Applied Machine Learning:
Turning Concepts into Code

Romina Soledad Molina, Ph.D.

School on Applied AI for Sustainable Development | smr 4210 | Trieste, Italy
March 2026

KaleidoForge
Lightning Bridges for Fast AI

The Abdus Salam International Centre for Theoretical Physics

Outline

- Edge AI Applications on FPGA-based Systems.
- Introduction to Applied Machine Learning.
- Artificial Neural Networks (ANNs).
- Training and Inference.
- Performance Metrics.
- Workflow with Keras and TensorFlow.
- Hands-on Demo: Training an MLP on the MNIST dataset.

Edge AI Applications
on FPGA-based Systems

Edge AI Applications on FPGA-based Systems

- Gamma/Neutron discrimination.
- Neutron Discrimination with Diamond Detectors.
- Physics-Guided Pile-up Rejection.
- Water quality monitoring.
- Pest classification in fruit crops.
- Pulse shape discriminator for cosmic rays studies.
- Volcanic seismic event detection.

Gamma/Neutron Discrimination

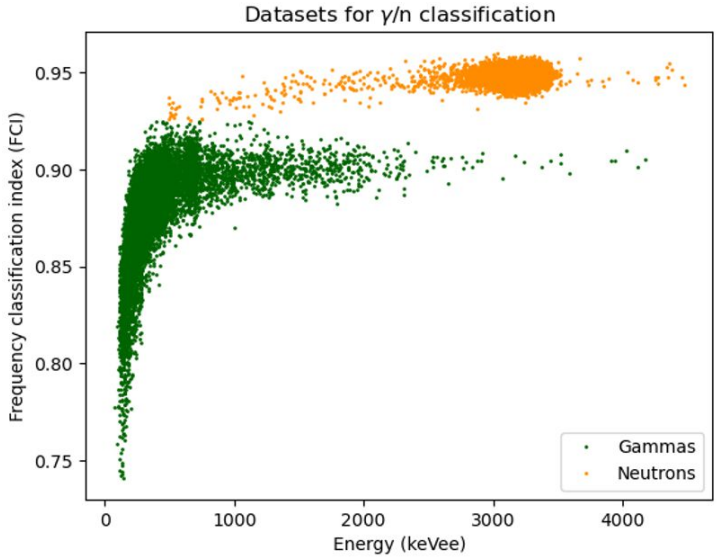
Edge AI Applications on FPGA-based Systems

Gamma/neutron discrimination

- **Collaboration between MLab - ICTP and NSIL - IAEA.**
- Tagged dataset of **gamma and neutron events** from Deuterium-Deuterium (DD) and Deuterium-Tritium (DT) generators.
- The dataset was recorded at the **Neutron Science Facility (NSF) of the Nuclear Science and Instrumentation Laboratory (NSIL), IAEA.**
- The detector is based on a small **CLYC** (Cs2LiYCl6:Ce) crystal (0.5 in diameter by 30 mm length) coupled to a 4-element SiPM array.
- The data were **sampled at 4 GSPS with 10-bits resolution** using a CAEN DT5761 digitizer.
- **The total gamma and neutron events in this dataset are 10,913 and 27,696, respectively.**

Edge AI Applications on FPGA-based Systems

Gamma/neutron discrimination

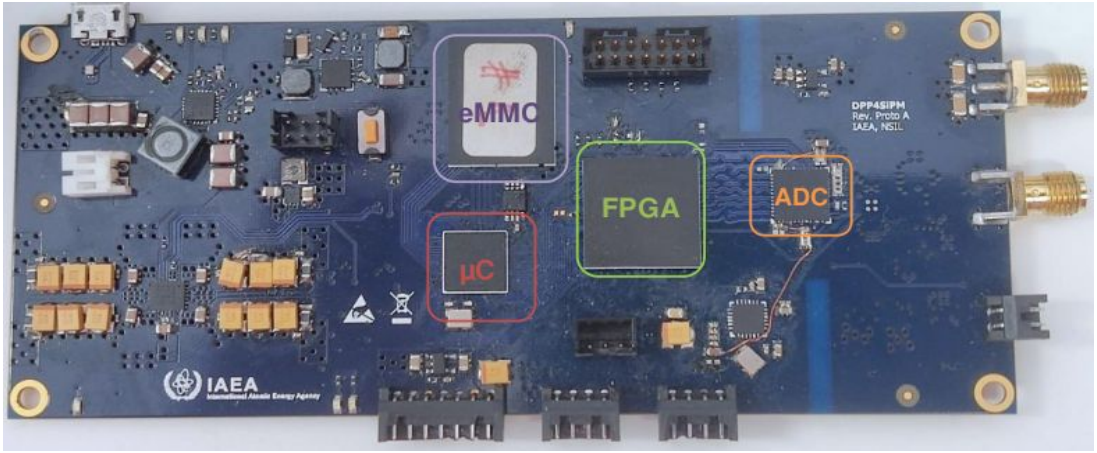


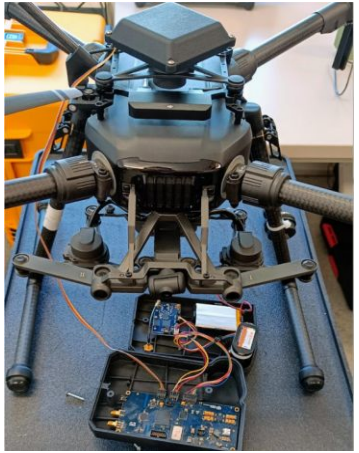
Morales, I. R., Crespo, M. L., Bogovac, M., Cicuttin, A., Kanaki, K., & Carrato, S. (2023). Gamma/neutron classification with SiPM CLYC detectors using frequency-domain analysis for embedded real-time applications. Nuclear Engineering and Technology.

Dataset from https://doi.org/10.5281/zenodo.8037059

Edge AI Applications on FPGA-based Systems

Gamma/neutron discrimination

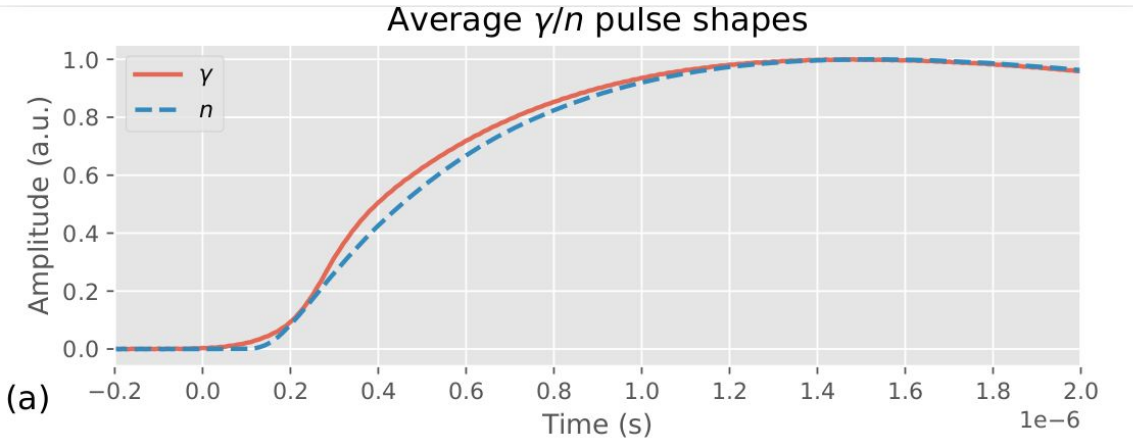




It accommodates various sub-assemblies, including: **(i)** input power monitoring and management sub-system, **(ii)** digital interface controller, **(iii)** digital signal processing block, **(iv)** data storage, and **(v)** analog interface domain.

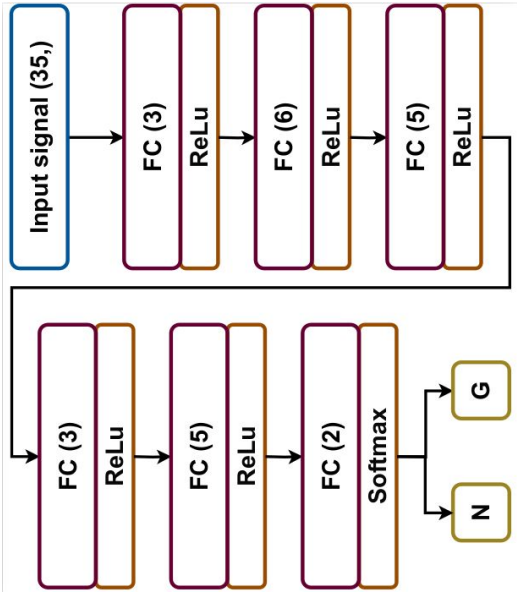
Edge AI Applications on FPGA-based Systems

Gamma/neutron discrimination



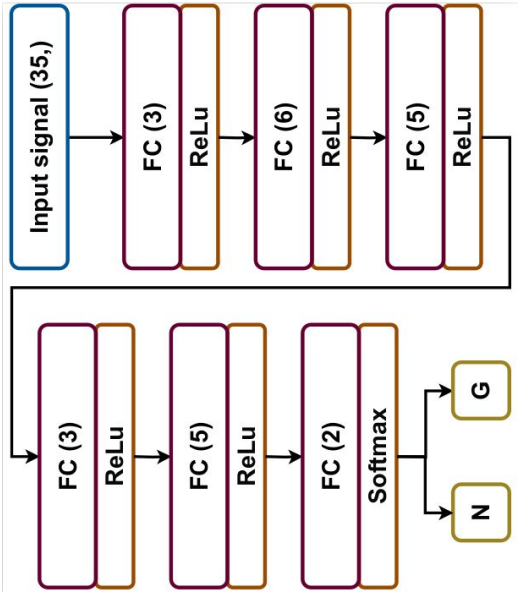
Edge AI Applications on FPGA-based Systems

Gamma/neutron discrimination



Edge AI Applications on FPGA-based Systems

Gamma/neutron discrimination



Teacher architecture with **2,623 parameters** distributed in 6 hidden layers (MLP).

Compressed architecture with **217 parameters**, distributed in 6 hidden layers (MLP).

Input size reduction:
35 samples of the leading edge of the pulse.

Edge AI Applications on FPGA-based Systems

Gamma/neutron discrimination

- **Overall accuracy**
  - Teacher architecture (original): **99.00%**
  - Student architecture (compressed): **98.20%**

Edge AI Applications on FPGA-based Systems

Gamma/neutron discrimination

- **Overall accuracy**
  - Teacher architecture (original): **99.00%**
  - Student architecture (compressed): **98.20%**

- **SoC memory footprint in terms of resource utilization @200MHz**
  - Artix-7 platform: **below 35%**

Edge AI Applications on FPGA-based Systems

Gamma/neutron discrimination

- **Overall accuracy**
  - Teacher architecture (original): **99.00%**
  - Student architecture (compressed): **98.20%**

- **SoC memory footprint in terms of resource utilization @200MHz**
  - Artix-7 platform: **below 35%**

- **SoC latency**
  - Artix-7 platform: **45 clk cycles (@200MHz)**

Neutron Discrimination with Diamond Detectors

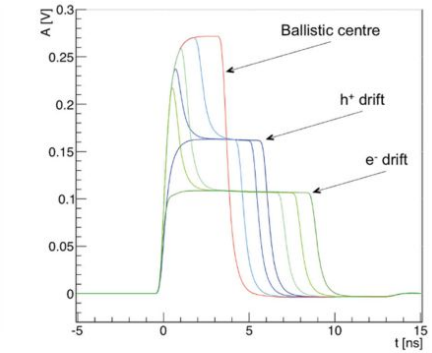
Edge AI Applications on FPGA-based Systems

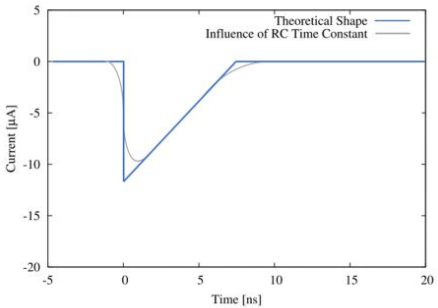
Neutron Discrimination with Diamond Detectors

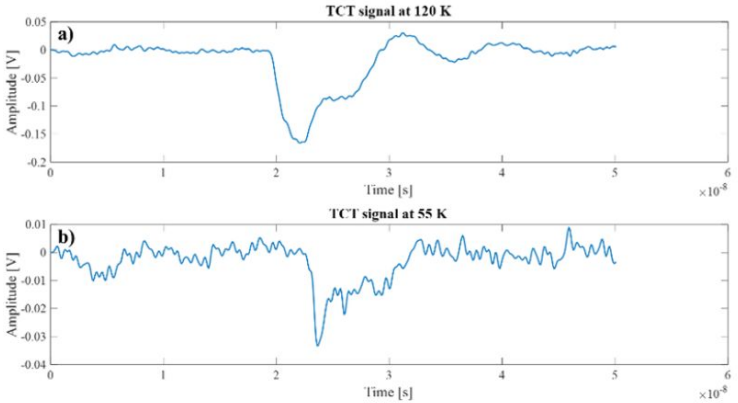
- **Collaboration between MLab - ICTP and Ion Beam Interactions (LIIS) at the Institude Ruđer Bošković (IRB).**

Edge AI Applications on FPGA-based Systems

Neutron Discrimination with Diamond Detectors

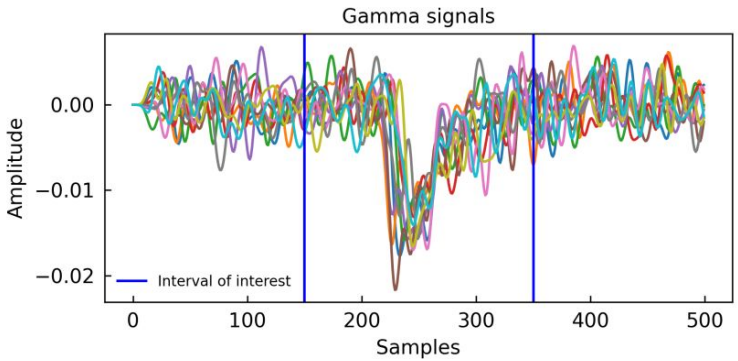


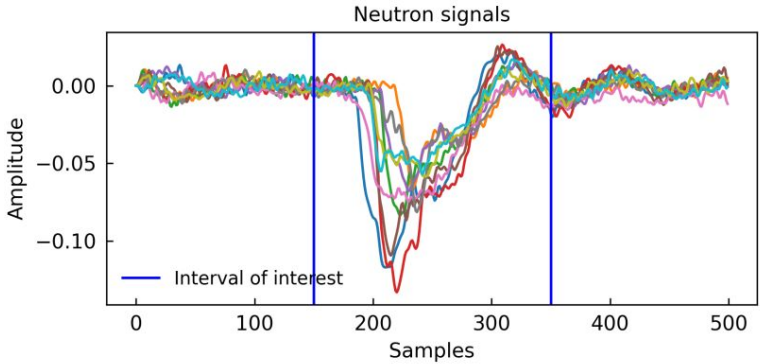




Edge AI Applications on FPGA-based Systems

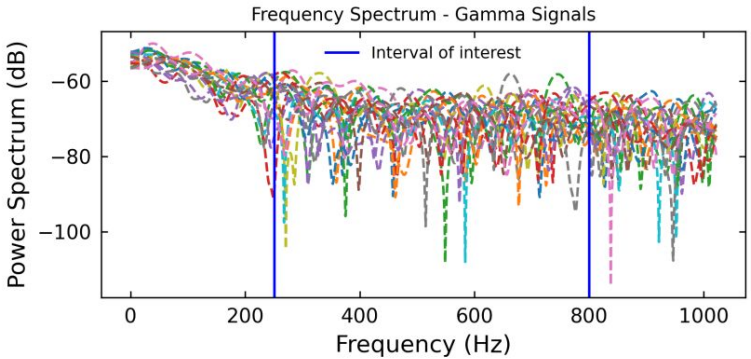
Neutron Discrimination with Diamond Detectors

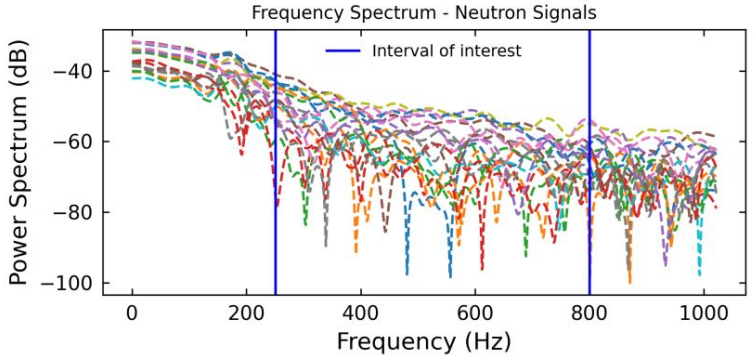




Edge AI Applications on FPGA-based Systems

Neutron Discrimination with Diamond Detectors





Edge AI Applications on FPGA-based Systems

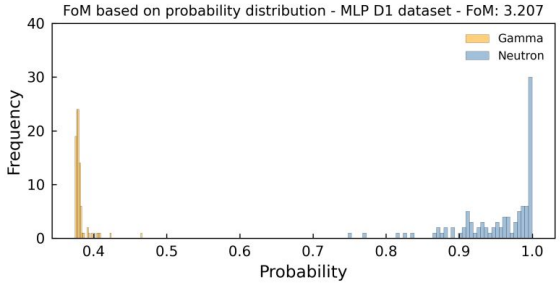
Neutron Discrimination with Diamond Detectors

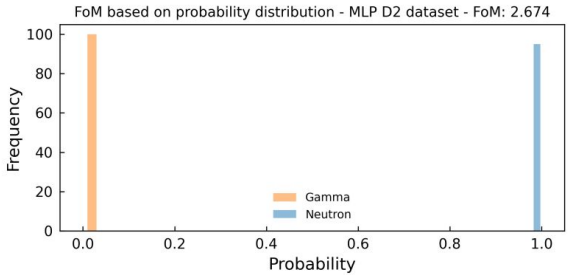
TABLE II
LIST OF HYPERPARAMETERS FOR MLP AND 1D-CNN MODELS FOR DATASETS D1, D2, AND D3FFT.

