating-point representation).

By **quantizing the model**, the storage size of weights is reduced by a factor of 4 (for a quantization from 32-bit to 8-bit values), or by a factor of 8 (for a quantization from 64-bit to 8-bit values) and the accuracy is often negligibly impacted **(often around less than 3%).**





*Data encoding is an optional step that is sometimes taken to further reduce the model size by storing the data in a maximally efficient way: often via the famed **Huffman encoding.**

https://towardsdatascience.com/tiny-machine-learning-the-next-ai-revolution-495c26463868

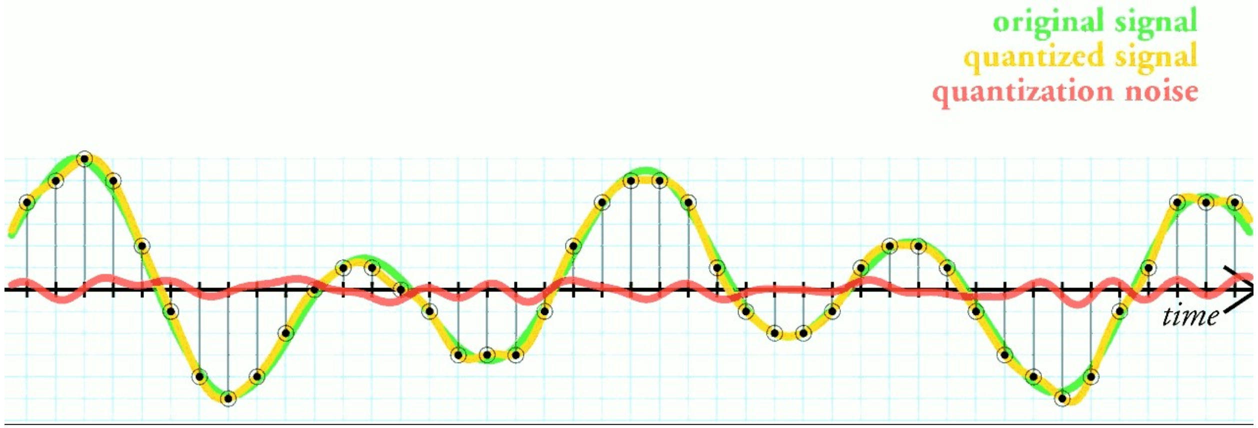
Quantization

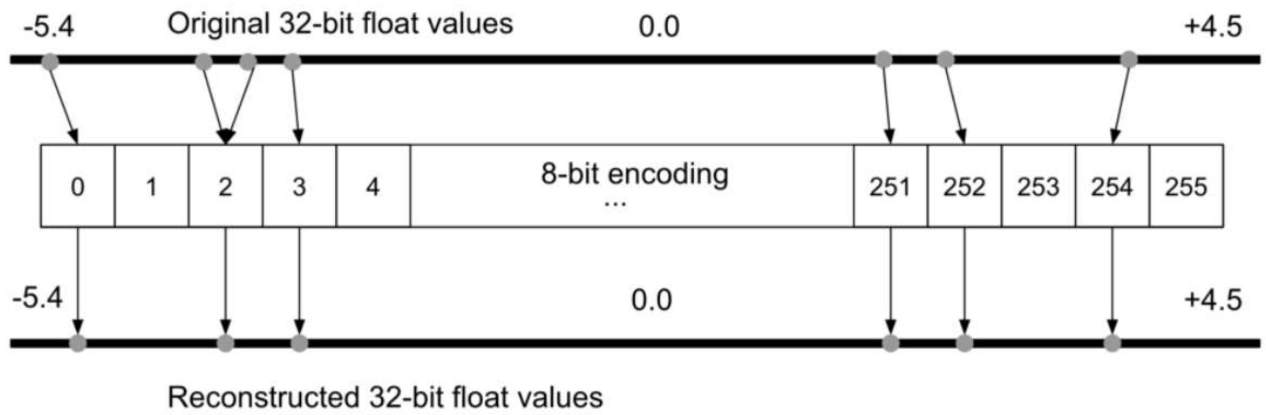
To run a model on the low-powered MCU, the model weights would ideally have to be stored as **8-bit integer values** (whereas many desktop computers and laptops use 32-bit or 64-bit floating-point representation).

By **quantizing the model**, the storage size of weights is reduced by a factor of 4 (for a quantization from 32-bit to 8-bit values), or by a factor of 8 (for a quantization from 64-bit to 8-bit values) and the accuracy is often negligibly impacted **(often around less than 3%).**

Some information may be lost during quantization due to quantization error. To combat this, **quantization-aware (QA) training** has also been proposed as an alternative. QA training essentially constrains the network during training to only use the values that will be available on the quantized device

https://towardsdatascience.com/tiny-machine-learning-the-next-ai-revolution-495c26463868





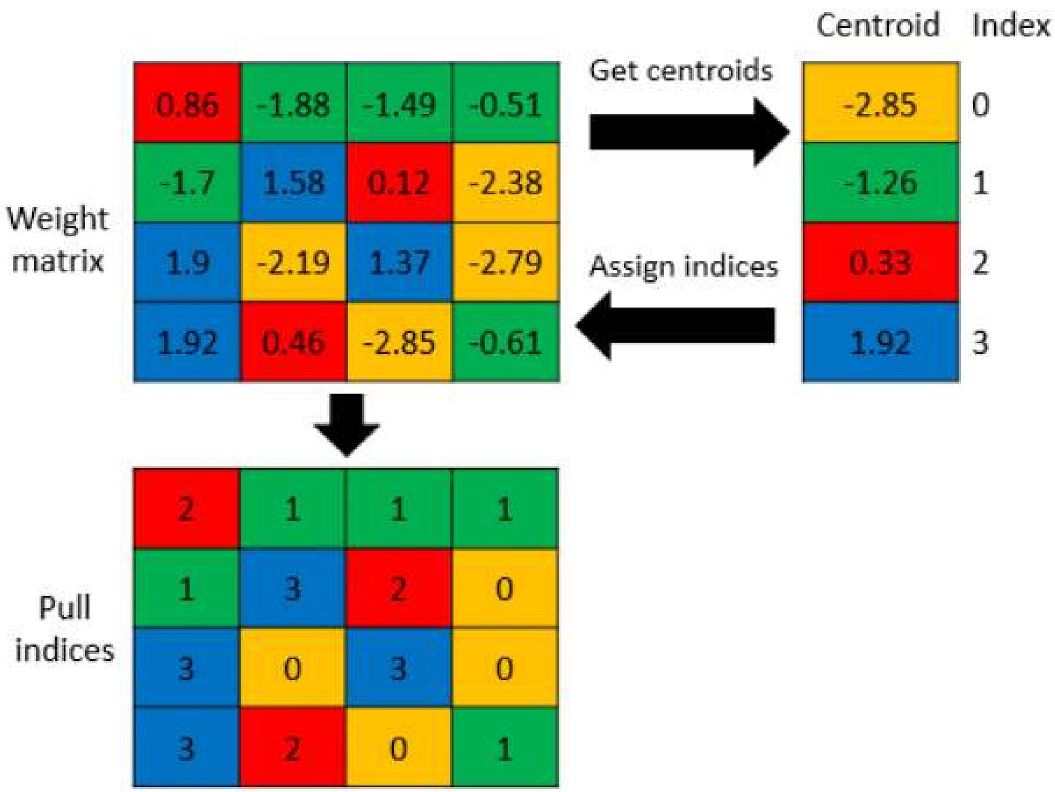
*Data encoding is an optional step that is sometimes taken to further reduce the model size by storing the data in a maximally efficient way: often via the famed **Huffman encoding.**

Weight clustering

Weight clustering reduces the size of your model by replacing similar weights in a layer with the same value. These values are found by running a clustering algorithm over the model's trained weights. The user can specify the number of clusters (in this case, 4). This step is shown in **"Get centroids"** in the diagram and the 4 centroid values are shown in the "Centroid" table. Each centroid value has an index (0-3).

