Matteo Abrate
Master's Degree in Physics of Advanced Technology
ID: 858346
Year: 2022/2023
Supervisor: Mario Bertaina
Co-supervisors: Antonio Montanaro, Emanuele Valpreda

Implementation and analysis of the Stack-CNN algorithm on FPGA board

Politecnico di Torino

Index:

Introduction

1. Space Debris
2. Stack CNN algorithm

Development

1. Pytorch CNN models
2. Quantization
3. Dataset – Stacking Method
4. Testing and Tweaking of models
5. Implementation
6. Conclusions & Future Steps

Introduction

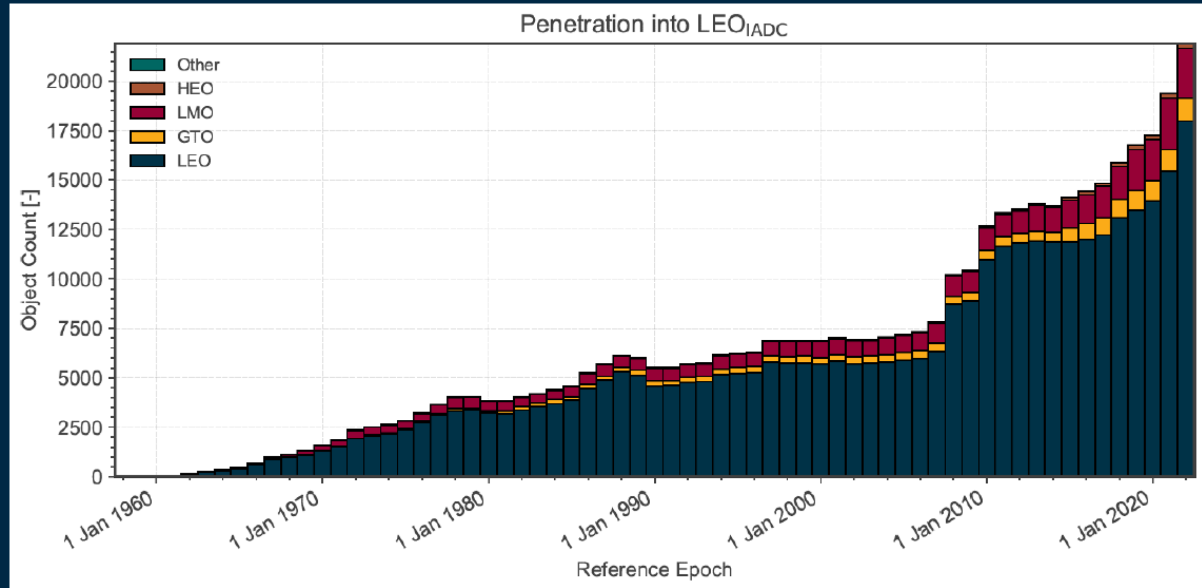
1. Space Debris
2. Stack CNN algorithm

1. Space Debris

Space debris are defunct human-made objects in space, principally in Earth orbit (less than 2,000 km), which no longer serve a useful function. These include **derelict spacecraft**, **nonfunctional spacecraft** and **abandoned launch vehicle stages**, mission-related debris, and particularly numerous in Earth orbit, fragmentation debris from the breakup of derelict rocket bodies and spacecraft.

Average impact speed in Low Earth Orbit (300-1000 km): 10 km/s.
Maximums: >14 km/s due to orbital eccentricity.
Estimated **number of debris** as of January 2019.

- 1 cm diameter: 128 million
- 1-10 cm diameter: 900,000



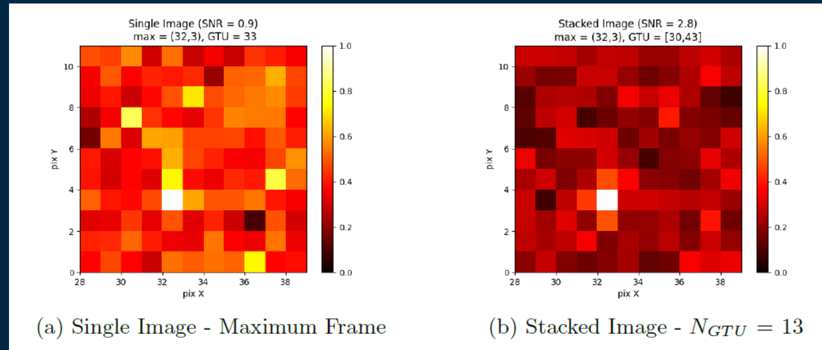
Evolution of (tracked) objects in Low Earth Orbit (2022 Annual Report by ESA)

2. Stack-CNN Algorithm

Stack-CNN is a trigger algorithm developed by the researcher Antonio Montanaro, which involves the use of two different techniques, a Stacking procedure and a Convolutional Neural Network.

Stacking Procedure

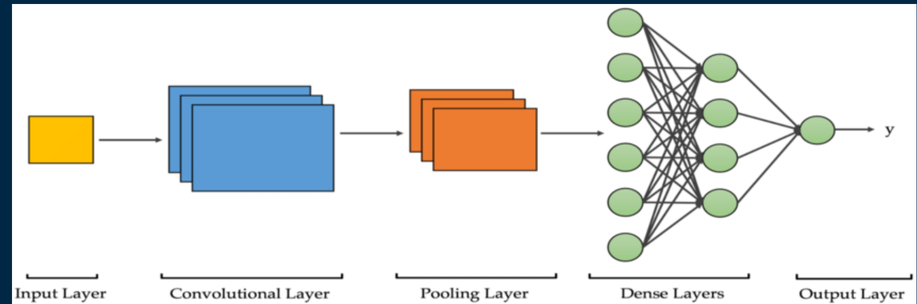
The Stacking Method is applied to the detection of moving objects moving linearly in the field of view of the detector. In our case objects are space debris.



SNR comparison between Stacked Image and Single Image (Simulated)

Convolutional Neural Network

Convolutional Neural Networks (CNNs or ConvNets) are a class of Neural Networks most commonly used in Computer Vision (image classification, video analysis). The advantage with respect to other algorithms is that the network is able to automatically extract image features without any prior knowledge.