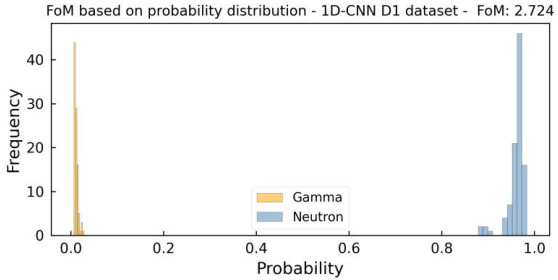
Hyperparameter	MLP-based		1D-CNN-based		
	D1	D2	D1	D2	D3FFT
Layers	D-D-D	D-D-D-D-D	CMB-CMB-CMB-F-D-D-D	CMB-CMB-F-D-D-D	CMB-CMB-CMB-CMB-F-D-D-D
Input shape	(200,)	(200,)	(200,)	(200,)	(550,)
Num. params	2,217	7,007	1,154	5,211	5,011
Batch size	32	32	8	32	32
Epochs	32	128	64	32	128
Optimizer	Adam	Adam	Adam	Adam	Adam
Learning rate	0.0001	0.0001	0.0001	0.0001	0.001

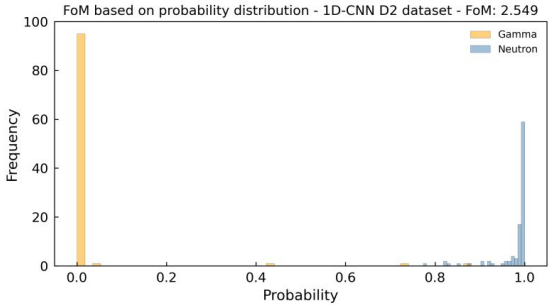
Edge AI Applications on FPGA-based Systems

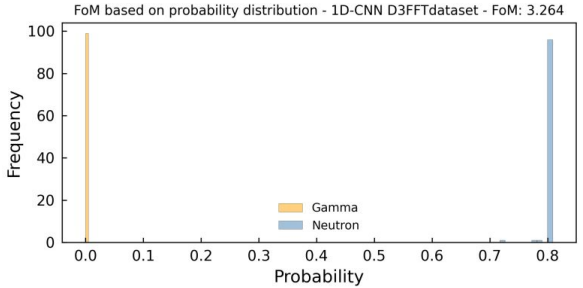
Neutron Discrimination with Diamond Detectors











Edge AI Applications on FPGA-based Systems

Neutron Discrimination with Diamond Detectors

TABLE V
IMPLEMENTATION RESULTS. FPGA PART: XC7Z020-CLG400-1.

Metrics	MLP D1	1D-CNN D1	1D-CNN D3FFT
BRAM	0%	21%	16%
DSP	0%	12%	7%
LUT	26%	30%	37%
FF	18%	11%	34%
Latency inference [μs]	1.6	2.230	4.3
Frequency [MHz]	100	100	100
FoM compressed model	2.242	2.211	2.230

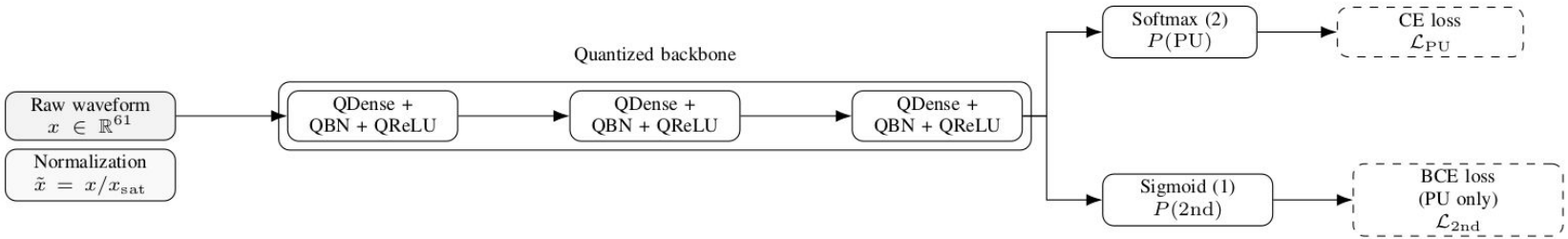
Physics-Guided
Pile-up Rejection

Edge AI Applications on FPGA-based Systems
Physics-Guided Pile-up Rejection

- **MLab - ICTP**

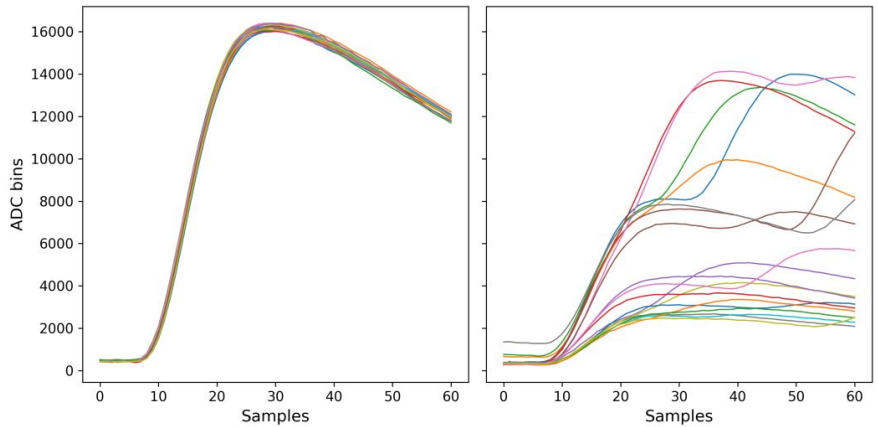
Edge AI Applications on FPGA-based Systems

Physics-Guided Pile-up Rejection



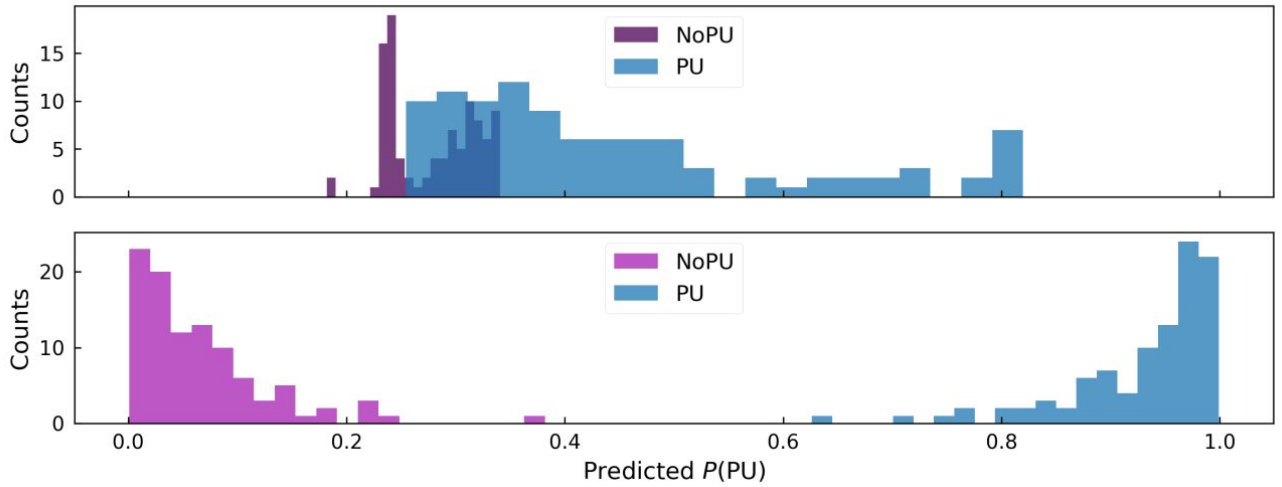
Edge AI Applications on FPGA-based Systems

Physics-Guided Pile-up Rejection



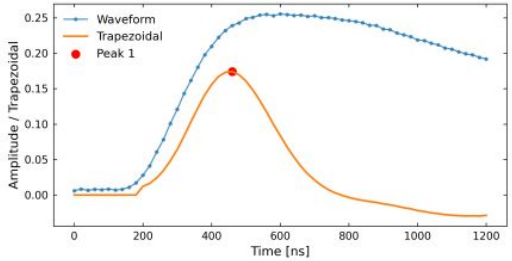
Edge AI Applications on FPGA-based Systems

Physics-Guided Pile-up Rejection

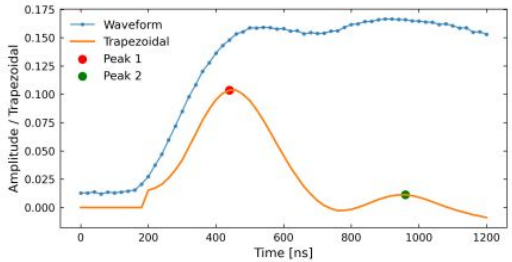


Edge AI Applications on FPGA-based Systems

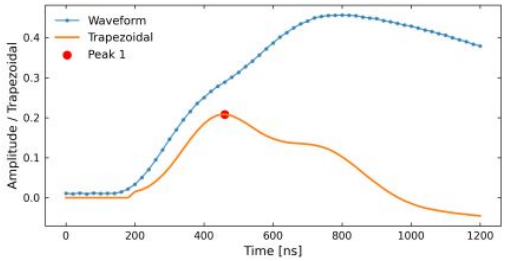
Physics-Guided Pile-up Rejection



(a) Single pulse ($P(\mathrm{PU}) = 0.008$).



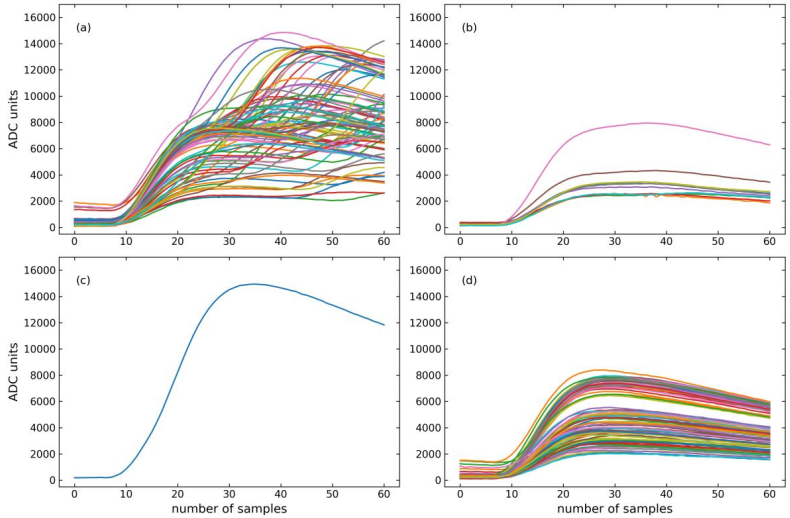
(b) Resolvable PU ($P(\mathrm{PU}) = 0.967$, $P(\mathrm{2nd}) = 0.695$).

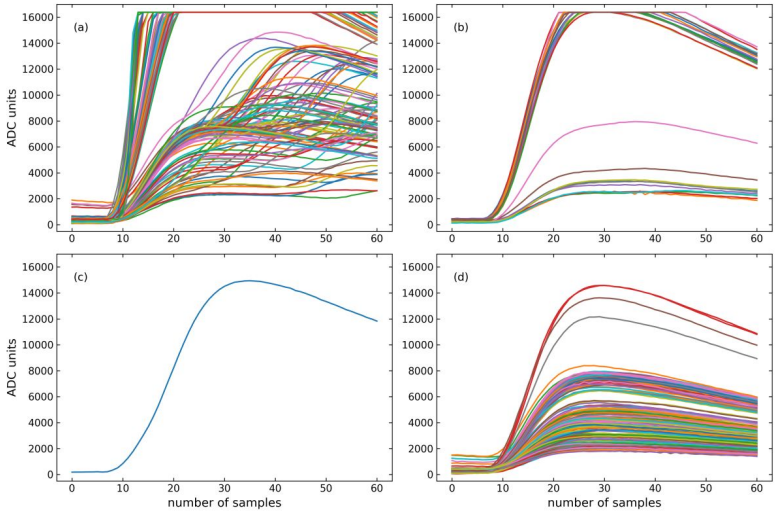


(c) Unresolvable PU ($P(\mathrm{PU}) = 0.962$, $P(\mathrm{2nd}) = 0.239$).

Edge AI Applications on FPGA-based Systems

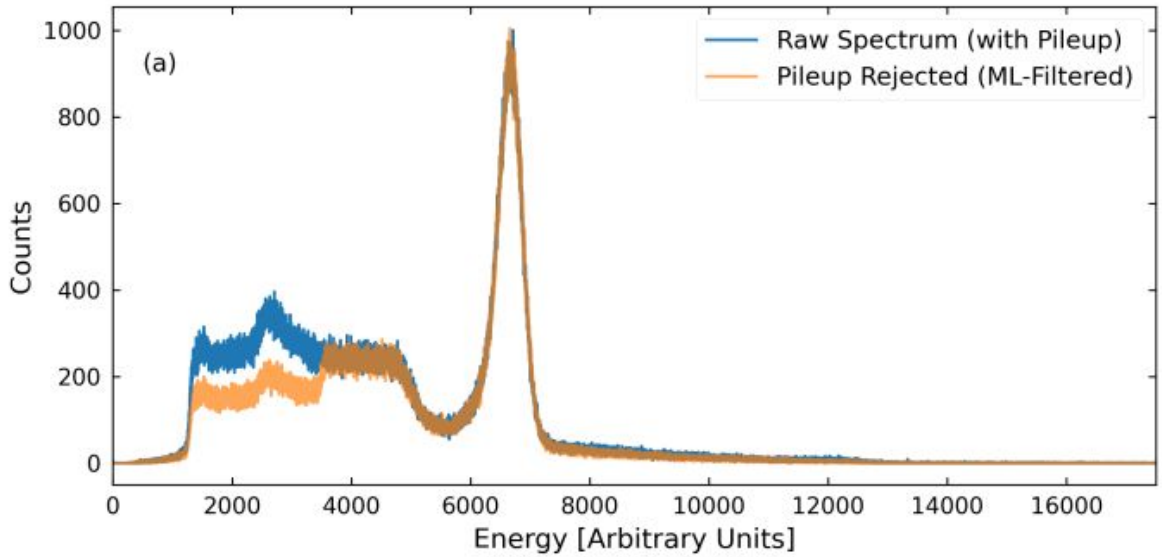
Physics-Guided Pile-up Rejection





Edge AI Applications on FPGA-based Systems

Physics-Guided Pile-up Rejection



Water Quality Monitoring

Edge AI Applications on FPGA-based Systems

Water Quality Monitoring

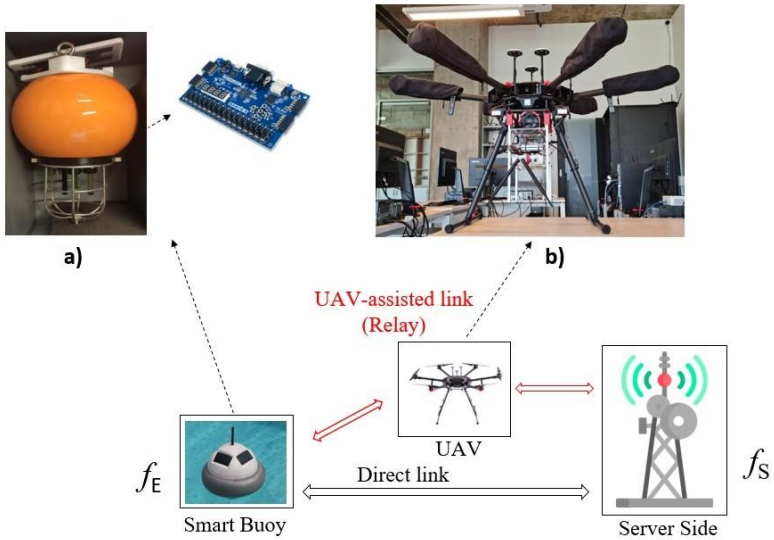
- **Collaboration between ICTP, University of Novi Sad - Serbia, and the Institute for Artificial Intelligence Research and Development of Serbia**

Edge AI Applications on FPGA-based Systems
Water Quality Monitoring

Dunav River,
Novi Sad, Serbia

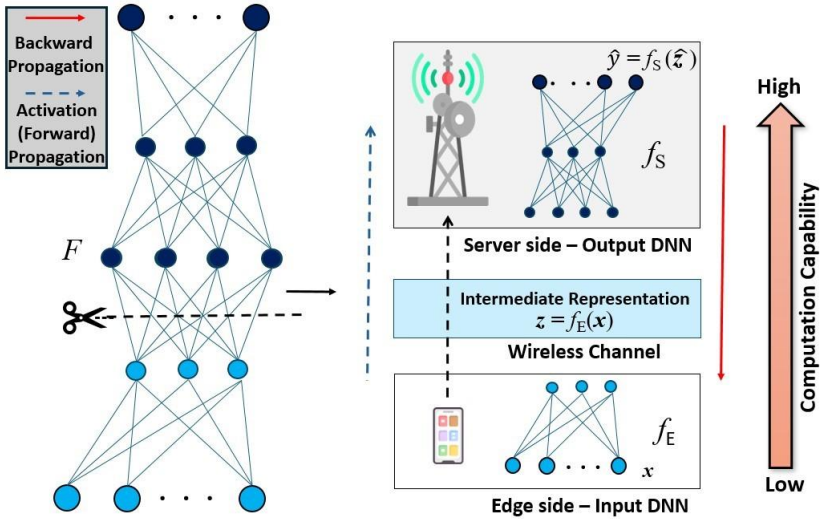
Edge AI Applications on FPGA-based Systems
Water Quality Monitoring

- **Hardware setup**



Edge AI Applications on FPGA-based Systems
Water Quality Monitoring

- **Split-inference stage**



Edge AI Applications on FPGA-based Systems
Water Quality Monitoring

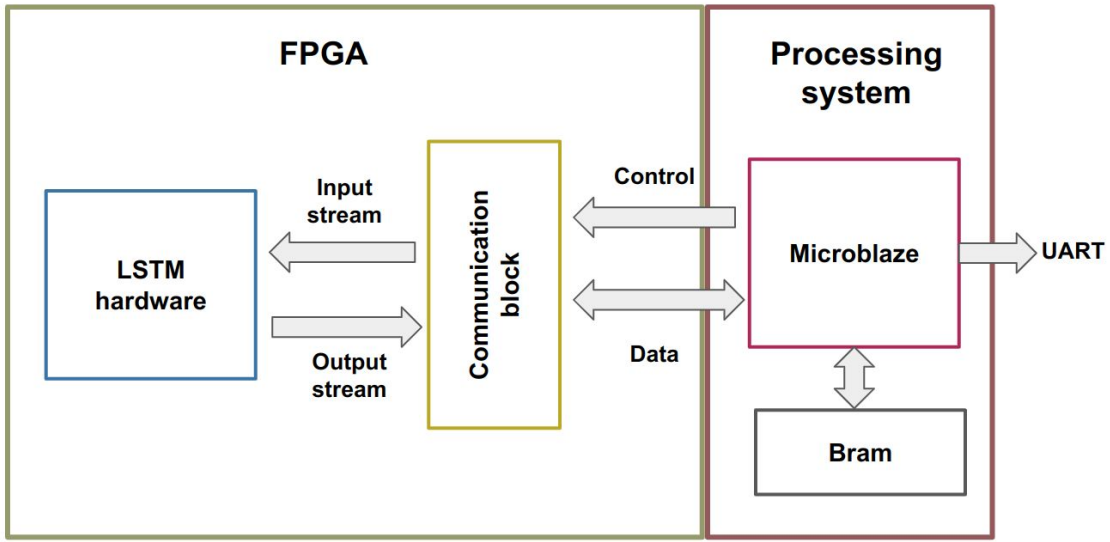
PERFORMANCE EVALUATION OF THE TEACHER (LSTM-DO-T) AND STUDENT (LSTM-DO-S) MODELS.

