



The Abdus Salam
**International Centre
for Theoretical Physics**

FPGA for Accelerating Machine Learning Algorithms

Romina Molina

Multidisciplinary Laboratory

Joint ICTP-IAEA School on
FPGA-based SoC and its
Applications to Nuclear and
Scientific Instrumentation



14 November - 2 December 2022
An ICTP - IAEA Hybrid Meeting
Trieste, Italy

Further information:
<http://indico.ictp.it/event/9933/>
smr3765@ictp.it

Outline

Outline

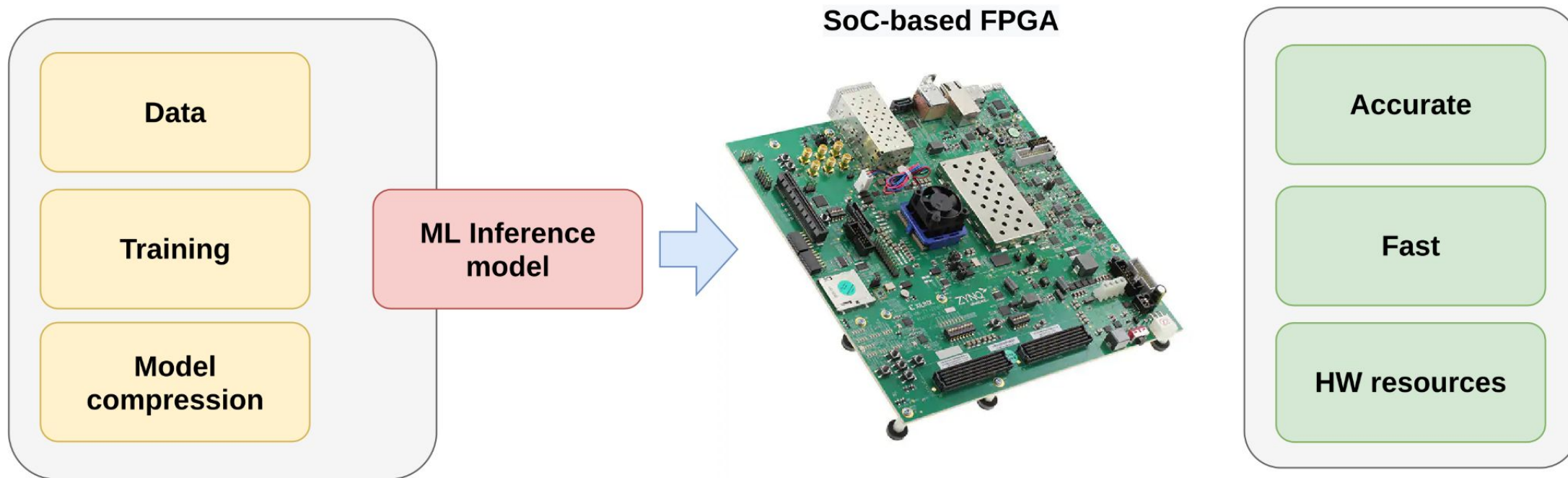
- Introduction.
- Machine Learning (ML).
- Acceleration of machine learning inference.
 - The general workflow.
- High-Level Synthesis for machine learning (hls4ml).
- Acceleration of ML-based applications.
 - Pulse Shape Discrimination for Water Cherenkov Detectors.
 - PYNQ-Z1 implementation.
 - Image classification based on CNN.