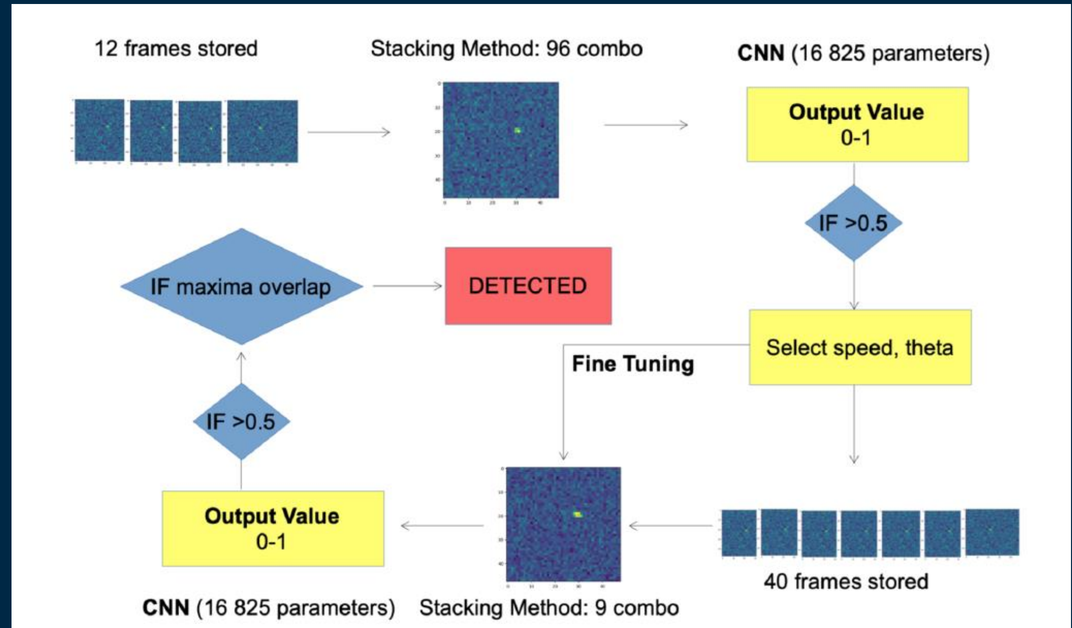


2.1 Stacking Method

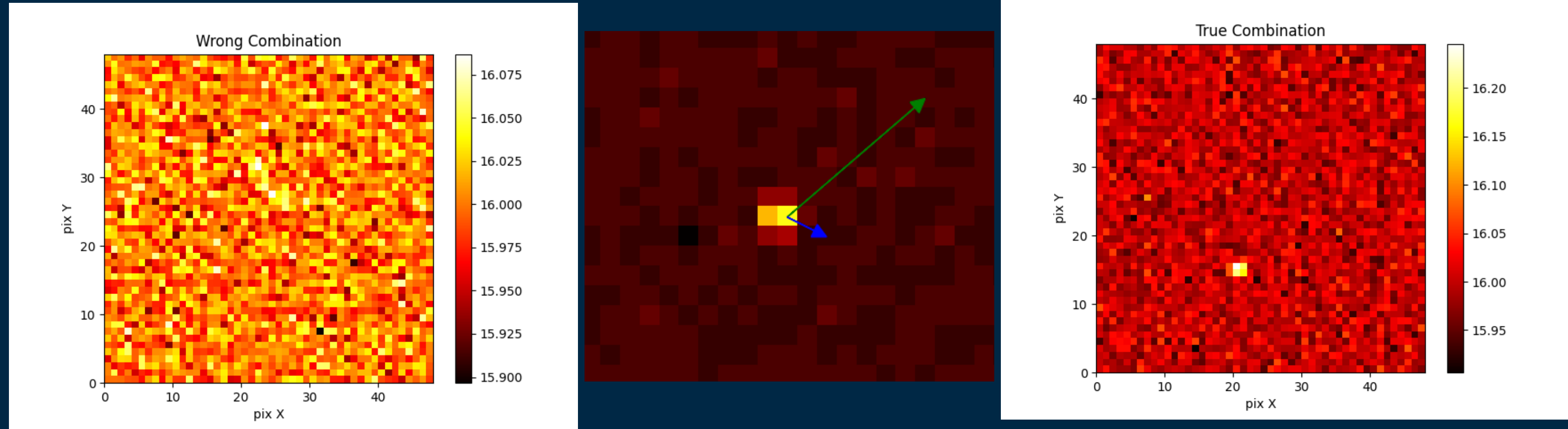
This powerful algorithm is capable of computing the speed and the angle of the detected object.
The stacking method applied to an object with fixed speed v and direction θ can be divided in the shifting and adding procedures.

- **Shifting**: Considering n frames of raw data the pixels are shifted in the opposite directions of the moving object's trajectory. The movement (dx, dy) depends on the time, speed and direction and it's used to roll the image back in the starting position (x_0, y_0).

- **Adding**: Sequentially the shifted images are summed in order to achieve a better SNR compared to a single image by a $\sqrt{n}$ factor.



2.2 Stacking Method



Images are shifted in θ direction through steps of 15°, from 0° to 360°, and with a step of 2 km/s for speed starting from 5 km/s until 11 km/s. This leads to:

- 4 speed combinations
- 24 combinations for direction
- Total number of combo: 96

For 80 SD, in total there are **7680 combinations**. There are just few right combinations (about 4%) among the entire set.

Development

1. Pytorch CNN Model
2. Quantization
3. Dataset – Stacking Method
4. Testing and Tweaking of Models
5. Implementation
6. Conclusions & Future Steps

1. Pytorch CNN model

PyTorch is a machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing.

Brevitas is a PyTorch library for neural network quantization, with support for both Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). Given a model made of PyTorch layers, the user has to replace them in the code with their Brevitas implementation. This library offers several quantized versions of the common PyTorch layers.