Model	Parameters	Size [KB]	MAE	MSE	R^2
LSTM-DO-T	39,951	156.06	0.0520	0.0081	0.97
LSTM-DO-S	871	3.40	0.0574	0.0087	0.96

- **Three LSTM inference variations**
  - LSTM-DO-S: the complete student network,
  - Split-A: a configuration with two stacked LSTM layers,
  - Split-B: a configuration containing a single LSTM layer.

Edge AI Applications on FPGA-based Systems
Water Quality Monitoring

- **Block design**



Edge AI Applications on FPGA-based Systems
Water Quality Monitoring

- **Results - LSTM Inference stage**

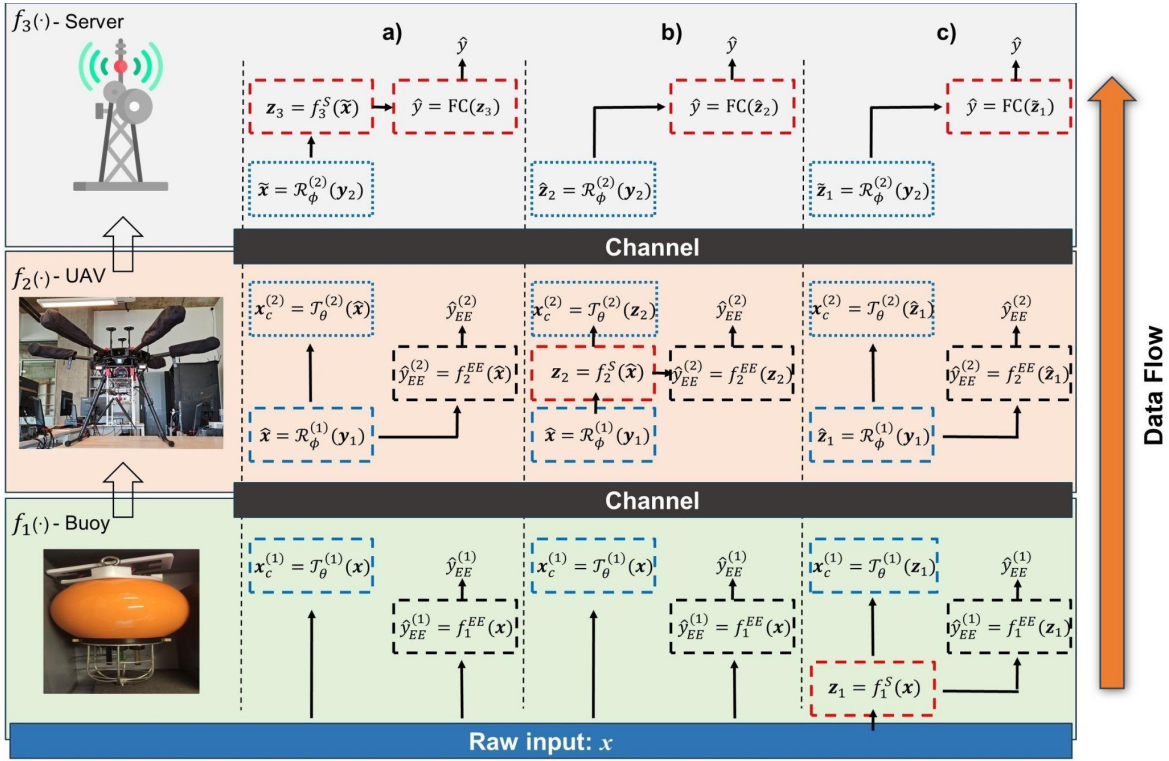
IMPLEMENTATION RESULTS FOR LSTM-BASED ACCELERATORS. FPGA PART: XC7A35TCSG325-1.

Metrics	LSTM-DO-S	Split-A	Split-B
BRAM	31%	16%	20%
DSP	55%	33%	17.7%
LUT	50%	40%	28%
FF	17%	16%	13%
Overall max utilization	61%	60%	48%
Latency [μs]	2.82	2.85	3.54
Frequency [MHz]	80	80	80
Scalability	1	1	2
Power consumption [W]	0.887	0.862	0.829

- **Design trade-offs:**
  - high performance that comes with higher resource and power demands,
  - balanced design that attempts to optimize both performance and resource utilization,
  - efficiency-oriented that prioritizes lower resource and power consumption, sometimes at the expense of speed.

Edge AI Applications on FPGA-based Systems

Water Quality Monitoring



Edge AI Applications on FPGA-based Systems

Water Quality Monitoring

Metric	Scenario a): LSTM @ Server		Scenario b): LSTM @ UAV		Scenario c): LSTM @ Buoy	
	Model A (Buoy)	Model B (UAV)	Model A (Buoy)	Model B (UAV)	Model A (Buoy)	Model B (UAV)
FPGA-bsaed Platform	Artix-7	Pynq-Z1	Artix-7	Pynq-Z1	Artix-7	Pynq-Z1
LUT	26%	27%	18%	27%	80%	16%
FF	20%	23%	10%	24%	28%	10%
DSP	6%	4%	33%	40%	3%	0%
BRAM	9%	1%	9%	7%	23%	0%
Reuse factor	16	16	1	8	8	8
Clock Period (ns)	10	12	10	12	10	12
Inference latency (clock cycles)	176	234	141	377	490	124
End-to-end latency (clock cycles)	276	334	241	477	590	224
Inference latency -models A + B- (μs)	4.568		5.934		6.383	
End-to-end inference latency -models A + B- (μs)	6.768		8.134		8.588	
Power Consumption (W)	0.234	1.556	0.295	1.501	0.265	1.434

Post-synthesis results for models implemented on FPGA, across deployment versions. For $n_1 = 5$ and $n_2 = 5$

Edge AI Applications on FPGA-based Systems
Water Quality Monitoring

Metric	Scenario a) 1: LSTM @ Server		Scenario b): LSTM @ UAV		Scenario c): LSTM @ Buoy	
	Model A (Buoy)	Model B (UAV)	Model A (Buoy)	Model B (UAV)	Model A (Buoy)	Model B (UAV)
FPGA-bsaed Platform	Artix-7	Pynq-Z1	Artix-7	Pynq-Z1	Artix-7	Pynq-Z1
LUT	21%	16%	22%	27%	80%	46%
FF	17%	12%	14%	16%	68%	32%
DSP	10%	12%	42%	46%	2%	12%
BRAM	1%	1%	82%	7%	23%	0%
Reuse factor	16	8	4	8	32	8
Clock Period (ns)	10	12	10	12	12	12
Inference latency (clock cycles)	461	488	332	687	560	336
End-to-end latency (clock cycles)	561	488	432	787	660	436
Inference latency -models A + B- (μs)	10.466		11.564		10.752	
End-to-end inference latency -models A + B- (μs)	11.466		13.764		11.952	
Power Consumption (W)	0.227	1.476	0.303	1.498	0.265	1.532

Post-synthesis results for models implemented on FPGA, across deployment versions. For n_1 = 15 and n_2 = 15

Image classification
based on CNN

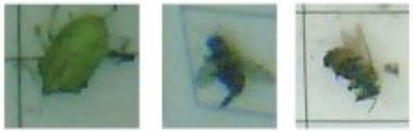
Edge AI Applications on FPGA-based Systems
Image classification based on CNN

- **Collaboration between MLab - ICTP, Universidad Nacional de San Luis, University of Trieste, Nectras**

Nectras IoT trap

Captured image

Other insects



Lobesia botrana

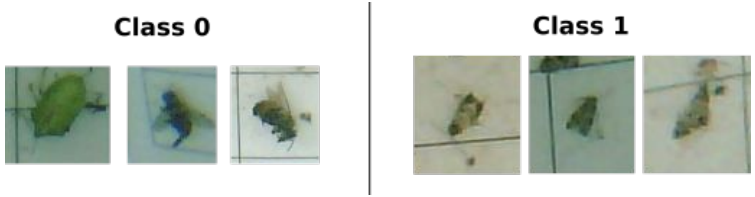
Edge AI Applications on FPGA-based Systems

Image classification based on CNN

Pest24 [6]

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

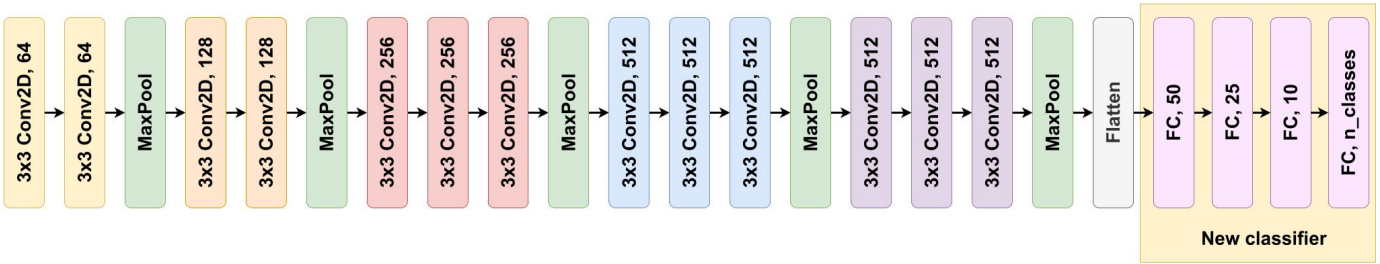
Argentina



Images provided by the system in Argentina.

Edge AI Applications on FPGA-based Systems

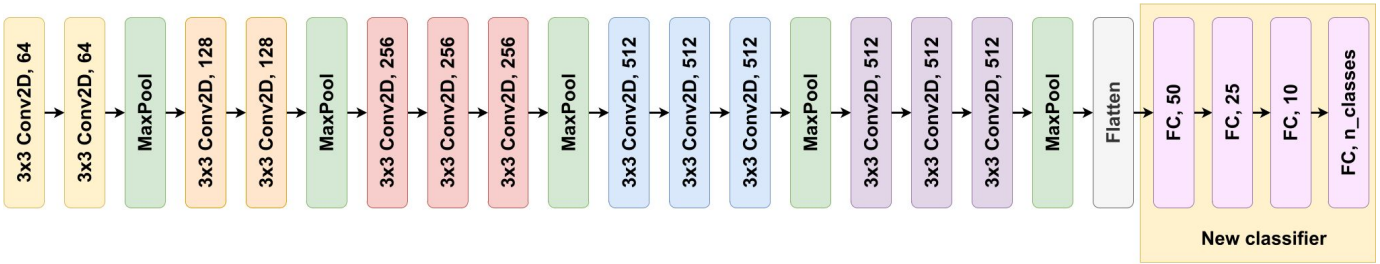
Image classification based on CNN



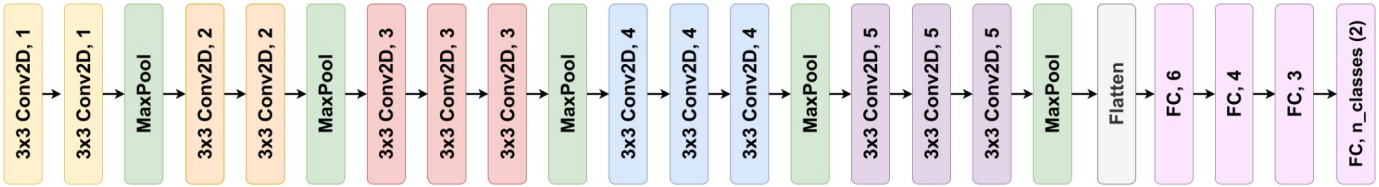
Teacher architecture based on VGG16 and obtained through transfer learning
– 14,818,706 parameters –

Edge AI Applications on FPGA-based Systems

Image classification based on CNN



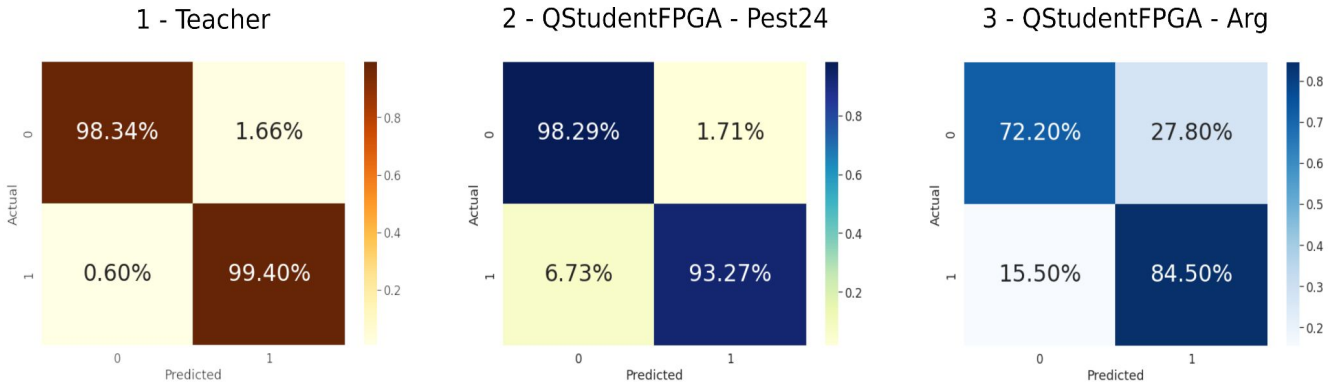
Teacher architecture based on VGG16 and obtained through transfer learning
– 14,818,706 parameters –



Compressed architecture.
Compression ratio: 7,409*x* – in number of parameters –

Edge AI Applications on FPGA-based Systems

Image classification based on CNN



Edge AI Applications on FPGA-based Systems
Image classification based on CNN

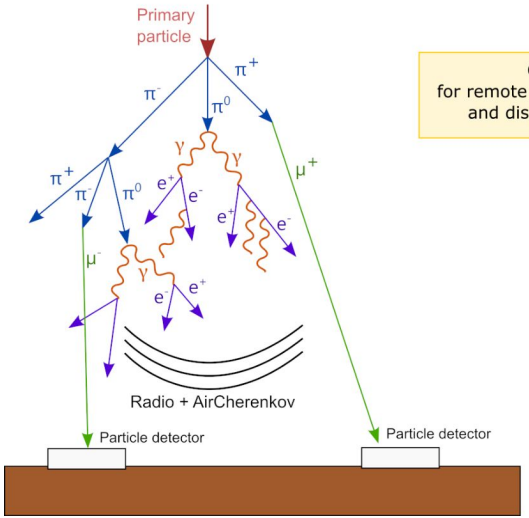
- **Overall accuracy**
  - Teacher architecture: **98.87%**
  - Student architecture: **95.78%**

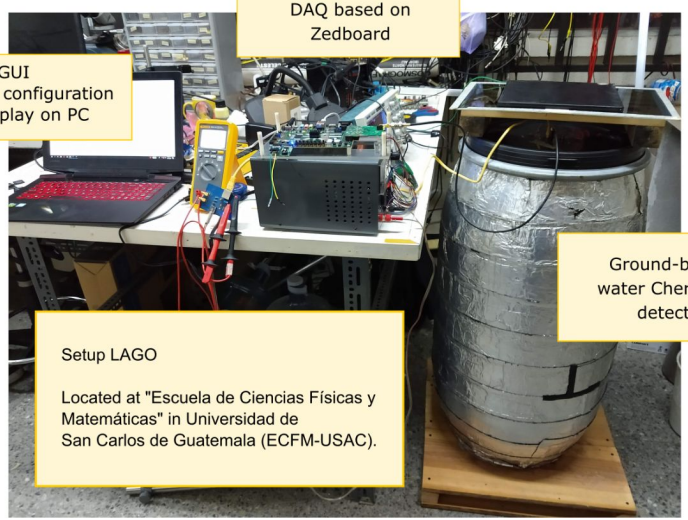
- **SoC memory footprint in terms of resource utilization @200MHz**
  - KRIA platform: **below 21%**
  - PYNQ-Z1 platform: **below 63%**

Pulse shape discriminator
for cosmic rays studies

Edge AI Applications on FPGA-based Systems

Pulse shape discriminator for cosmic rays studies





GUI
for remote configuration
and display on PC

DAQ based on
Zedboard

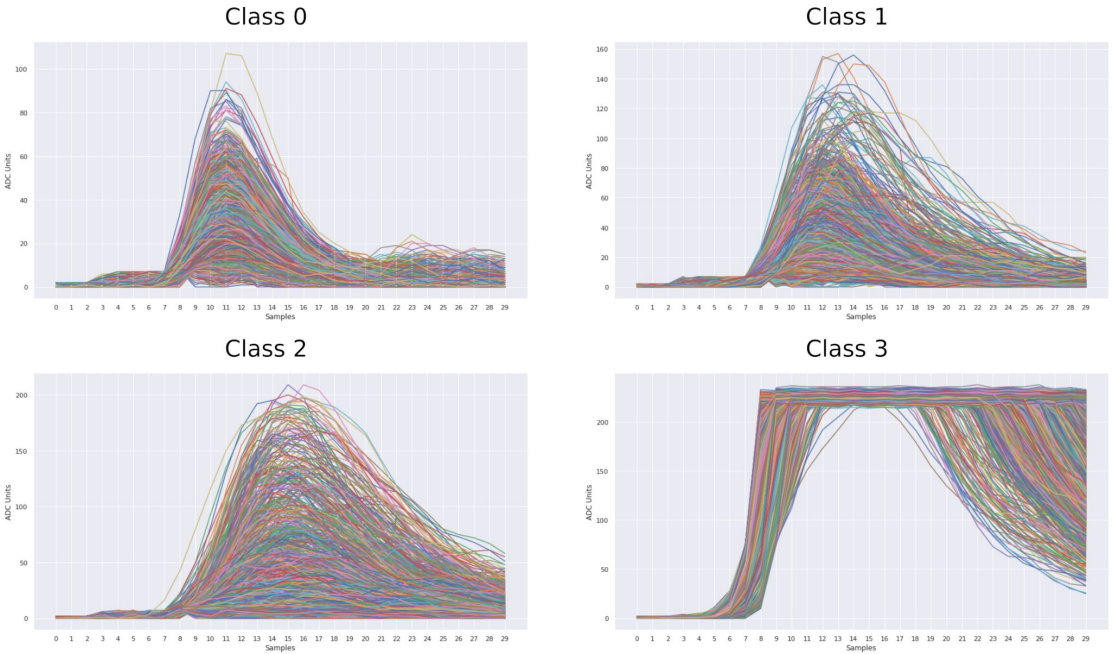
Ground-based
water Cherenkov
detector

Setup LAGO

Located at "Escuela de Ciencias Físicas y Matemáticas" in Universidad de San Carlos de Guatemala (ECFM-USAC).

