Introduction to Resource Constrained Embedded Deep Learning

Mehrdad Yaghoobi
m.yaghoobi-vaighan@ed.ac.uk,
Lecturer at the Institute for Digital Communications,
University of Edinburgh

UDRC Summer School, Machine Learning Track, 30 June 2021

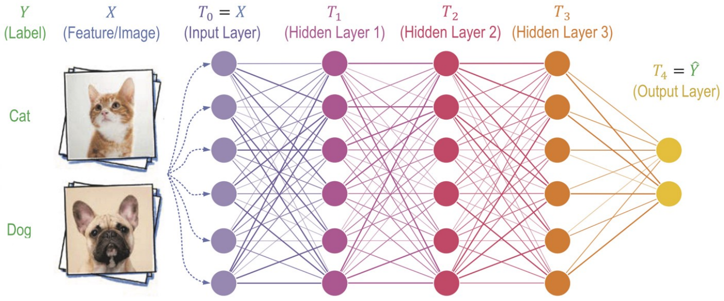
EPSRC
Engineering and Physical Sciences Research Council

Deep Neural Networks Deployment

- Brining the success of deep learning to the ”sensor” side
- Running machine learning tasks on the edge, in real time
- Preserving data privacy and reducing the dependency on the network access

A Recap on Deep Neural Networks Inference and Deployment

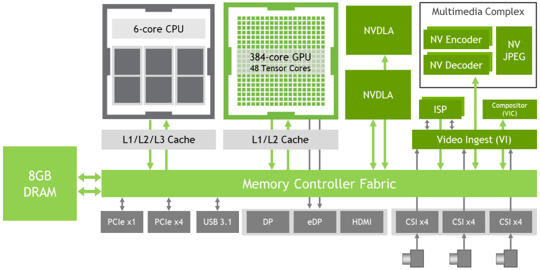
Deep Neural Networks



https://bit.ly/30V55AC

- Consecutive **multiplication**, **additions**, and **non-linear operators**; $\hat{Y} = f_4(f_3(\dots f_2(\mathbf{W2} * f_1(\mathbf{W1} * X + \mathbf{b1}) + \mathbf{b2})))$.

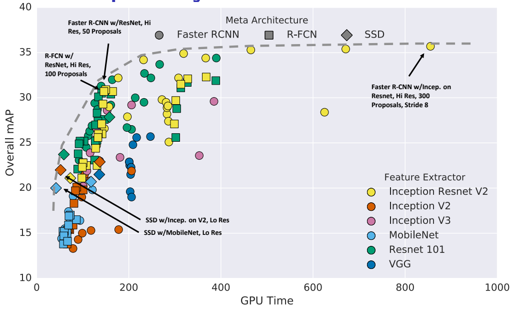
Embedded Deep Neural Networks Deployments



Jetson Xavier NX processor engines, high-speed I/O, and memory fabric

- Challenges to be addressed:
  1. Computational complexity
  2. Memory limitation
  3. Communication bandwidth

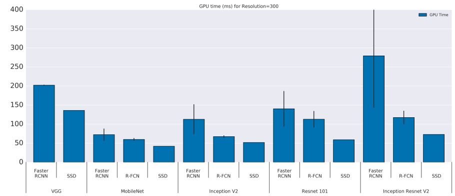
Computational Complexity

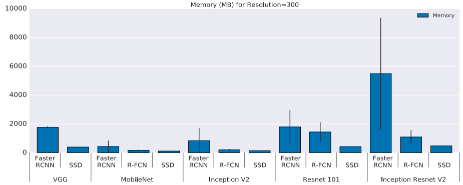


arXiv:1611.10012v3

- While DNN inference needs much less computational resources than learning, **real time** implementations on power constrained embedded platforms are still challenging.
- More computation gives more accuracy, *i.e.* mean Average Precision (mAP)