PyTorch Layer	Brevitas Layer
Convolutional Layers	
nn.Conv1d	QuantConv1d
nn.Conv2d	QuantConv2d
nn.ConvTranspose1d	QuantConvTranspose1d
nn.ConvTranspose2d	QuantConvTranspose2d
Pooling Layers	
nn.MaxPool1d	QuantMaxPool1d
nn.MaxPool2d	QuantMaxPool2d
nn.AvgPool2d	QuantAvgPool2d
nn.AdaptiveAvgPool2d	QuantAdaptiveAvgPool2d
Non-linear Activations	
nn.Hardtanh	QuantHardTanh
nn.ReLU	QuantRelu
nn.Sigmoid	QuantSigmoid
nn.Tanh	QuantTanh
Dropout Layers	
nn.Dropout	QuantDropout

```
class brevitas_model(Module):
    def __init__(self):
        super(brevitas_model, self).__init__()
        self.quant_inp = qnn.QuantIdentity(bit_width=8, return_quant_tensor=True)
        self.conv1 = qnn.QuantConv2d(in_channels=1, out_channels=10, kernel_size=(3,3),
                 stride=(1,1), padding=(1,1), weight_bit_width=8, bias_quant=Int8Bias)
        self.relu1 = qnn.QuantReLU(bit_width=8, return_quant_tensor=True)
        self.conv2 = qnn.QuantConv2d(in_channels=10, out_channels=5, kernel_size=(3,3),
                 stride=(1,1), padding=(1,1),weight_bit_width=8, bias_quant=Int8Bias)
        self.relu2 = qnn.QuantReLU(bit_width=8, return_quant_tensor=True)
        self.conv3 = qnn.QuantConv2d(in_channels=5, out_channels=1, kernel_size=(3,3),
                 stride=(1,1), padding=(1,1), weight_bit_width=8, bias_quant=Int8Bias)
        self.relu3 = qnn.QuantReLU(bit_width=8, return_quant_tensor=True)
        self.flatten = nn.Flatten()
        self.fc1 = qnn.QuantLinear(144, 72, bias=True, weight_bit_width=8,
                          bias_quant=Int8Bias)
        self.relu4 = qnn.QuantReLU(bit_width=8, return_quant_tensor=True)
        self.fc2 = qnn.QuantLinear(72, 72, bias=True, weight_bit_width=8,
                          bias_quant=Int8Bias)
        self.relu5 = qnn.QuantReLU(bit_width=8, return_quant_tensor=True)
        self.fc3 = qnn.QuantLinear(72, 1, bias=True, weight_bit_width=8,
                          bias_quant=Int8Bias)
```

2. Quantization

Scale (sc) and Threshold (th) are two parameters that are necessary for quantization.

- The Scale parameter (sc) is used to scale low-precision data back to floating-point values, it is stored with complete precision.
- The Threshold (th) is defined as the maximum absolute value in the input tensor X.
- int_th is the integer representation of the threshold value.
- IntW is the quantized weight value.

$$sc = \frac{th}{int_th}$$

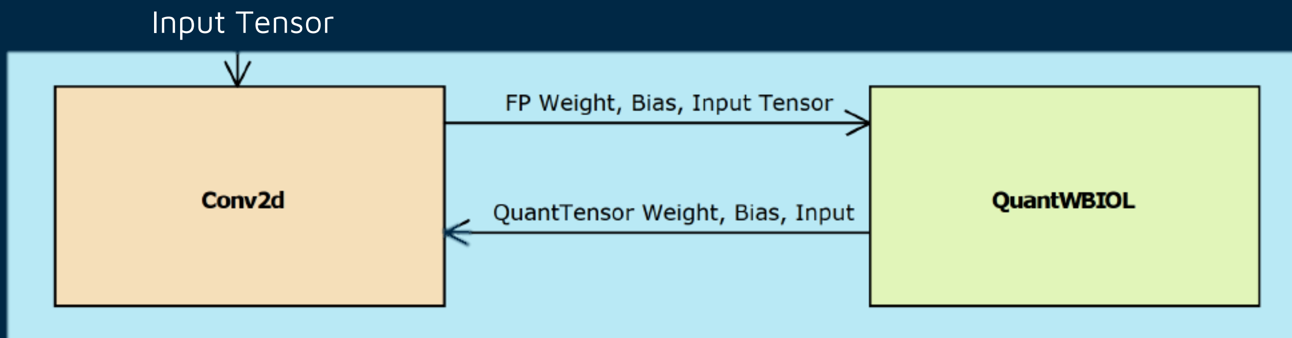
$$th = \max_{i,j=1,..,dim(X)} \{|x_{i,j}|\}$$

$$int_th = \begin{cases} 2^{N-1} - 1 & \text{if signed=True} \\ 2^{N} - 1 & \text{if signed=False} \end{cases}$$

$$IntW = \frac{FPV}{sc}$$

2.1 Quantization

The QuantConv2d layer is implemented inheriting two classes: Conv2d, the class that implements the convolution in PyTorch and that instantiates the weight and bias parameters, and QuantWBIOL which receives the weight and bias of Conv2d and compute its quantized version, so that the convolution is performed using quantized parameters.

