Application of hls4ml for Astronomical Radio Signals

Yunpeng Men, Andrei Kazantsev,
Ramesh Karuppusamy, Michael Kramer

Max Planck Institute for Radio Astronomy

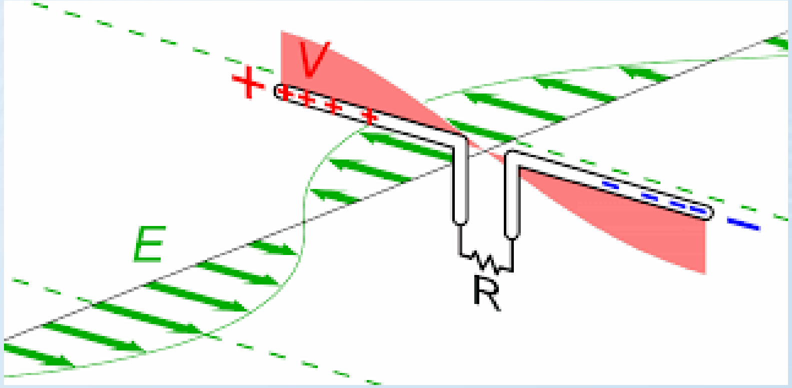
Outline

- Radio astronomical signals
- hls4ml framework
- Test on mnist dataset
- Future work

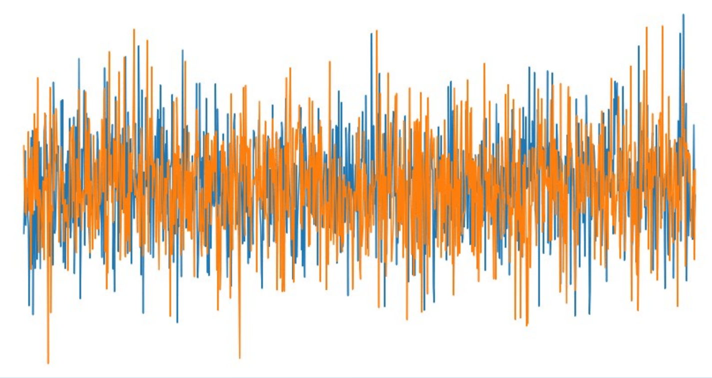
Radio Astronomical Signals

Effersberg 100m

MeerKAT



Our sensor looks like!



2 polarizations for each antenna

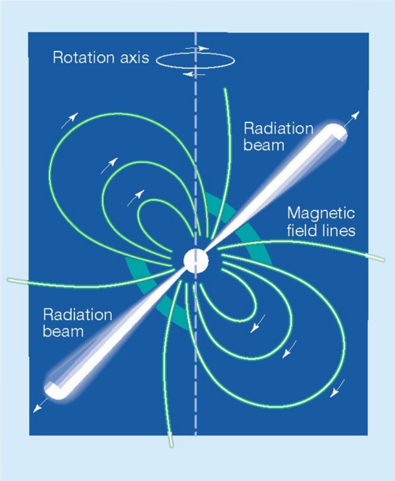
Data Rate

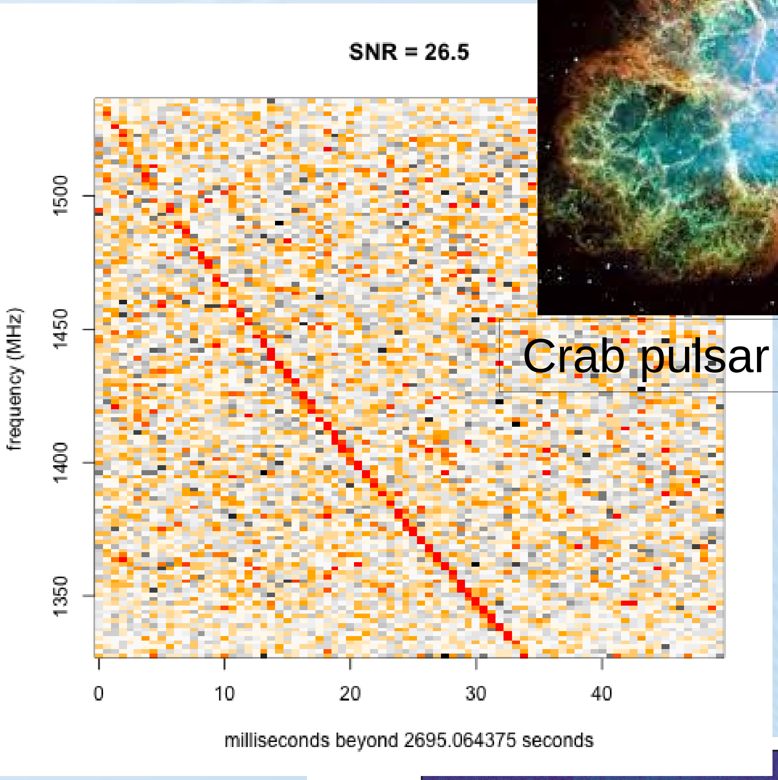
~28GB/s (2GHz*7beam*2B)