Introduction

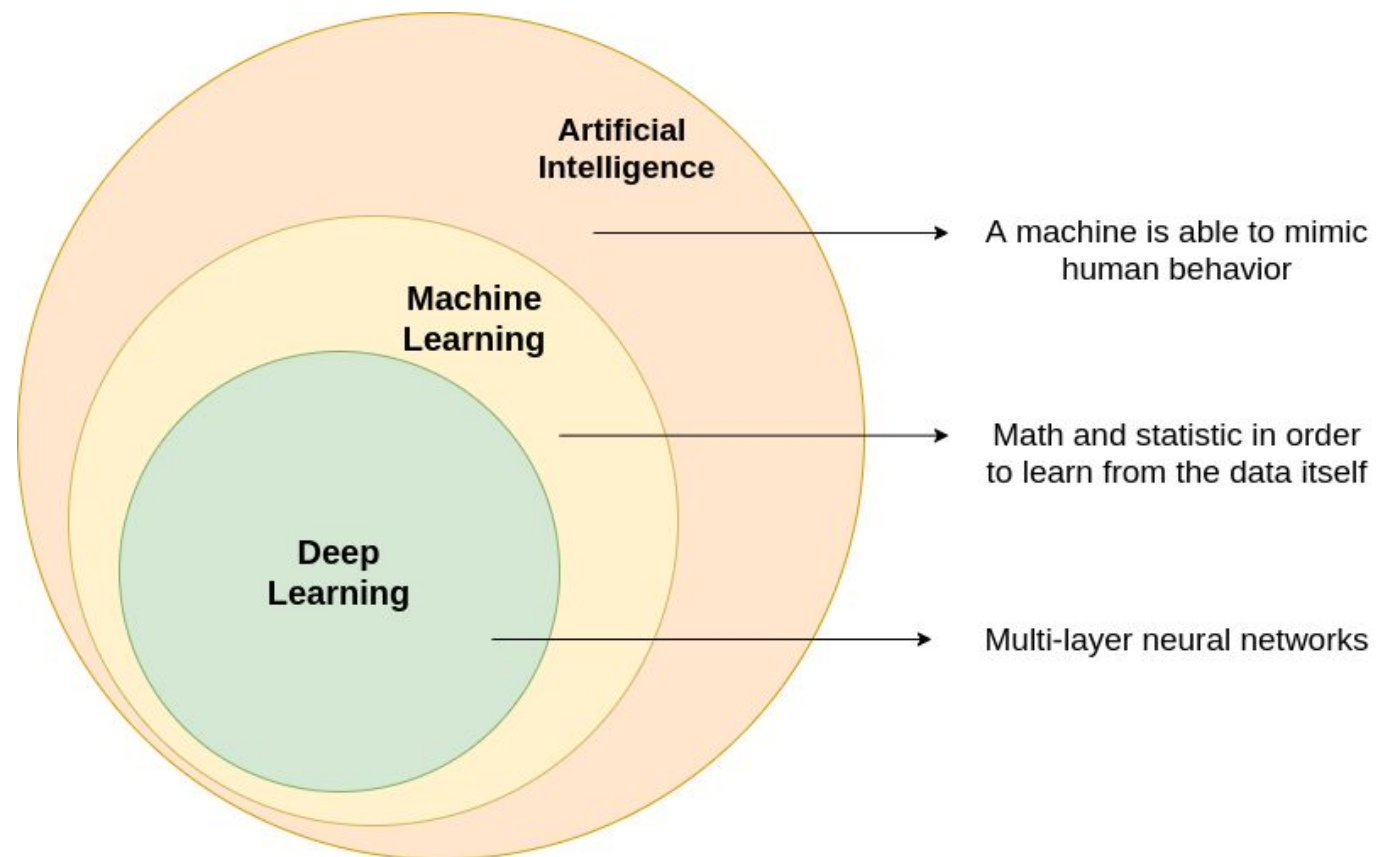
Introduction

Machine Learning and System On Chip

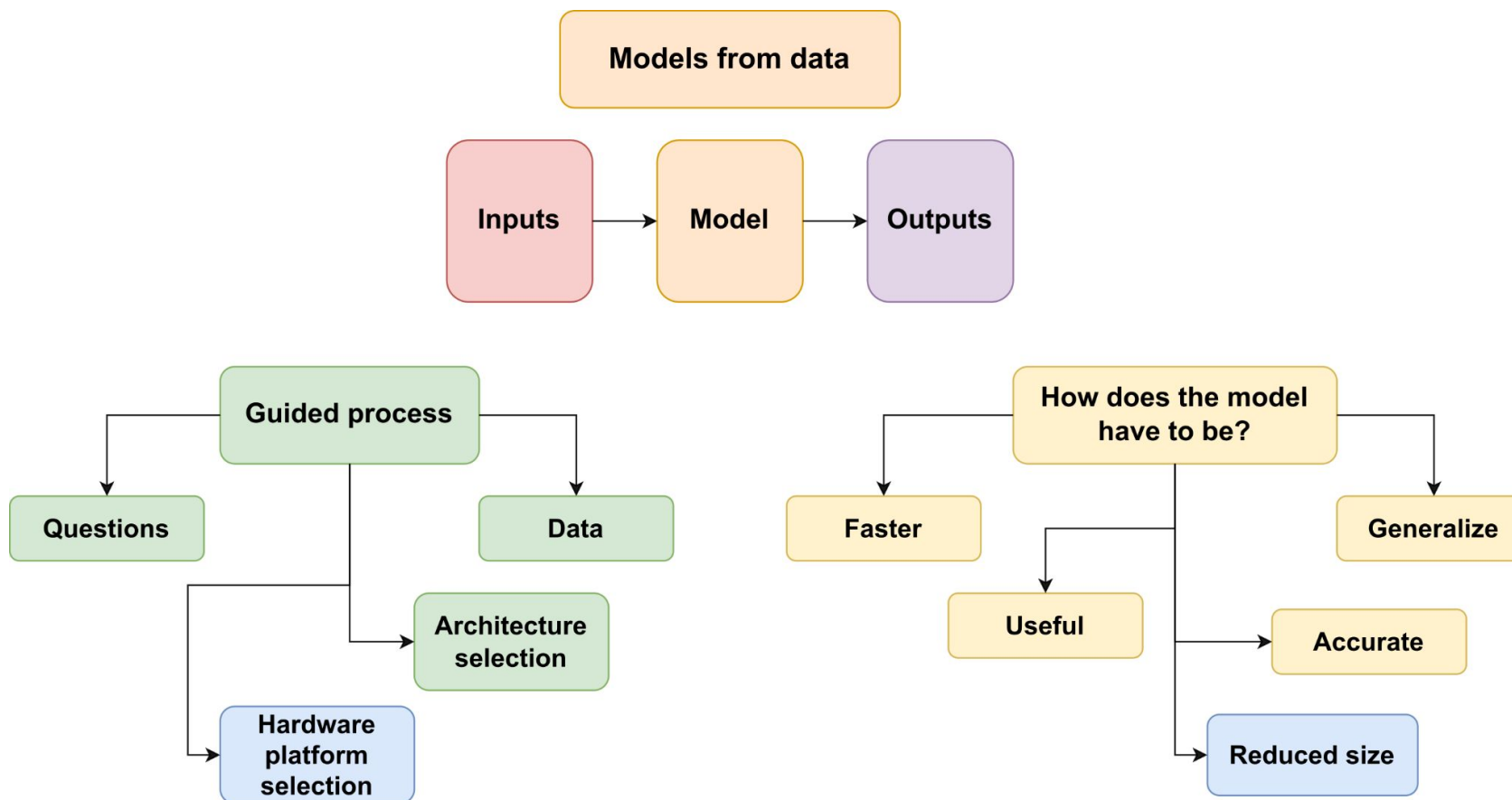


Machine Learning

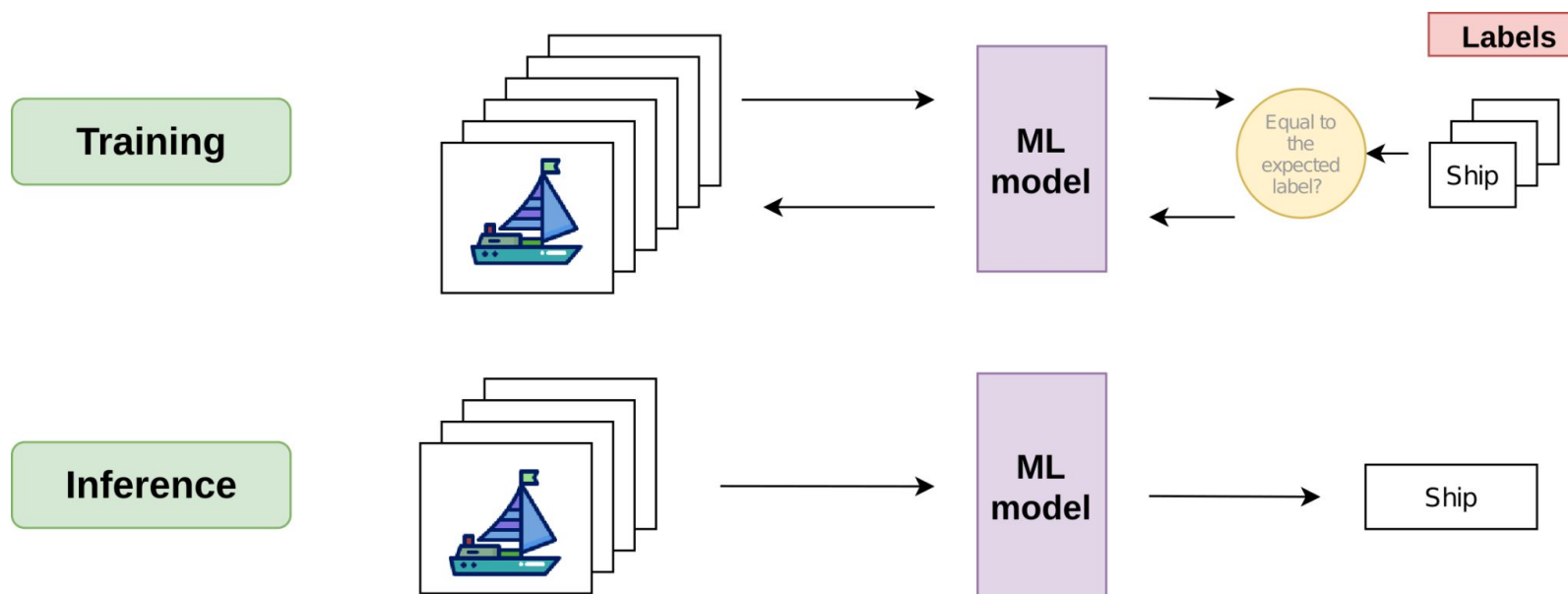
Machine Learning



Machine Learning



Machine Learning

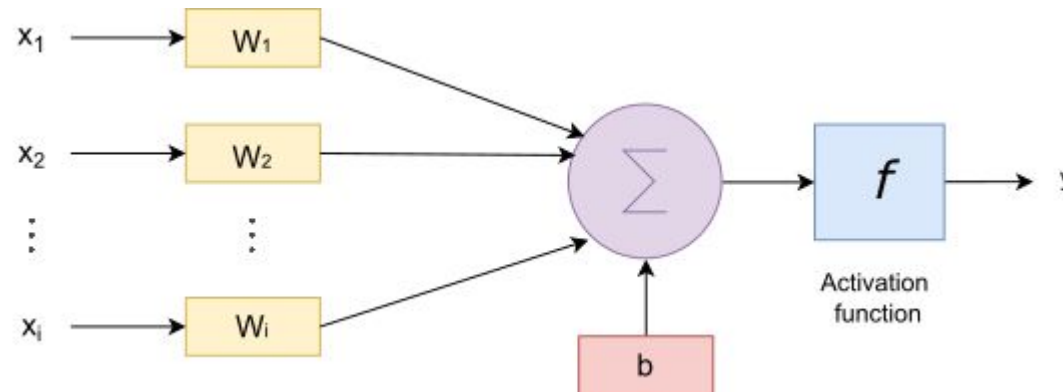


- In a classifier, an input is mapped into a specific class.
- Supervised training step to recognize patterns: the network compares its actual output with the desired output. The difference between these two values is adjusted with backpropagation.

Machine Learning

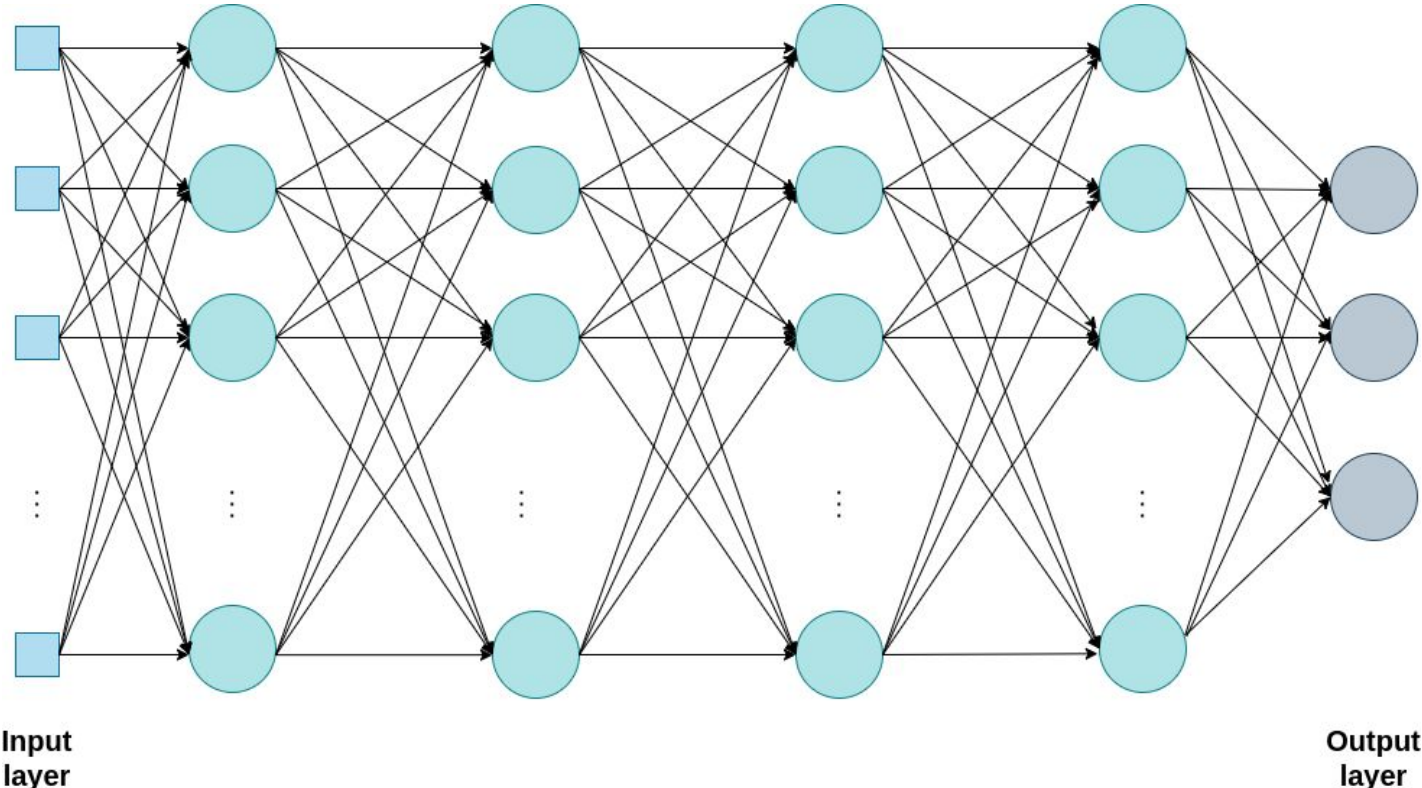
Artificial Neural Network

An Artificial Neural Network (ANN) is composed of neuron (or node) interconnections arranged in different layers.



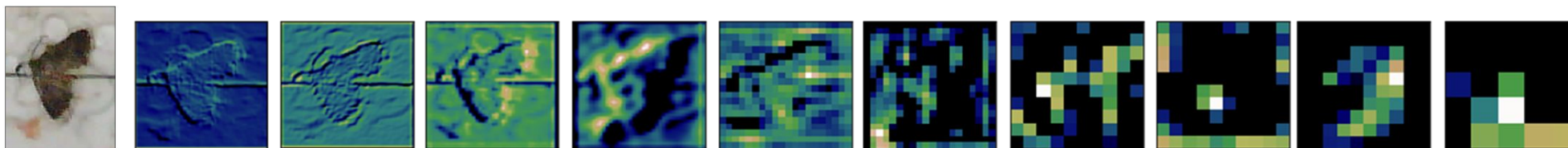
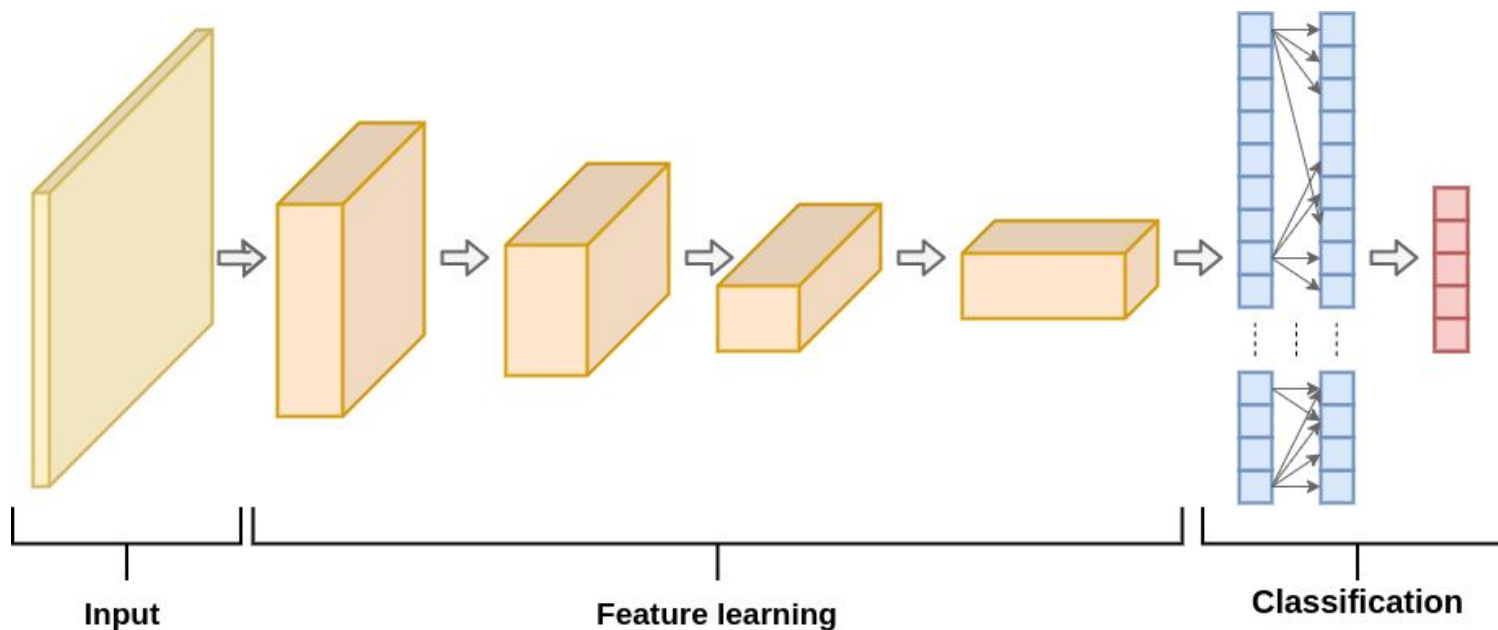
Machine Learning

Multi-Layer Perceptron (MLP) architecture



Machine Learning

Convolutional Neural Networks (CNN) architecture



A solid blue vertical bar on the left side of the slide.

Acceleration of machine learning inference

Acceleration of ML Inference

Considerations to map inference into FPGAs



Clock frequency

Bandwidth off-chip memory

DSP, LUT, BRAM, FF

Fixed point

Power consumption

Acceleration of machine learning inference

The general workflow

The general workflow

Considerations to map inference into FPGAs

- **Low-precision arithmetic** to reduce power consumption and increase throughput.
- **Reduce memory footprint**
 - ML models can be deployed into on-chip memory, avoiding DDR access and bottlenecks.
- Model **compression techniques** [1]

The general workflow

Ensemble of compression techniques.