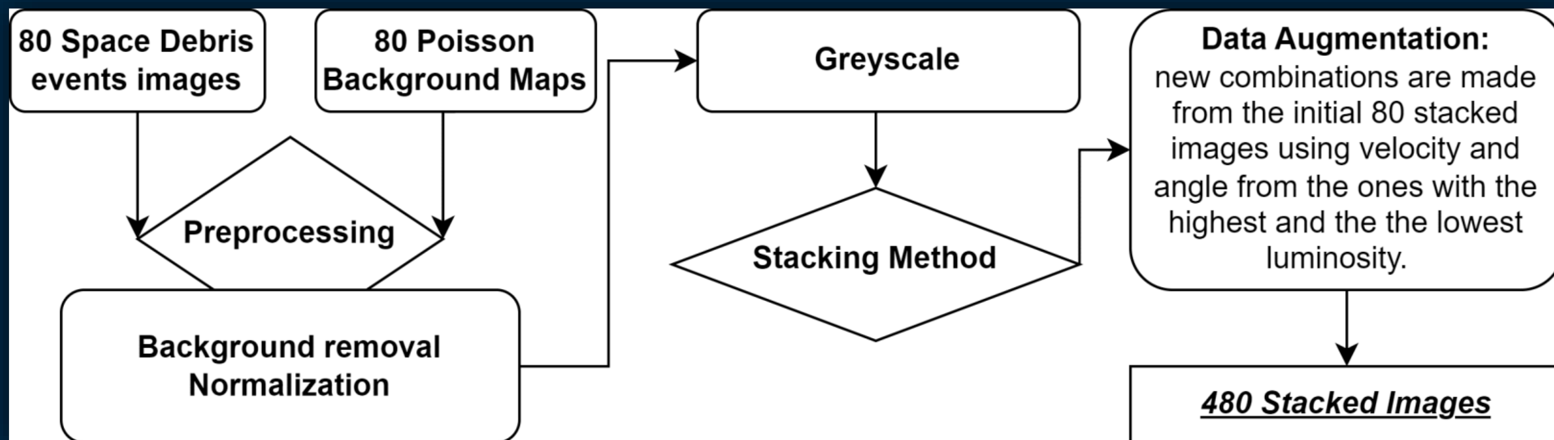


Scheme of the implementation of the *QuantConv2d* layer in Brevitas, made inheriting the standard PyTorch Conv2.

3. Dataset – Stacking Method

One of the most important tasks in working with neural networks is the dataset organization.
It has to be:

- **Statistical:** The set must include data from a statistical sample with the main features to be learnt.
- **Big**: More data are provided, more easily the network will learn to generalize.
- **Preprocessed**: All the inputs have to be preprocessed in the same way.



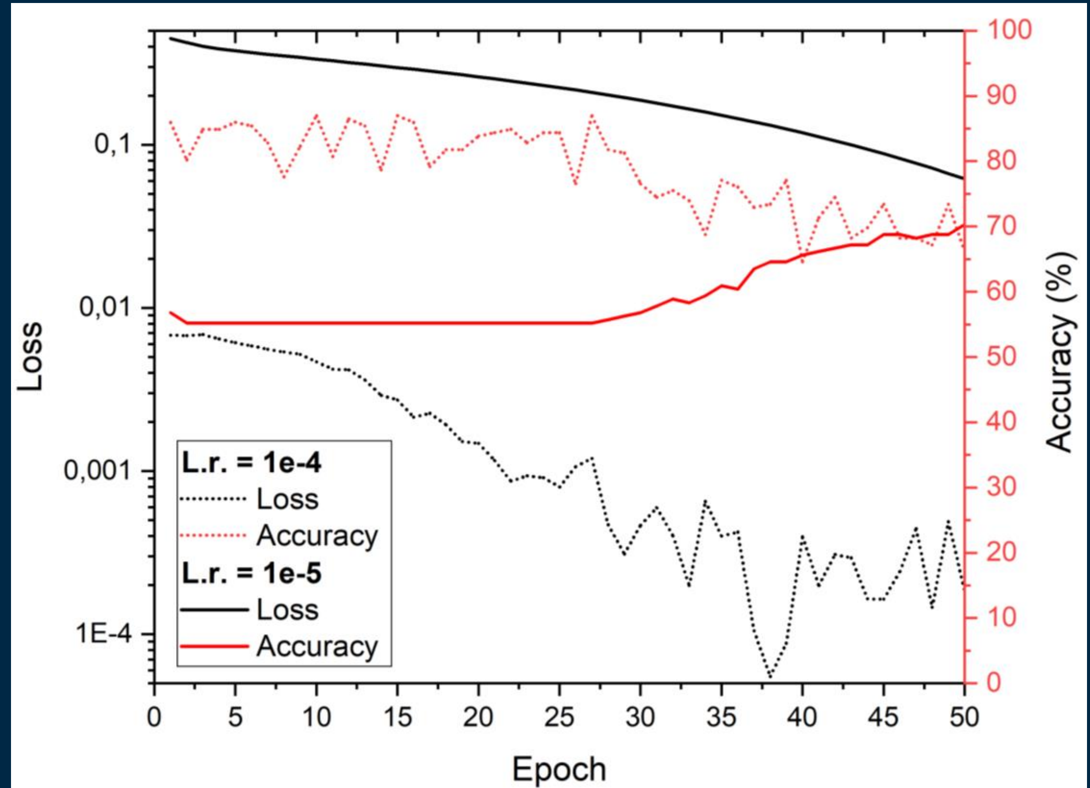
4. Testing and Tweaking of models

In the graph is shown the training for the Brevitas CNN, it was carried over the same 480 stacked images dataset.

Parameters of the model:

Optimizer: ADAM
Learning Rate: 1e-4 & 1e-5
Betas = (0.9, 0.999)
Epsilon = 1e-5
Weight decay = 1e-5
L.R. momentum = 0.9
Loss Function = MSE

Also L.R.=0.001 was tested but the model failed the training.