~256GB/s (2GHz*64*2 B)

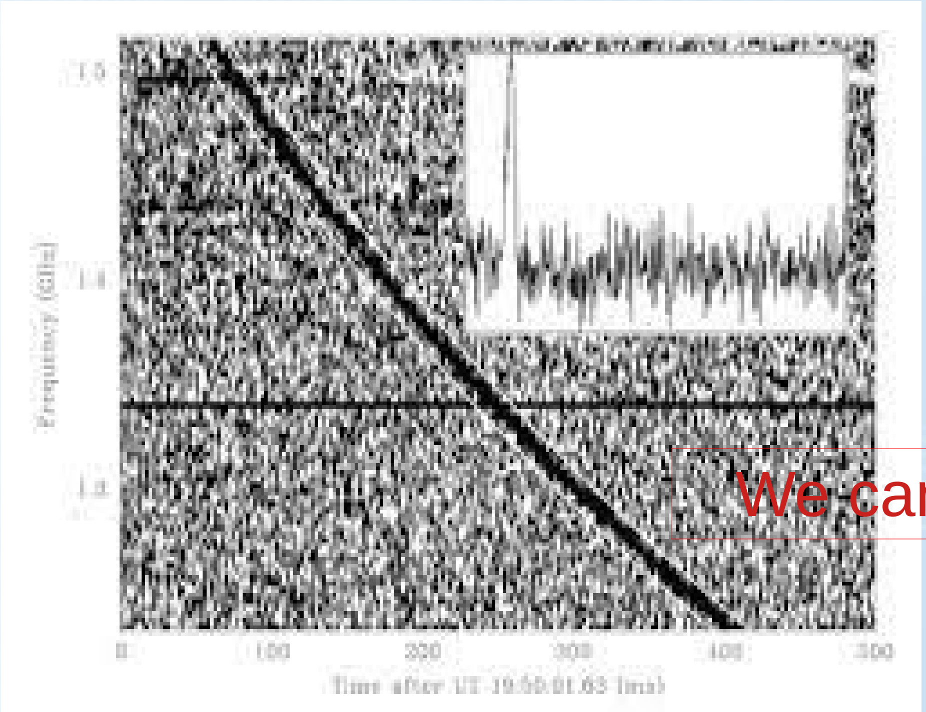
We need high throughput system!