Next, each weight in the weight matrix is replaced with its centroid's index. This step is shown in **"Assign indices".** Now, instead of storing the original weight matrix, the weight clustering algorithm can store the modified matrix shown in **"Pull indices"** and the centroid values themselves.

Economy of memory increases with larger matrix sizes.



Knowledge distillation approach

ITMO UNIVERSITY

Knowledge distillation is a generalization of models compression approach, introduced by Geoffrey Hinton et al. in 2015, in a preprint that formulated the concept and showed some results achieved in the task of image classification.

Knowledge distillation is the process of transferring knowledge from a large model - “**teacher**” **model** to a smaller one – “**student**” **model**.

It differs from transfer learning approach: we don’t use pretrained weights of large network, we train small network from scratch using outputs (predictions on data) from large network.

IT's MOre than a UNIVERSITY

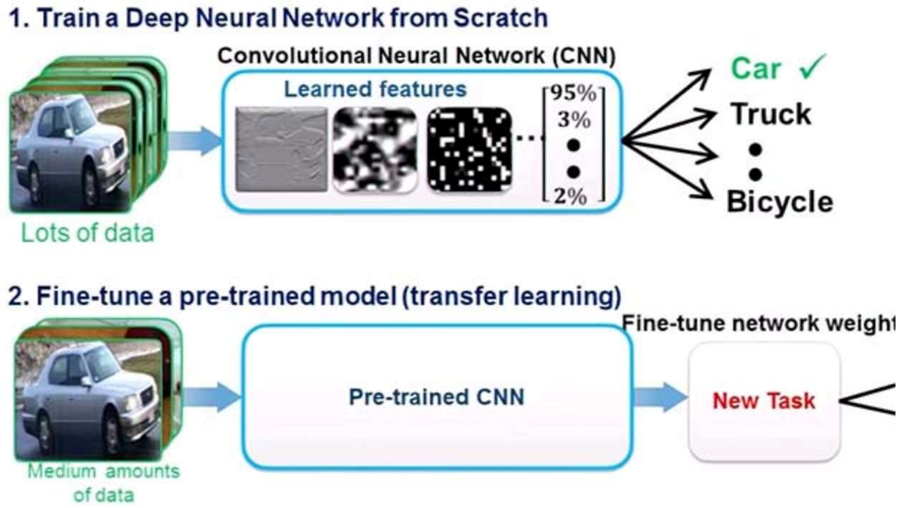
Knowledge distillation approach

Knowledge distillation is a generalization of models compression approach, introduced by Geoffrey Hinton et al. in 2015, in a preprint that formulated the concept and showed some results achieved in the task of image classification.

Knowledge distillation is the process of transferring knowledge from a large model - **“teacher” model** to a smaller one – **“student” model**.

It differs from transfer learning approach: we don’t use pretrained weights of large network, we train small network from scratch using outputs (predictions on data) from large network.

Transfer Learning:



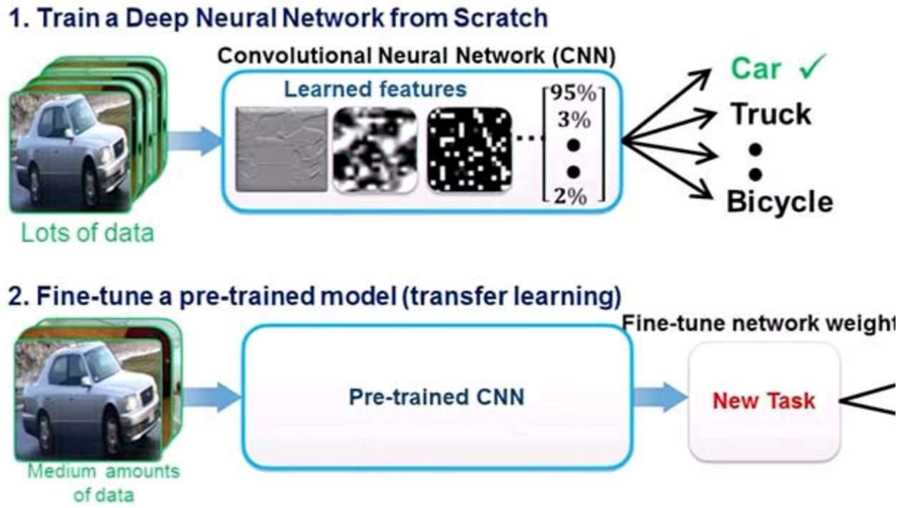
Knowledge distillation approach

Knowledge distillation is a generalization of models compression approach, introduced by Geoffrey Hinton et al. in 2015, in a preprint that formulated the concept and showed some results achieved in the task of image classification.

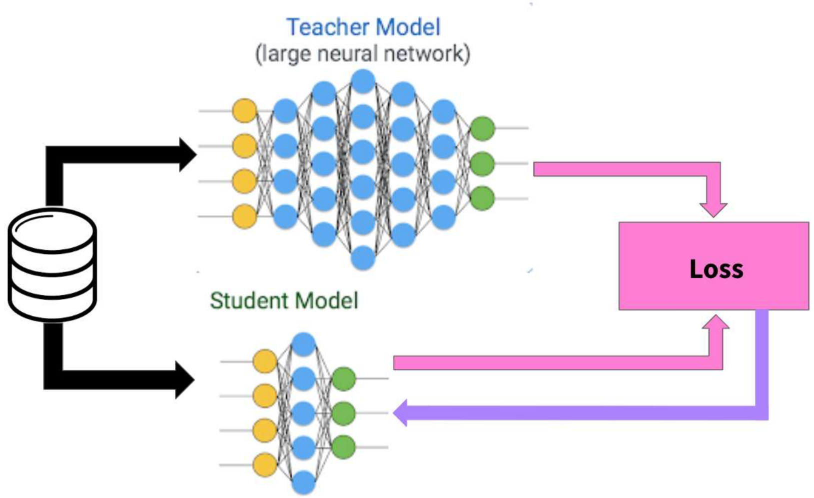
Knowledge distillation is the process of transferring knowledge from a large model - **“teacher” model** to a smaller one – **“student” model**.

It differs from transfer learning approach: we don’t use pretrained weights of large network, we train small network from scratch using outputs (predictions on data) from large network.