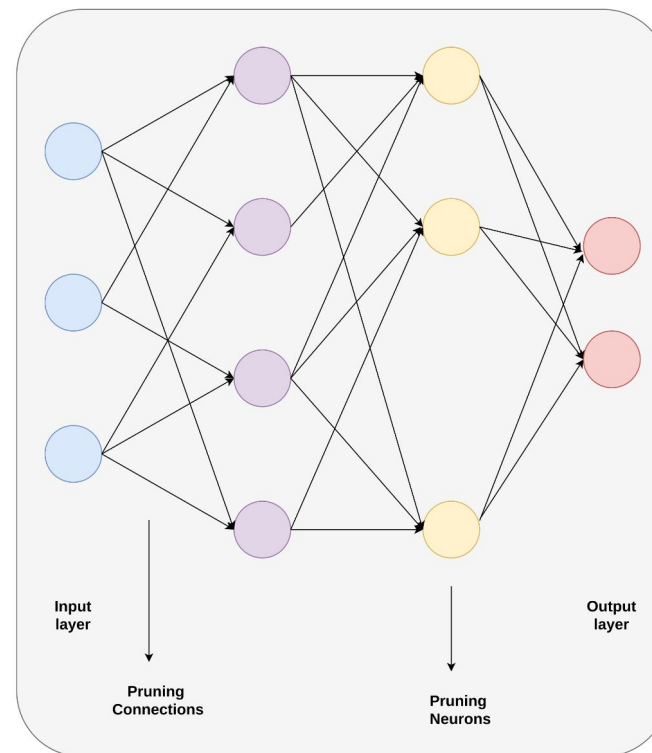
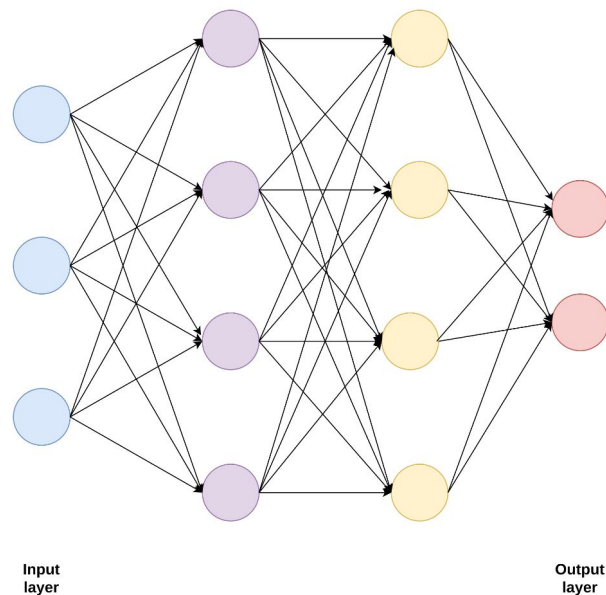
Exploration of the interplay between:

- **Pruning** [3] aims to reduce the number of operations by **removing neurons and connections**, and **quantization** [4] reduces the memory and computational complexity by selecting the **number of bits to represent the weights and bias**.
- **Knowledge distillation** transfers the knowledge (or "dark knowledge" according to Hinton [2]) from a **teacher network** (a single large model or an ensemble of models) to a **smaller and faster target network** (distilled or student) that is able to mimic the teacher's behaviour, being computationally less expensive.

Acceleration of ML Inference

Considerations to map inference into FPGAs

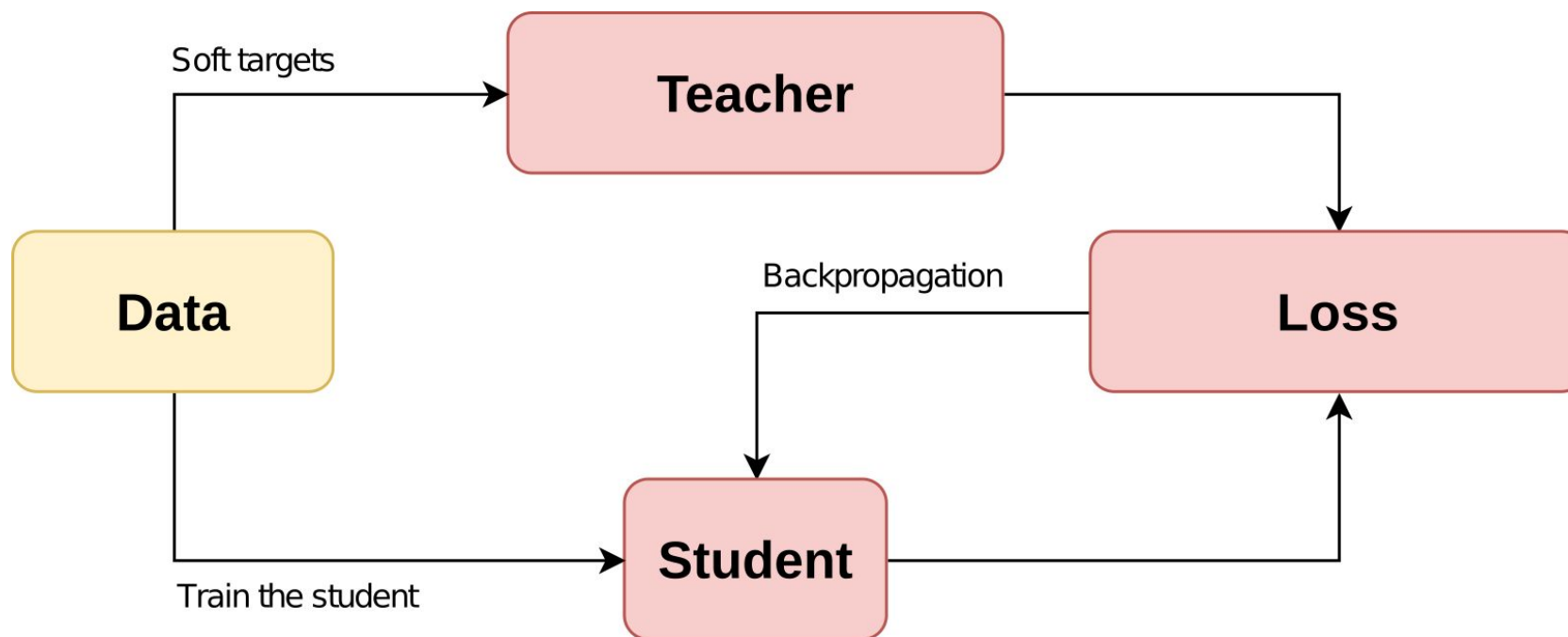
- Model Compression: Pruning



Acceleration of ML Inference

Considerations to map inference into FPGAs

- Model Compression: Knowledge Distillation



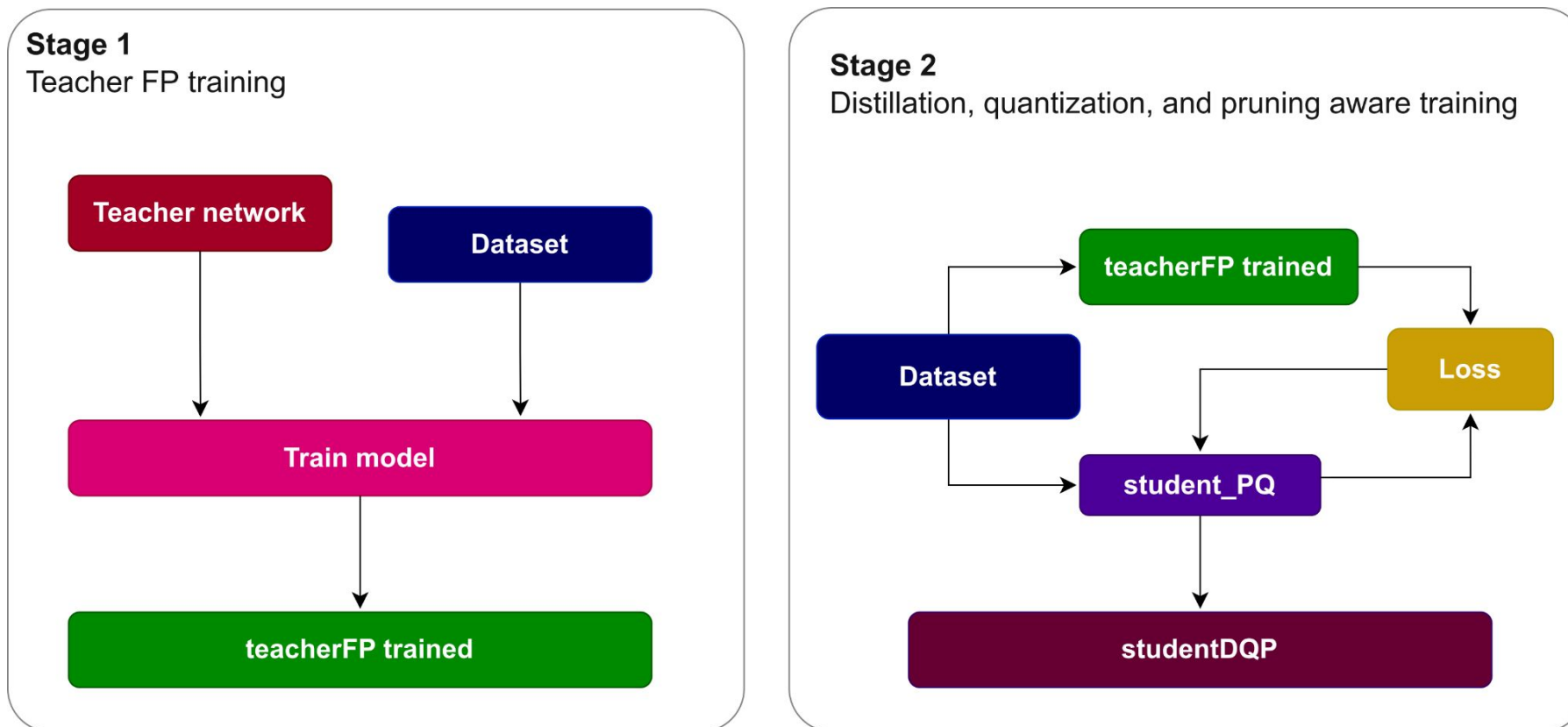
The general workflow


Ensemble of compression techniques.

- **Pruning and quantization are orthogonal to distillation**, helping to achieve a better performance, reducing the size of the model with minimum loss of accuracy.
- Quantization and pruning (train from scratch and pre-trained model)
- Knowledge distillation (train from scratch)

The general workflow

Deploy ML-based application on embedded systems



A solid blue vertical bar on the left side of the slide.