Radio Transient Signals



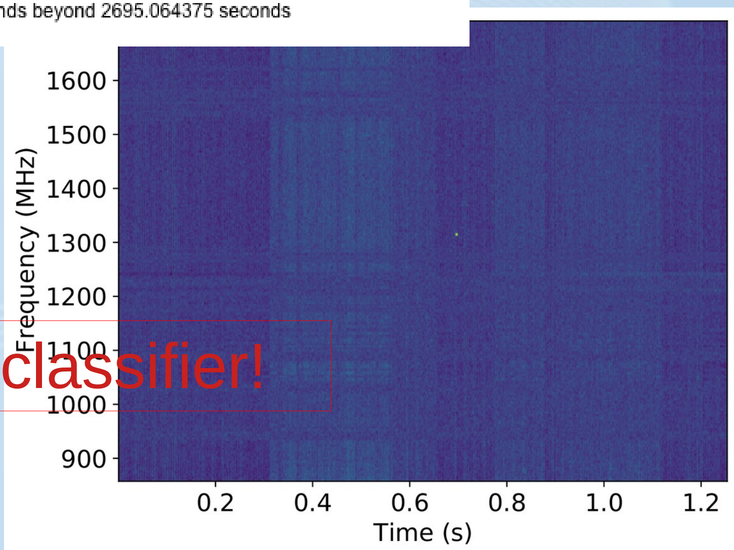


Crab pulsar giant pulse



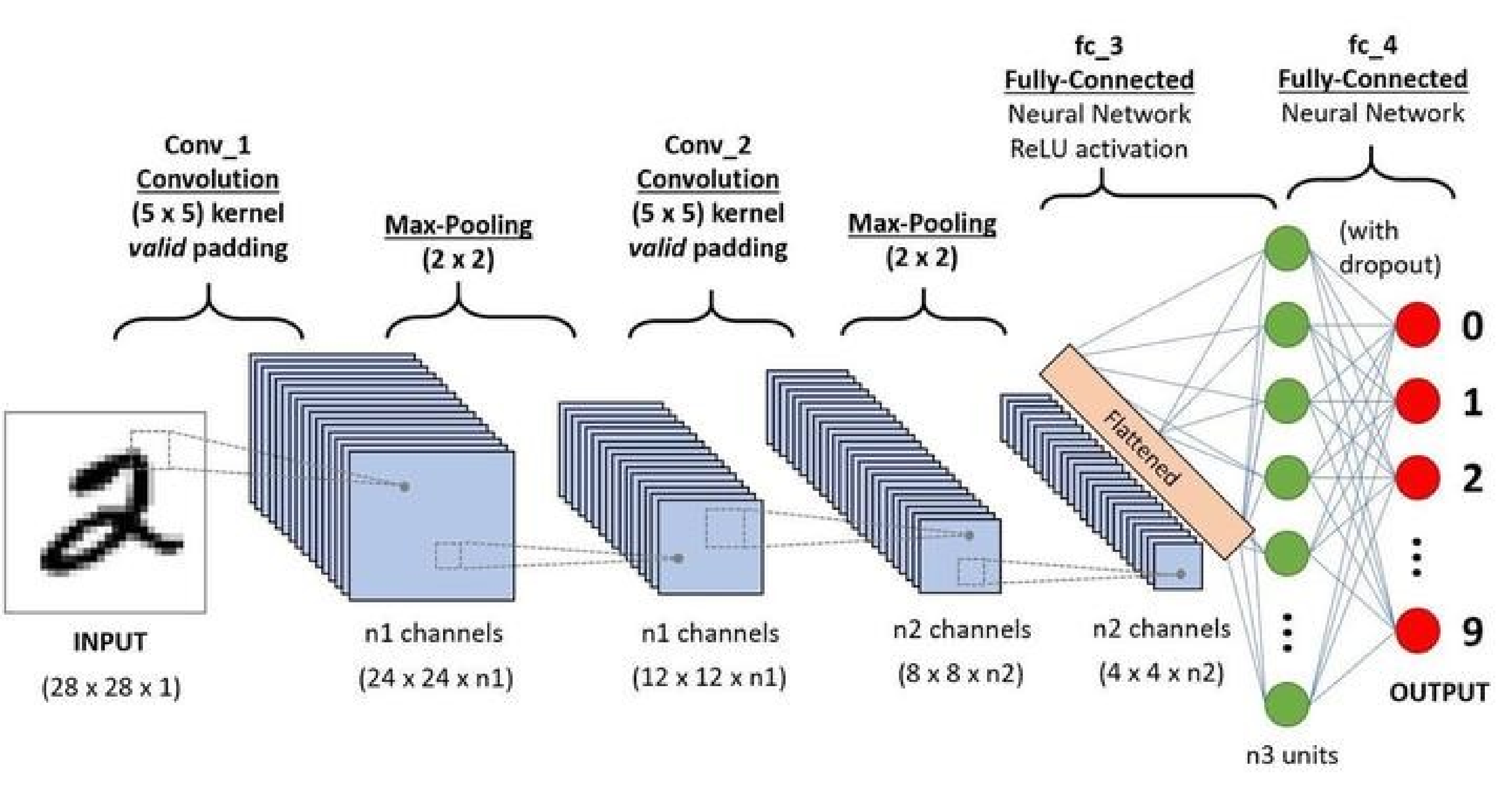
Fast radio burst (FRB)

We can use image classifier!



