On-Site Gamma-Hadron Separation with Deep Learning on FPGAs

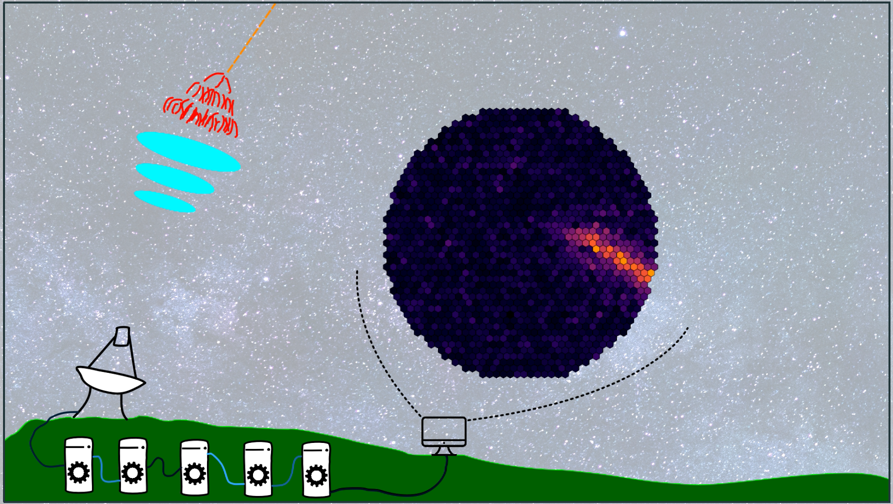
ECMLPKDD2020

Sebastian Buschjäger, Lukas Pfahler, Jens Buss, Katharina Morik and Wolfgang Rhode

July 19, 2020

TU Dortmund University - Artifical Intelligence Group - Collaborative Research Center 876

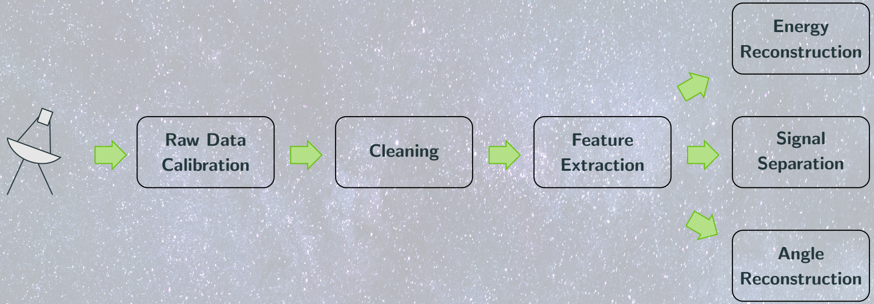
Gamma-Ray Astronomy



FACT First G-APD Cherenkov Telescope continuously monitors the sky for gamma rays

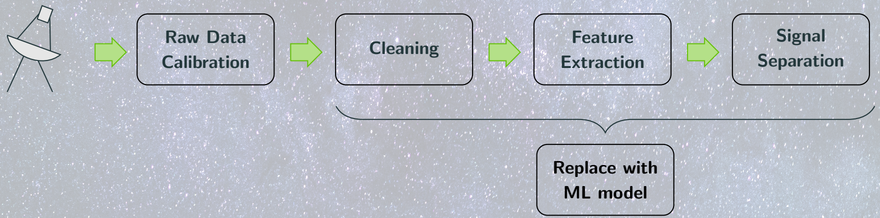
- It produces roughly 180 MB/s of data
- Only 1 in 10.000 measurements is interesting
- Bandwidth / computation power / physical space is limited

Telescope processing pipeline



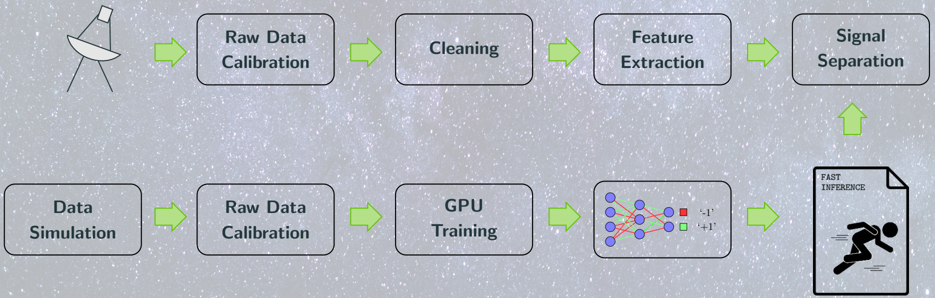
Telescope processing pipeline

Telescope processing pipeline

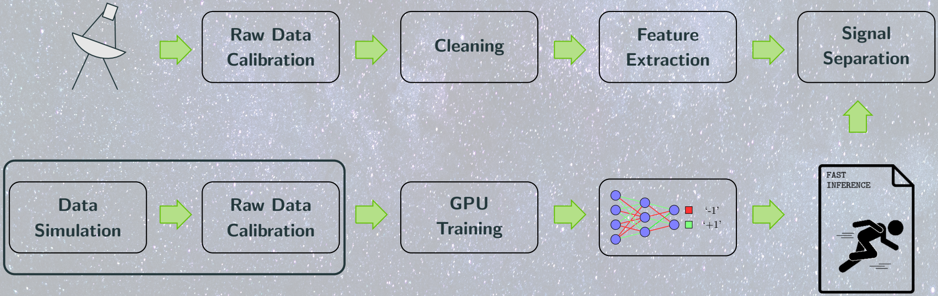


Telescope processing pipeline

Telescope processing pipeline



Telescope processing pipeline



Simulation Data

- CORSIKA simulation
- with and without quality cuts
- downsampled to 200K/100K train/test data, equal distribution