High-Level Synthesis for machine learning (hls4ml)

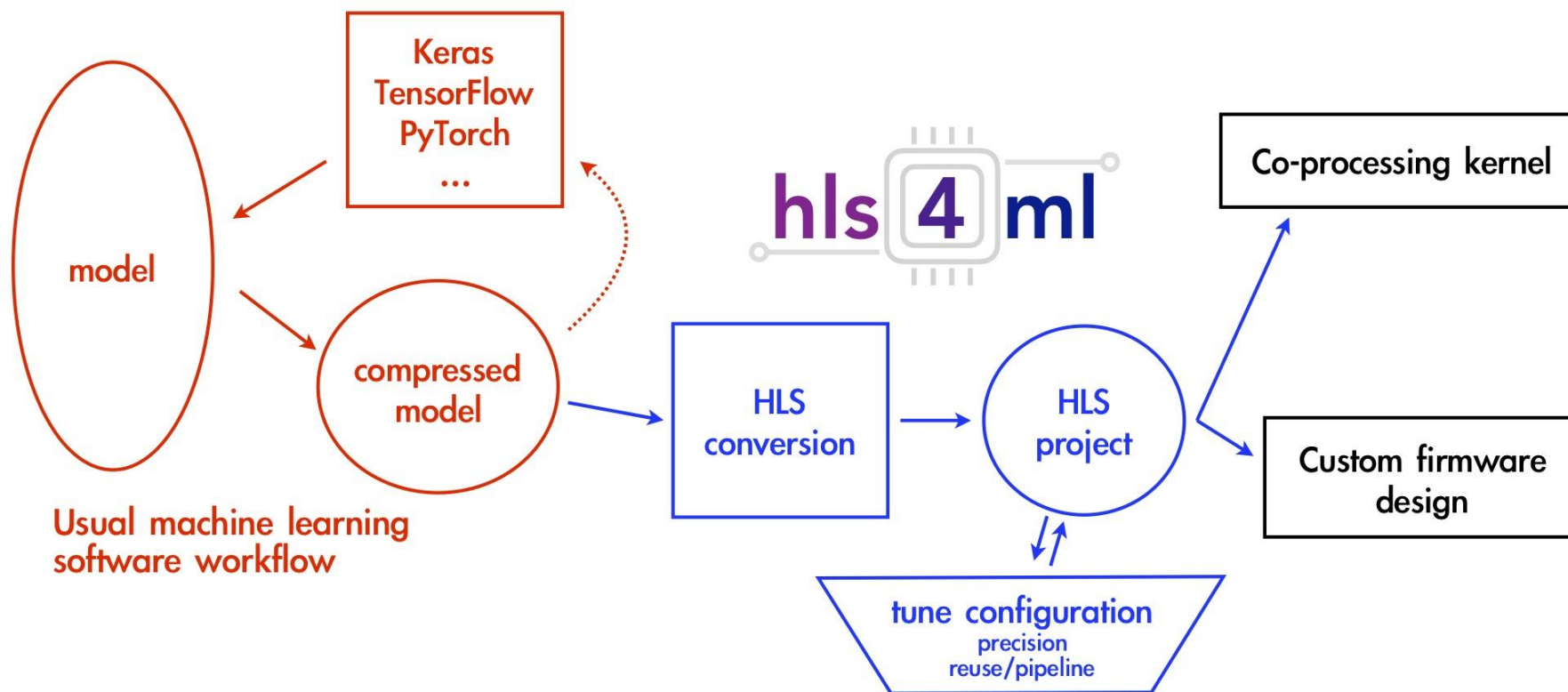
High-Level Synthesis for machine learning (hls4ml)

- Package for ML inference on SoC-FPGAs using HLS. (Duarte et. al)
- "Fast inference of deep neural networks (DNN) in FPGAs for particle physics" Duarte et al.[5]
- GitHub: <https://github.com/fastmachinelearning/hls4ml-tutorial>
- <https://fastmachinelearning.org/hls4ml/>



High-Level Synthesis for ML (hls4ml)

Design flow



From [5]

High-Level Synthesis for ML (hls4ml)

Features:

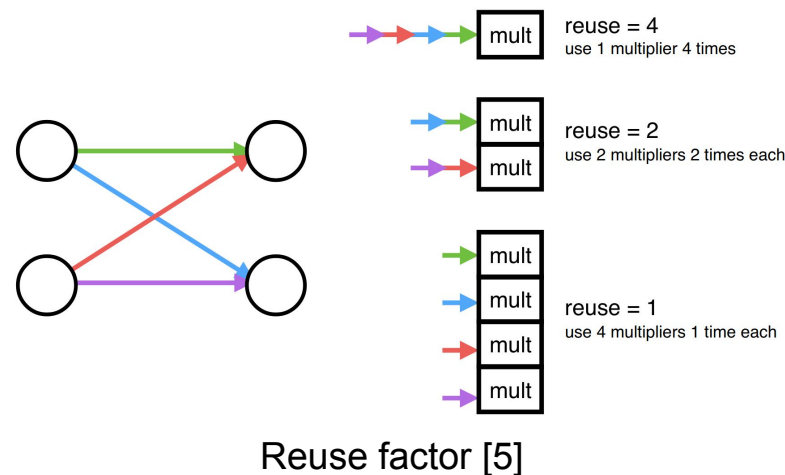
- HLS to create IP Core.
- Keras, TensorFlow, Pytorch.
- On-chip data structures.
- Quantization through `ap_fixed` data type in HLS.
 - typedef **`ap_ufixed<10,8> din`** (A 10-bit input: 8-bit integer value with 2 decimal places)
- Trade-off between resource utilization and latency/throughput.



High-Level Synthesis for ML (hls4ml)

Features:

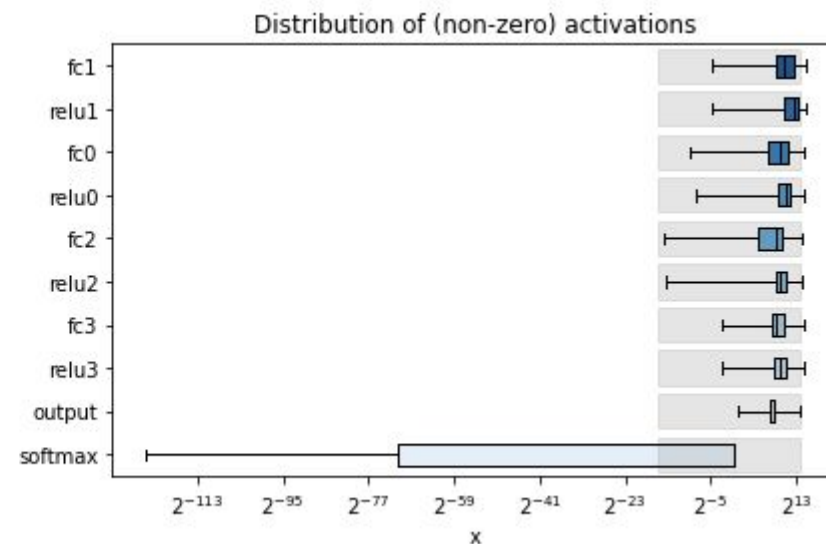
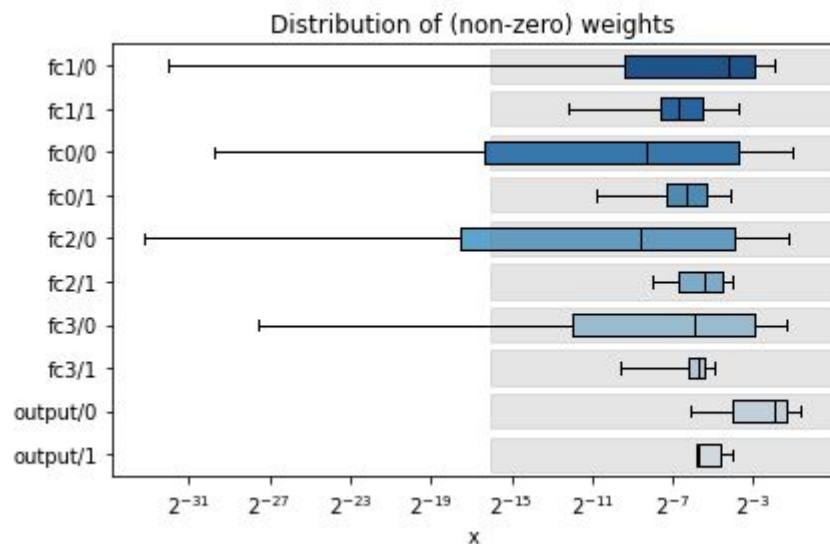
- Pipelining to speed up the process by accepting new inputs after an initiation interval.
- Size/Compression
- Precision
- Dataflow/Resource Reuse
- Quantization-aware training:
 - **Qkeras**



High-Level Synthesis for ML (hls4ml)

Features - Profiling

- Profiling to adjust precision



High-Level Synthesis for ML (hls4ml)

```

1 from tensorflow.keras.models import load_model
2 from sklearn.metrics import accuracy_score
3 model = load_model('model_keras_MLP.h5')
4 model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
fc1 (Dense)	(None, 60)	3900
relu1 (Activation)	(None, 60)	0
fc0 (Dense)	(None, 40)	2440
relu0 (Activation)	(None, 40)	0
fc2 (Dense)	(None, 30)	1230
relu2 (Activation)	(None, 30)	0
fc3 (Dense)	(None, 10)	310

High-Level Synthesis for ML (hls4ml)

```
import hls4ml
import plotting
hls4ml.model.optimizer.OutputRoundingSaturationMode.layers = ['Activation']
hls4ml.model.optimizer.OutputRoundingSaturationMode.rounding_mode = 'AP_RND'
hls4ml.model.optimizer.OutputRoundingSaturationMode.saturation_mode = 'AP_SAT'

config = hls4ml.utils.config_from_keras_model(model, granularity='name')

config['Model'] = {'Precision' : 'ap_fixed<17,16>', 'ReuseFactor' : 1, 'Strategy' : 'Latency'}

config['LayerName']['fc1']['Precision']['weight'] = 'ap_fixed<9, 1>'

config['LayerName']['softmax']['Precision'] = 'ap_fixed<32,15>'

print("-----")
plotting.print_dict(config)
print("-----")
hls_model = hls4ml.converters.convert_from_keras_model(model,
                                                       hls_config=config,
                                                       output_dir='model_3/MLP_student_smr3765'
                                                       )

hls_model.compile()
```

High-Level Synthesis for ML (hls4ml)

```
import hls4ml
import plotting

config = hls4ml.utils.config_from_keras_model(teacherMLP, granularity='model')
config['Model'] = {'Precision' : 'ap_fixed<24,16>', 'ReuseFactor' : 1, 'Strategy' : 'Latency'}
print("-----")
plotting.print_dict(config)
print("-----")
hls_model = hls4ml.converters.convert_from_keras_model(teacherMLP,
                                                       hls_config=config,
                                                       output_dir='model_3/hls_model'
                                                       )

hls_model.compile()
```

High-Level Synthesis for ML (hls4ml)

Network description generated inside HLS project

```

63
64 layer3_t layer3_out[N_LAYER_3];
65 #pragma HLS ARRAY_PARTITION variable=layer3_out complete dim=0
66 nnet::dense_latency<input2_t, layer3_t, config3>(input1, layer3_out, w3, b3);
67
68 layer5_t layer5_out[N_LAYER_3];
69 #pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
70 nnet::relu<layer3_t, layer5_t, relu_config5>(layer3_out, layer5_out);
71
72 layer6_t layer6_out[N_LAYER_6];
73 #pragma HLS ARRAY_PARTITION variable=layer6_out complete dim=0
74 nnet::dense_latency<layer5_t, layer6_t, config6>(layer5_out, layer6_out, w6, b6);
75
76 layer8_t layer8_out[N_LAYER_6];
77 #pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
78 nnet::relu<layer6_t, layer8_t, relu_config8>(layer6_out, layer8_out);
79
80 layer9_t layer9_out[N_LAYER_9];
81 #pragma HLS ARRAY_PARTITION variable=layer9_out complete dim=0
82 nnet::dense_latency<layer8_t, layer9_t, config9>(layer8_out, layer9_out, w9, b9);
83
84 layer11_t layer11_out[N_LAYER_9];
85 #pragma HLS ARRAY_PARTITION variable=layer11_out complete dim=0
86 nnet::relu<layer9_t, layer11_t, relu_config11>(layer9_out, layer11_out);
87
88 layer12_t layer12_out[N_LAYER_12];
89 #pragma HLS ARRAY_PARTITION variable=layer12_out complete dim=0
90 nnet::dense_latency<layer11_t, layer12_t, config12>(layer11_out, layer12_out, w12, b12);
91
92 layer14_t layer14_out[N_LAYER_12];
93 #pragma HLS ARRAY_PARTITION variable=layer14_out complete dim=0
94 nnet::relu<layer12_t, layer14_t, relu_config14>(layer12_out, layer14_out);
95
96 layer15_t layer15_out[N_LAYER_15];
97 #pragma HLS ARRAY_PARTITION variable=layer15_out complete dim=0
98 nnet::dense_latency<layer14_t, layer15_t, config15>(layer14_out, layer15_out, w15, b15);
99
100 nnet::softmax<layer15_t, result_t, softmax_config17>(layer15_out, layer17_out);
101

```


High-Level Synthesis for ML (hls4ml)

Qkeras for quantization-aware training