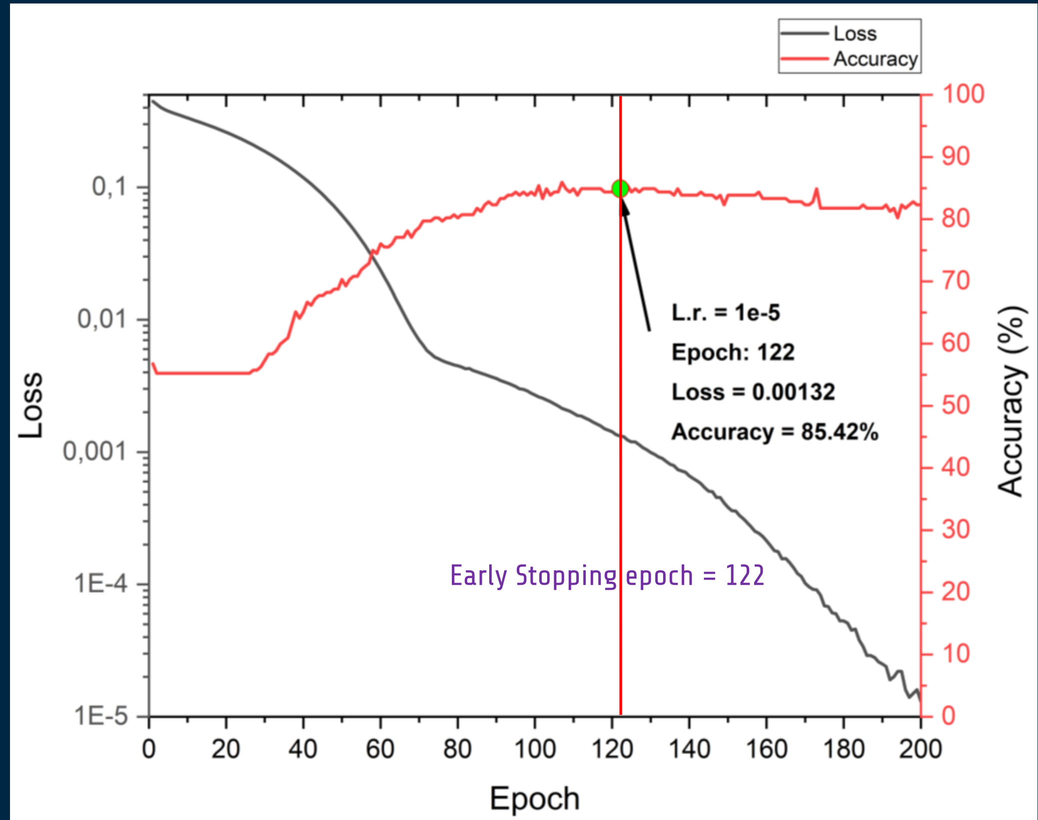
4.1 Best model

In the graph is shown the best training for the Brevitas CNN, it was carried over the same 480 stacked images dataset.

Parameters of the model:

Optimizer: ADAM
Learning Rate: 1e-5
Betas = (0.9, 0.999)
Epsilon = 1e-5
Weight decay = 1e-5
L.R. momentum = 0.9
Loss Function = MSE

This 200 epochs training was primarly used to determine at which epoch doing Early Stopping.
It is a form of regularization used to avoid overfitting when training a learner with an iterative method. The best results were obtained at the epoch number 122.

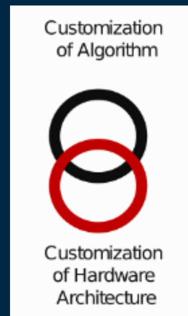


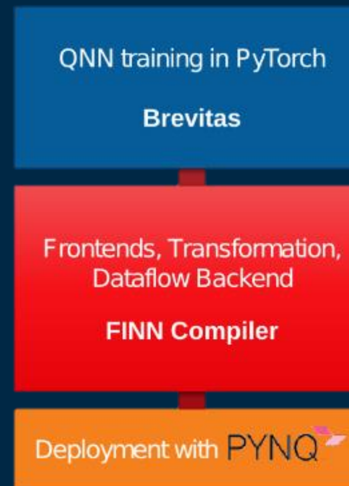
5. Implementation

The FINN project is an experimental framework from Xilinx Research Labs to explore deep neural network inference on FPGAs.

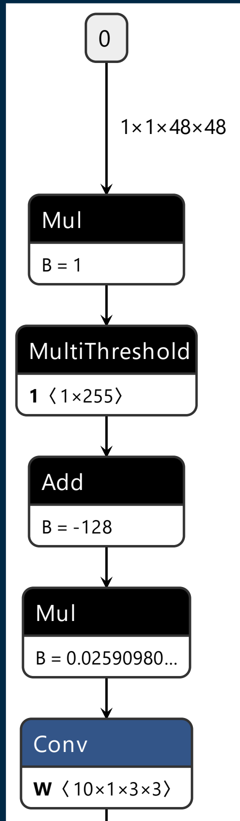
It specifically targets quantized neural networks (QNNs), with emphasis on generating dataflow-style architectures customized for each network. The key components are illustrated in the figure, including tools for training quantized neural networks (Brevitas), the FINN compiler, and the finn-hlslib Vivado HLS library of FPGA components for QNNs.

On a FPGA platform drawing less than 25 W total system power, FINN demonstrate up to 12.3 million image classifications per second with 0.31 μs latency on the MNIST dataset with 95.8% accuracy.





5.1 Implementation – ONNX



Brevitas Model

Brevitas Export

Brevitas FINN-ONNX Export

Trained Network in Pytorch/Brevitas

Network Preparation

Network of high-level ONNX layers

Streamlining Transformations

Streamlined network of high-level ONNX layers

Convert to HLS Layers

Mixture of HLS and non-HLS layers

Dataflow Partitioning

Parent Graph (non-HLS layers)