Memory Limitation

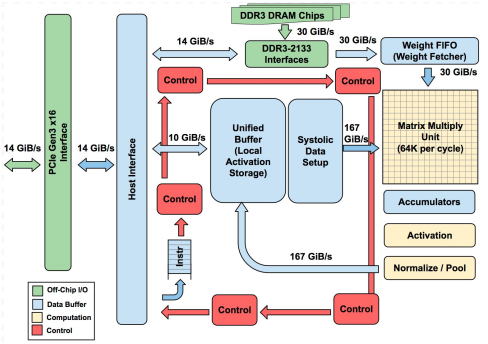




arXiv:1611.10012v3

- Higher capacity networks often need more memory
- Latency in memory read-out can be the bottleneck

I/O and Communication Bandwidths and Delays

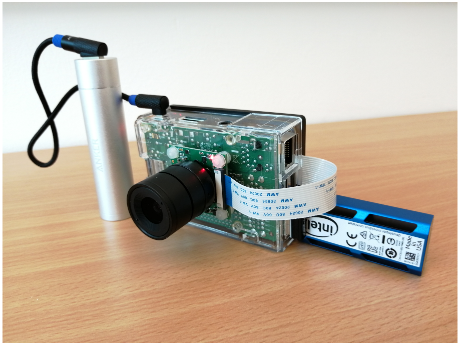


Google (HPC) TPU structure. This bandwidth is not achievable in the embedded TPUs.

- Issues in acquiring full rate signals/images
- Asynchronous sensor measurements
- Time delays in measurements

ASIC/FPGA AI Accelerators

Hardware Platforms and AI Accelerators

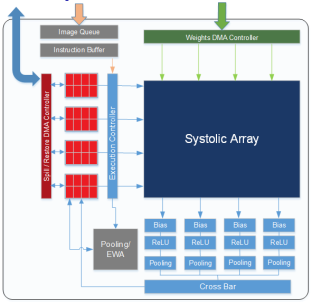


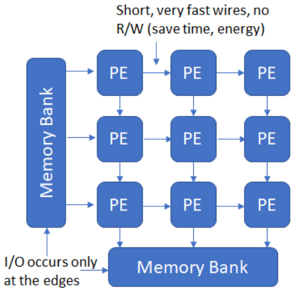
- Low power CPU or Microcontroller, *e.g.* ARM, RISC-V
- I/O interfaces and sensors, *e.g.* camera(s), microphones, depth-sensors
- ASIC or FPGA accelerators, *e.g.* GPU, TPU and RISC many-core processors

AI Accelerators

- Matrix-matrix multiplication accelerator,
  Multiplier-Accumulator (MAC), Systolic MAC
- Structured fast memory,
  Multi-level dedicated/shared caches (L1/L2 cache GPU)
- Recursive implementation facilitators

Systolic Multiplier-Accumulator

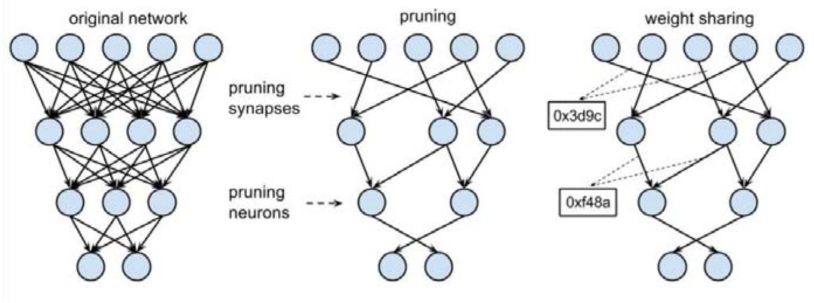



Short, very fast wires, no R/W (save time, energy)
Memory Bank
PE
I/O occurs only at the edges
Memory Bank

"Exploration and Tradeoffs of Different Kernels in FPGA Deep Learning Applications" by Elliott Delaye, 2018

- The breakthrough in computation is in the systolic MAC or massive parallel processing elements (PE),
- PEs in a systolic MAC are configured for one- (few-) shot tensor multiplications
- PEs are suitable for scaling up array sizes.