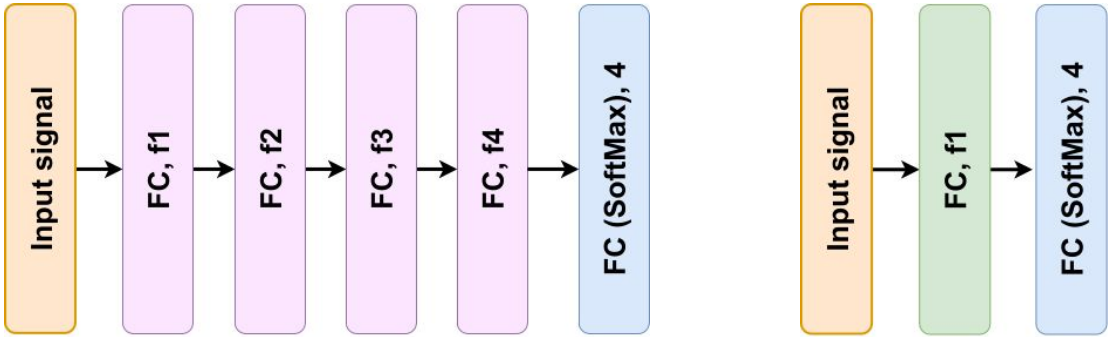
Edge AI Applications on FPGA-based Systems

Pulse shape discriminator for cosmic rays studies



Edge AI Applications on FPGA-based Systems

Pulse shape discriminator for cosmic rays studies



Left. Teacher architecture based on MLP - **16,352** parameters.

Right: Distilled architecture - **529** parameters
Compression ratio: 30.91*x*.

Edge AI Applications on FPGA-based Systems

Pulse shape discriminator for cosmic rays studies

- **Overall accuracy**
  - Teacher architecture: **99.70%**
  - Student architecture: **98.96%**
    - **8-bit fixed point**
    - **Target sparsity: 20%**

- **SoC memory footprint in terms of resource utilization @200MHz**
  - **Below 27%**

- **SoC latency @200MHz**
  - **10 clock cycles**

Edge AI Applications on FPGA-based Systems

Pulse shape discriminator for cosmic rays studies

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™

Edge AI Applications on FPGA-based Systems

Pulse shape discriminator for cosmic rays studies

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™

Copahue volcano
seismic event detection

Edge AI Applications on FPGA-based Systems

Copahue volcano seismic event detection

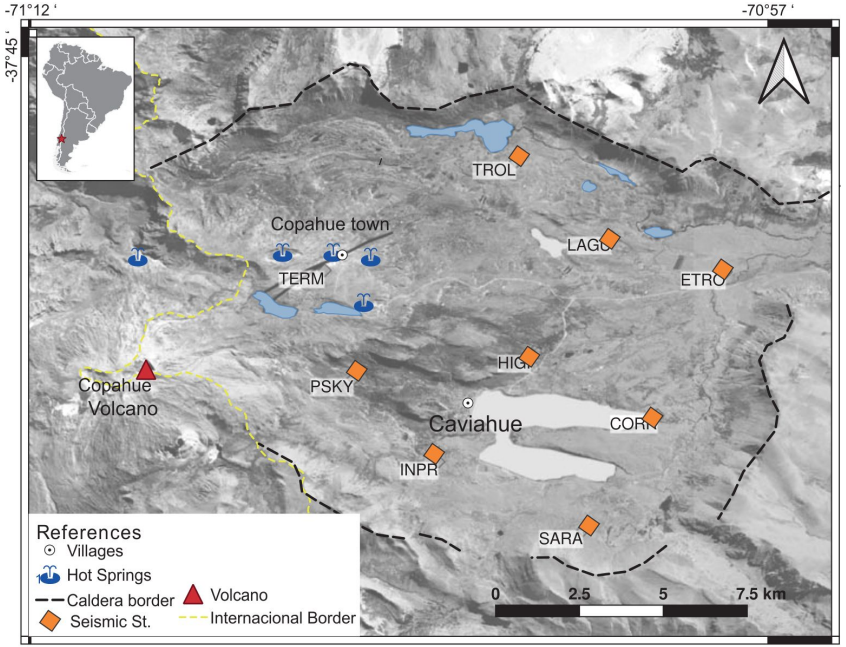
Copahue Volcano

- Active stratovolcano located on the border between Argentina and Chile.
- Its summit is at an altitude of 2,997 m.

Edge AI Applications on FPGA-based Systems

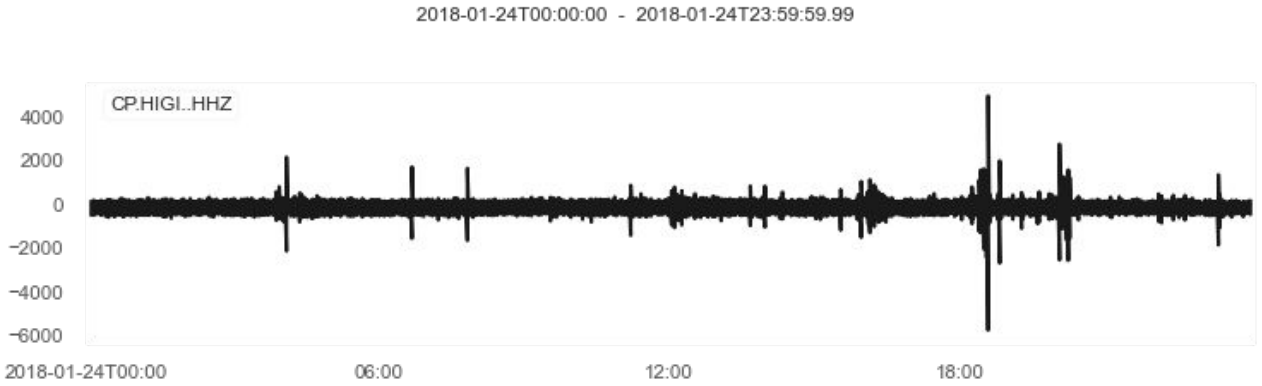
Copahue volcano seismic event detection

Copahue Volcano



Edge AI Applications on FPGA-based Systems

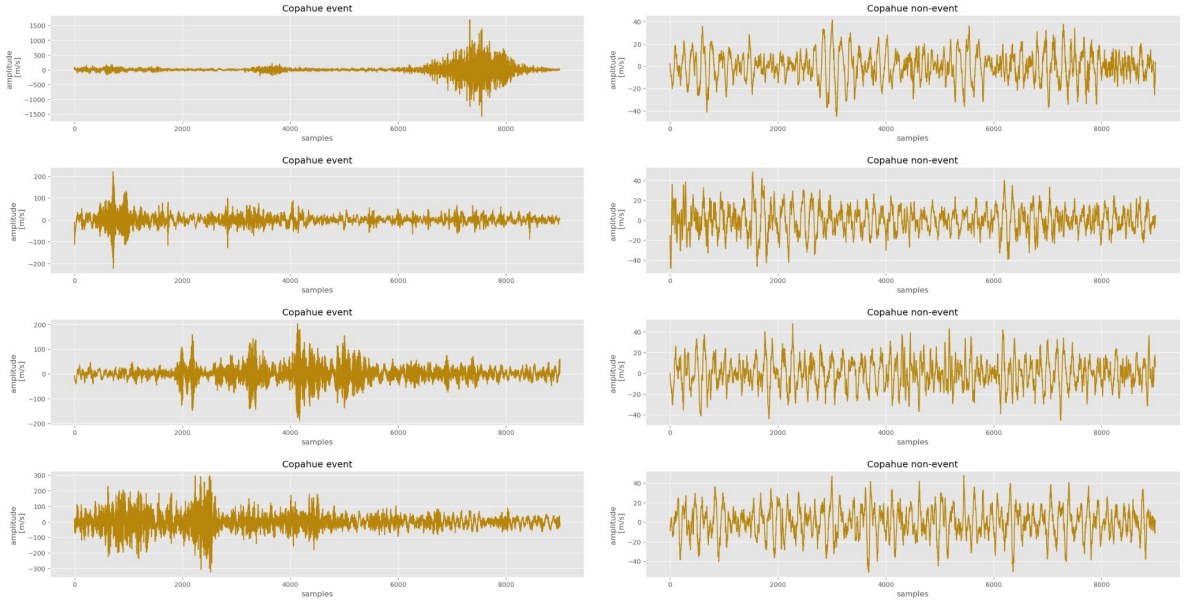
Copahue volcano seismic event detection



The signals used in this research were obtained from Montenegro's Ph.D. thesis [1], and were provided by "Laboratorio de Estudios y Seguimientos de Volcanes Activos' (LESVA), Universidad Nacional de Rio Negro.

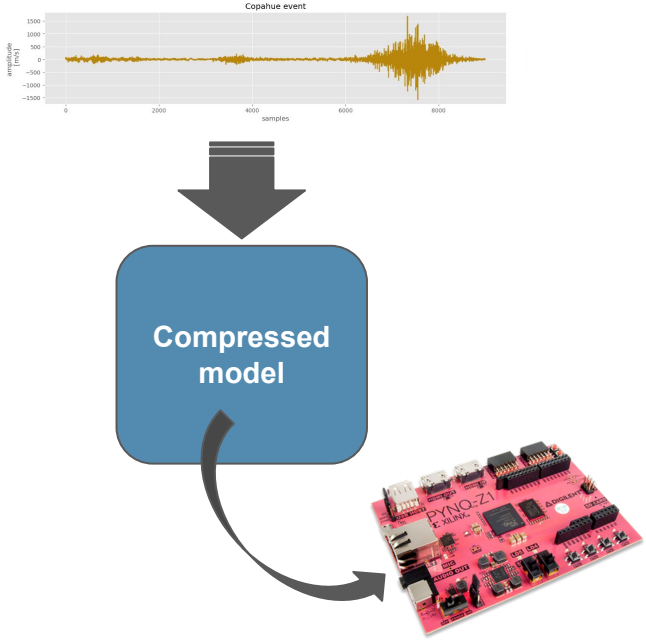
Edge AI Applications on FPGA-based Systems

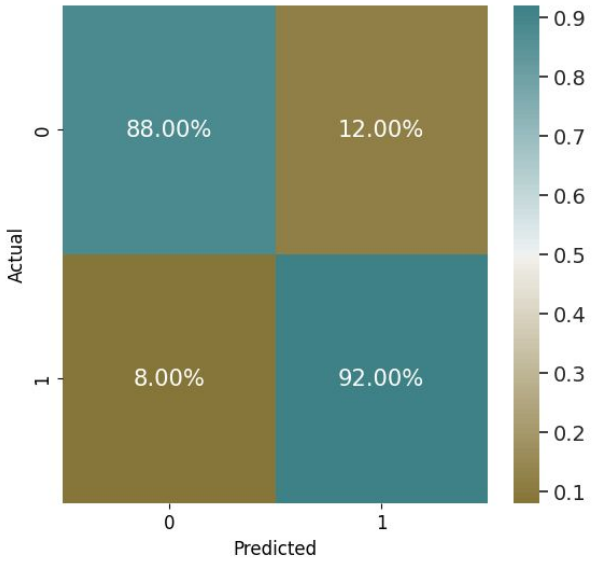
Copahue volcano seismic event detection



Edge AI Applications on FPGA-based Systems

Copahue volcano seismic event detection

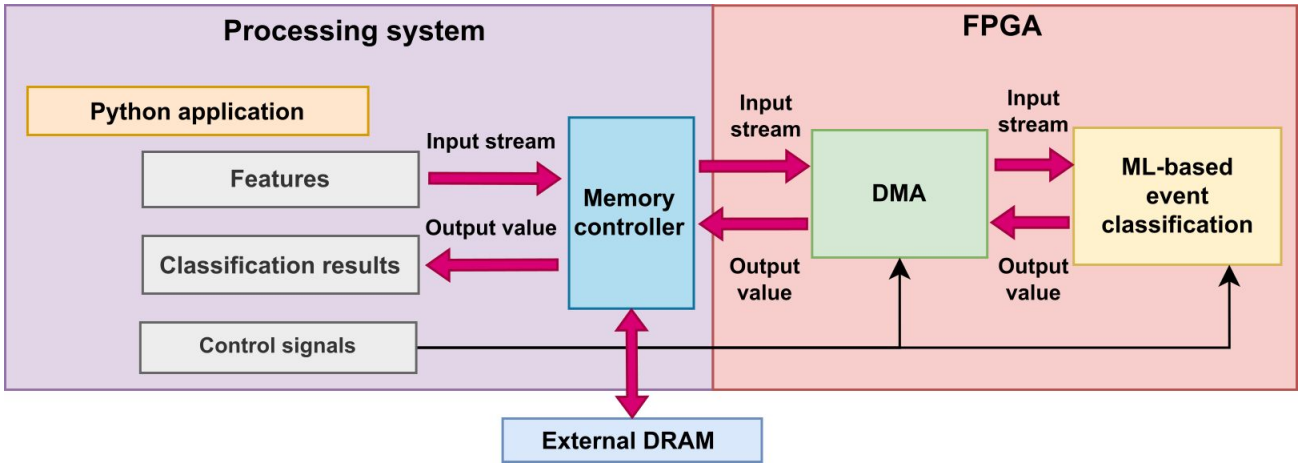




Edge AI Applications on FPGA-based Systems

Copahue volcano seismic event detection

PYNQ



Introduction to
Applied Machine Learning

Introduction to Applied ML

Definition

Machine Learning

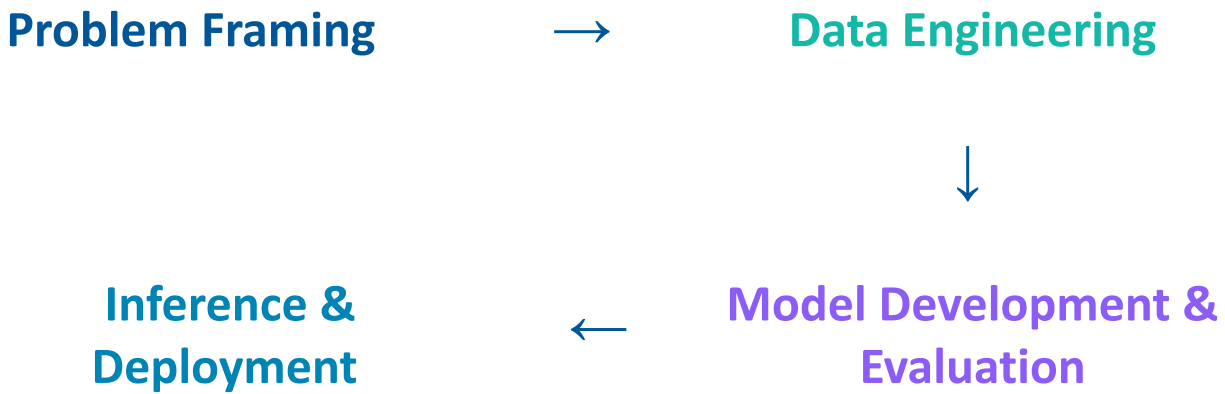
> *"A set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty (such as planning how to collect more data!)."* [1]

> *"...what is the best prediction about the future given some past data? what is the best model to explain some data? what measurement should I perform next? etc."* [1]

[1] *Murphy, K. P., & Learning, M. (2012). A probabilistic perspective. Text book.*

Introduction to Applied ML

Machine learning lifecycle, which forms the data-driven core of modern AI systems.