Network of HLS layers, maximum folding

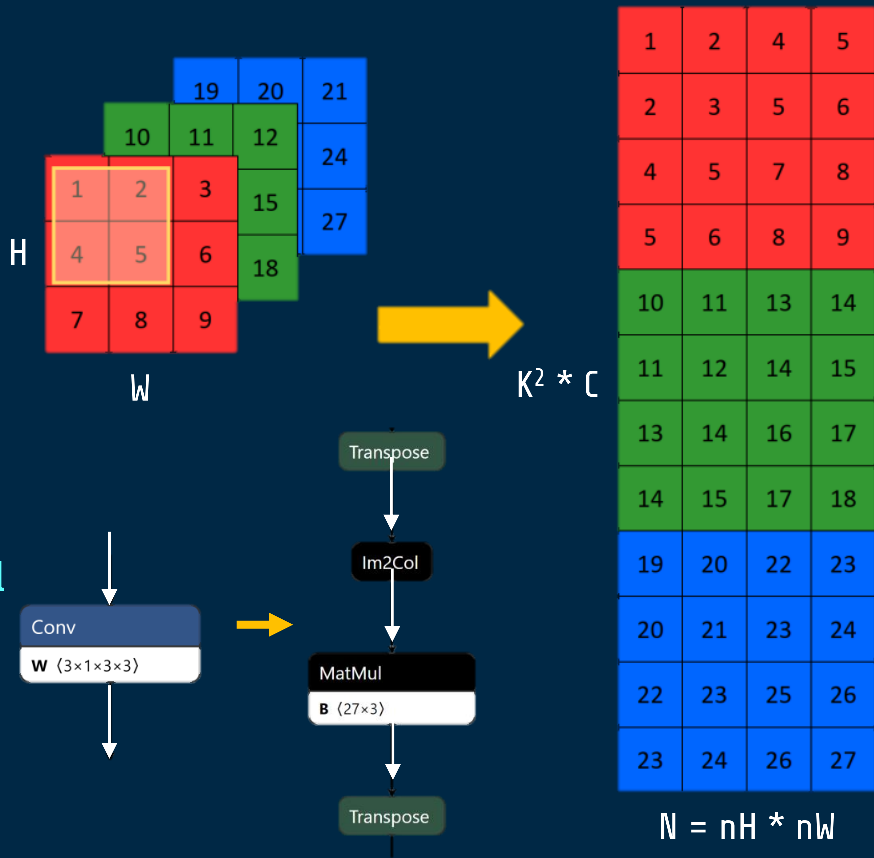
Adjust folding to maximize performance

Network of HLS layers, desired folding

5.2 LowerConvsToMatMul Transformation

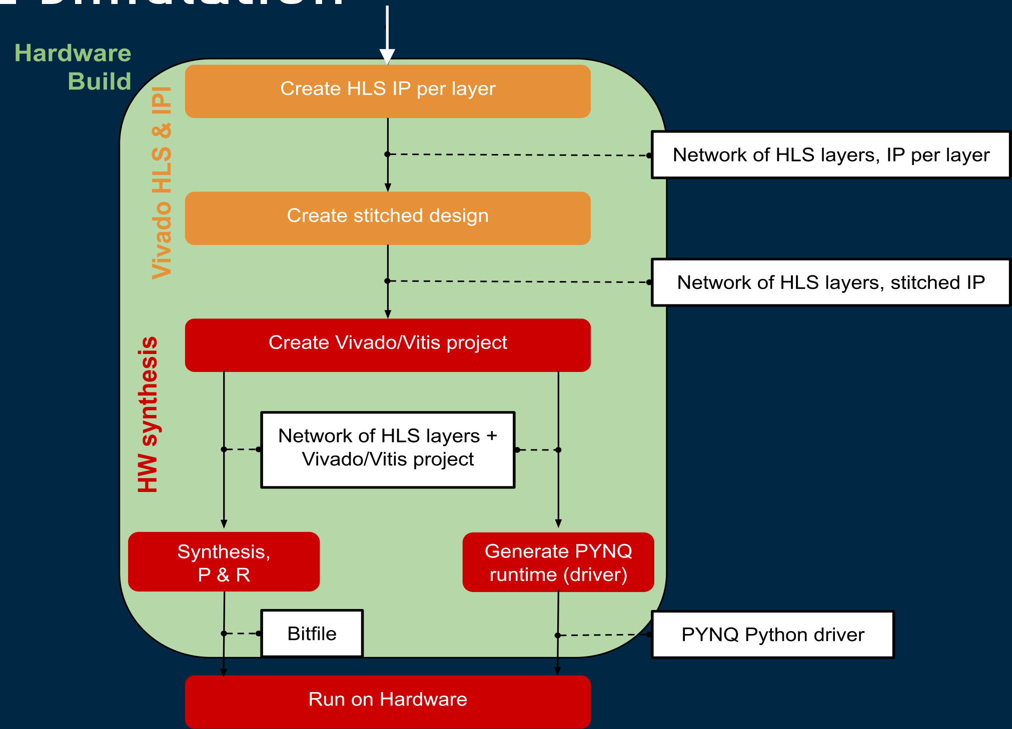
Our starting ONNX model presents Conv nodes, they have to be replaced using the LowerConvsToMatMul transformation. This transformation is one of the most relevant from the hardware point of view, since it is strictly related on how finn-hls library performs the convolution.

When executing LowerConvsToMatMul, FINN searches in the model for Conv nodes and replaces them with a pair of Im2Col→MatMul nodes in case of depthwise convolution. The input tensor is reshaped in a matrix of dimension $K^2 * C \times N$.



5.3 Implementation – RTL Simulation

RTL Synthesis Main Results		
	Utilized	Total
LUT	28189	504K
LUTRAM	1655	–
FF	15661	461K
DSP	37	1,728
BRAM	17	–
Estimated Throughput [images/s]	9585.4	–
Throughput [images/s]	4605.9	–
Clock Frequency [mhz]	185.19	–



0.13 ms to process 1 stacked image through the CNN *(Simulation)*.

5.4 Implementation Results

Implementation Main Results	
Throughput [images/s]	1822,02
Clock Frequency [Mhz]	100
copy_input_data_to_device[ms]	0.85
copy_output_data_from_device[ms]	0.25

0.5 ms to process 1 stacked image through the CNN *(Simulation)*.

48 ms to process all the 96 combinations of stacked image through the CNN *(Real Time)*.

6.1 Conclusions

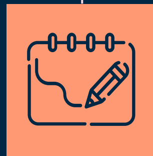
Implementation on FPGA of the Stack-CNN algorithm for Space Debris tracking.

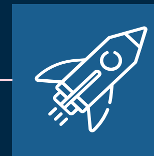
6.2 Future Steps

Test inference accuracy directly on board.

Increasing the performance of the CNN.

Developing the quantized online Stacking Module inside a custom Brevitas module.





Testing the performances of the complete system for online detection.

THANKS FOR YOUR ATTENTION

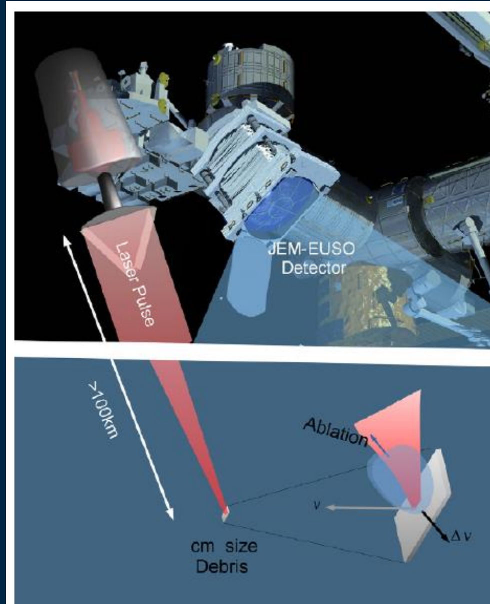
JEM-EUSO project for SD removal

JEM-EUSO is a space-based detector that records fluorescence light in UV bandù (290 - 430 nm).