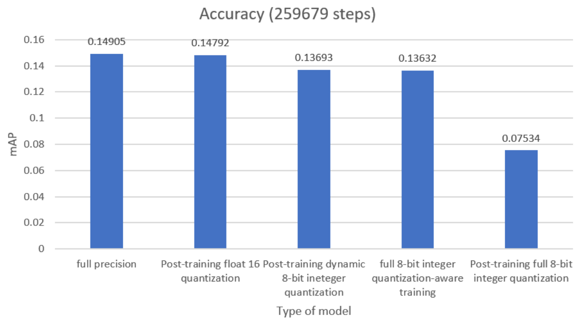
Embedded Deep Models



"Exploration and Tradeoffs of Different Kernels in FPGA Deep Learning Applications" by Elliott Delaye, 2018

- Learned models with highest mAP: over-parametrized with double precision weights
- Model simplification with pruning: cutting ineffective weights and sharing weights
- Model quantization: not losing much using quantized models

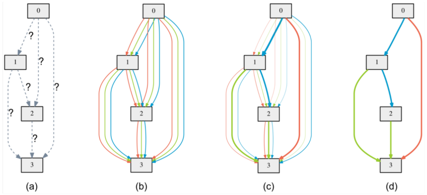
Quantised Models



mAP of MobileNetV2 SSD in different settings

- Full precision models: computationally and memory expensive
- Simple *post-quantization*: degrades mAP performance
- Quantized learning and dynamic post-quantization : showing close performance in a comparison with full precision.

Neural Architecture Search (NAS)



An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

"DARTS: Differentiable Architecture Search", by Hanxiao Liu, et al., 2019

- α_is: architecture parametrises, ω_is: network weights
- NAS task to find the pair (α, ω). Computationally extremely expensive
- DART makes the parameters continues and solve the problem by bilevel optimisation

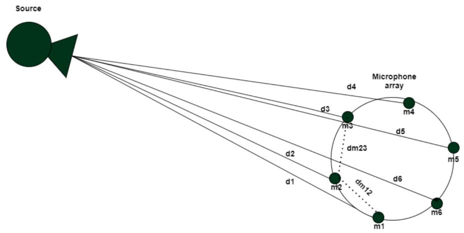
Take-Home Messages from First Part

- Various benefits in **on-the-device** and **on-the-edge** computing
- **Embedded DNN Deployment Hardware**
  - Powerful **AI accelerators** are available now and they will be highly optimised in the near future
  - Each structure has its pros and cones. They mainly accelerate **parallel MAC** operations
  - Integration of multiple AI units accelerates almost linearly
- **Embedded DNN Deployment Models**
  - **Pruning:** sparse and shared weights
  - **Quantization:** acceleration given a fixed IC footprint and lower power consumption
  - **NAS:** search for very efficient network, if we can afford the computation

Example and Demo

Prepared by:
Theodoros Papaiakovou and Tomek Horszczaruk

Direction of Arrival Estimations with Circular Microphone Arrays I

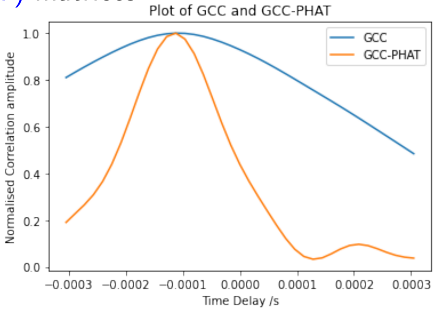


A typical circular microphone array w/o centre microphone.