```
# MLP architecture
# Create the student QKERAS
studentQ_MLP = keras.Sequential(
    [
        Input(shape=(30,)),
        QDense(20, name='fc1',
              kernel_quantizer=quantized_bits(9,1,alpha=1), bias_quantizer=quantized_bits(23,15,alpha=1)),
        QActivation(activation=quantized_relu(16,15), name='relu1'),
        QDense(10, name='fc2',
              kernel_quantizer=quantized_bits(9,1,alpha=1), bias_quantizer=quantized_bits(23,15,alpha=1)),
        QActivation(activation=quantized_relu(16,15), name='relu2'),
        QDense(10, name='fc6',
              kernel_quantizer=quantized_bits(9,1,alpha=1), bias_quantizer=quantized_bits(23,15,alpha=1)),
        QActivation(activation=quantized_relu(16,15), name='relu3'),

        QDense(4, name='output',
              kernel_quantizer=quantized_bits(32,15,alpha=1), bias_quantizer=quantized_bits(32,15,alpha=1)),
        Activation(activation='softmax', name='softmax')

    ],
    name="student",
)

print_qstats(studentQ_MLP)
```

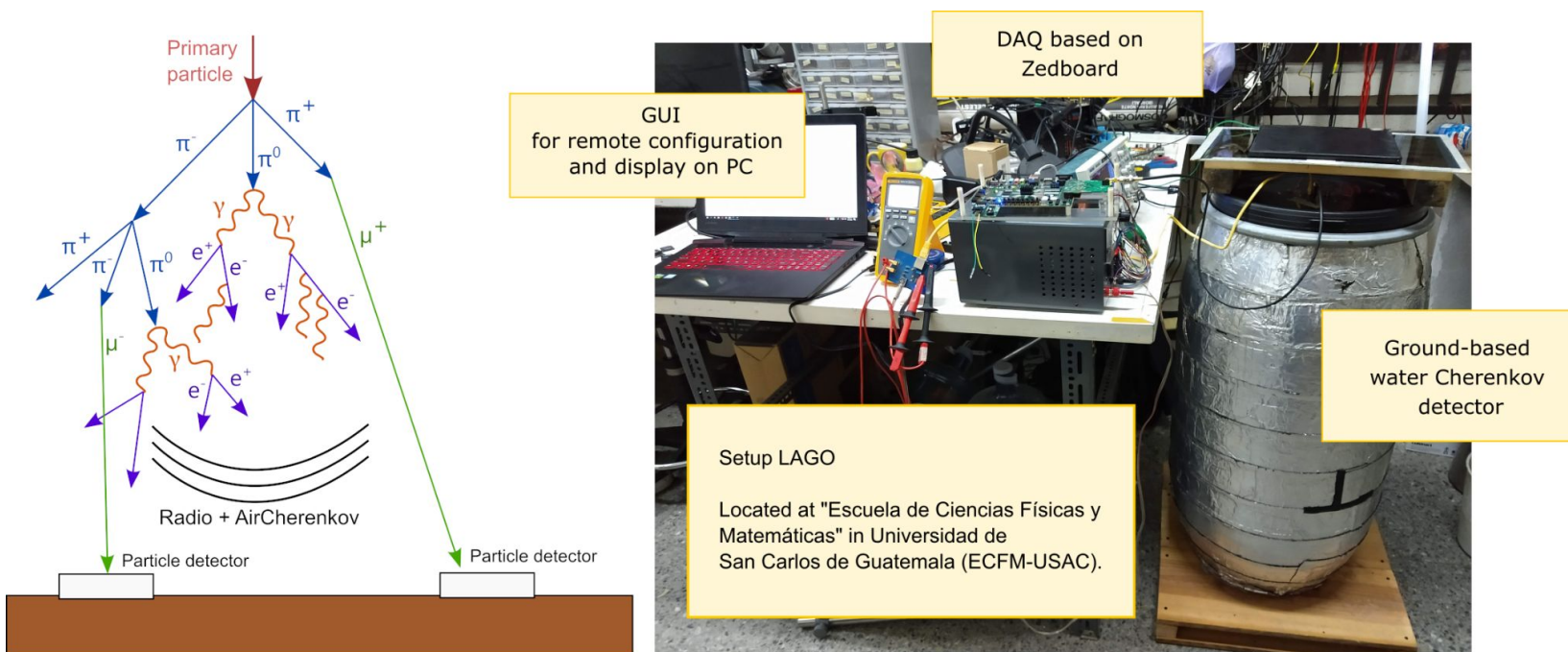
A solid blue vertical bar on the left side of the slide.

Acceleration of ML-based applications

**Pulse shape
discrimination for water
Cherenkov detectors
(WCD)**

Pulse shape discrimination for WCD

Experimental Setup



- Molina, R.S. et al. 2022. "Compression of NN-Based Pulse-Shape Discriminators in Front-End Electronics for Particle Detection". In: Saponara, S., De Gloria, A. (eds) Applications in Electronics Pervading Industry, Environment and Society. ApplePies 2021. Lecture Notes in Electrical Engineering, vol 866. Springer, Cham.
- Garcia, L. G.; Molina, R.S. ; Crespo, M. L.; Carrato S.; Ramponi, G.; Cicuttin, A.; Morales, I. R., Perez, H. "Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors Based on FPGA/SoC". In: Electronics. 2021; 10(3):224.

Pulse shape discrimination for WCD

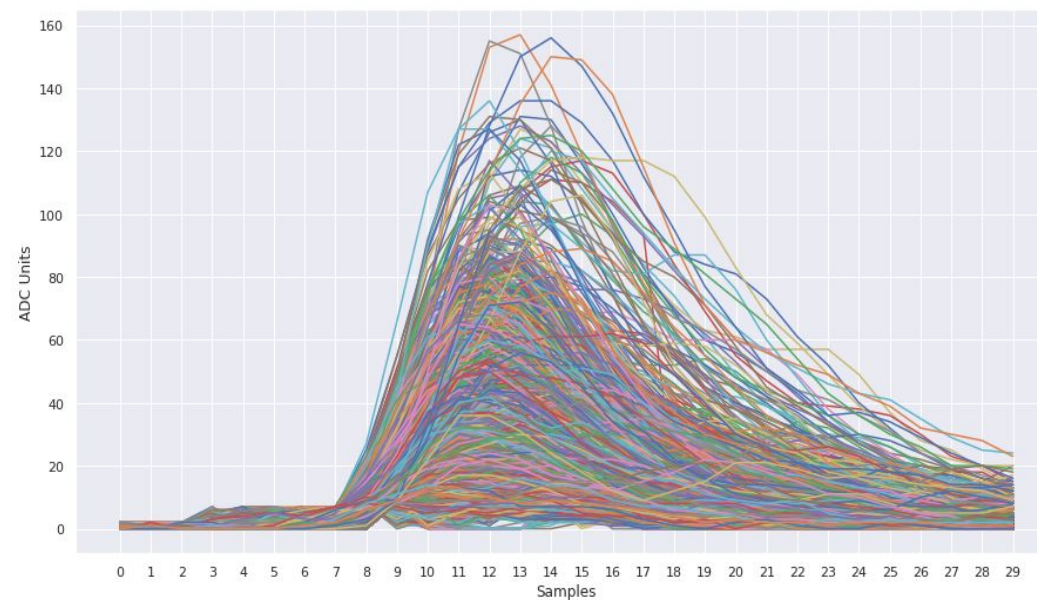
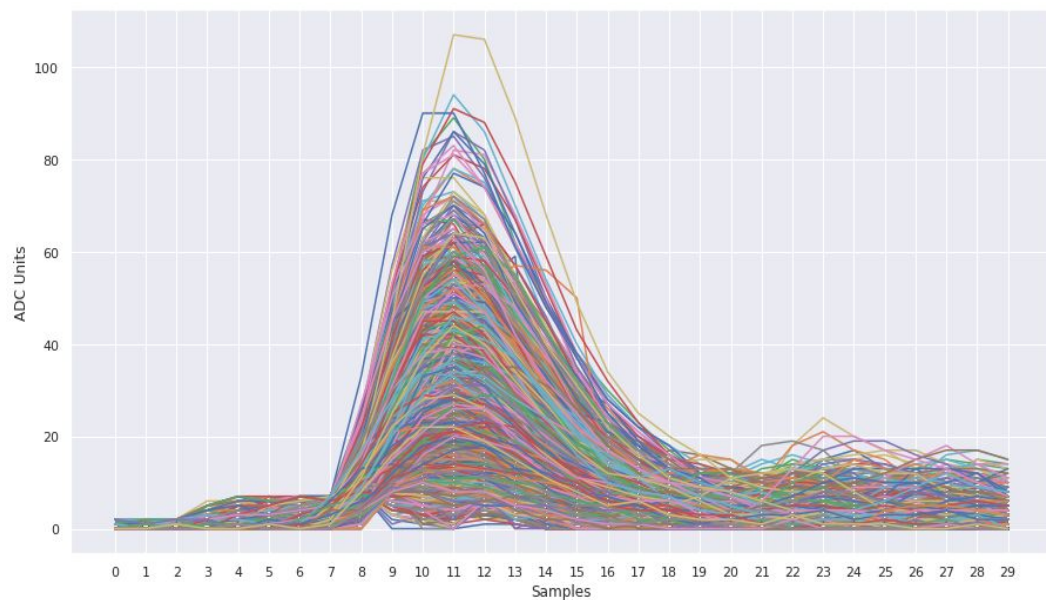
Experimental Setup

- **Data acquisition systems (DAQ)** based on FPGAs and System-on-Chip (SoC) are often used in experimental physics.
- **Water Cherenkov detectors (WCD)** consist of a pure water tank used as a scintillator material coupled to a photomultiplier tube, which is connected to a high-voltage power supply and to an analog front-end.