Transfer Learning:



Knowledge distillation:

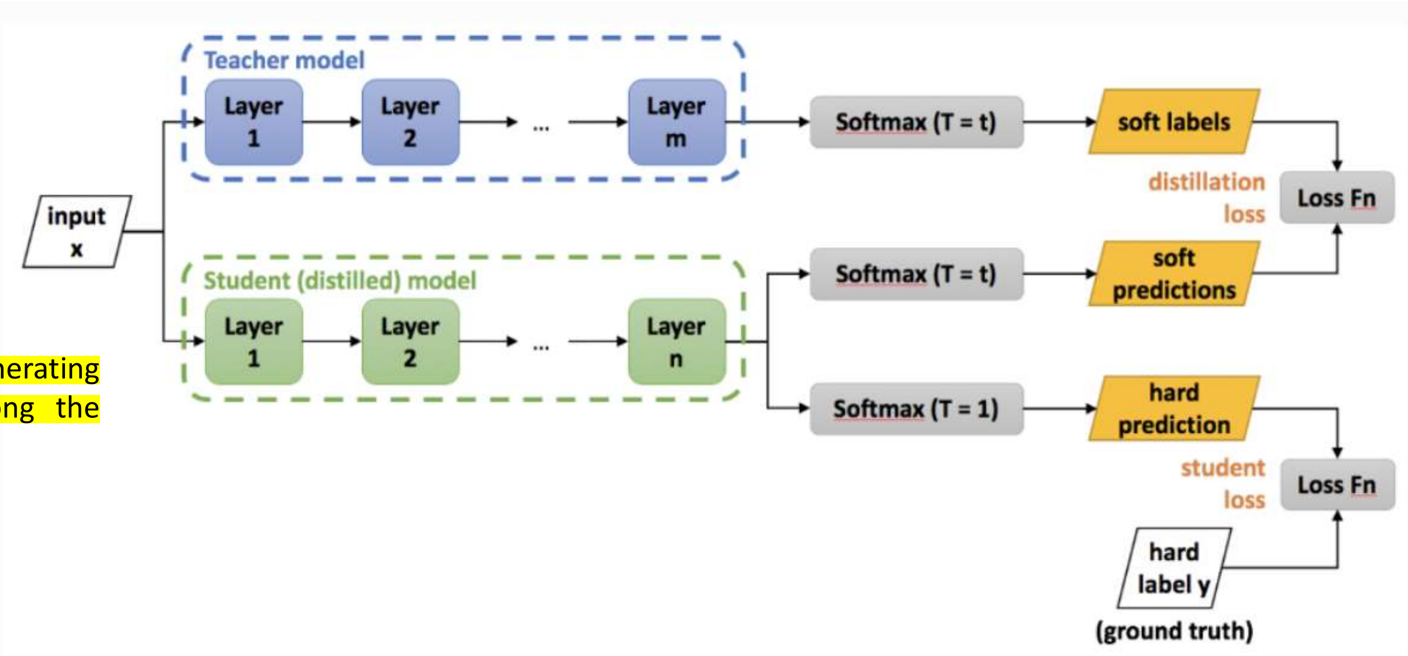


Knowledge distillation approach

Given a large model as a function of the vector variable x, trained for a specific classification task, typically the final layer of the network is a softmax in the form:

$$y_i(\mathbf{x}|t) = \frac{e^{\frac{z_i(\mathbf{x})}{t}}}{\sum_j e^{\frac{z_j(\mathbf{x})}{t}}}$$

Higher values of **temperature t** have the effect of generating a softer distribution of pseudo-probabilities among the output classes.



Knowledge distillation approach

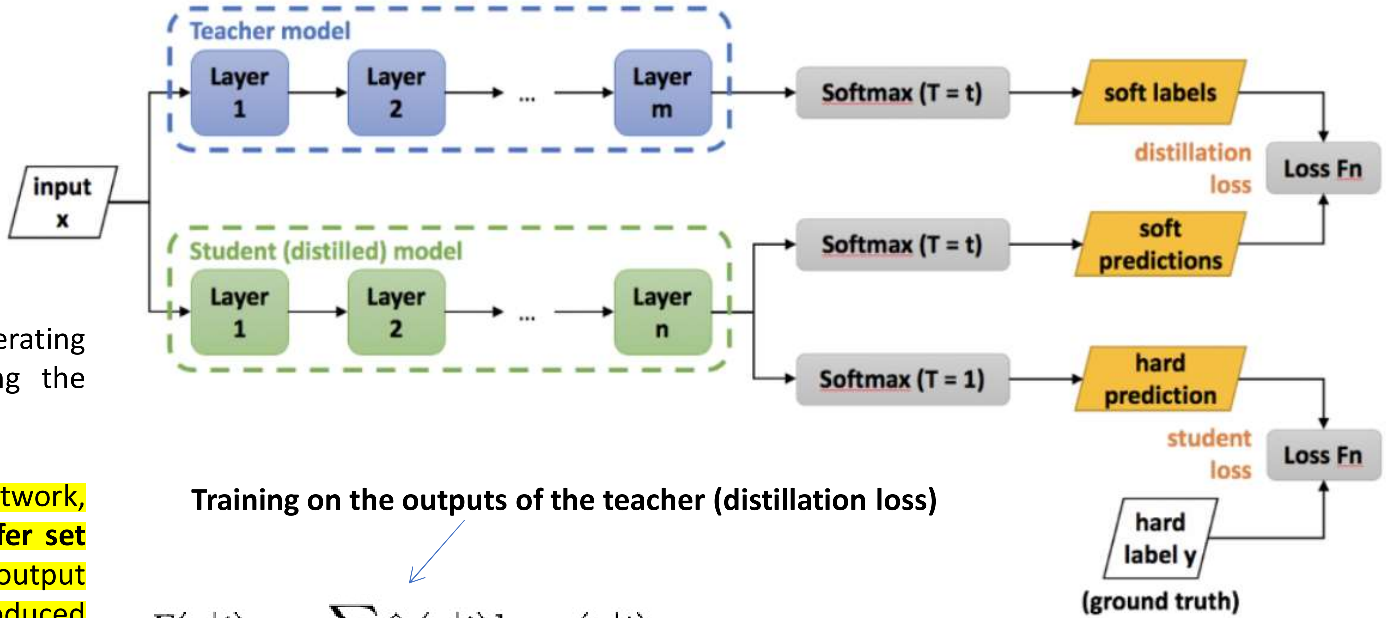
ITMO UNIVERSITY

Given a large model as a function of the vector variable x, trained for a specific classification task, typically the final layer of the network is a softmax in the form:

$$y_i(\mathbf{x}|t) = \frac{e^{\frac{z_i(\mathbf{x})}{t}}}{\sum_j e^{\frac{z_j(\mathbf{x})}{t}}}$$

Higher values of **temperature t** have the effect of generating a softer distribution of pseudo-probabilities among the output classes.

Knowledge distillation consists of training a smaller network, called the **distilled model**, on a dataset called **transfer set** using the cross entropy as loss function between the output of the distilled model y(x|t) and the output ŷ(x|t) produced by the large model on the same record **using a high value of temperature t** for both models.