Facts about the FACT Pipeline

Simulation Data

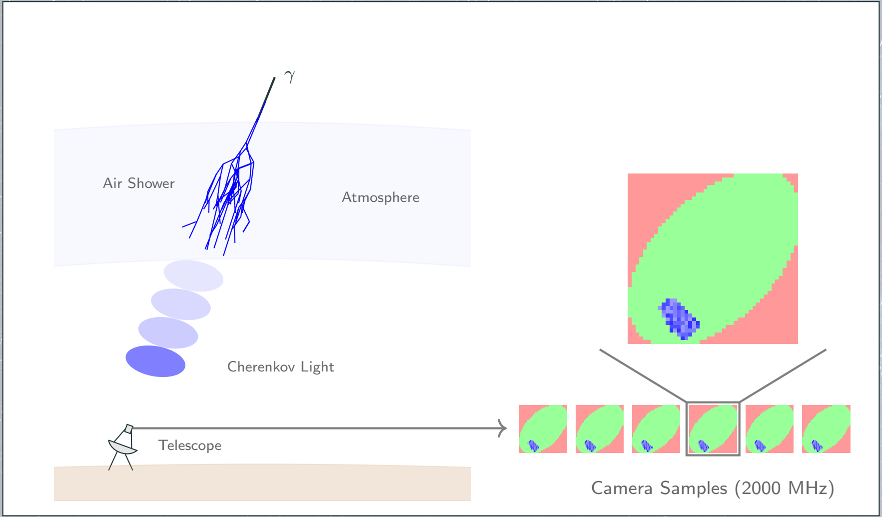
- CORSIKA simulation
- with and without quality cuts
- downsampled to 200K/100K train/test data, equal distribution

Pre-processing

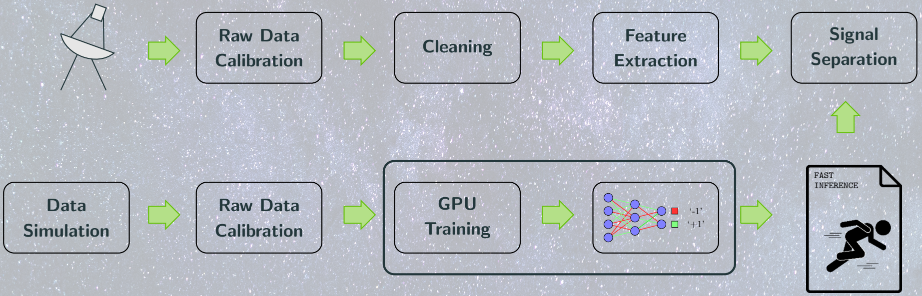
- Subtract reference voltage curves from measurement to count no of photons
- Focus on 50 ns ROI from 300ns time series

Result 45×45 images where each pixel contains a photon count

The FACT data



Telescope processing pipeline



Deep Learning on FACT

General approach Start with something simple and gradually increase complexity

Input data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_i \in \mathbb{N}^{45\times45}, y_i \in \{0,1\}, N_{train} = 200K, N_{test} = 100K$

Take-Aways In total 1178 experiments performed

- Smaller architectures work better
- Early stopping / Learning-rate scheduler helps
- ResNet does not seem to improve performance

General approach Start with something simple and gradually increase complexity

Input data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_i \in \mathbb{N}^{45 \times 45}$, $y_i \in \{0, 1\}$, $N_{train} = 200K$, $N_{test} = 100K$

Take-Aways In total 1178 experiments performed

- Smaller architectures work better
- Early stopping / Learning-rate scheduler helps
- ResNet does not seem to improve performance

Final model Simple VGG-like architecture

Binarized Neural Networks on FACT

Binarized Neural Networks Use weights from $\{-1, 1\}$ instead of $\mathbb{R}$

For training Use deterministic binarization + full precision SGD

Binarized Neural Networks Use weights from $\{-1, 1\}$ instead of $\mathbb{R}$

For training Use deterministic binarization + full precision SGD

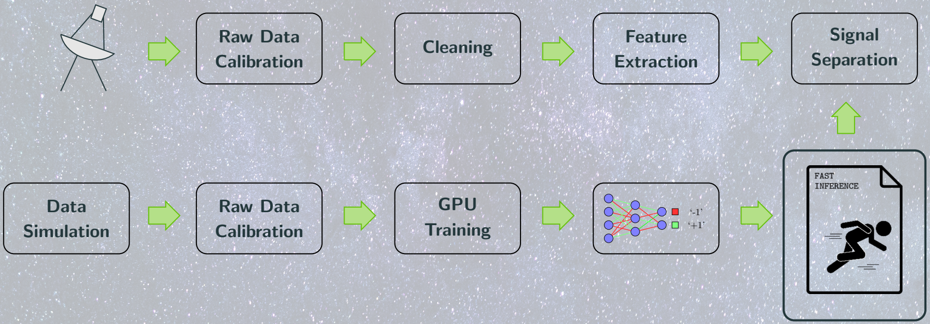
Why BNNs? Only 2 clocks needed to process 32/64/128/256 bits (= weights)