- Water Cherenkov detector (WCD) at the Escuela de Ciencias Físicas y Matemáticas - Universidad de San Carlos de Guatemala (ECFM-USAC).
- Signal classification of the incoming signal (raw data - 30 samples).
- Pulse shape discrimination based on MLP architecture.

Pulse shape discrimination for WCD

Multi-class classification

Different types of signals - Class 0 and 1

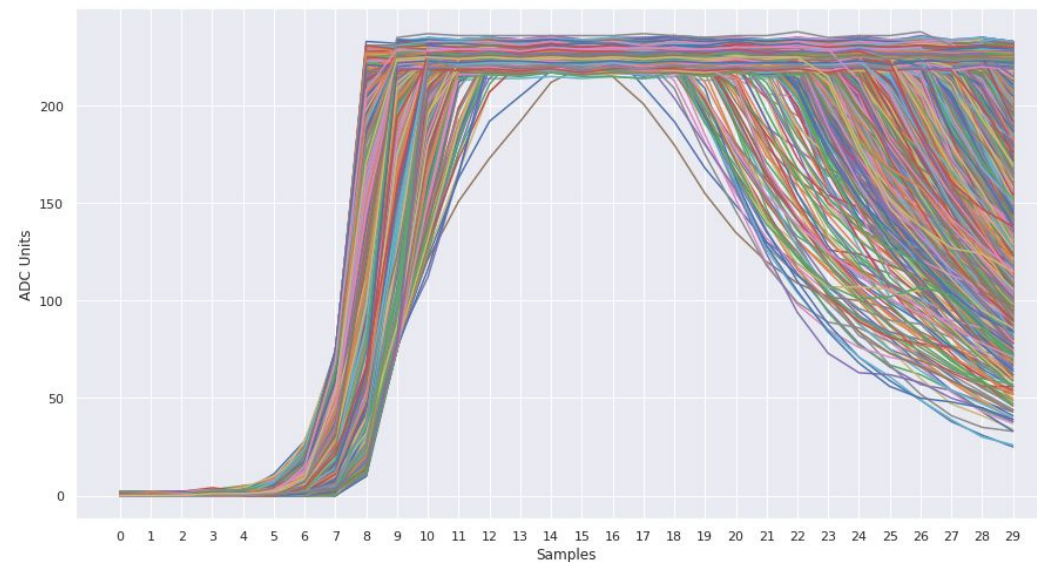
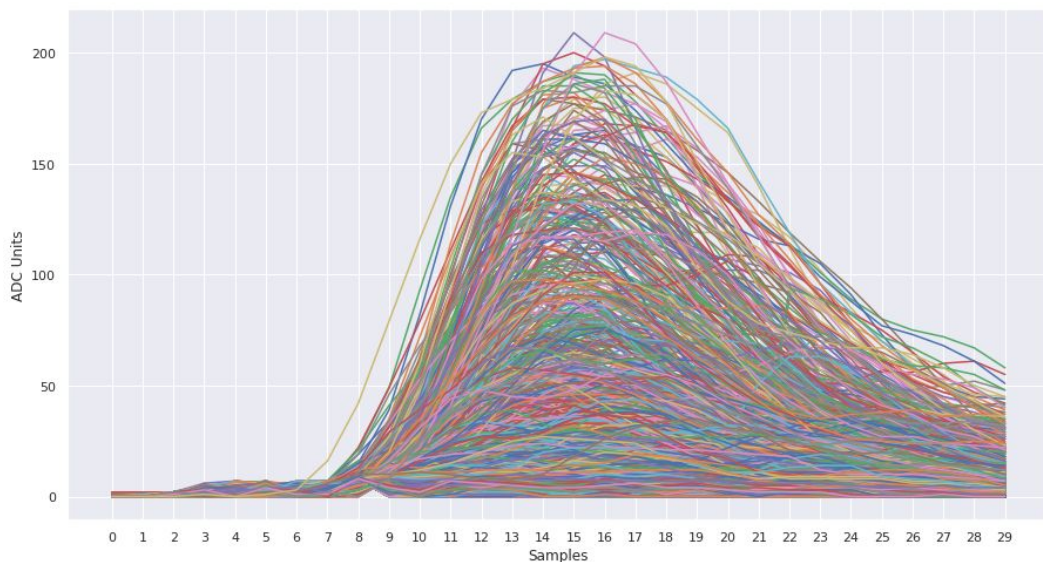


From [8]

Pulse shape discrimination for WCD

Multi-class classification

Different types of signals - Class 2 and 3

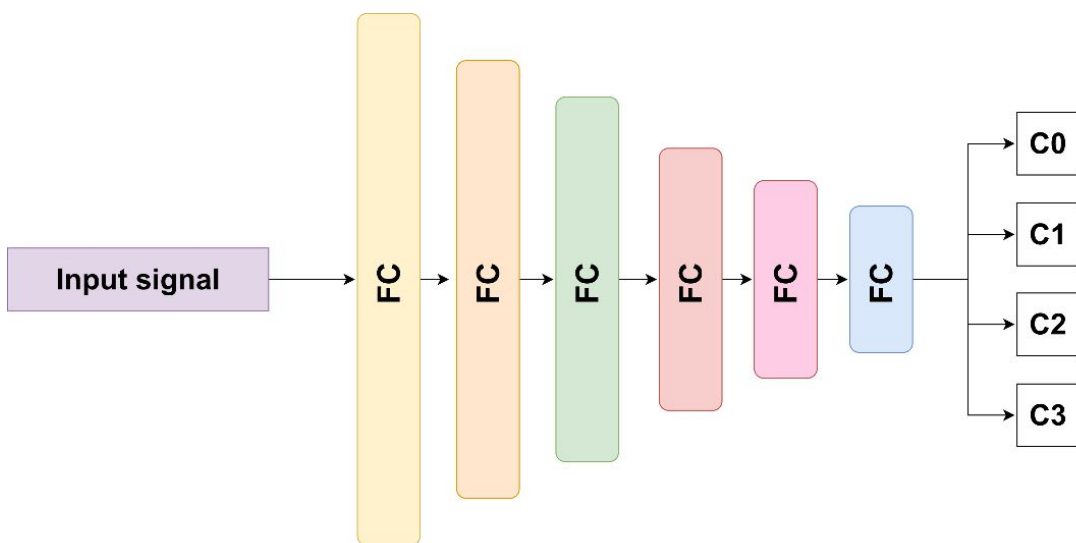


From [8]

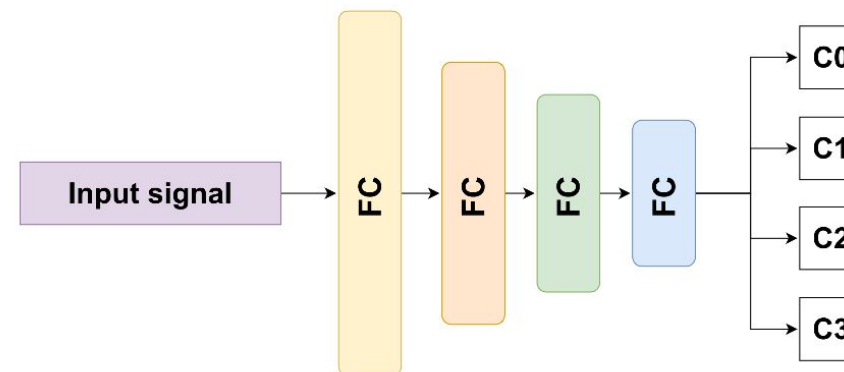
Pulse shape discrimination for WCD

Multi-class classification

MLP model through an ensemble of compression techniques: distillation, quantization, and pruning.



Teacher architecture



Student architecture

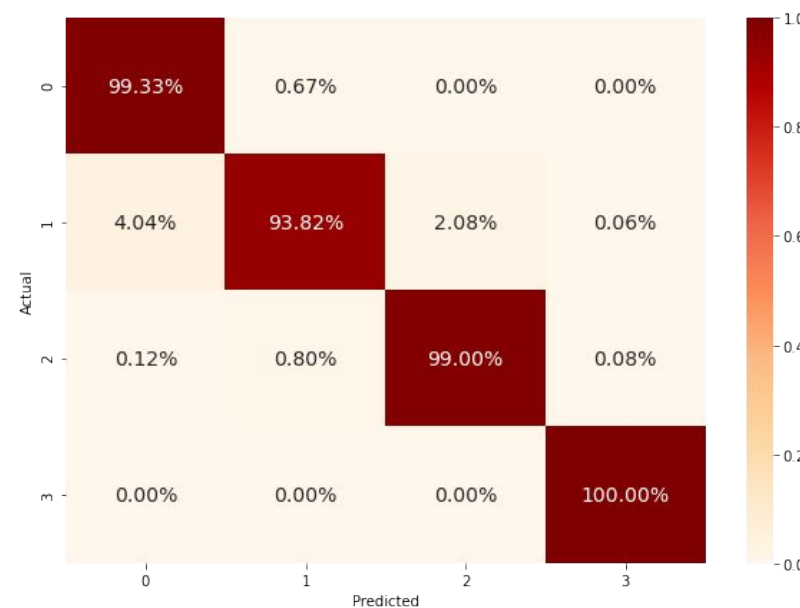
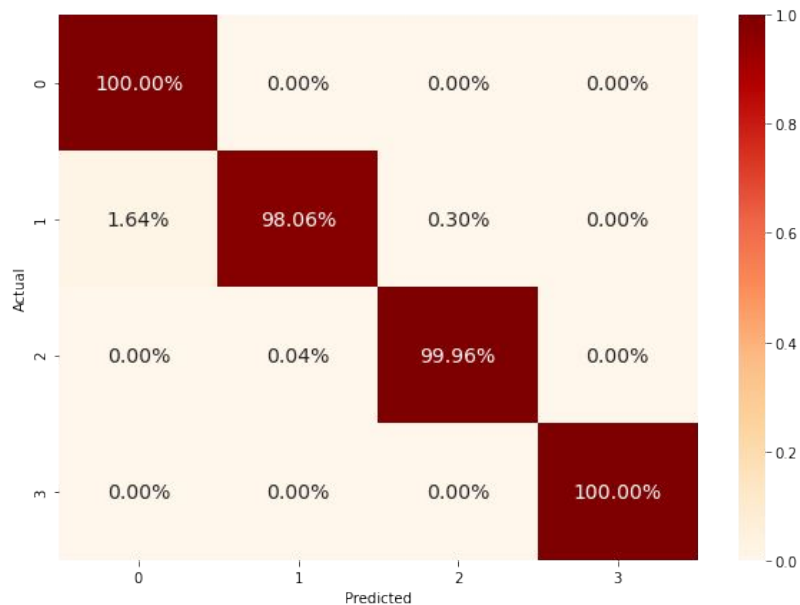
Pulse shape discrimination for WCD

Multi-class classification

Confusion Matrix before (left) and after (right) compression

Total params reduction: From 31,514 to 984

Overall accuracy: From 99.4% to 97%



Pulse shape discrimination for WCD

Define the SoC architecture



Pulse shape discrimination for WCD

Metrics

HLS reports - Clock @5ns

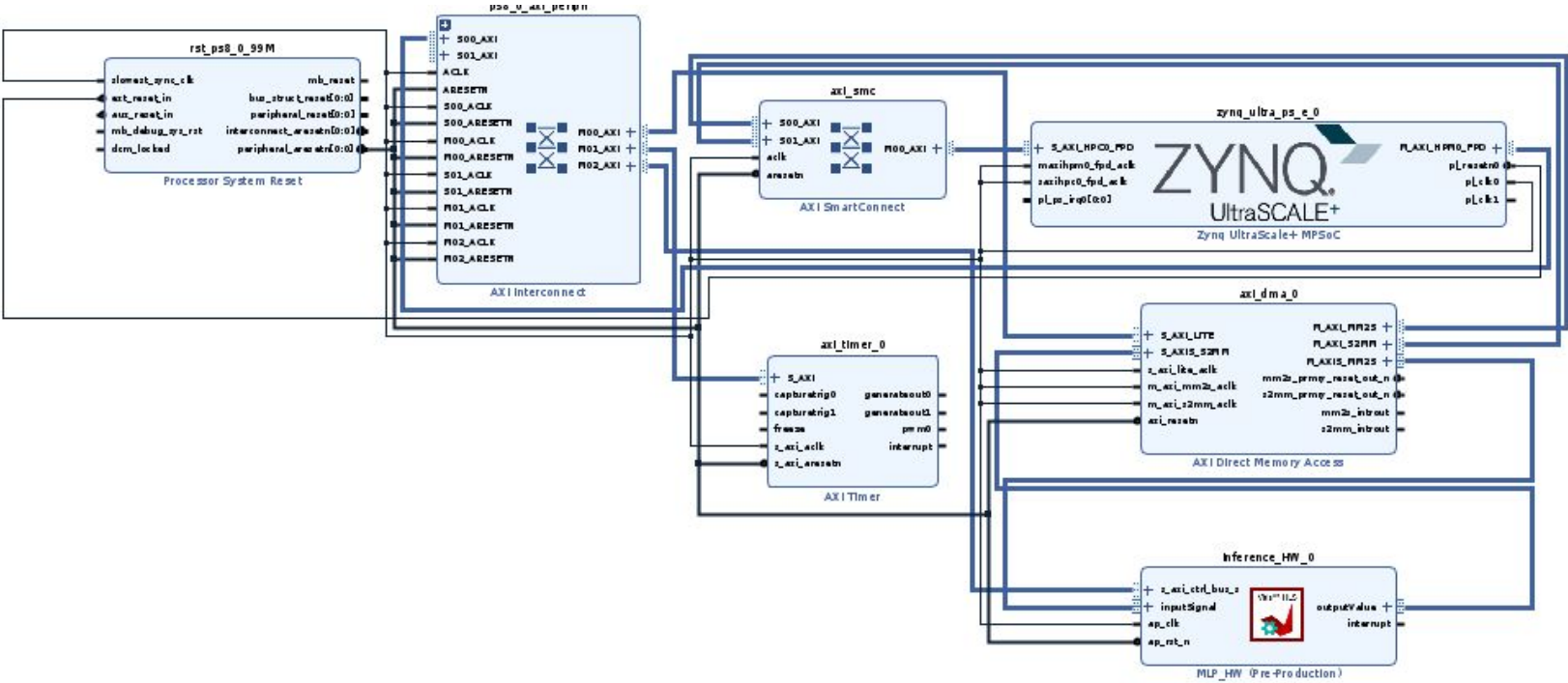
	Latency [clk]*	LUT	FF	BRAM	DSP
PYNQ					
Sol_1_rf1	39	69%	75%	0%	369%
Sol_2_rf8	55	72%	24%	0%	50%
KRIA					
Sol_3_rf1	20	49%	12%	0%	69%
ZCU102					
Sol_4_rf1	20	20%	5%	0%	34%

*Latency only for inference

Pulse shape discrimination for WCD

Implementation on SoC-based FPGA

Integration with Vivado IP Integrator



Pulse shape discrimination for WCD

Implementation on SoC-based FPGA

Final resource usage reported by Vivado

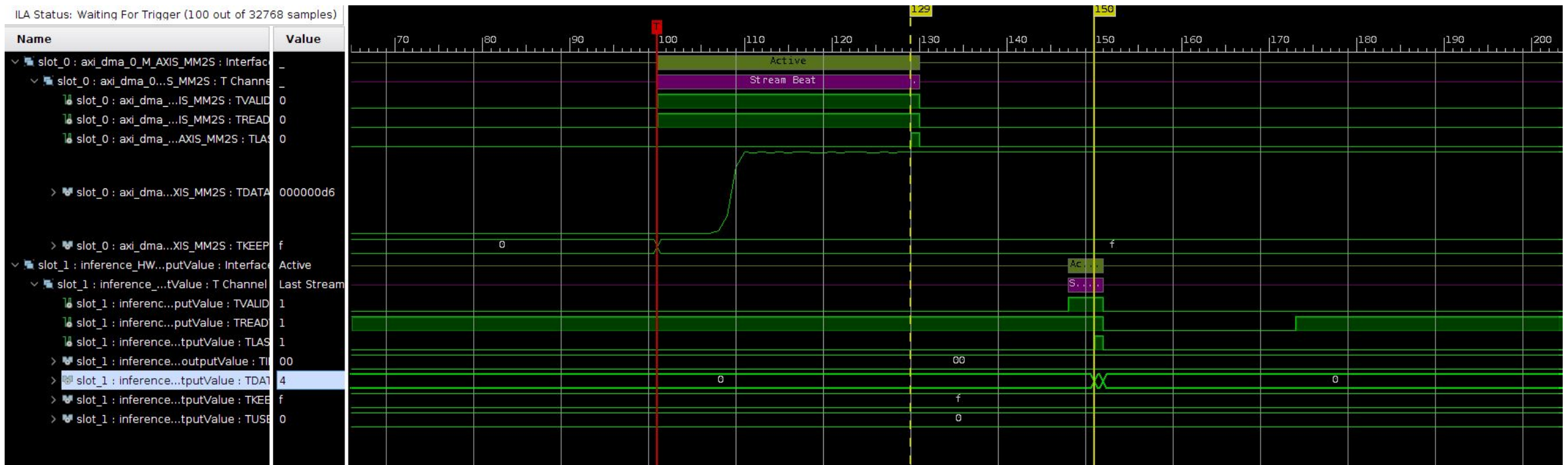
	LUT	FF	BRAM	DSP
PYNQ	44.6%	23%	34%	50%
KRIA	30%	7.8%	33%	69%
ZCU102	7.8%	2.8%	9.9%	27%

Pulse shape discrimination for WCD

Implementation on SoC-based FPGA

Family / Part: zynqplus / xczu9eg-ffvb1156-2-e

Clock cycles for inference: 21 (Estimated by HLS: 20)



PYNQ-Z1 implementation

PYNQ-Z1 implementation

Pulse shape discriminator for cosmic rays | smr3765