Training on the outputs of the teacher (distillation loss)

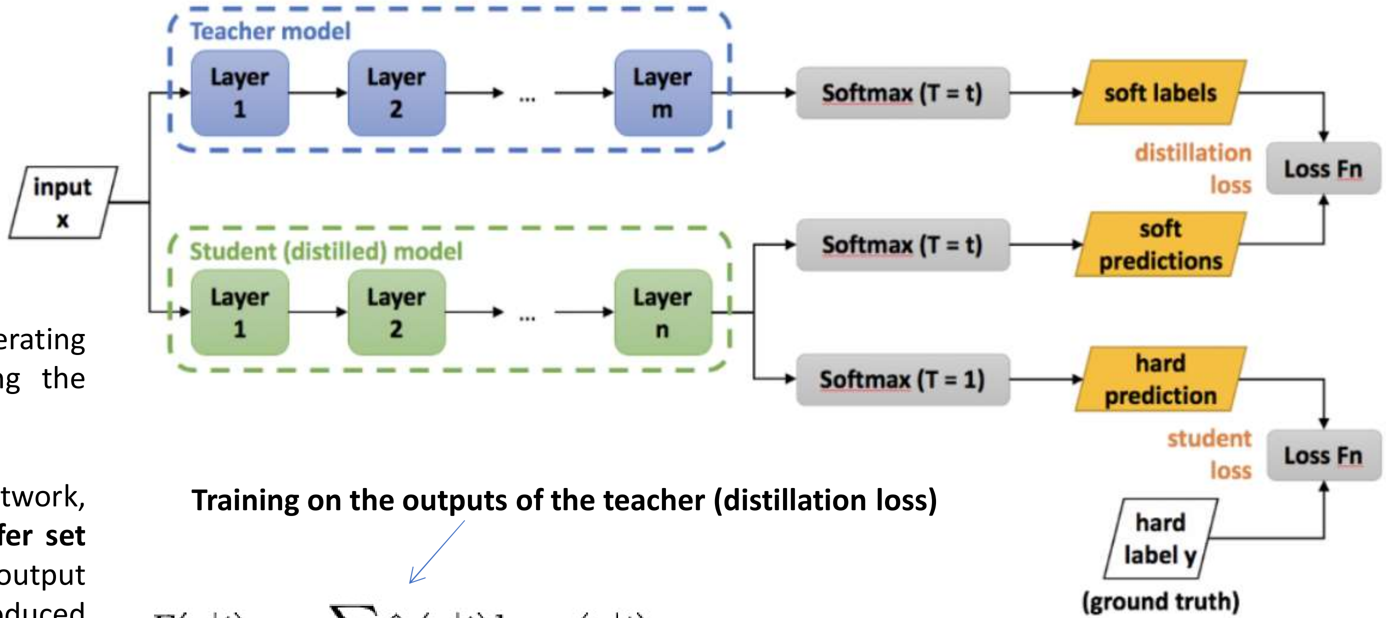
$$E(\mathbf{x}|t) = -\sum_i \hat{y}_i(\mathbf{x}|t) \log y_i(\mathbf{x}|t)$$

Knowledge distillation approach

ITMO UNIVERSITY

Given a large model as a function of the vector variable x, trained for a specific classification task, typically the final layer of the network is a softmax in the form:

$$y_i(\mathbf{x}|t) = \frac{e^{\frac{z_i(\mathbf{x})}{t}}}{\sum_j e^{\frac{z_j(\mathbf{x})}{t}}}$$



Higher values of **temperature t** have the effect of generating a softer distribution of pseudo-probabilities among the output classes.

Knowledge distillation consists of training a smaller network, called the **distilled model**, on a dataset called **transfer set** using the cross entropy as loss function between the output of the distilled model y(x|t) and the output ŷ(x|t) produced by the large model on the same record **using a high value of temperature t** for both models.

Training on the outputs of the teacher (distillation loss)

$$E(\mathbf{x}|t) = -\sum_i \hat{y}_i(\mathbf{x}|t) \log y_i(\mathbf{x}|t)$$

High temperature increases the entropy of the output, and therefore provides more information to learn for the distilled model compared to hard targets, and reducing the variance of the gradient between different records and therefore allowing higher learning rates.
If ground truth is available for the transfer set, the process can be strengthened by adding to the loss the cross-entropy between the output of the distilled model (with t=1) and GT label **("hard label")** ȳ:

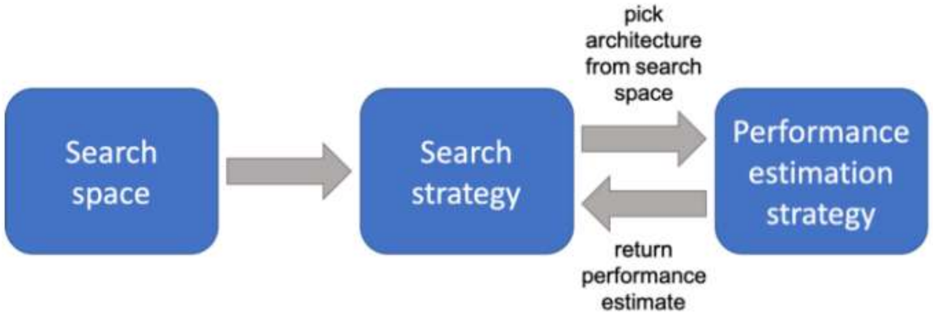
$$E(\mathbf{x}|t) = -t^2 \sum_i \hat{y}_i(\mathbf{x}|t) \log y_i(\mathbf{x}|t) - \sum_i \bar{y}_i \log y_i(\mathbf{x}|1)$$

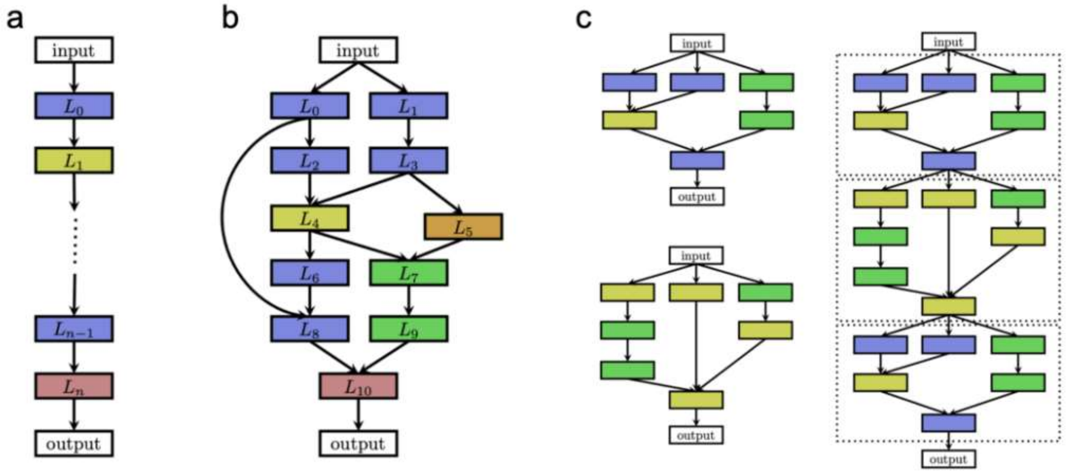
Training on the outputs of the teacher and GT labels (distillation + student loss)

Architecture design and Neural Architecture Search (NAS)

ITMO UNIVERSITY

- **NAS** is actually a sub-field of **AutoML**;
- **NAS** finds an ideal solution from a large set of candidates and selects the one that best meets the objectives of a given problem
- The most popular ways for optimizing this search are:
  - Optimization-based algorithms
  - Reinforcement Learning
  - Evolutionary algorithms
- Very Computationally expensive
- Hard to estimate how it will behave with real data