- A popular configuration for home assistances, e.g. Alexa, Google Echo
- Have some defence applications, e.g. gunshot DoA estimation

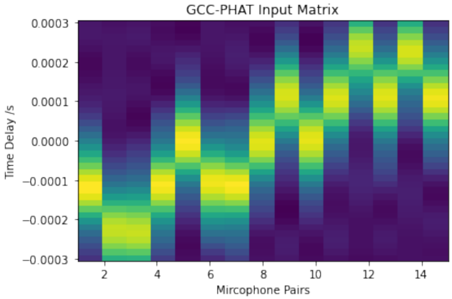
Direction of Arrival Estimations with Circular Microphone Arrays II

- Conventional techniques: incorporating time-delays on different microphones, e.g. MUSIC, ESPRIT, and many more
- Computationally cheap methods often rely on Generalised Cross Correlation (GCC) and GCC with Phase Transform (GCC_PHAT) matrices



Direction of Arrival Estimations with Circular Microphone Arrays III

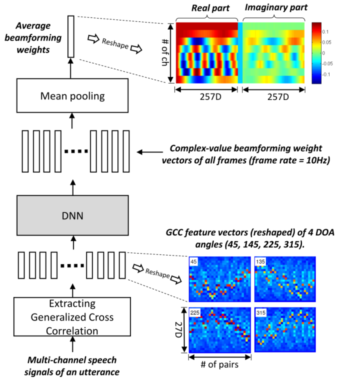
- Have some limitations form complex cases of very noisy and/or reverberant environments.
- Can some of these ambiguities be leaned from data and mitigated? Some studies have done, but not fully understood!



39 x 15 GCC-PHAT inputs (yellow stripes indicate TDOA peaks for different microphone pairs)

Deep Direction of Arrival Estimations

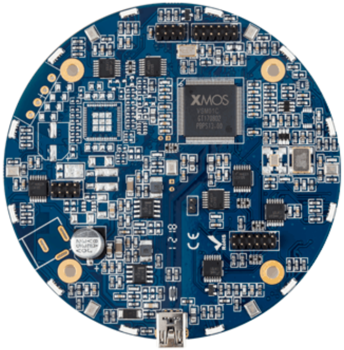
- GCC_PHAT matrix can be used as the input of DNN to construct beamforming weights and estimate DoA and extract the signal
- The task of DoA estimation from GCC_PHAT matrix is a classification task with [0, 360), and desirable resolution, class labels
- Depending on the single or multiple source setting, it is a **multiclass** or **multilabel** classification task


Real part
Imaginary part
Average beamforming weights
Reshape
# of ch
257D
257D
Mean pooling
Complex-value beamforming weight vectors of all frames (frame rate = 10Hz)
DNN
GCC feature vectors (reshaped) of 4 DOA angles (45, 145, 225, 315).
Reshape
45
135
225
315
27D
# of pairs
Extracting Generalized Cross Correlation
Multi-channel speech signals of an utterance

Microphone Array and Computation Setup

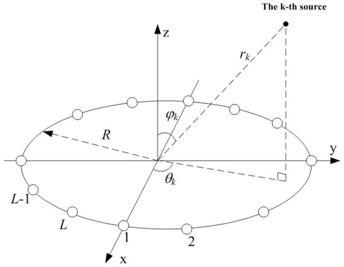
- MiniDSP UMA-8 USB mic array - V2.0: A 6 way microphone array
- Raspberry Pi 4 - 4 GB RAM, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Tensorflow (TF), Keras and Tensorflow Lite (TFLite)
- PyRoomAcoustics for simulation and training data generation

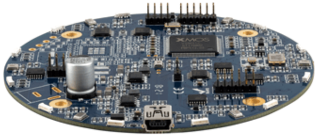
pyroomacoustics



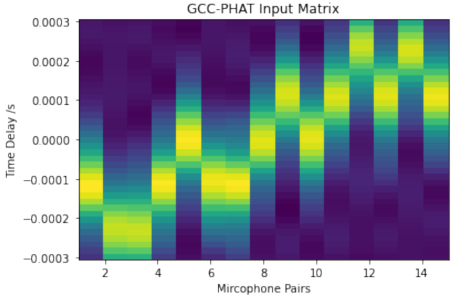
Deep 3D Direction of Arrival Estimation

- The task is **only** DoA estimation with no beamforming
- Due to circular configuration we aim for 3D DoA, which is theoretically possible but generally difficult for azimuth angle estimation
- The problem is broken to estimation of θ_k and ϕ_k for each source
- The resolution chosen for this demonstration is 10 degrees