Introduction to Applied ML

Data-driven model

Input → Model → Output

Introduction to Applied ML

Data-driven model

Input → **Model** → **Output**

Guided process

- Problem definition
- Data & questions
- Architecture selection
- Hardware platform
- Training & validation
- Deployment & iteration

Introduction to Applied ML

Data-driven model

Input → Model → Output

Guided process

- Problem definition
- Data & questions
- Architecture selection
- Hardware platform
- Training & validation
- Deployment & iteration

Model requirements

- Generalizable
- Efficient
- Accurate
- Practical

Introduction to Applied ML

Generalization

Image from
Togootogtokh, E., & Amartuvshin, A. (2018). Deep Learning Approach for Very Similar Objects Recognition Application on Chihuahua and Muffin Problem. *ArXiv, abs/1801.09573*.

Introduction to Applied ML

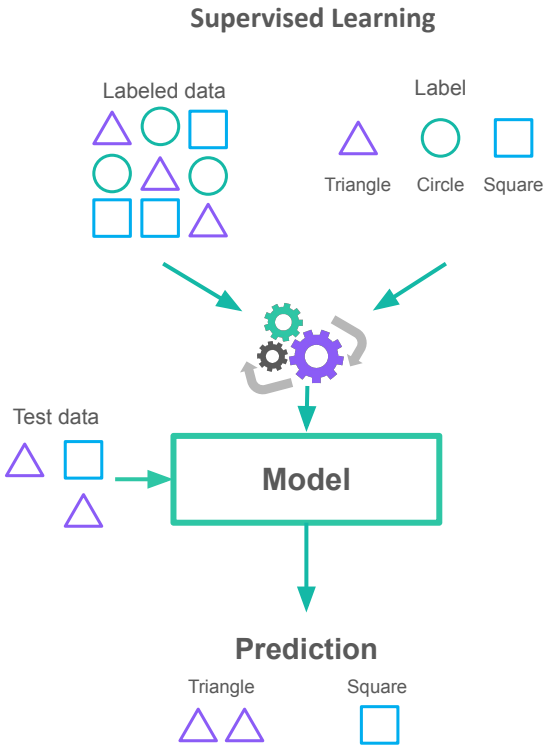
Classification

Machine learning methods are commonly classified into three main categories:

- supervised,
- unsupervised,
- reinforcement learning.

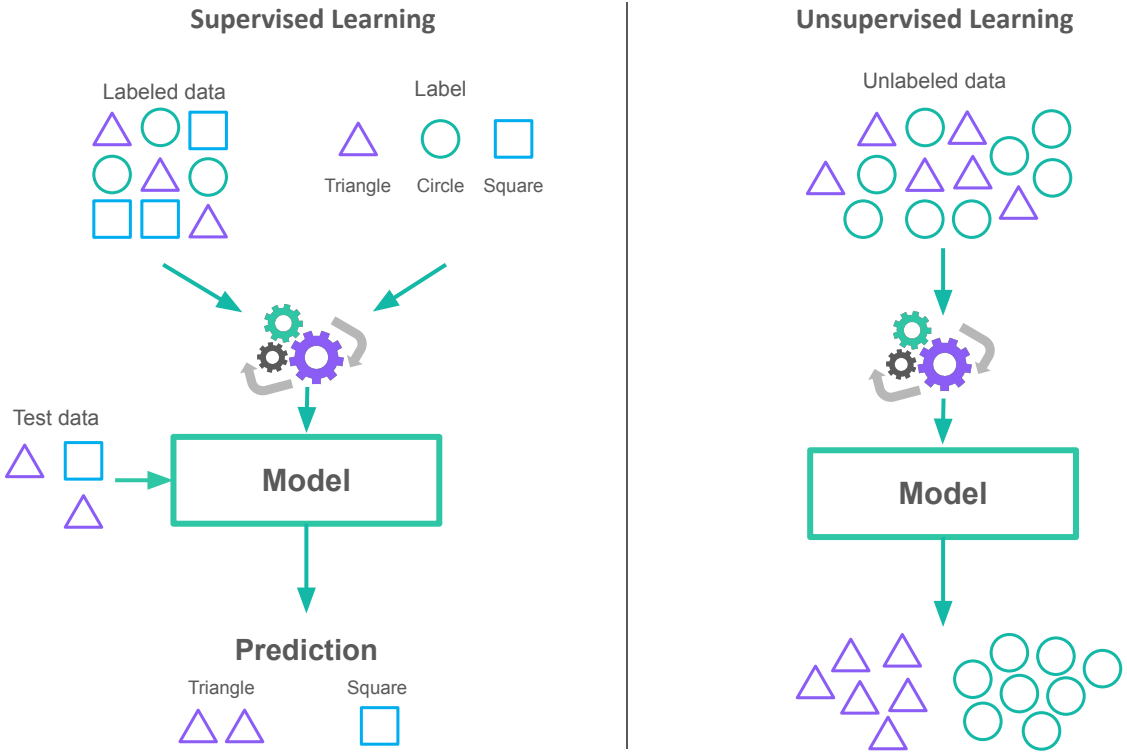
Introduction to Applied ML

Classification



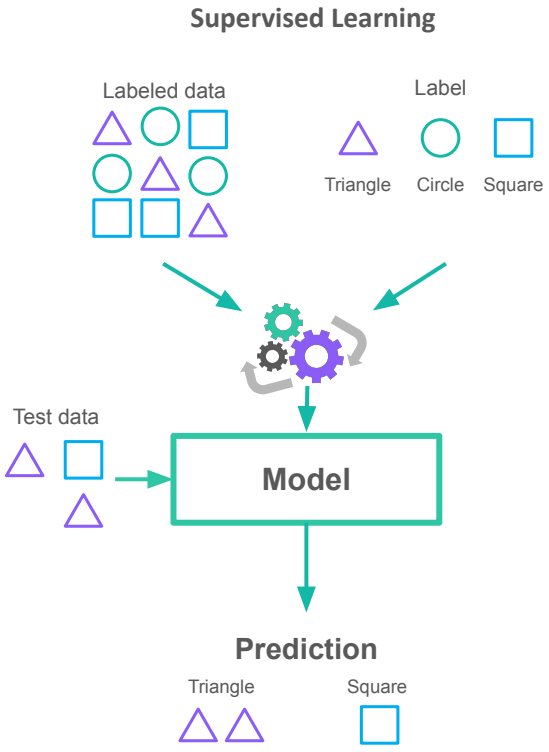
Introduction to Applied ML

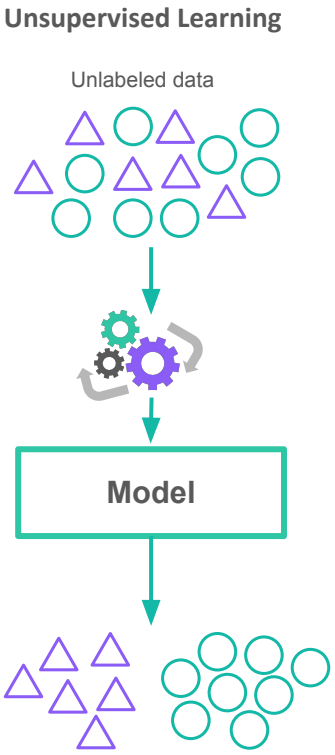
Classification

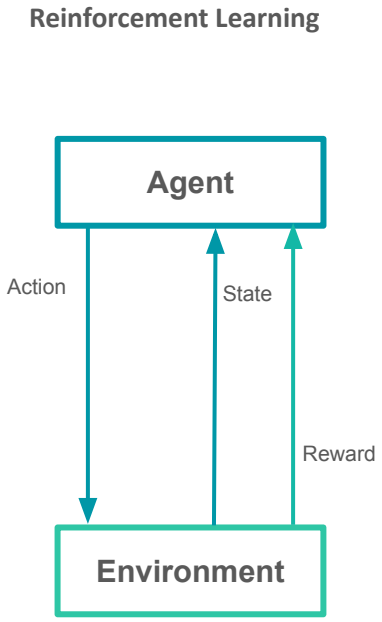


Introduction to Applied ML

Classification







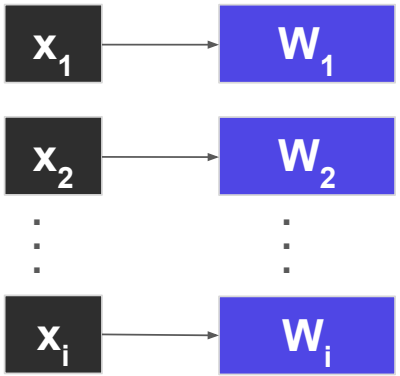
Artificial Neural Networks

Artificial Neural Networks

An artificial neural network (ANN) consists of neurons (nodes) organized into layers. Each neuron applies a transformation to its inputs and passes the result to connected neurons in the following layer.

Artificial Neural Networks

An artificial neural network (ANN) consists of neurons (nodes) organized into layers. Each neuron applies a transformation to its inputs and passes the result to connected neurons in the following layer.

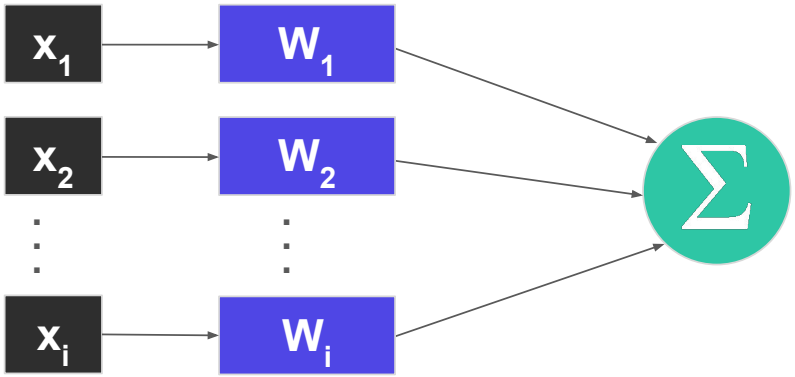


Weights are numerical parameters that capture the importance of each input within a model.

During training, the network adjusts these values to reduce errors and refine its understanding of the data.

Artificial Neural Networks

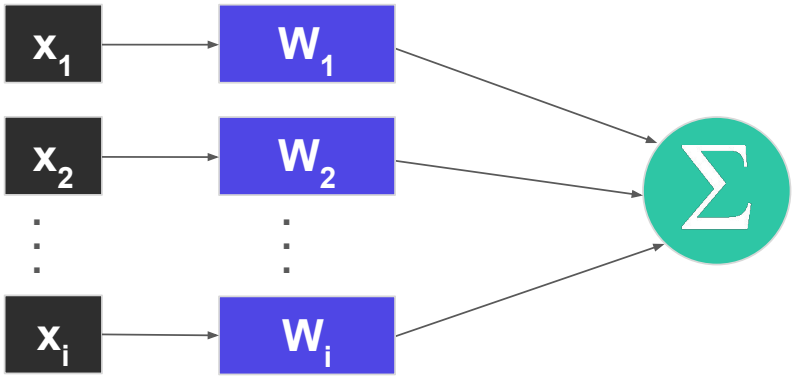
An artificial neural network (ANN) consists of neurons (nodes) organized into layers. Each neuron applies a transformation to its inputs and passes the result to connected neurons in the following layer.



The neuron **multiplies** each input by its weight and then **sums** all the results.

Artificial Neural Networks

An artificial neural network (ANN) consists of neurons (nodes) organized into layers.
Each neuron applies a transformation to its inputs and passes the result to connected neurons in the following layer.

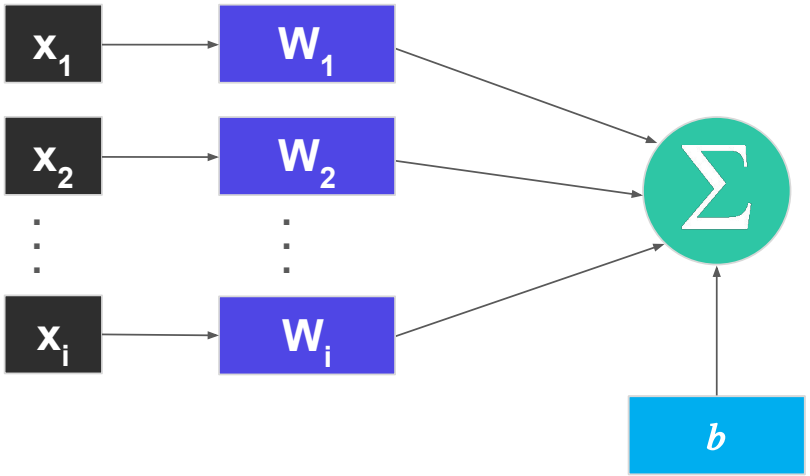


The neuron **multiplies** each input by its weight and then **sums** all the results.

$$z = \sum_{i=1}^{n} x_i w_i$$

Artificial Neural Networks

An artificial neural network (ANN) consists of neurons (nodes) organized into layers. Each neuron applies a transformation to its inputs and passes the result to connected neurons in the following layer.

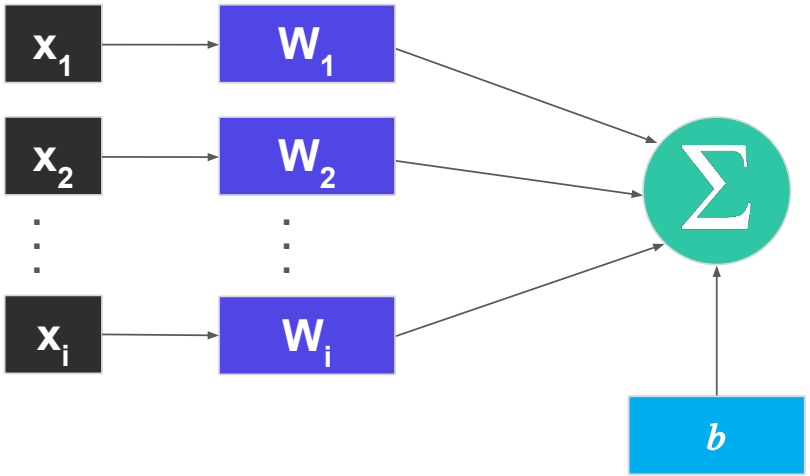


A **bias** is a trainable value added to a neuron's output before the activation function.

It allows the model to shift decision boundaries and learn patterns that do not pass through the origin.

Artificial Neural Networks

An artificial neural network (ANN) consists of neurons (nodes) organized into layers.
Each neuron applies a transformation to its inputs and passes the result to connected neurons in the following layer.

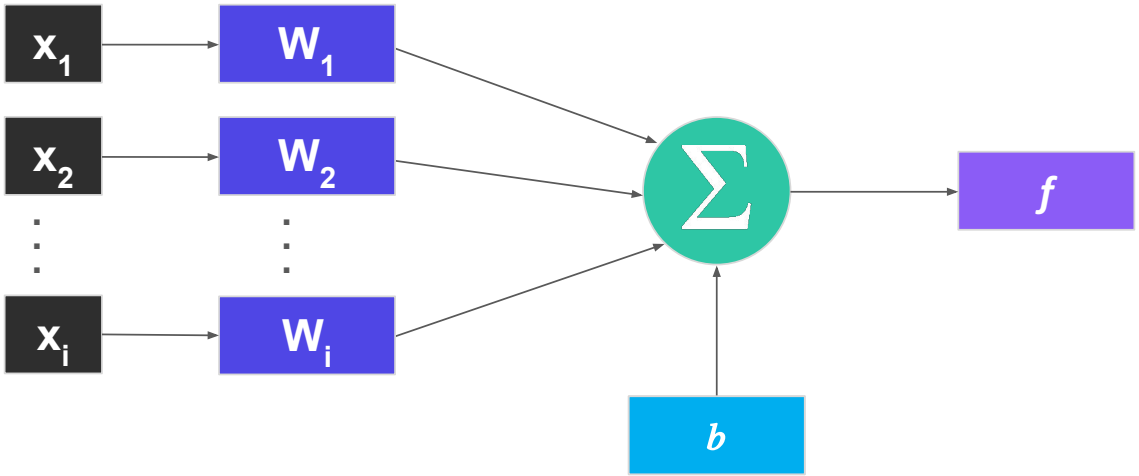


After computing the weighted sum of the inputs, a **bias** term b is added

$$z = \sum_{i=1}^{n} x_i w_i + b$$

Artificial Neural Networks

An artificial neural network (ANN) consists of neurons (nodes) organized into layers.
Each neuron applies a transformation to its inputs and passes the result to connected neurons in the following layer.

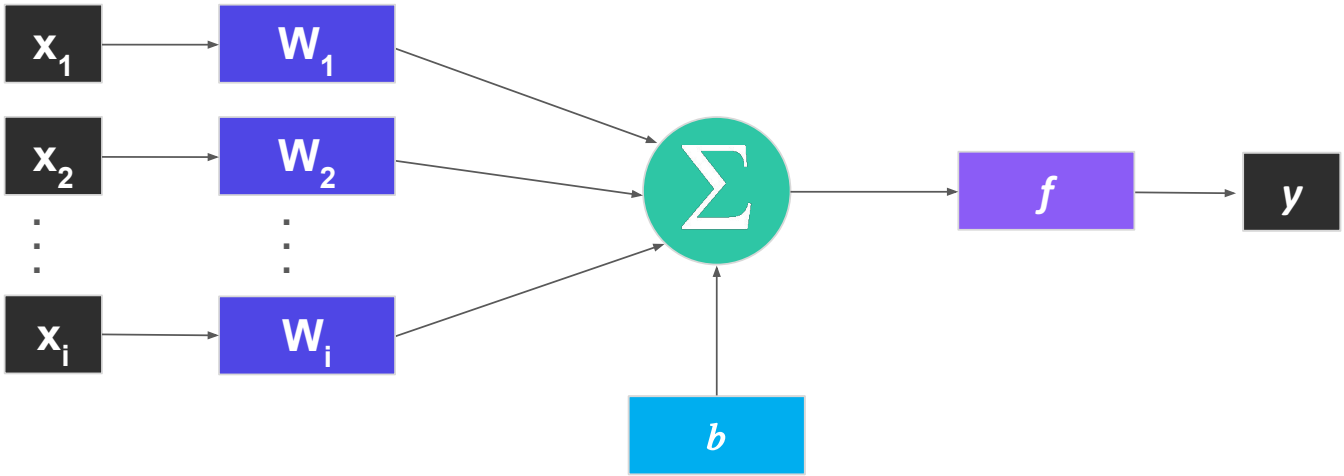


An **activation function** defines how a neuron transforms its input into an output.