Approach Map weights/inputs to bitstring '$-1 \rightarrow 0$' and '$+1 \rightarrow 1$'

- $f_i w_i$ is '+1' if same sign, else '0'. This is an XOR operation
- $\sum_i f_i w_i$ counts occurrences of same sign. This is the popcount operation.

$$\sum_i f_i w_i = \mathrm{POPCNT}(f \text{ XOR } W)$$

Telescope processing pipeline



FastInference: Workflow and Capabilities

JSON
ONNX

Read File
- Linear Model
- Ensemble
- Decision Tree
- Neural Network

Optimize Model
- Weight clustering
- Tree Framing

Choose Target Architecture
- X86
- ARM
- FPGA

Combine Templates
- templates/NN/binary/gmm.j2
- templates/NN/binary/conv.j2
- templates/NN/float/gmm.j2
- templates/NN/float/conv.j2
- ...

FAST INFERENCE

1. How do RandomForest, CNNs and Binary CNNs perform on simulation data?
   $\rightarrow$ CNNs should outperform Binary CNNs should outperform RF on simulation

2. How do RandomForest, CNNs and Binary CNNs perform on real-world data?
   $\rightarrow$ (Binary) CNNs hopefully outperform RF on real-world data

3. Is the implementation of FastInference real-time capable?
   $\rightarrow$ Binary CNNs should be faster than regular CNNs, regardless the target architecture

Deep Learning vs Random Forest on Simulation Data

Model	Data	Accuracy, no QC		Accuracy, QC	
		epochs:100	epochs:10	epochs:100	epochs:10
RF	DL2	0.70959		0.78483	
RF	PhC	0.74711		0.78839	
CNN(small)	PhC	0.90825	0.88867	0.93441	0.93846
BNN(small)	PhC	0.90861	0.88644	0.90440	0.88866
CNN(large)	PhC	**0.91094**	0.90251	0.93735	**0.94228**
BNN(large)	PhC	0.90011	0.89925	0.93112	0.91369

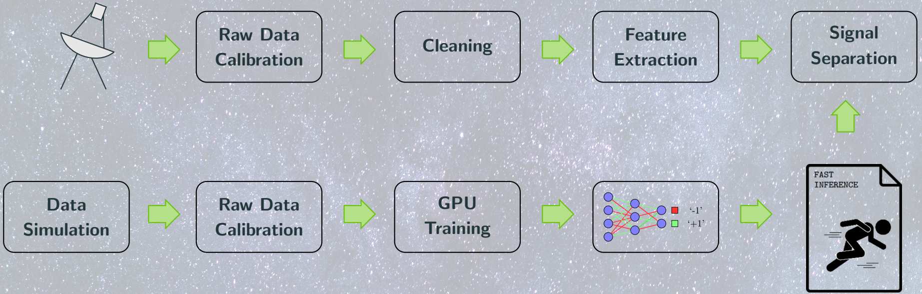
Deep Learning vs Random Forest on Crab Nebula Data

Model	Data	$S_{li\&ma}$, no QC		$S_{li\&ma}$, QC	
		epoch: 100	epoch: best loss	epoch: 100	epochs:best loss
RF	DL2	22.86σ		23.82σ	
RF	PhC	2.09σ		3.35σ	
CNN(small)	PhC	24.09σ	25.83σ	24.12σ	24.89σ
BNN(small)	PhC	19.55σ	25.87σ	22.96σ	21.67σ
CNN(large)	PhC	23.68σ	24.64σ	24.20σ	23.17σ
BNN(large)	PhC	22.70σ	22.92σ	22.35σ	22.26σ

X86 CPU vs FPGAs for Deep Learning

System	Type	Runtime [ms/event]	
		float	binary
ONNX Runtime	large	21.083 ± 0.078	26.642 ± 0.100
	small	0.957 ± 0.020	1.861 ± 0.037
Generated Code	large	78.583 ± 1.704	11.250 ± 0.077
	small	2.757 ± 0.026	1.574 ± 0.014
FPGA	large	-	561.588 ± 0.000
	small	-	4.221 ± 0.000
FPGA pipelined	large	-	72.657 ± 0.000
	small	-	0.662 ± 0.000

Recap: (Binarized) CNNs work well on simulated and real-world FACT data



- ✓ Excellent performance on training data
- ✓ Improved performance on real-world data
- ✓ Real-time capabilities on small devices
- ✓ FPGA implementation available if necessary