Possible search space of architectures

ITsMOre than a UNIVERSITY

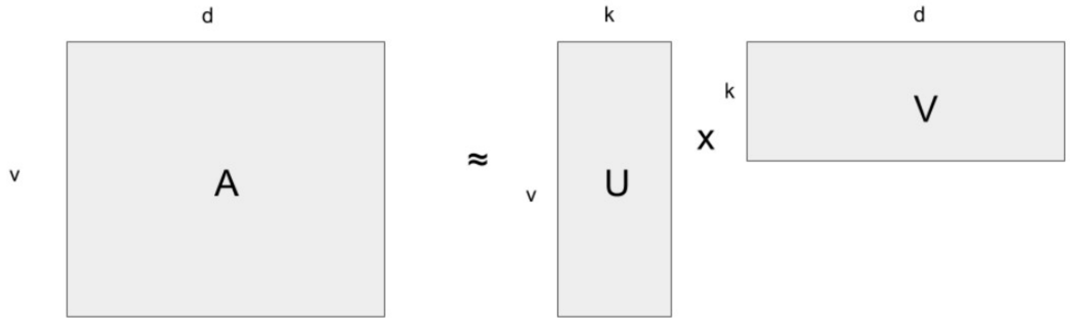
Low-rank factorization

- **Low-rank matrix factorization (MF)** is an important technique in data science.
- The key idea of MF is that there exists latent structures in the data, by uncovering which we could obtain a **compressed representation of the data**.
- By factorizing an original matrix to low-rank matrices, MF provides a unified method for dimension reduction, clustering, and matrix compression.

Low-rank factorization

- **Low-rank matrix factorization (MF)** is an important technique in data science.
- The key idea of MF is that there exists latent structures in the data, by uncovering which we could obtain a **compressed representation of the data**.
- By factorizing an original matrix to low-rank matrices, MF provides a unified method for dimension reduction, clustering, and matrix compression.

- Original matrix A is factored into two thinner matrices by minimizing the Frobenius error $|A-UV^T|_F$ where **U, V** are low rank (rank k) matrices. This minimization can be solved optimally by using **SVD (Singular Value Decomposition).**



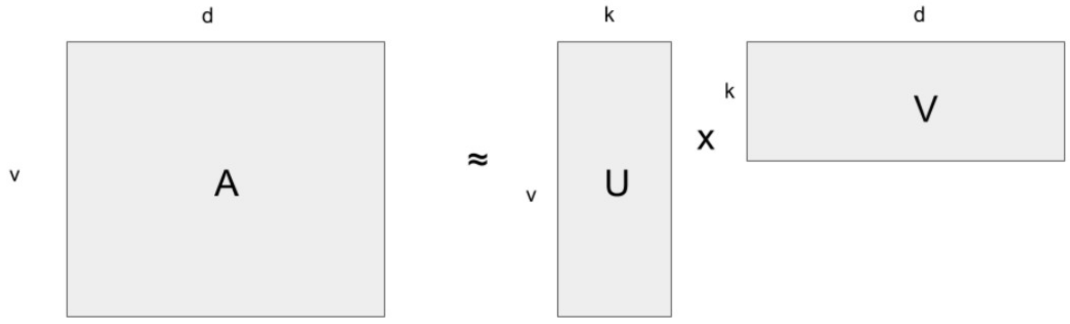
Low rank approximation: Write matrix M as a product of two thin (rank k) matrices U and V. Space goes down from vd to k(v+d). Not very useful for embedding matrix: is thin (low rank) to begin with.

Low-rank factorization

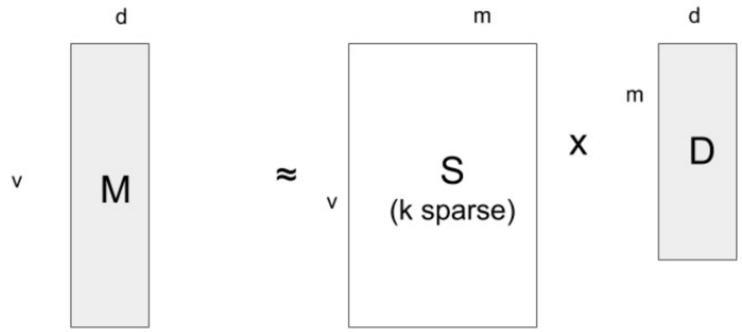
- **Low-rank matrix factorization (MF)** is an important technique in data science.
- The key idea of MF is that there exists latent structures in the data, by uncovering which we could obtain a **compressed representation of the data**.
- By factorizing an original matrix to low-rank matrices, MF provides a unified method for dimension reduction, clustering, and matrix compression.

- Original matrix A is factored into two thinner matrices by minimizing the Frobenius error $|A\text{-}UV^T|_F$ where **U, V** are low rank (rank k) matrices. This minimization can be solved optimally by using **SVD (Singular Value Decomposition).**

- **Sparse factorization via dictionary learning** – is another way to perform low-rank factorization: it exploits the possibility that there may be a smaller dictionary of basis vectors such that each embedding vector is some sparse combination of a few of these dictionary vectors -- thus it decomposes the embedding matrix into a product of smaller dictionary table and a sparse matrix that specifies which dictionary entries are combined for each embedding entry.



Low rank approximation: Write matrix M as a product of two thin (rank k) matrices U and V. Space goes down from vd to k(v+d). Not very useful for embedding matrix: is thin (low rank) to begin with.



Dictionary Learning: Write matrix M as a product of two matrices S and D. S is sparse. Space goes down from vd to m*d + k*v * log m.