It introduces **non-linearity**, allowing the network to learn complex patterns.

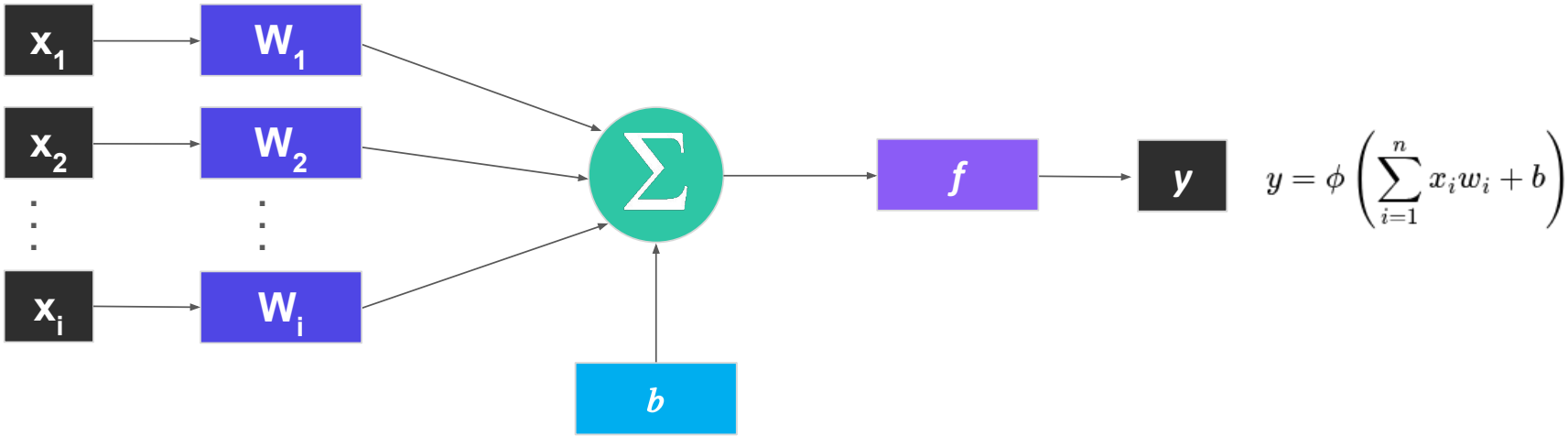
Artificial Neural Networks

An artificial neural network (ANN) consists of neurons (nodes) organized into layers.
Each neuron applies a transformation to its inputs and passes the result to connected neurons in the following layer.



Artificial Neural Networks

An artificial neural network (ANN) consists of neurons (nodes) organized into layers.
Each neuron applies a transformation to its inputs and passes the result to connected neurons in the following layer.



Artificial Neural Networks

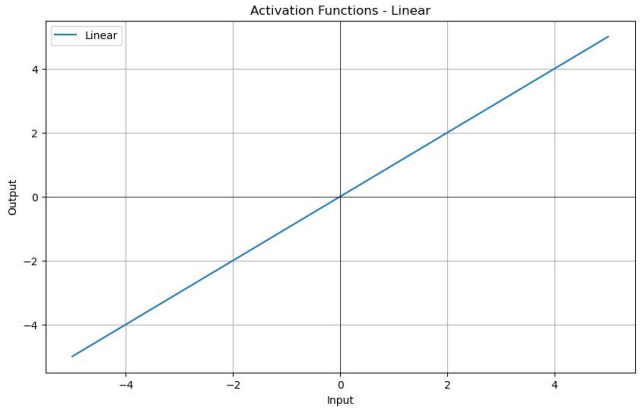
- **Types of activation functions**
  - Linear
  - Non-Linear: Sigmoid, ReLu, Leaky ReLu, Softmax.

- **Tasks:**
  - Classification: Sigmoid (binary), Softmax (multi-class).
  - Hidden layers: ReLU, Leaky ReLU.
  - Regression: Linear activation or ReLU.

Artificial Neural Networks

Linear

$$f(x) = ax$$

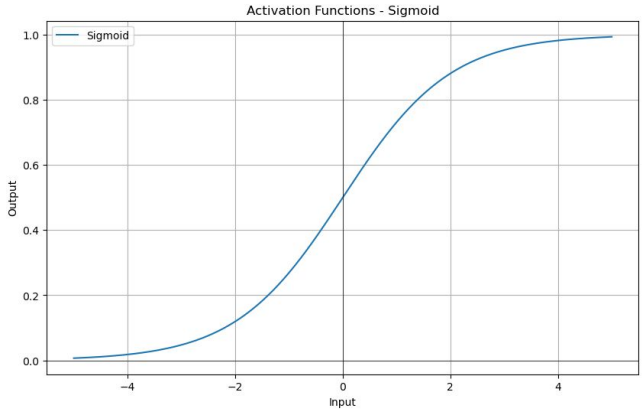


- Simple
- Useful for regression
- Cannot model non-linearity

Artificial Neural Networks

Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

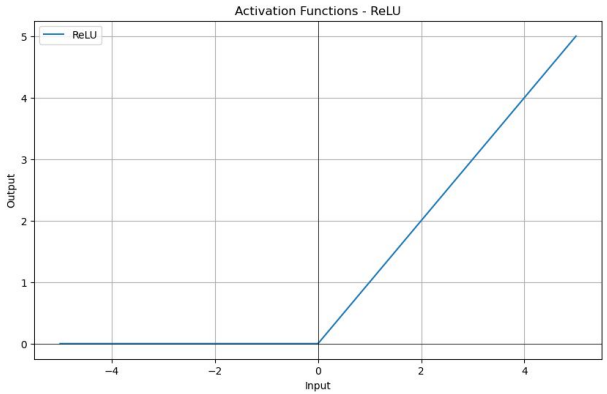


- Binary classification
- Slow training
- Good for probabilities

Artificial Neural Networks

ReLu

$$f(x) = \max(0, x)$$

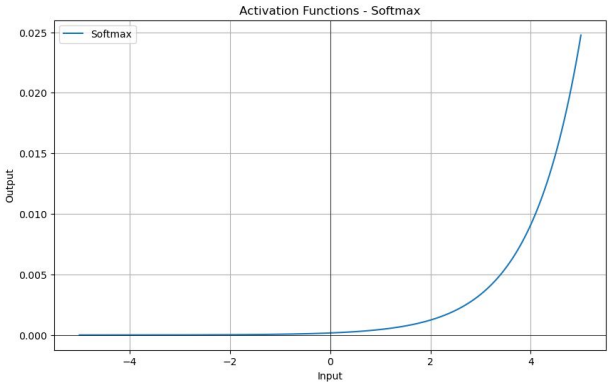


- Deep networks
- Neurons can stuck at zero
- Fast

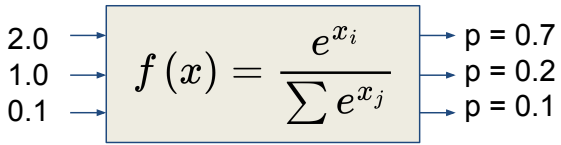
Artificial Neural Networks

Softmax

$$f(x) = \frac{e^{x_i}}{\sum e^{x_j}}$$



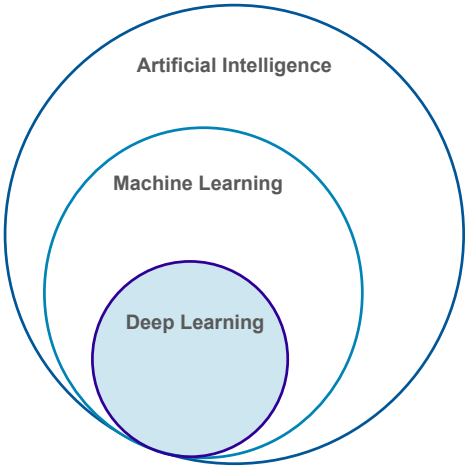
- Multi-class classification
- Computationally expensive
- Converts outputs into probabilities



Artificial Neural Networks

Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN)

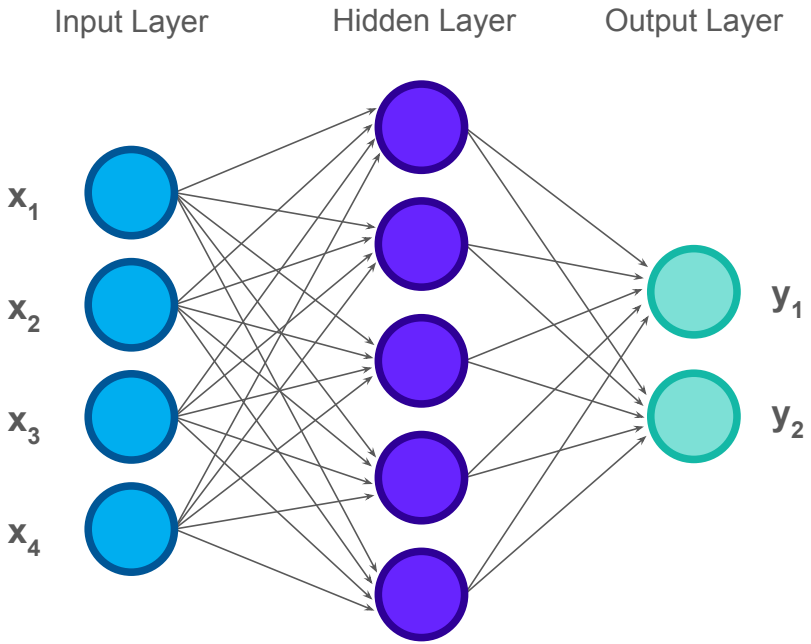
MLPs and CNNs fall under the category of **Deep Learning**, as they are neural networks with multiple layers.



Artificial Neural Networks

Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron is a feed-forward artificial neural network composed of fully connected layers.



Artificial Neural Networks

Multi-Layer Perceptron (MLP)

Considering a given layer *l*, the output $\mathbf{a}^{(l)}$ can be defined as:

$$\mathbf{a}^{(l)} = f(z^{(l)})$$

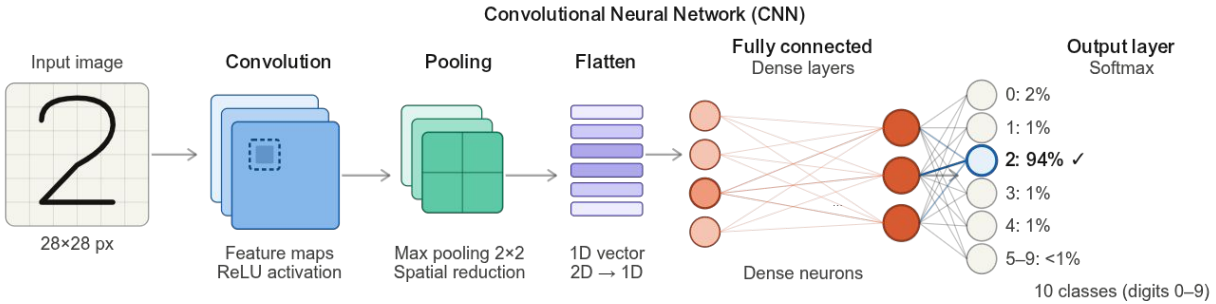
$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

$W^{(l)}$: weights matrix of the layer *l*.
$a^{(l-1)}$: output of the previous layer (*l-1*)
$b^{(l)}$: bias vector of layer *l*.
$f(.)$: activation function

Artificial Neural Networks

Convolutional Neural Network (CNN)

A CNN is a neural network that uses convolutional layers to extract features from images for classification tasks.



What each layer does

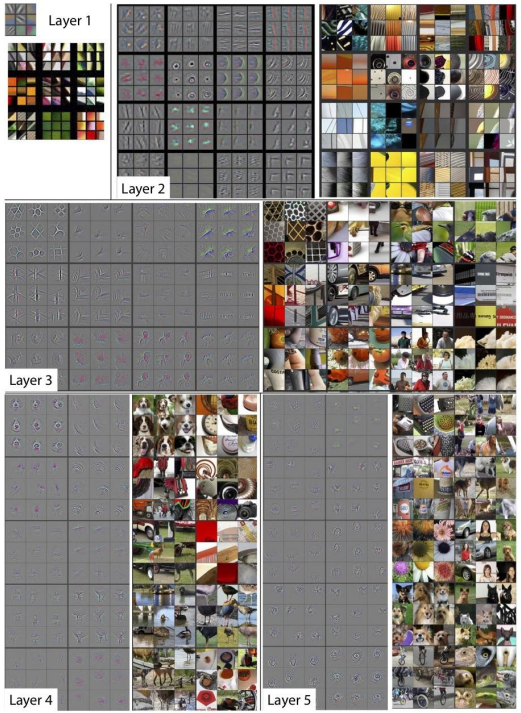
- Convolution: applies learned filters to detect local features (edges, textures, shapes)
- Pooling: downsamples feature maps, preserving the most important activations
- Flatten: reshapes 2D feature maps into a single 1D vector for classification
- Fully connected + Softmax: every neuron connects to all others, outputs a probability per class

Artificial Neural Networks

Convolutional Neural Network (CNN)

Basic layers that make up the CNN architecture

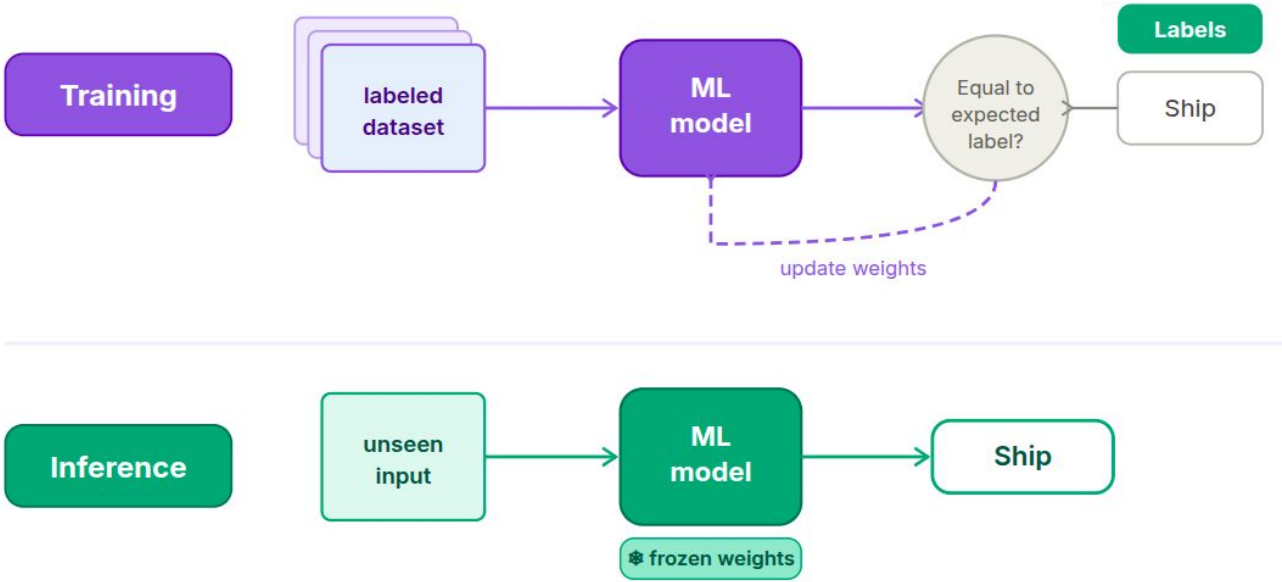
- The first layer: captures basic features.
  - For example, horizontal and vertical edges.
- The output moves to the next layer, which identifies more complex features.
  - For instance, corners or combinations of edges.
- As the network deepens, it becomes capable of recognizing even more intricate features, such as faces, objects, and more.



Training and Inference

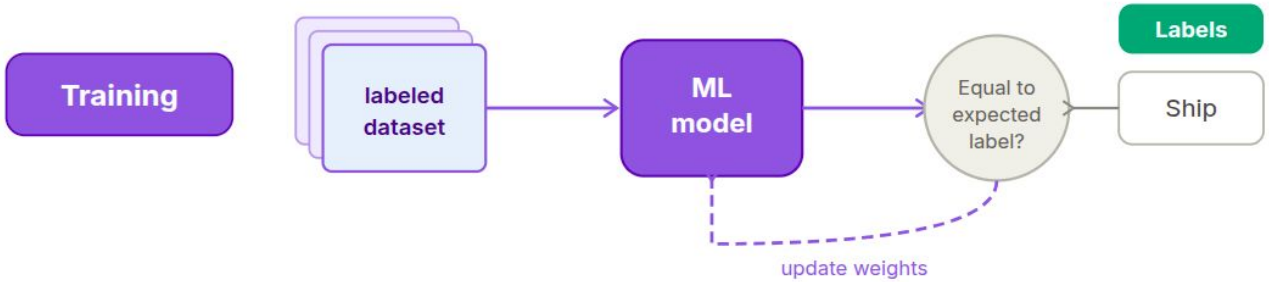
Artificial Neural Networks

Training and Inference



Artificial Neural Networks

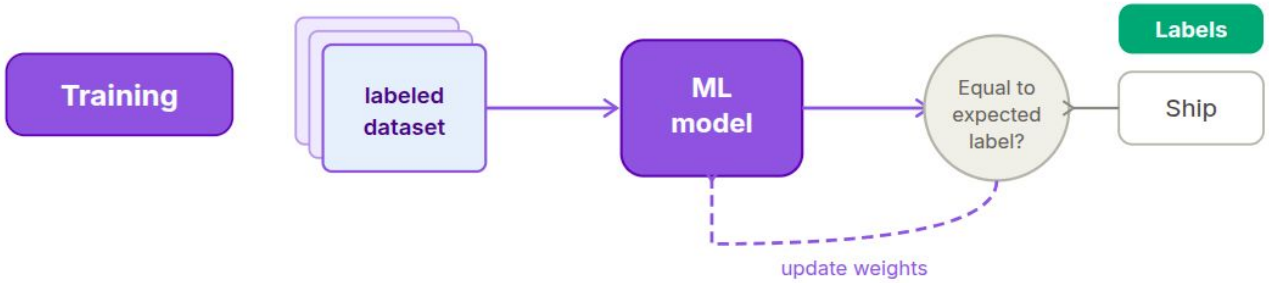
Training



The **training phase** involves adjusting the network's weights, allowing the neural network to learn from data.