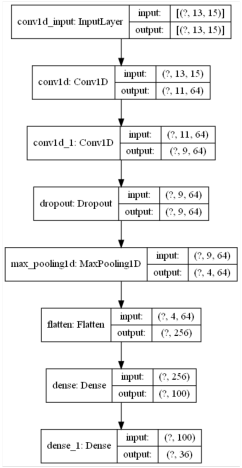




Convolutional Neural Network for DoA Estimation I



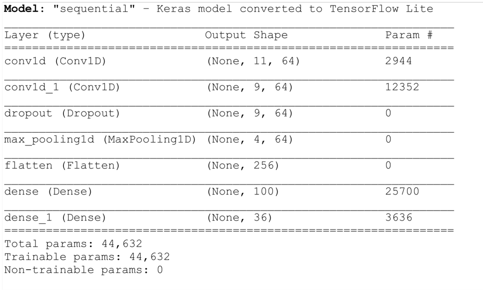
39 x 15 GCC-PHAT inputs (yellow stripes indicate TDOA peaks for different microphone pairs)



- A LeNet5 type CNN with GCC_PHAT and 36-class output for bearing angle
- A similar topology but with [0:10:90]-classes for the azimuth angle

Convolutional Neural Network for DoA Estimation II

- Overparametrization of CNNs helps to overcome the complex optimisation landscape
- We need a rich training set including different angles and source to microphone distances



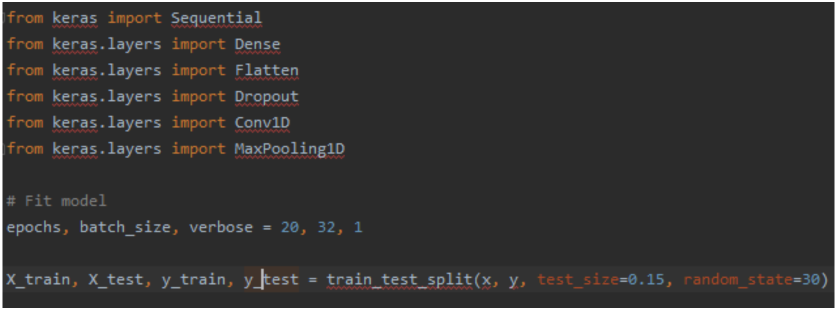
```
Model: "sequential" - Keras model converted to TensorFlow Lite
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 11, 64)            2944
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 9, 64)             12352
_________________________________________________________________
dropout (Dropout)            (None, 9, 64)             0
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 4, 64)             0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 100)               25700
_________________________________________________________________
dense_1 (Dense)              (None, 36)                3636
=================================================================
Total params: 44,632
Trainable params: 44,632
Non-trainable params: 0
```

- Such a dataset can be generated using acoustic simulators, e.g. pyroomacoustics
- Noisy and reverberant environments can be included to provide robustness to the algorithm

Setup

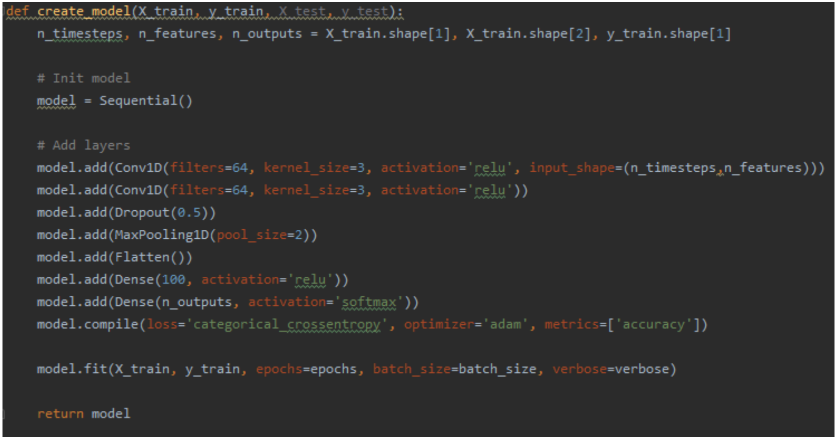
- We here rely on synthetically generated training-data and fine tuning with a few real samples, i.e. using pre-training model
- If enough real training dataset is available, no need for **data augmentation** or **model pre-training**
- We only consider supervised learning here
- We here are not discussing about **imbalanced classification**, **overfitting/underfitting**, and **generalisation** issues or alternative learning regimes, e.g. **few shot**, **semi-supervised** and **self-supervised** learnings
- Keras/TF development platform is selected for the ease of use and embedding

