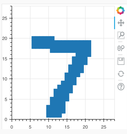
Welcome to the EGGNET 1.0.

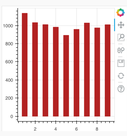
This is a student project from TU Vienna in the course 384.157 SoC Design Laboratoy. Goal was to develop a full system focusing on a specific topic.

1. MNIST-Data Set

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

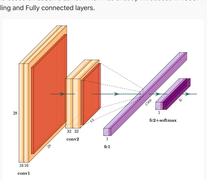
On the left we can see an image looks like when plotted and on the right we can see Histogram of the Test set!





2. The designed Network

The networks design is based on LeCun's LeNet which was already introduced in 1999. We used a combination of Convolutional, Pooling and Fully connected layers.



The famous EggNet designed by Yellow et AL.

We use a four layer structure as seen in the picture. The first layers are *Convolutional layers* where the output is defined by:

$$z(i,j) = (f * g)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m,n)g(m-i,n-j)$$

Here f and g are some arbitrary two dimensional tensors. For our neural network one of the tensors is a 3-by-3 kernel, where the values are learned. The first layer convolutional features 16 of this operations and the second convolutional layer 24. Convolutional layers are a common, state-of-the-art way to achieve good results in image recognition tasks. To reduce the computational workload, the image is downsampled after each convolutional step. Although different methods exist, we choose the *Max. Pooling* approach for 2x2 image patches.

The second important layer type is the *Fully Connected Layer*. Here the output z for an two dimensional input x is defined by

$$z = xW + b$$

where W and b are learned parameters. Those layers have high computation and memory demands and should therefore used with care. In our case we used them to interpret the features that are extracted from the convolutional layers and for final classification.

Another important factor of Deep Neural Networks are non-linear activation functions to give the network the ability to learn arbitrary functions. We used the *Rectified Linear Unit (ReLU)* which leads to good result and is also computational very cheap.

$$f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ 0 & \text{else} \end{cases}$$

This function was used for every layer output, which also enabled us to save an extra bit, because the values could be stored as signed integers.

2.1. Quantization of the network

For a seal floating point number we can represent it via

$$v = Q \cdot 2^{-m}$$

where Q and m are integers. We used a per layer quantization and achieved a near floating point performance with just minor performance drops. We achieved this by analyzing the distribution of the weights and biases as well as the input and output data of the network for each layers. This showed us the expected value range of this parameters and enabled to use to select good values for m and Q.

Quantization Details, 4 Bit

Type	$log_2(Q)$ (Total Bits)	m (Fraction)
Layer 1: Weights	4	2
Layer 1: Output	4	2
Layer 2: Weights	4	5
Layer 2: Output	4	0
Layer 3: Weights	4	5
Layer 3: Output	4	0
Layer 4: Weights	4	5
Layer 4: Output	4	2

3. Upload Image and Resize

Choose File no file selected Click to upload

3.1. Downscaled / Numpy Array / Network Solution

4. Eggnet Benchmark!

Here is the chance to run a simulation of the FPGA. A single batch consists of 100 images.

Run Benchmark

Dataset Choose... Network Choose... Batch Size Choose...

Run Benchmark

Latest Results

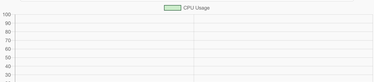
Index	Data Set	# Batches	Network	Accuracy	Time
0	Train Set	1	CPU	0.999	0.01

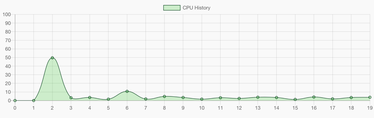
5. System Information

This sections shows current stats of the system.

System Information

```
System: Linux
Node Name: linaro-server
Release: 4.9.0-xilinx-v2017.4
Version: #1 SMP PREEMPT Wed Feb 5 13:41:53 CET 2020
Machine: armv7l
Processor: armv7l


========================================= Boot Time ==========================================
Boot Time: 2020/3/12 12:29:59

=========================================    CPU    ==========================================
Physical cores: 2
Total cores: 2
Total CPU Usage: 4.8


======================================== Memory Information ========================================
Total: 498.50MB
Available: 363.83MB
Used: 77.36MB
Percentage: 27.0

========================================== SWAP ==========================================
Total: 0.00B
Free: 0.00B
Used: 0.00B
Percentage: 0.0
```





6. Documentation

The full documentation in tex format as well as the source code of this project is available on Github. A precompiled PDF document can be downloaded here.

Documentation

7. Credits

This project was created by Baischer Lukas, Leitner Anton, Kulnik Benjamin, Marschner Stefan and Cerv Miha for a SoC Lab Project at TU Wien.

Third Parties

This project uses third party tools and libraries. Namely:

- MNIST Dataset
- Xilinx Vivado and SDK 2017.4
- Ubuntu 14.04
- Linaro Toolchains for Ubuntu 14.04
- Python v3.6
- SWIG v4.1
- GHDL v0.33
- pdftex
- Any many python software packages. See the corresponding requirement files