Artificial Neural Networks

Training



Training steps:

- Training the model on the training dataset
- Computing the loss and updating model parameters
- Evaluating performance using a validation dataset

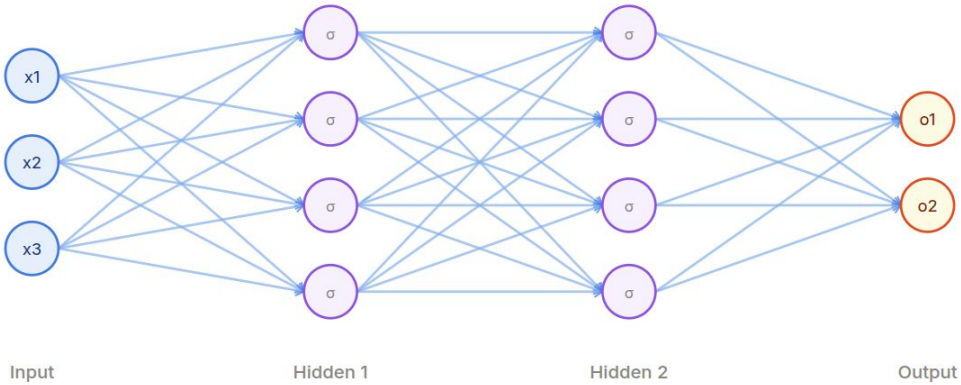
These steps are repeated iteratively until the model reaches satisfactory performance.

Artificial Neural Networks

Training Loop

1 · Forward pass | 2 · Loss | 3 · Backward pass | 4 · Weight update

Forward pass: signal flows from input to output →



Forward pass

Input data flows layer by layer. Each neuron computes $z = \Sigma\, w_i x_i + b$ and applies $a = \sigma(z)$. The network produces a prediction $\hat{y}$.

Loss

The loss function measures how wrong the prediction is: $L = \frac{1}{2}(y - \hat{y})^2$. The higher the loss, the further the model is from the correct answer.

Backward pass

Using the chain rule, gradients $\partial L/\partial W$ are computed from output back to input. Each weight learns how much it contributed to the error.

Weight update

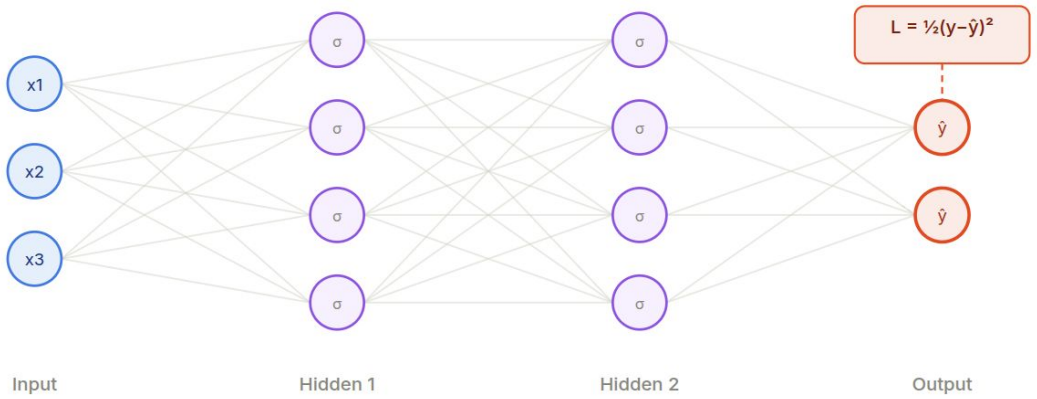
The optimizer updates each weight: $W \leftarrow W - \eta \cdot \partial L/\partial W$. The learning rate η controls the step size. Repeats for every batch and epoch.

Artificial Neural Networks

Training Loop

1 · Forward pass | **2 · Loss** | 3 · Backward pass | 4 · Weight update

Loss: error between prediction ŷ and target y is computed



Forward pass

Input data flows layer by layer. Each neuron computes $z = \Sigma\, w_i x_i + b$ and applies $a = \sigma(z)$. The network produces a prediction ŷ.

Loss

The loss function measures how wrong the prediction is: $L = \frac{1}{2}(y - \hat{y})^2$. The higher the loss, the further the model is from the correct answer.

Backward pass

Using the chain rule, gradients $\partial L/\partial W$ are computed from output back to input. Each weight learns how much it contributed to the error.

Weight update

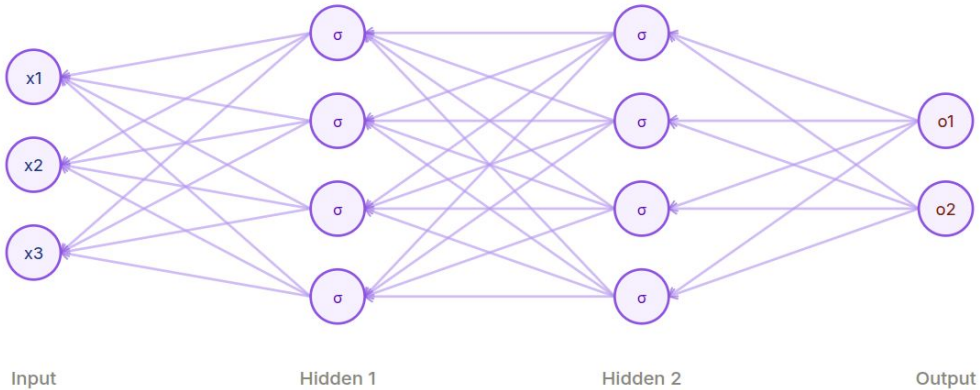
The optimizer updates each weight: $W \leftarrow W - \eta \cdot \partial L/\partial W$. The learning rate η controls the step size. Repeats for every batch and epoch.

Artificial Neural Networks

Training Loop

1 · Forward pass | 2 · Loss | 3 · Backward pass | 4 · Weight update

Backward pass: gradients flow from output back to input ←



Forward pass

Input data flows layer by layer. Each neuron computes $z = \Sigma\, w_i x_i + b$ and applies $a = \sigma(z)$. The network produces a prediction $\hat{y}$.

Loss

The loss function measures how wrong the prediction is: $L = \frac{1}{2}(y - \hat{y})^2$. The higher the loss, the further the model is from the correct answer.

Backward pass

Using the chain rule, gradients $\partial L/\partial W$ are computed from output back to input. Each weight learns how much it contributed to the error.

Weight update

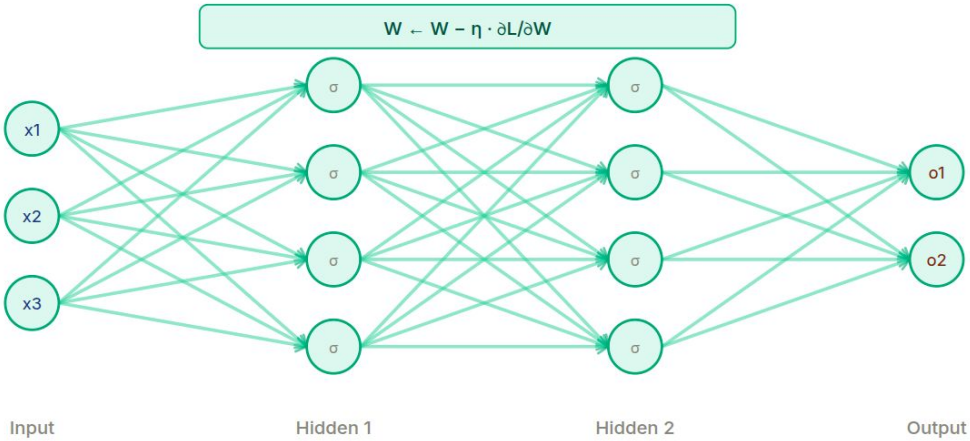
The optimizer updates each weight: $W \leftarrow W - \eta \cdot \partial L/\partial W$. The learning rate η controls the step size. Repeats for every batch and epoch.

Artificial Neural Networks

Training Loop

1 · Forward pass | 2 · Loss | 3 · Backward pass | 4 · Weight update

Weight update: $W \leftarrow W - \eta \cdot \partial L/\partial W$



Forward pass

Input data flows layer by layer. Each neuron computes $z = \Sigma\, w_i x_i + b$ and applies $a = \sigma(z)$. The network produces a prediction $\hat{y}$.

Loss

The loss function measures how wrong the prediction is: $L = \frac{1}{2}(y - \hat{y})^2$. The higher the loss, the further the model is from the correct answer.

Backward pass

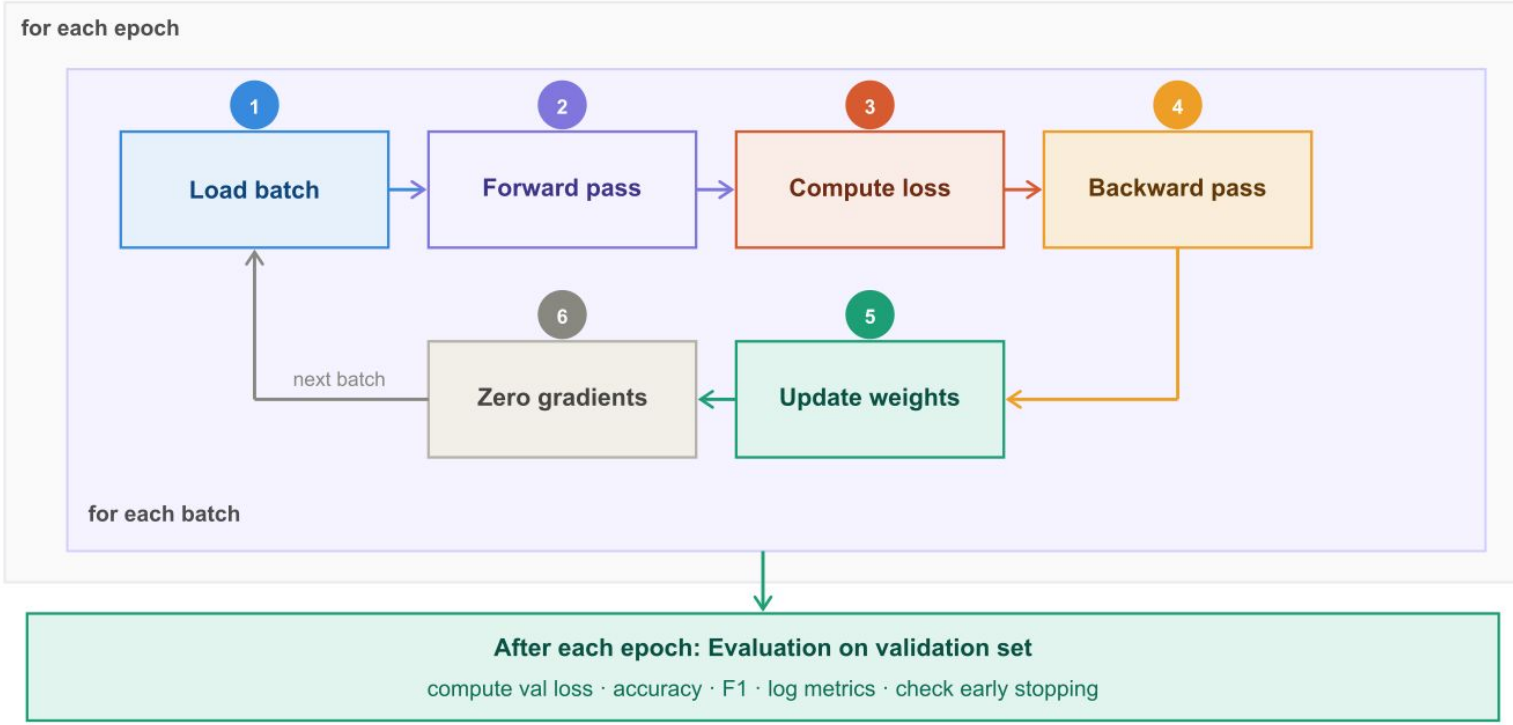
Using the chain rule, gradients $\partial L/\partial W$ are computed from output back to input. Each weight learns how much it contributed to the error.

Weight update

The optimizer updates each weight: $W \leftarrow W - \eta \cdot \partial L/\partial W$. The learning rate η controls the step size. Repeats for every batch and epoch.

Artificial Neural Networks

Training Loop



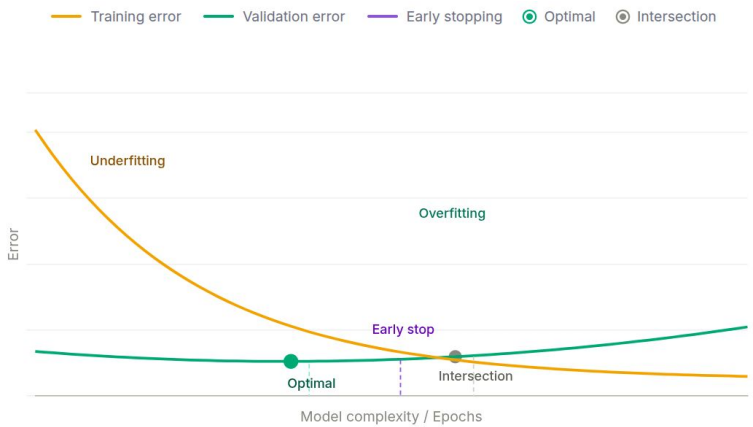
Artificial Neural Networks

Generalization

Underfitting, Overfitting, and Optimal

Underfitting: model too simple to capture patterns.
Overfitting: model memorizes training data.
Optimal fit: balance between bias and variance.

As model complexity increases, training error always decreases, but validation error reaches a minimum and then increases due to overfitting.



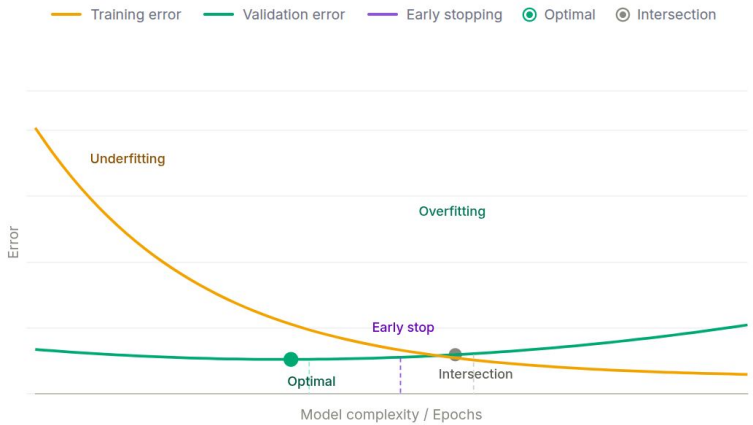
Artificial Neural Networks

Generalization

Underfitting, Overfitting, and Optimal

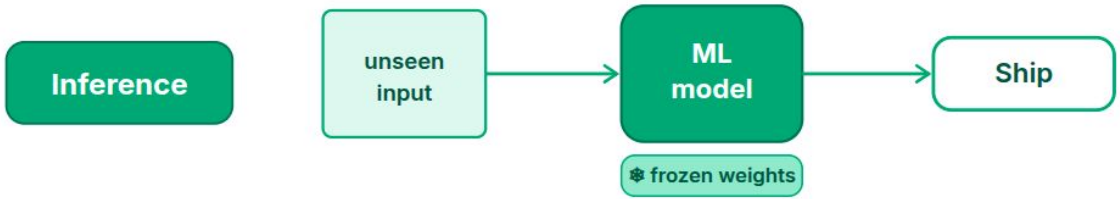
Underfitting and overfitting are training-time phenomena.

Once the model is trained and deployed, inference operates at a fixed point on this curve.



Artificial Neural Networks

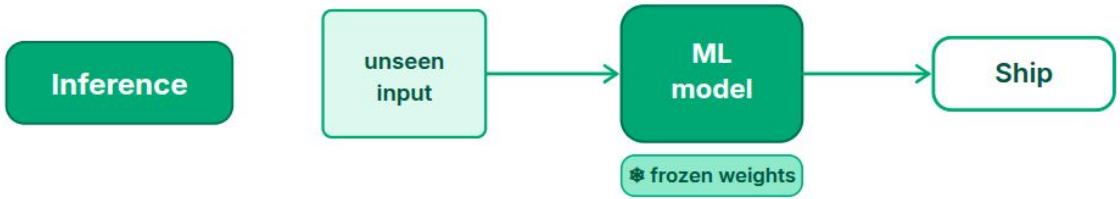
Inference



- **Model Inference**
  - Uses a trained model to make predictions on new data.
  - Critical for latency, efficiency, and deployment.
  - Drives hardware and system-level design choices.

Artificial Neural Networks

Inference



- Edge AI enables **AI model inference directly on the device**, without relying on remote servers or cloud infrastructure.
- It is a powerful alternative for applications where tasks are not computationally intensive and low latency is critical.