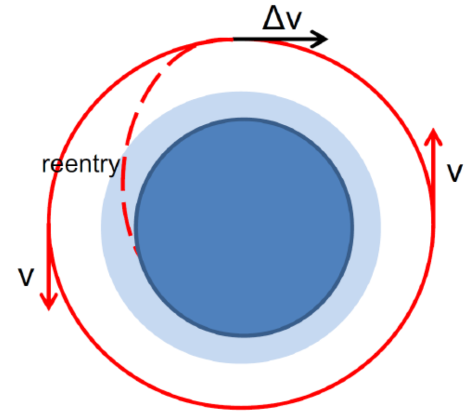
JEM-EUSO, looking down at Earth's atmosphere, is also able to detect SD through albedo phenomenon.

Light reaches only SD and the detector is covered by Earth's shadow.

The proposal is then building in tandem a space-borne pulsed-laser system.



(a) EUSO telescope for acquisition and CAN laser system for tracking and impulse delivery of cm-sized SD



(b) Laser-ablation impulses reducing the speed of debris and causing its atmospheric reentry

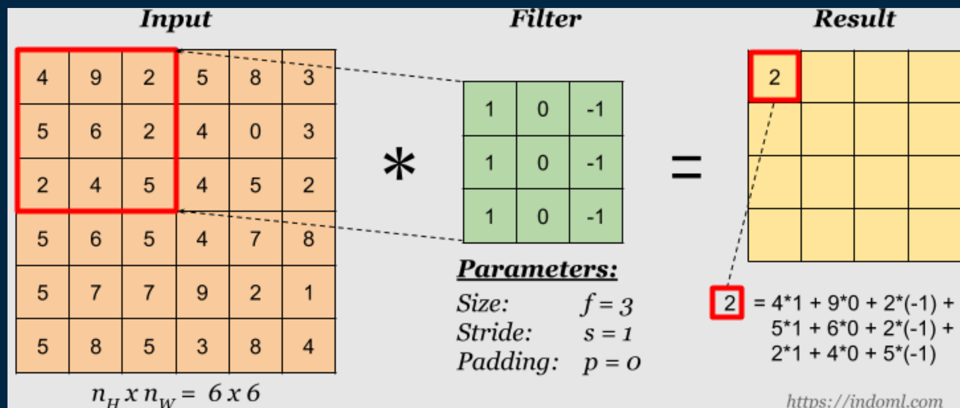
T.Ebisuzaki et al. Demonstration designs for the remediation of space debris from the International Space Station, 2015.

Convolutional Neural Networks

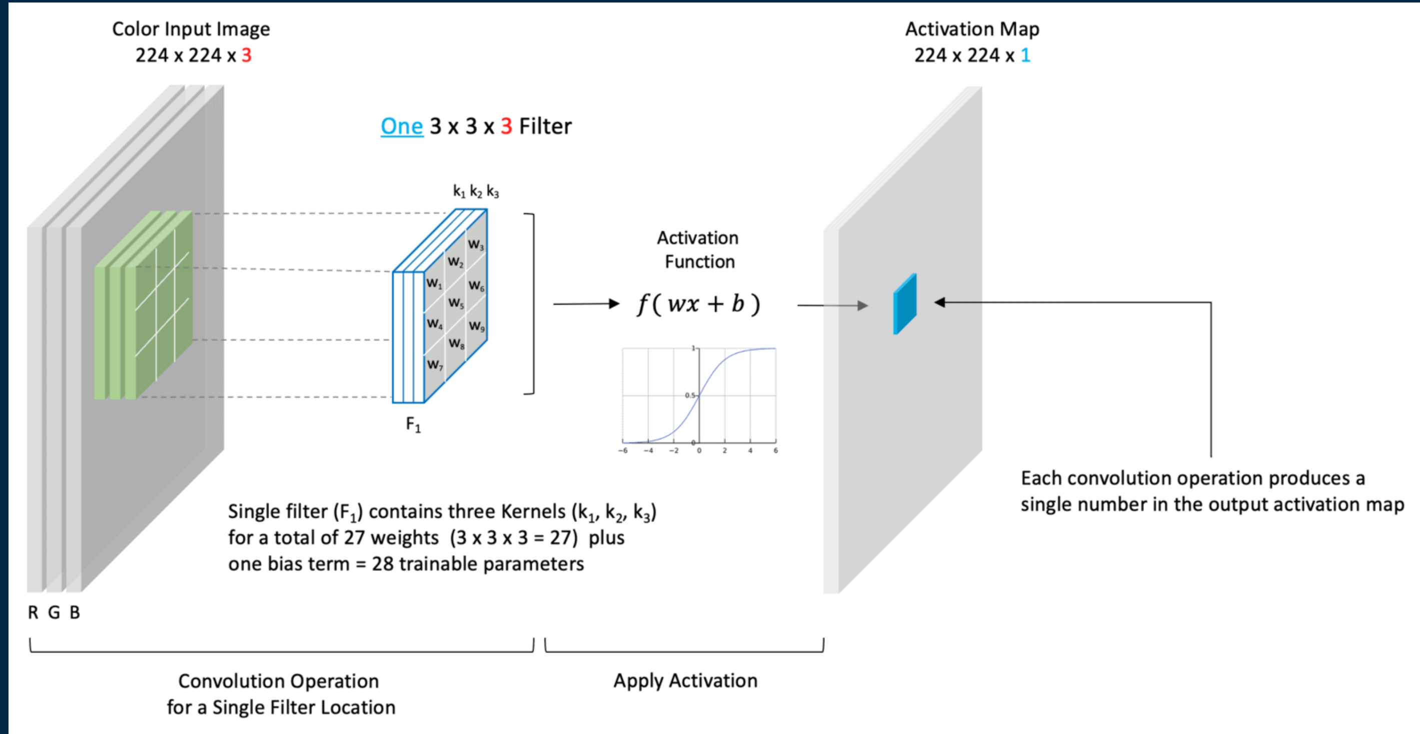
1. **Learning rate**: is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

2. Loss Function: $MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y}_i)^2$ in classification, it is the penalty for an incorrect classification of an example.