```
In [ ]: from pynq import Overlay

In [ ]: ol = Overlay("hw/inference_PYNQ.bit")

In [ ]: ol.ip_dict

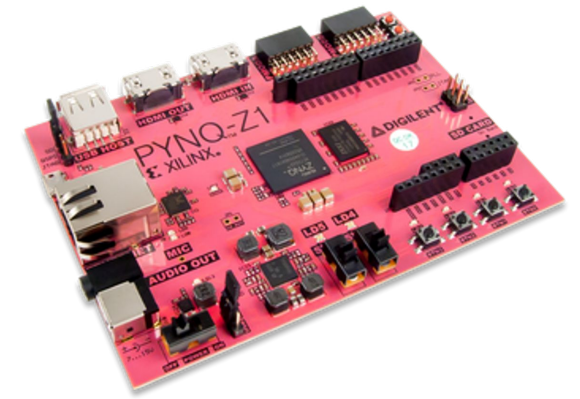
In [ ]: dma = ol.axi_dma_0
dma_send = ol.axi_dma_0.sendchannel
dma_recv = ol.axi_dma_0.recvchannel

In [ ]: from pynq import allocate
import numpy as np

data_size = 30
input_buffer = allocate(shape=(data_size,), dtype=np.uint32)

In [ ]: x3 = [0, 2, 0, 0, 0, 0, 2, 14, 68, 231, 232, 232, 232, 230, 232,
            231, 233, 232, 231, 231, 232, 232, 231, 230, 232, 231, 232, 231, 231, 230]
for i in range(0, data_size):
    input_buffer[i] = x3[i]

In [ ]: import matplotlib.pyplot as plt
plt.figure(figsize=(15,7))
plt.xlabel('Samples', fontsize=11)
plt.ylabel('Amplitude', fontsize=11)
plt.grid(True, alpha=1.0)
plt.plot(x3, 'o', label="Signal 1", color='navy', markersize=7, lw=1)
```



 PYNQ™

PYNQ-Z1 implementation

```
In [ ]: hls_ip = ol.inference_HW_0
```

```
In [ ]: hls_ip.register_map
```

```
In [ ]: # Initialize HLS IP core
```

```
CONTROL_REGISTER = 0x0
hls_ip.write(CONTROL_REGISTER, 0x81) # 0x81 will set bit 0
```

```
In [ ]: hls_ip.register_map
```

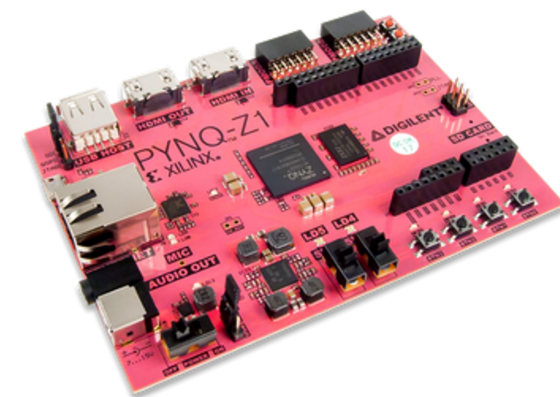
```
In [ ]: # Start the DMA transfer
```