https://blog.tensorflow.org/2020/02/matrix-compression-operator-tensorflow.html

Example

Once-for-all model

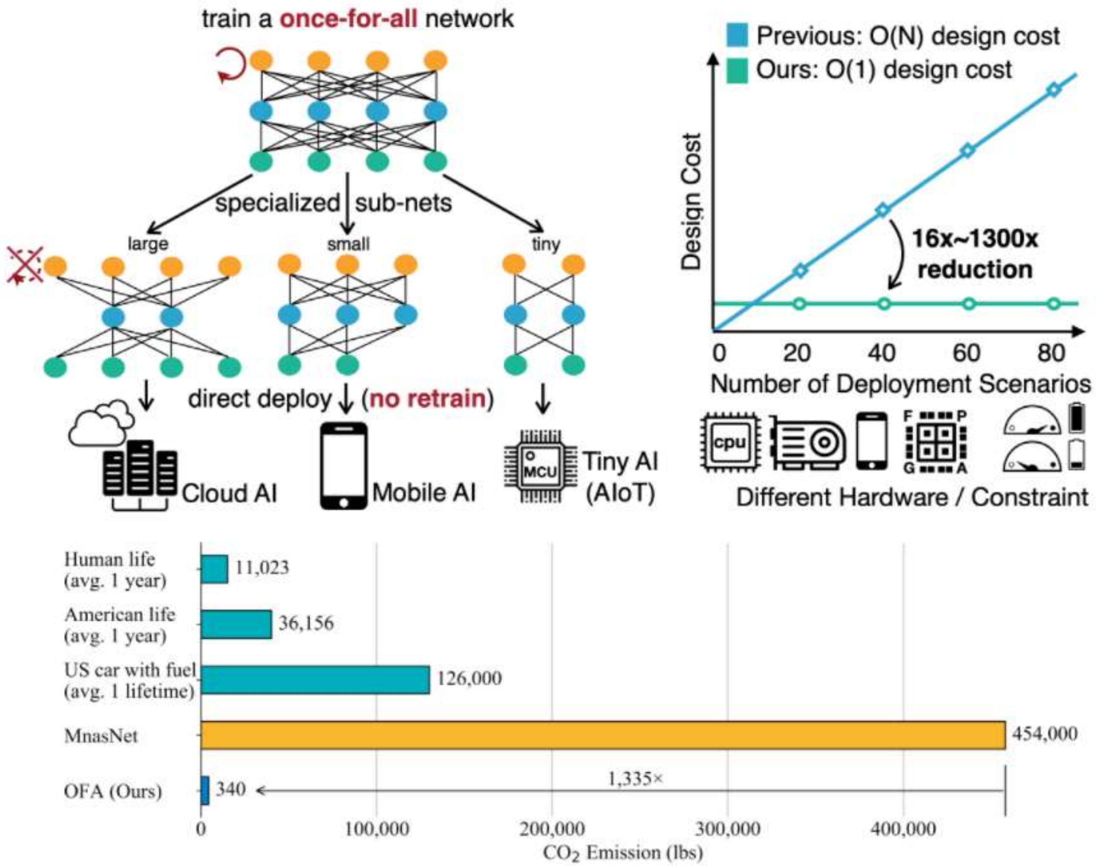
- **Once-for-all (OFA)** network is trained to support versatile architectural configurations including depth, width, kernel size and resolution;
- Given a deployment scenario, a specialized **subnetwork is directly selected** from the base OFA network without training;
- Approach reduces the cost of specialized deep learning deployment from O(N) to O(1);
- The winner of Low Power Computer Vision Challenge (2020);

Can get **>10^{19} subnetworks** that can fit different hardware platforms and latency constraints;

OFA saves up to 3 orders of magnitude design cost compared to NAS methods;

But: need >4K hours GPU for training base model.

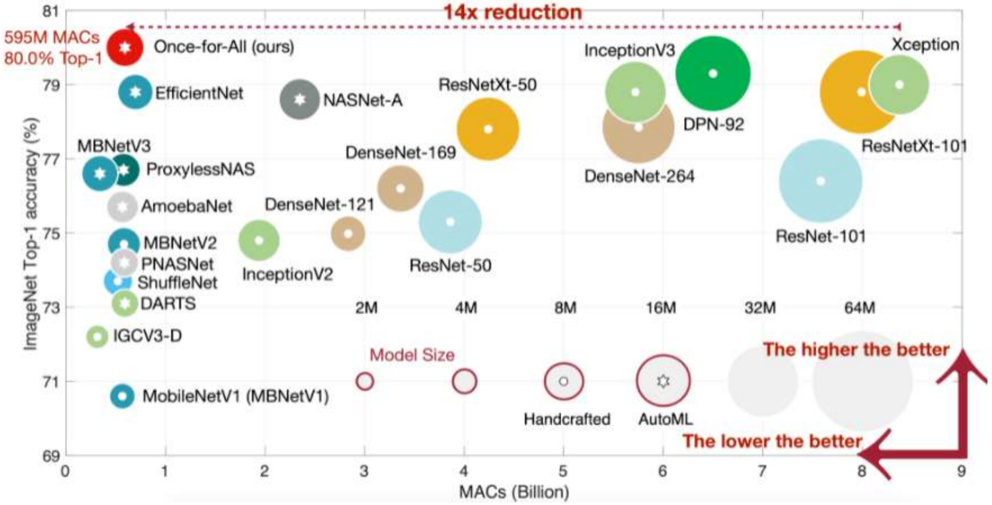
Train once, specialize for many deployment scenarios

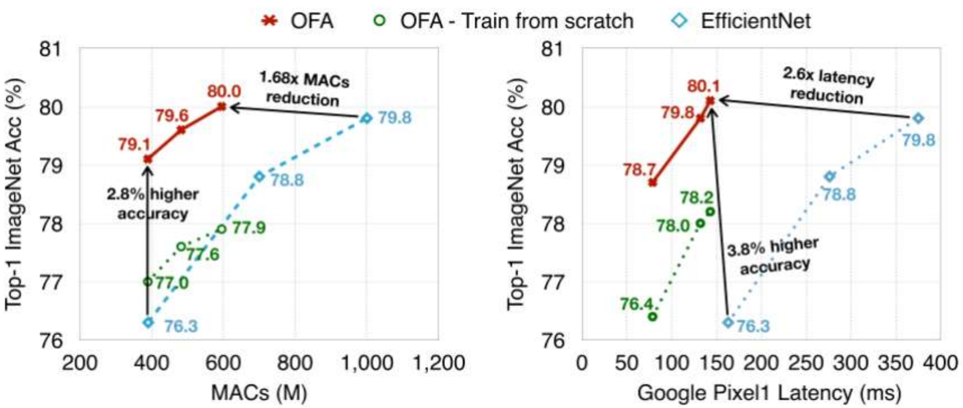


Cai et.al., Once-for-All: Train One Network and Specialize it for Efficient Deployment, 2019

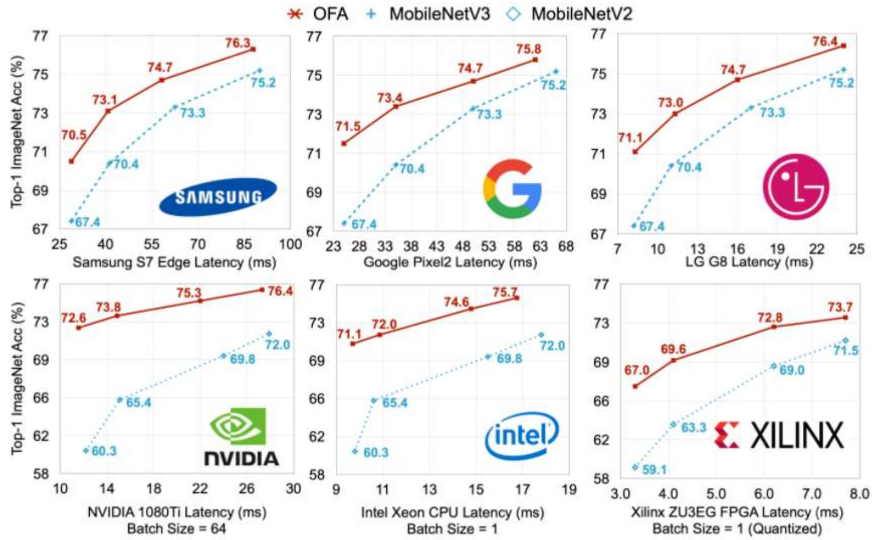
Once-for-all model

ITMO UNIVERSITY





Consistently outperforms MobileNetV3/MobileNetV2 on diverse hardware platforms



OFA approach achieves State-of the art results for many mobile hardware platforms on the different image-related tasks.