Keras Code Structure: Importing Packages



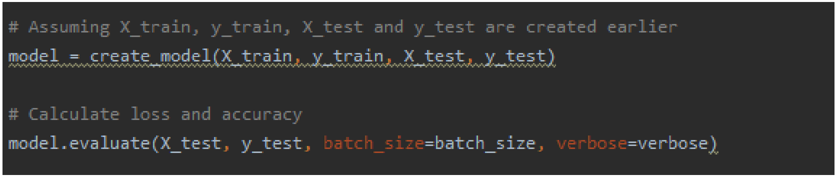
```
from keras import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import Conv1D
from keras.layers import MaxPooling1D

# Fit model
epochs, batch_size, verbose = 20, 32, 1

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.15, random_state=30)
```

Keras Code Structure: Model Generation



```
def create_model(X_train, y_train, X_test, y_test):
    n_timesteps, n_features, n_outputs = X_train.shape[1], X_train.shape[2], y_train.shape[1]

    # Init model
    model = Sequential()

    # Add layers
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(n_timesteps,n_features)))
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
    model.add(Dropout(0.5))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=verbose)

    return model
```

Keras Code Structure: Training and Evaluation

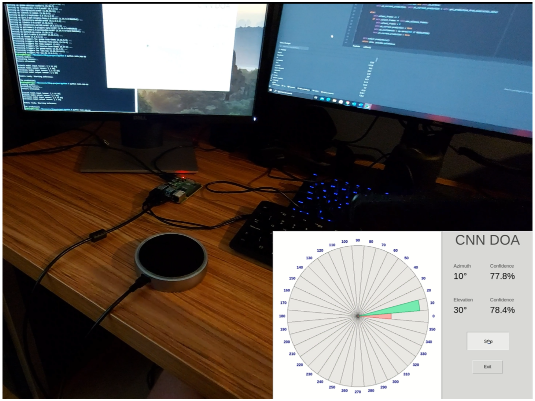


```
# Assuming X_train, y_train, X_test and y_test are created earlier
model = create_model(X_train, y_train, X_test, y_test)

# Calculate loss and accuracy
model.evaluate(X_test, y_test, batch_size=batch_size, verbose=verbose)
```

From Learning to Deployment

- Learned TF models can be deployed on full precision platforms, e.g. RPi
- TFLite has some tools to optimise the leaned model for faster deployments on embedded devices, e.g. weight pruning, etc.
- Deployment on AI accelerators often needs model quantisation, e.g. Embedded TPU, or special multi-core optimisation tools, e.g. Intel Movidius.
- **Quantise learning** is possible in TFLite and sometime benefits

Audio DoA Estimation on a RPi 4



Click for video

Created by Tomek Horszczaruk

Take-Home Messages from Second Part

- Knowing the **physics of problem** and **computational platform** are useful for appropriate ML algorithm selection
- Part of model based solutions, e.g. GCC_PHAT, can help ML by reducing sample complexity
- Try to simplify the DNN/CNN as much as we are happy with accuracy, then try to deploy it
- DL could learn to conduct difficult tasks for model based methods, e.g. azimuth angle estimation
- Some DL tasks cannot be done with only real data, i.e. unavailability, and we need to deal with the only few real data and use prior domain knowledge, data augmentation, or few shot learning
- Test possible model quantization and quantized learning, if the target embedded platform has limited precision

UDRC

EPSRC
Engineering and Physical Sciences
Research Council

dstl