Artificial Neural Networks

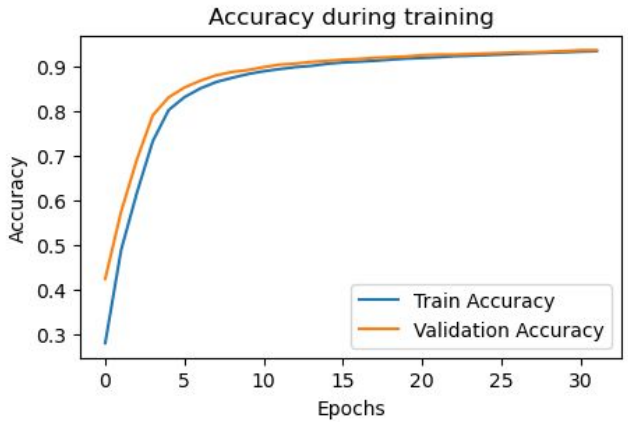
Inference

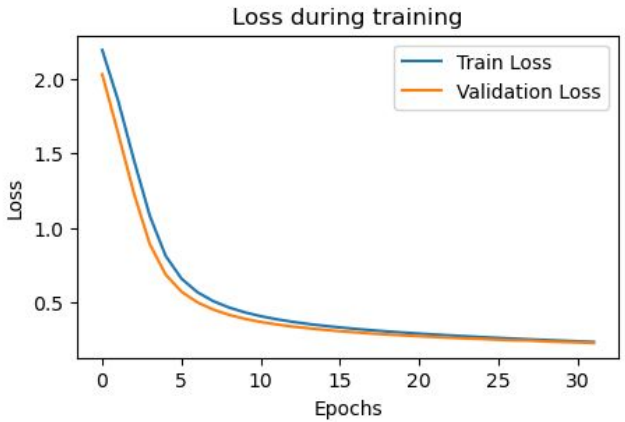
Training	Inference
Uses labeled data	Uses unseen input data
Learns model parameters (weights, bias)	Uses fixed parameters
Computes loss and gradients	No loss or gradients
Updates model parameters iteratively	No parameter updates
Computationally intensive	Optimized for low latency
Typically performed offline	Executed in deployment (edge / cloud)

Training and Inference
Metrics

Training and Inference Metrics

Accuracy and Loss during training





Training and Inference Metrics

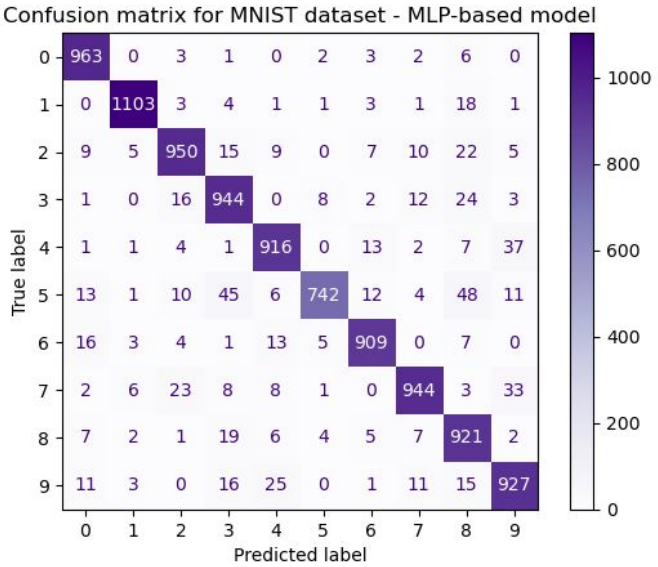
Confusion Matrix

- **True Positives (TP)**
  - The number of instances where the model correctly predicted the positive class.
- **True Negatives (TN)**
  - The number of instances where the model correctly predicted the negative class.
- **False Positives (FP)**
  - The number of instances where the model incorrectly predicted the positive class.
- **False Negatives (FN)**
  - The number of instances where the model incorrectly predicted the negative class.

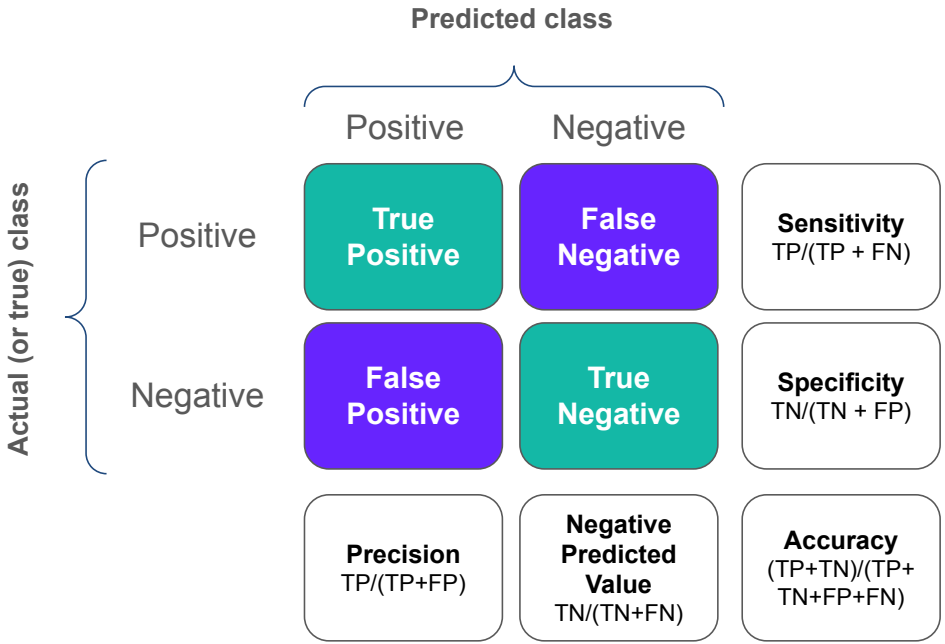
Training and Inference Metrics

Confusion Matrix

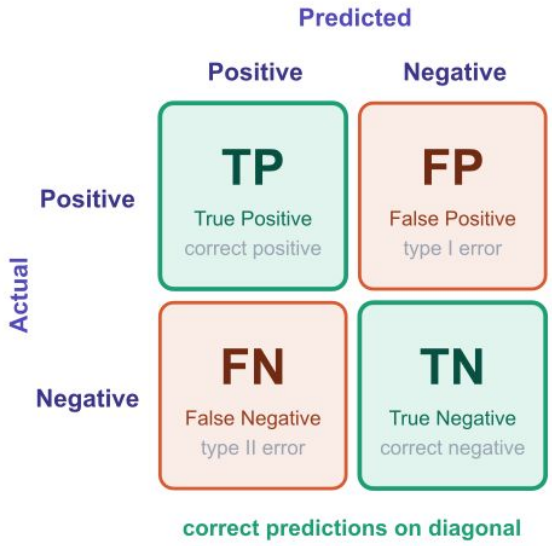
True Positive	False Negative
False Positive	True Negative



Training and Inference Metrics



Training and Inference Metrics



Accuracy — overall correctness

`(TP + TN) / (TP + TN + FP + FN)`

Precision — of all predicted POSITIVE, how many correct?

`TP / (TP + FP)`

Recall (Sensitivity) — of all actual POSITIVE, how many found?

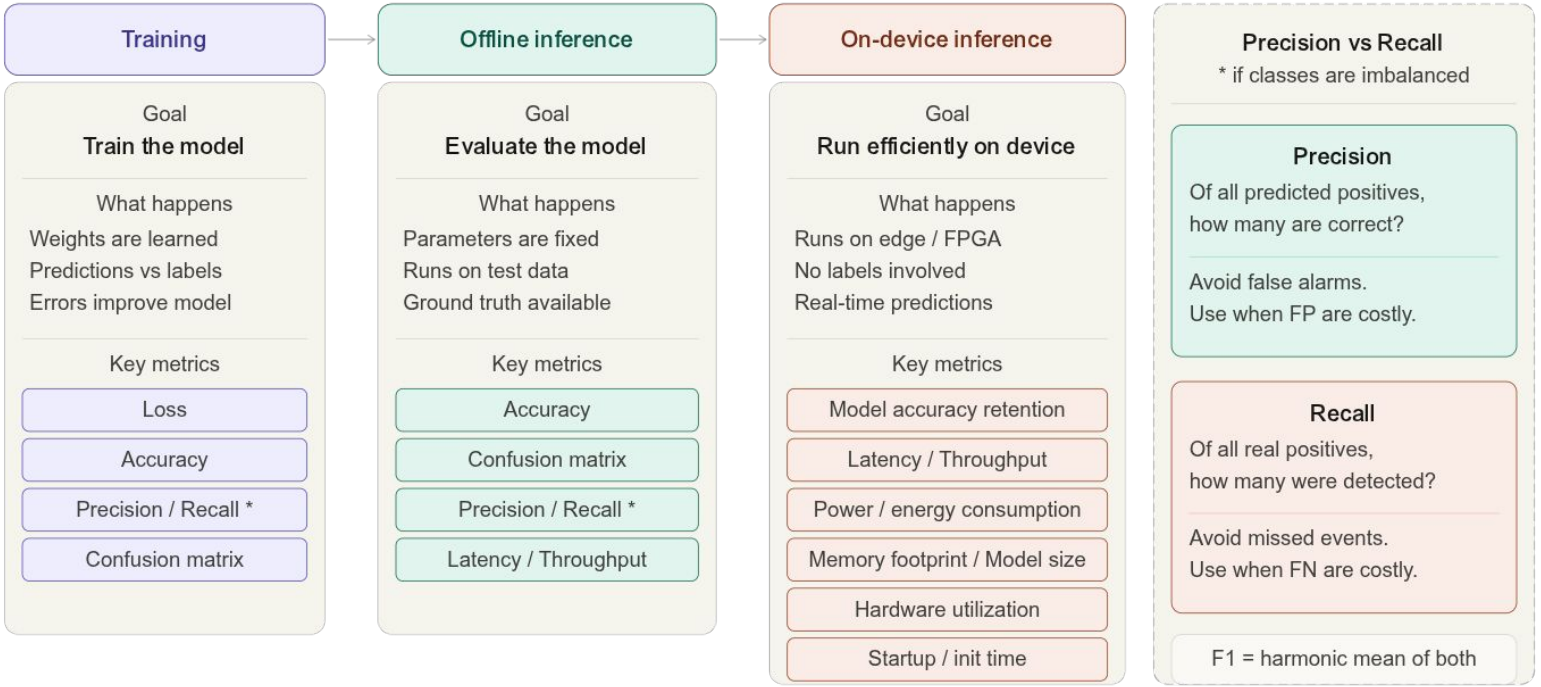
`TP / (TP + FN)`

F1 Score — harmonic mean · balances P and R

`2 · (Precision · Recall) / (P + R)`

Precision focuses on avoiding false alarms | **Recall** focuses on avoiding missed events.

Training and Inference Metrics



Metrics shown apply to classification tasks

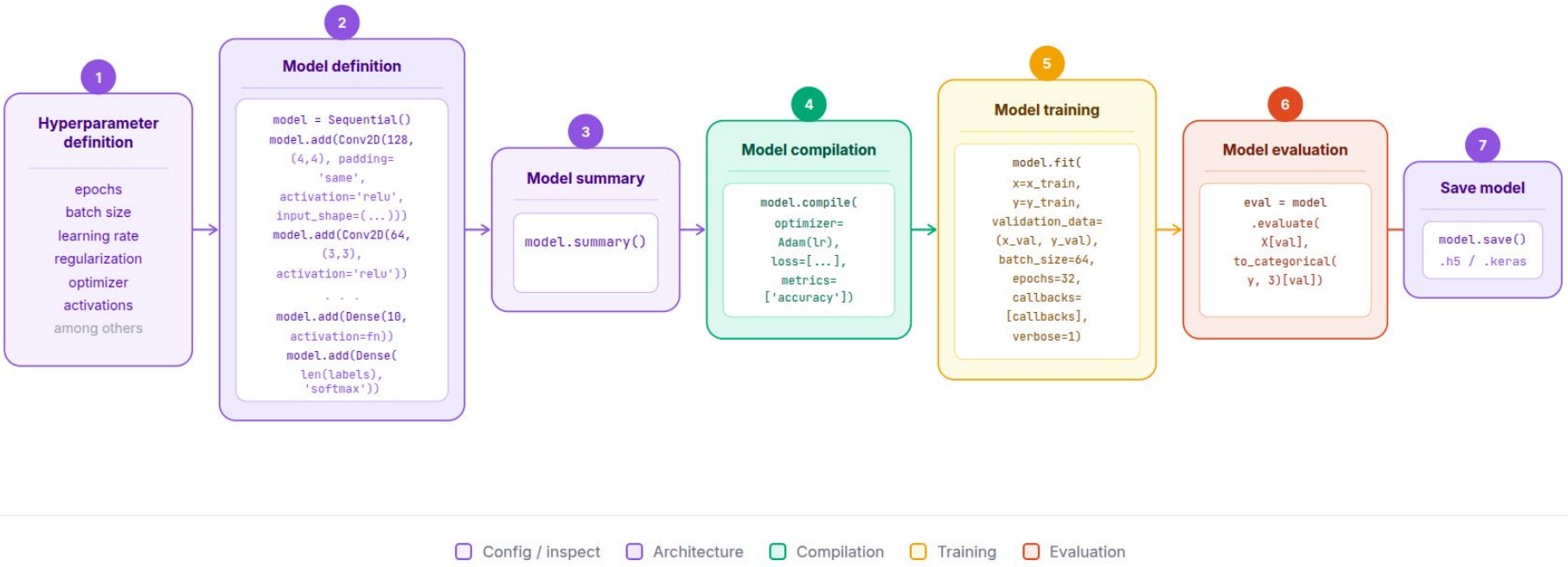
Training and Inference Metrics

Training vs. Inference

Phase	Training / Validation	Inference (Offline)	On-Device Inference
Goal	Learn and evaluate the model	Verify model behavior before deployment	Execute the model in real-world conditions
Model parameters	Updated iteratively	Fixed	Fixed
Ground truth labels	Available	Available (for evaluation)	Not available
Learning	Yes	No	No
Confusion matrix	Yes	Optional	No
Key metrics	Loss, Accuracy	Accuracy, Latency	Latency, Power, Memory
Main focus	Generalization	Performance verification	Efficiency and feasibility

Workflow with
Keras and TensorFlow

Workflow with Keras and TensorFlow



Workflow with Keras and TensorFlow

General overview

- The first two steps focus on **defining the hyperparameters and configuring the machine learning architecture**. Afterward, a model summary provides an overview of how the model was constructed.

- Once the model is created, parameters such as the optimizer, loss function, and metrics are configured using the **model.compile()** function.

- Finally, training is performed with the **model.fit()** function, where the dataset, batch size, number of epochs, and callbacks, among other settings, are specified.

Workflow with Keras and TensorFlow

```
model= Sequential([

    Flatten(input_shape=(w, h)),
    Dense(256, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(n_classes, activation='softmax')

])
```

Model definition

```
model.summary()
```

Model summary

Workflow with Keras and TensorFlow

```
learningRate = 0.001
optimizer = Adam(learningRate)
Epochs = 32
Batch = 16
```

Defining some of the hyperparameters

Workflow with Keras and TensorFlow

```
model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(0.001), metrics=['accuracy'])
```

Model compile

Loss: A metric that measures how far the model's predictions are from the actual values.

Optimizer: An algorithm that adjusts the weights of the neural network to minimize the loss function.

Learning Rate: A hyperparameter that controls the size of the adjustments the optimizer makes to the model's weights during each iteration.

Metrics: Additional values monitored during training to evaluate the model's performance. For example, accuracy (used in classification).

Workflow with Keras and TensorFlow

```
history = model.fit(x_train_norm, y_train, epochs= 32, batch_size = 50, validation_split=0.2)
```

Model fit

x_train_norm: normalized dataset obtained by applying a transformation to x_train.

y_train: labels (or expected values) corresponding to the training data.

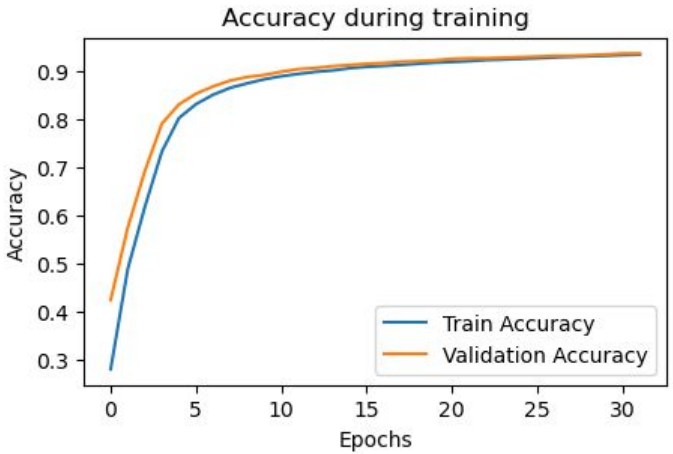
batch: number of samples processed before updating the model's weights.

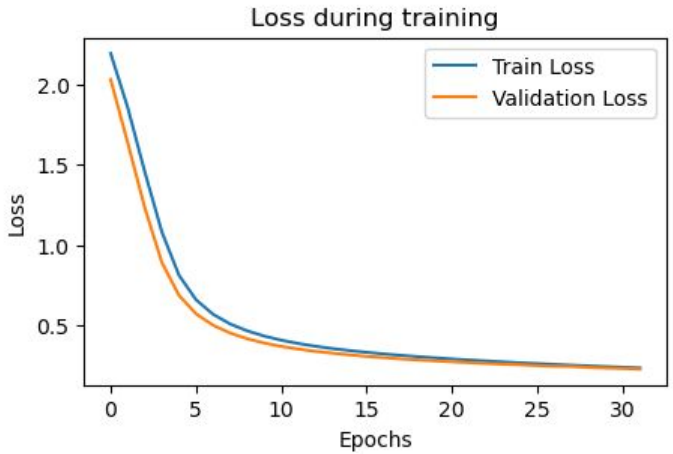
epochs: number of times the model will go through the entire training dataset.

validation_split: percentage of the training dataset (x_train, y_train) reserved for validation.

Workflow with Keras and TensorFlow

Plot the Accuracy and Loss from the **history** variable during training





Applied Machine Learning:
Turning Concepts into Code

Romina Soledad Molina, Ph.D.

School on Applied AI for Sustainable Development | smr 4210 | Trieste, Italy
March 2026

KaleidoForge
Lightning Bridges for Fast AI

The Abdus Salam International Centre for Theoretical Physics