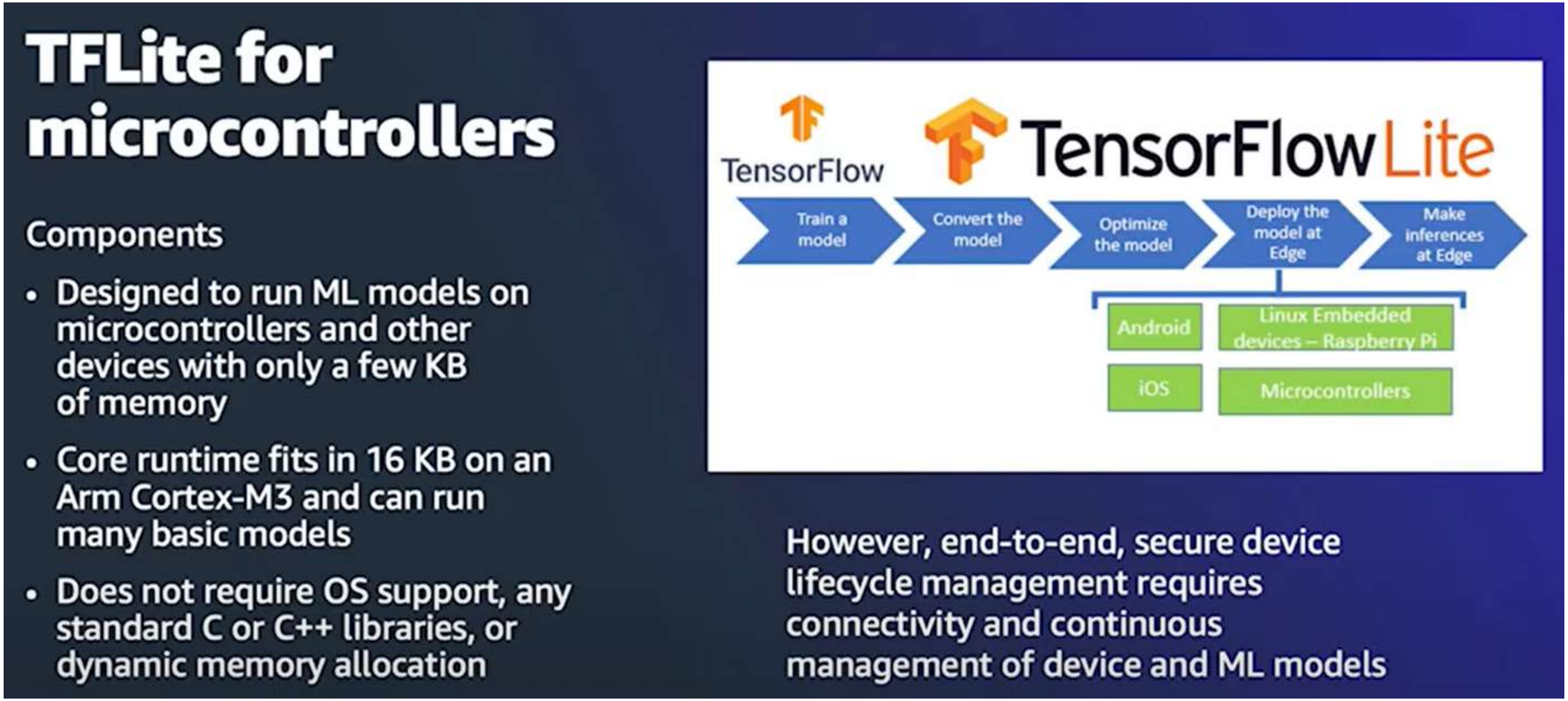
Cai et.al., Once-for-All: Train One Network and Specialize it for Efficient Deployment, 2019

ITMO UNIVERSITY

Frameworks

ITsMOre than a UNIVERSITY

Tensorflow Lite (TFLite)



Source: Embedded World 2021: TinyML for AI at the Very, Very Edge

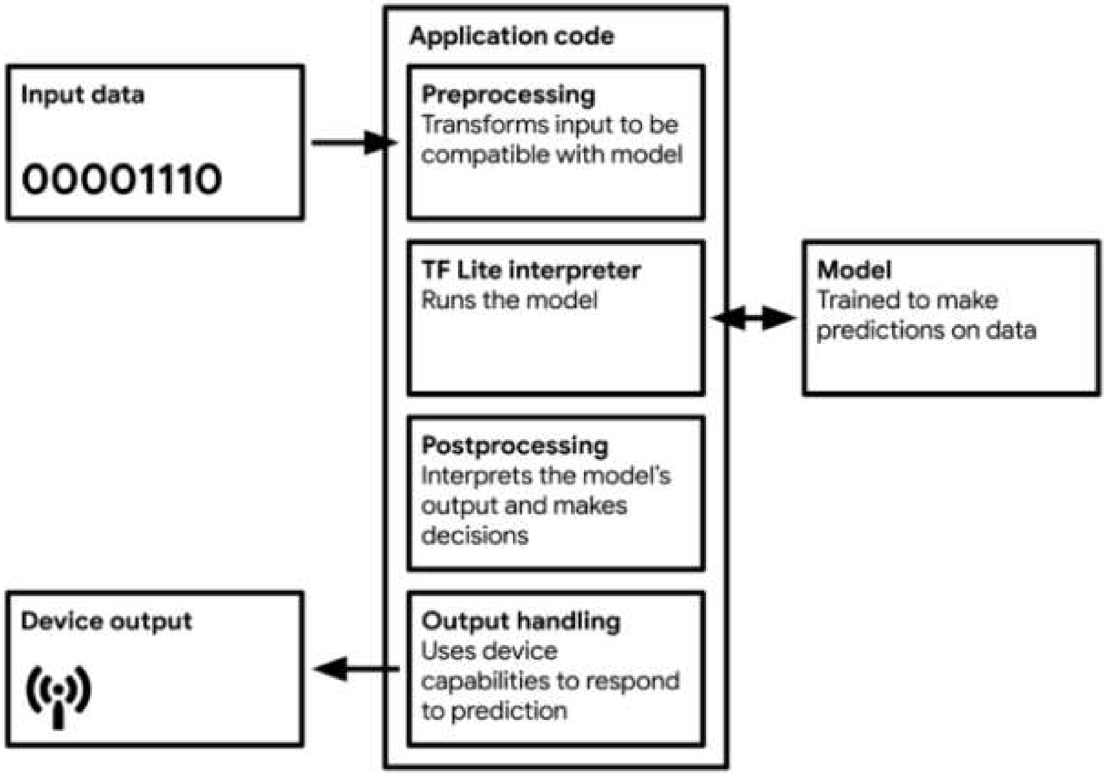
Tensorflow Lite (TFLite)

ITMO UNIVERSITY

- Lite model should be converted to a format that can be interpreted by some form of light neural network interpreter: the most popular of which are TF Lite (~500 KB in size) and TF Lite Micro (~20 KB in size);
- The model is then compiled into C or C++ code (the languages most microcontrollers work in for efficient memory usage) and run by the interpreter on-device.