```
dma_send.transfer(input_buffer)
```

```
In [ ]: output_buffer = allocate(shape=(4,), dtype=np.uint32)
```

```
In [ ]: dma_recv.transfer(output_buffer)
```

```
In [ ]: for i in range(4):
        print((output_buffer[i]))
```



PYNQ™

Image classification based on CNN

Image classification based on CNN

Experimental setup

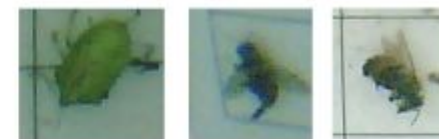
Nectras IoT trap



Captured image



Other insects



Lobesia botrana



- Romina Molina, Valentina Carrer, Maynor Ballina, Maria Liz Crespo, Luciana Bollati, Daniel Sequeiro, Stefano Marsi and Giovanni Ramponi. "ML-based classifier for precision agriculture on embedded systems". ApplePies2022. [Accepted].

- A. Suárez, R. S. Molina, G. Ramponi, R. Petrino, L. Bollati and D. Sequeiros, "Pest detection and classification to reduce pesticide use in fruit crops based on deep neural networks and image processing," 2021 XIX Workshop on Information Processing and Control (RPIC), 2021, pp. 1-6.

Image classification based on CNN

Methodology

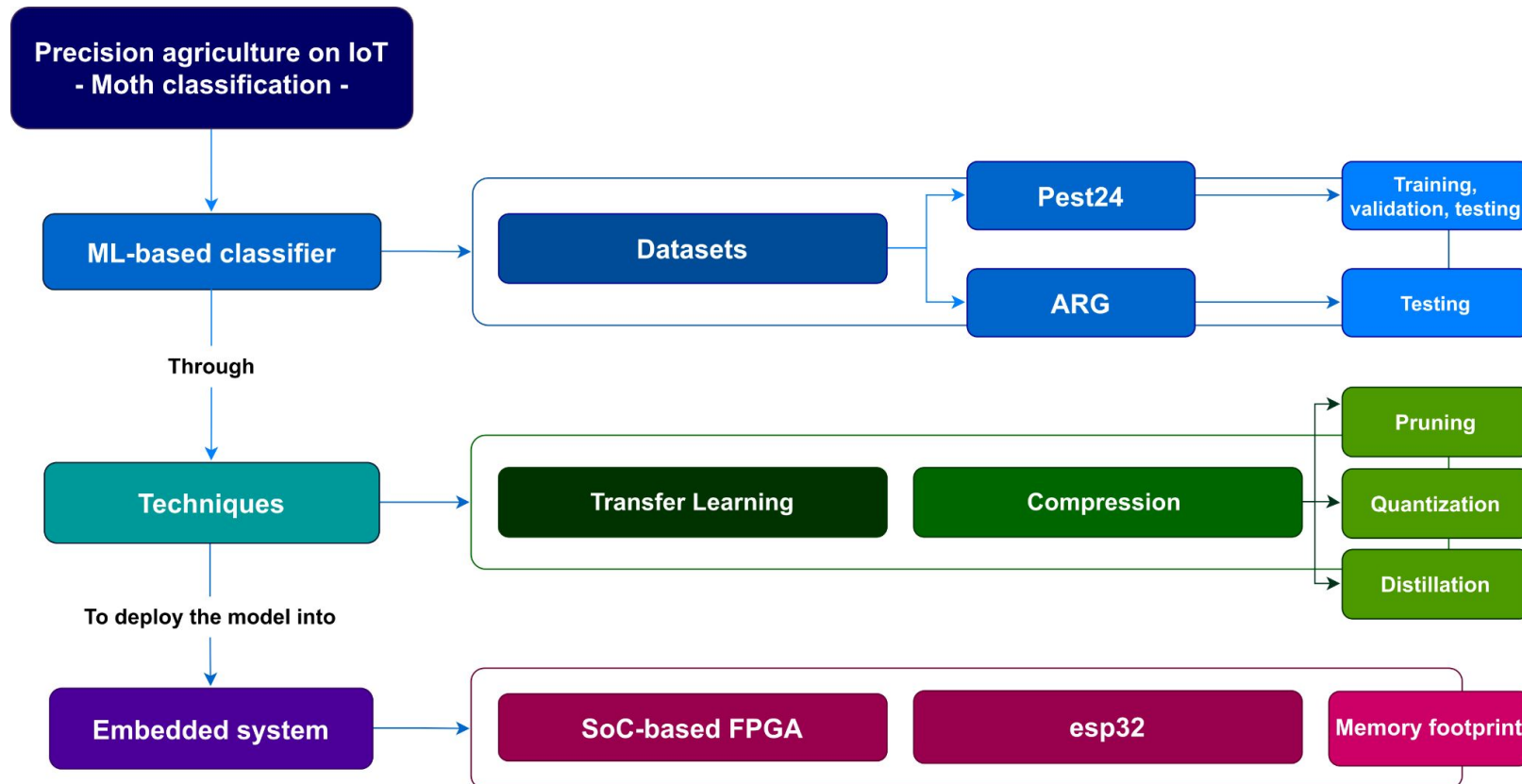
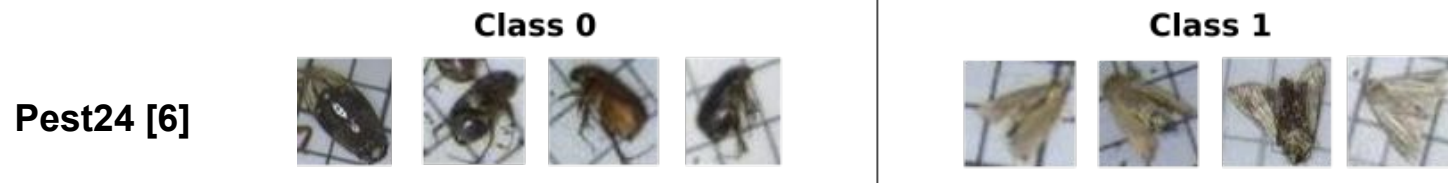


Image classification based on CNN

Datasets



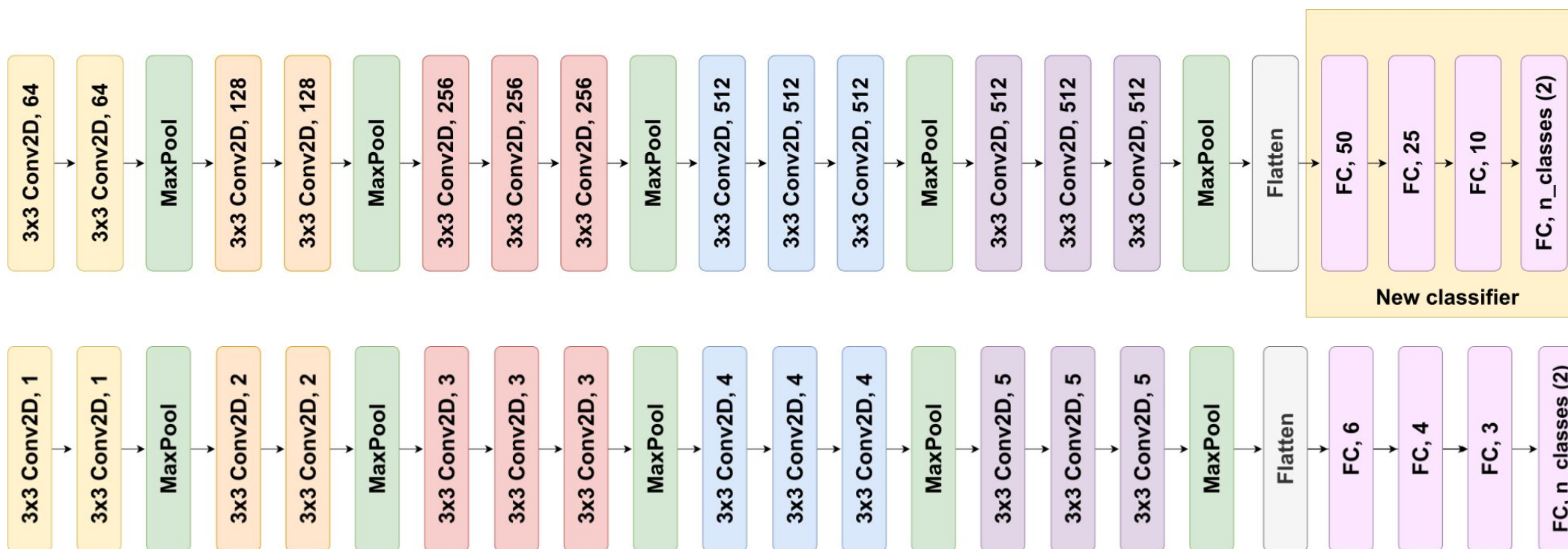
A standard dataset available in the literature for training, granting a stable and effective performance.



Images provided by the current system in Argentina.

Image classification based on CNN

ML-based architectures



Based on [9]

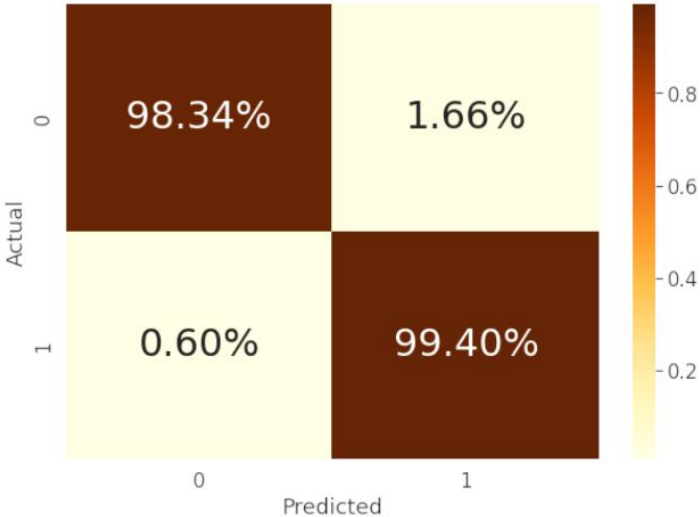
Top. Teacher architecture based on VGG16 and obtained through transfer learning - 14,818,706 parameters (Model size: 177.6Mb)

Bottom: Distilled architecture. Compression ratio: 7409x – in number of parameters –

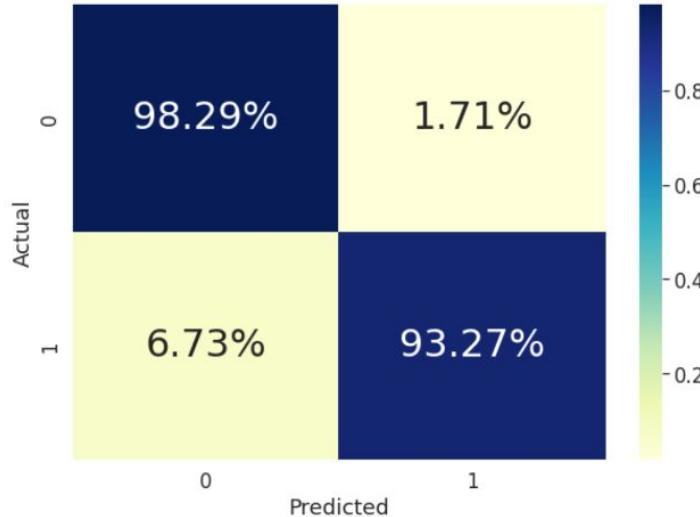
Image classification based on CNN

Confusion matrix

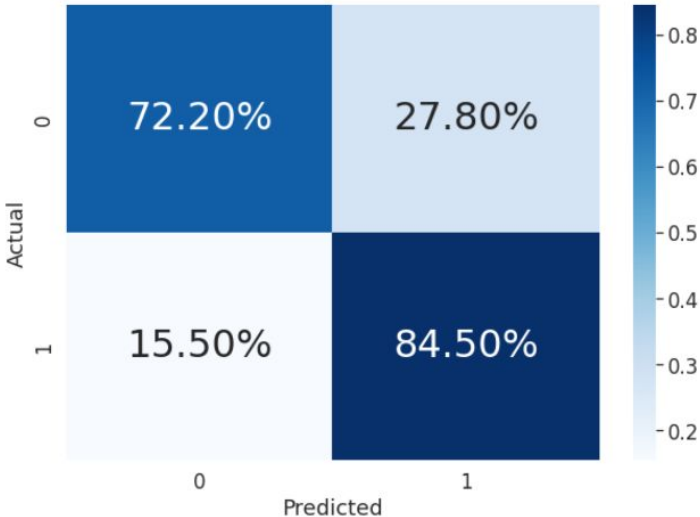
1 - Teacher



2 - QStudentFPGA - Pest24



3 - QStudentFPGA - Arg



Based on [9]

Image classification based on CNN

Experimental results

- SoC-based FPGA

	LUT	FF	BRAM	DSP
PYNQ	41%	28%	88%	12%
KRIA	20%	28%	22%	2%

Utilization from place & route reports (post-implementation)

Based on [9]

References

- [1] Choudhary, T., Mishra, V., Goswami, A., Sarangapani, J.: A comprehensive survey on model compression and acceleration. In: Artificial Intelligence Review, 53(7), pp. 5113--5155, 2020, doi: <https://doi.org/10.1007/s10462-020-09816-7>.
- [2] Hinton, G.; Vinyals, O.; Dean, J.: Distilling the knowledge in a neural network. In: arXiv preprint arXiv:1503.02531, 2(7), 2015, doi: <https://doi.org/10.48550/arXiv.1503.02531>.
- [3] Blalock, D.; Gonzalez Ortiz, J. J.; Frankle, J.; Gutttag, J. (2020).: What is the state of neural network pruning?. In: Proceedings of machine learning and systems, 2, pp. 129--146, 2020.
- [4] Liang, T.; Glossner, J.; Wang, L.; Shi, S.; Zhang, X.: Pruning and quantization for deep neural network acceleration: A survey. In: Neurocomputing, 461, pp. 370--403, 2021, doi: <https://doi.org/10.1016/j.neucom.2021.07.045>.
- [5] Duarte, J.; Han, S.; Harris, P.; Jindariani, S.; Kreinar, E.; Kreis, B.; Ngadiuba, J.; Pierini, M.; Rivera, R.; Tran, N.; Wu, Z. : Fast inference of deep neural networks in FPGAs for particle physics. In: Journal of Instrumentation, 13(07), 2018, <https://doi.org/10.1088/1748-0221/13/07/P07027>.
- [6] Wang, Qi-Jin; Zhang, Sheng-Yu; Dong, Shi-Feng; Zhang, Guang-Cai; Yang, J.; Li, R., Wang, Hong-Qiang: Pest24: A large-scale very small object dataset of agricultural pests for multi-target detection. In: Computers and Electronics in Agriculture, Volume 175, 2020, doi: <https://doi.org/10.1016/j.compag.2020.105585>.
- [7] Molina, R. S.; Gil-Costa, V.; Crespo, M. L.; Ramponi, G.: High-Level Synthesis Hardware Design for FPGA-based Accelerators: Models, Methodologies, and Frameworks. In: IEEE Access, 2022, doi:10.1109/ACCESS.2022.3201107.
- [8] García Ordóñez, L.G.; Molina, R.S.; Moreales Argueta, I.R. Morales; Crespo, M.L.; Cicuttin, A. ; S. Carrato, G. Ramponi, H.E. Pérez Figueroa, and M.G. Ballina Escobar: Pulse Shape Discrimination for Online Data Acquisition in Water Cherenkov Detectors Based on FPGA/SoC. In: 37th International Cosmic Ray Conference (ICRC2021), 2021, doi: <https://doi.org/10.22323/1.395.0274>
- [9] Romina Molina, Valentina Carrer, Maynor Ballina, Maria Liz Crespo, Luciana Bollati, Daniel Sequeiro, Stefano Marsi and Giovanni Ramponi. "ML-based classifier for precision agriculture on embedded systems". ApplePies2022. [Accepted].



The Abdus Salam
**International Centre
for Theoretical Physics**

Thank you!

**Joint ICTP-IAEA School on
FPGA-based SoC and its
Applications to Nuclear and
Scientific Instrumentation**



**14 November - 2 December 2022
An ICTP - IAEA Hybrid Meeting
Trieste, Italy**

Further information:
<http://indico.ictp.it/event/9933/>
smr3765@ictp.it