3. Convolution:

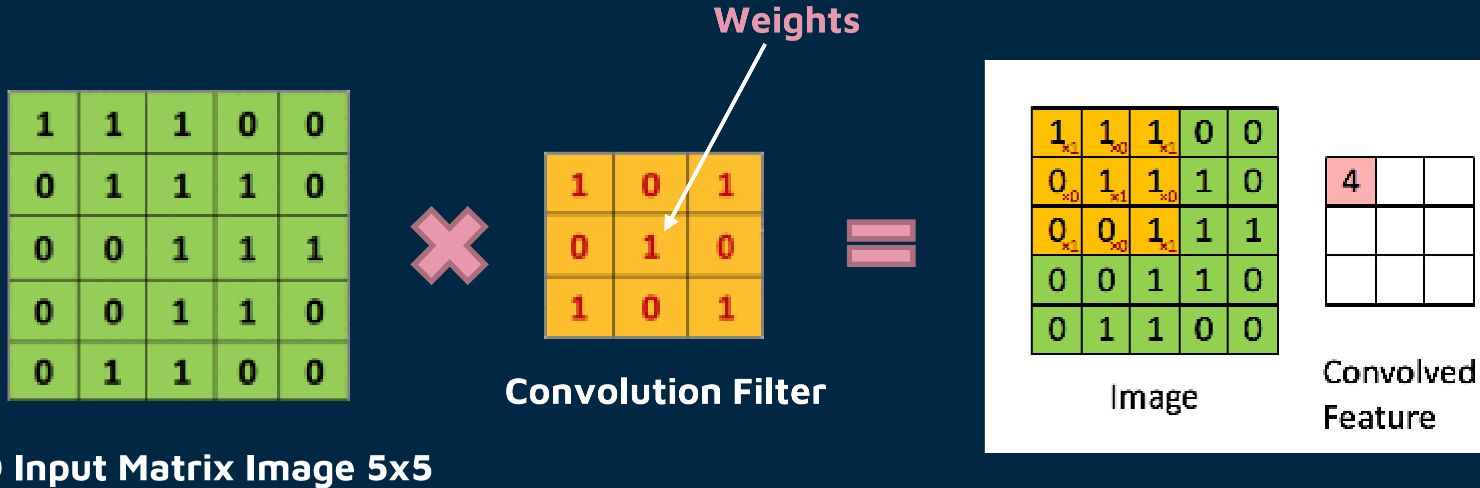


Activation Tensors & Weight Tensors



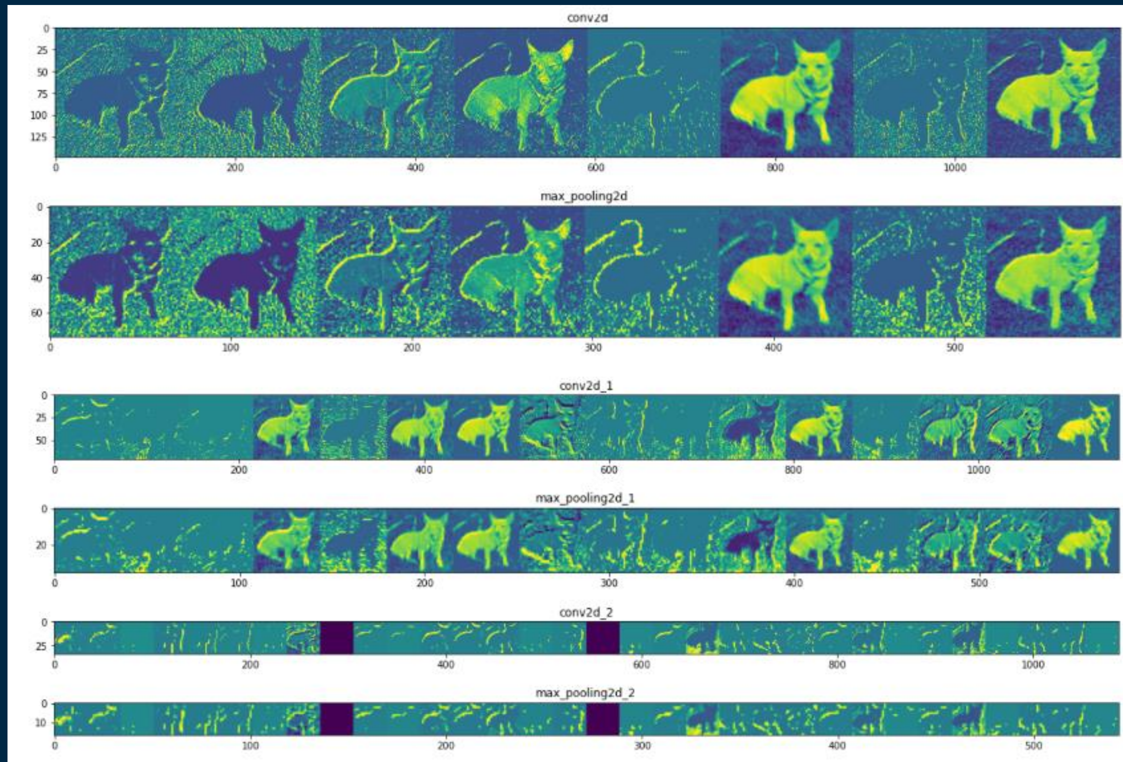
[https://learnopencv.com/understanding-convolutional-neural-networks-cnn/]

Convolution Operation



[https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26]

Feature Masks CNN



[https://www.analyticsvidhya.com/blog/2020/11/tutorial-how-to-visualize-feature-maps-directly-from-cnn-layers/]

CubeSat & Requirements

If the detection could be achieved within a time scale of tens of milliseconds an online trigger system could be implemented in a CubeSat.

3U Cubesat solar panel	
Mass	127 g
Maximum power	Up to 8.4 W in LEO per side
Power efficency	29+% (at EOL)
Current	< 504 mA
Voltage	16.8 V (for 7 cells)

https://satsearch.co/products/endurosat-3u-cubesat-solar-panel

1U CubeSat CP1 (left) 10x10x11cm
3U CubeSat CP10 (right) 10x10x34cm (NASA)