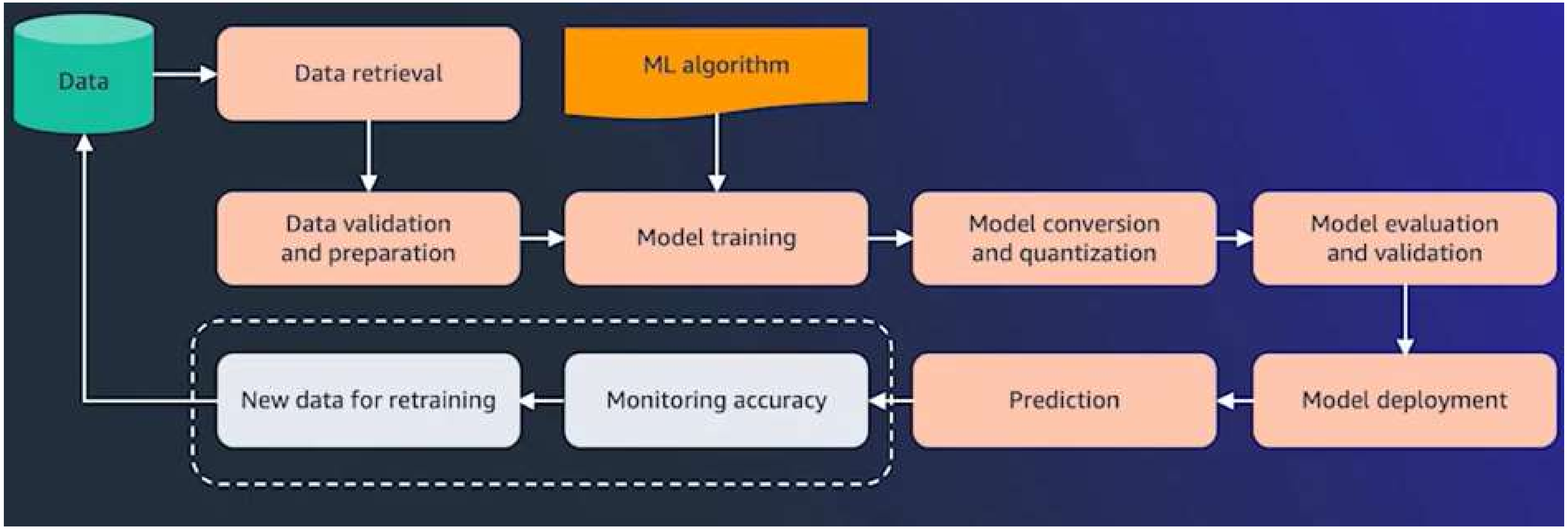
- The model on the device has to be able to perform inference. Microcontroller must have a memory large enough that it can run:
  (1) its operating system and libraries,
  (2) a neural network interpreter such as TF Lite,
  (3) the stored neural weights and neural architecture,
  (4) the intermediate results during inference.

- The peak memory usage of a quantized algorithm is often quoted in tinyML research papers, along with memory usage, the number of multiply-accumulate units (MACs), accuracy, etc.



The workflow of TinyML application (Source: TinyML book by Pete Warden and Daniel Situnayake)

Basic pipeline for inference ML model on edge device



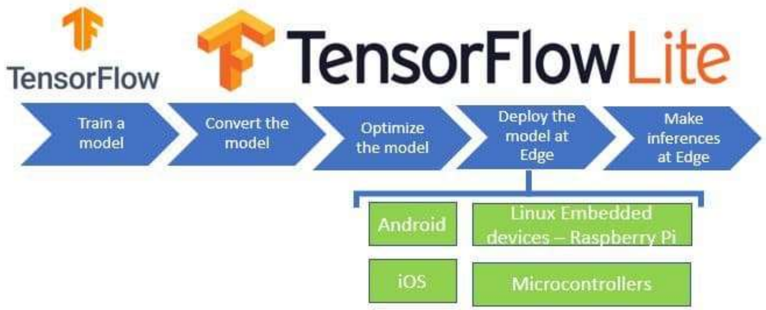
Source: Embedded World 2021: TinyML for AI at the Very, Very Edge

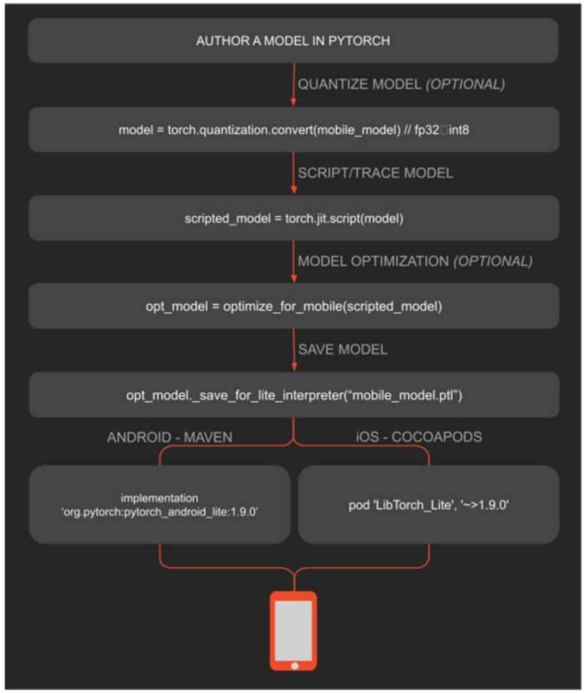
Popular frameworks for embedded ML and DL

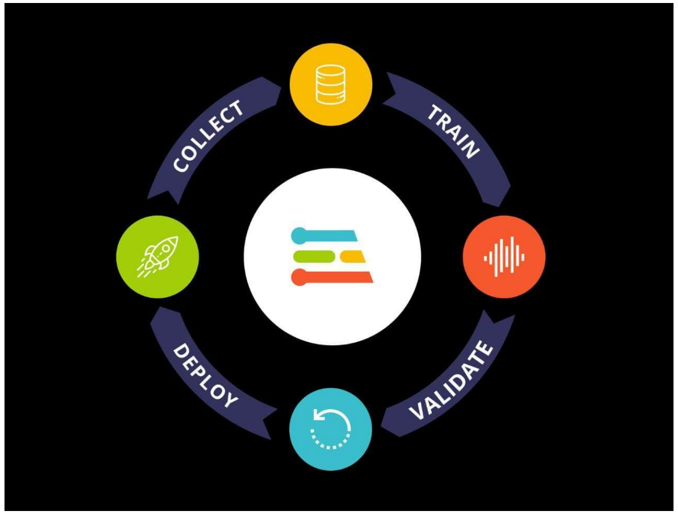
- **TensorFlow Lite** https://www.tensorflow.org/lite
  Provides a set of tools that enables on-device machine learning by allowing developers to run their trained models on mobile, embedded, and IoT devices and computers. It supports platforms such as embedded Linux, Android, iOS, and MCU.

- **Pytorch mobile** https://pytorch.org/mobile/home/
  Provides an efficient mobile interpreter in Android and iOS. Also supports build level optimization and selective compilation depending on the operators needed for user applications (i.e., the final binary size of the app is determined by the actual operators the app needs).

- **Edge Impulse** https://www.edgeimpulse.com/
  Enables the easy collection of real sensor data, live signal processing from raw data to neural networks, testing and deployment to any target device. Open source SDKs allow you to collect data from or deploy code to any device.







ITMO UNIVERSITY

Saint Petersburg, Russia

Thank you!