RFI

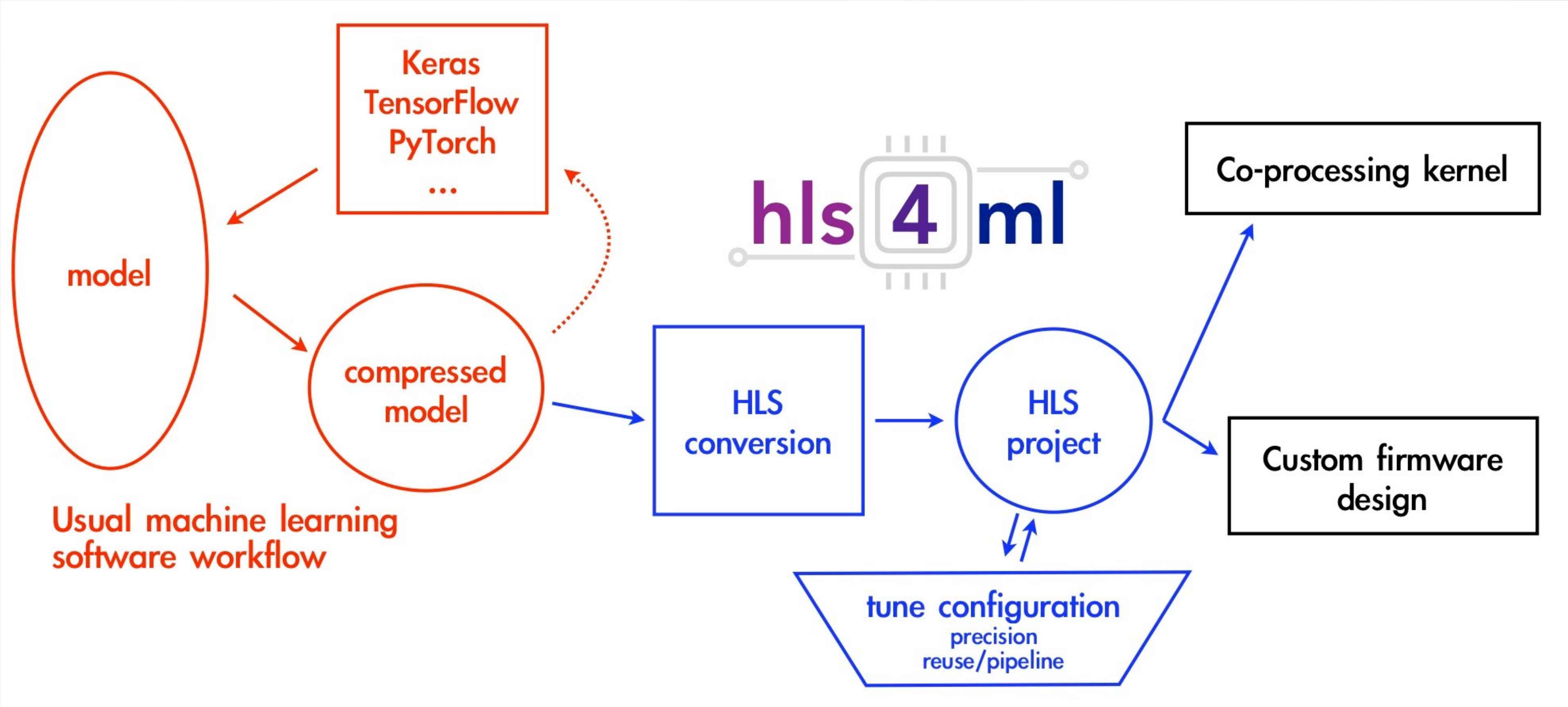
Convolutional Neural Network (CNN)



See Andrei's talk

hls4ml

https://github.com/fastmachinelearning/hls4ml



High-Level-Synthesis (HLS) C++

GCD(a, b) = GCD(b, a mod b)

a	b	a mob b
105	77	28
77	28	21
28	21	7
21	7	0

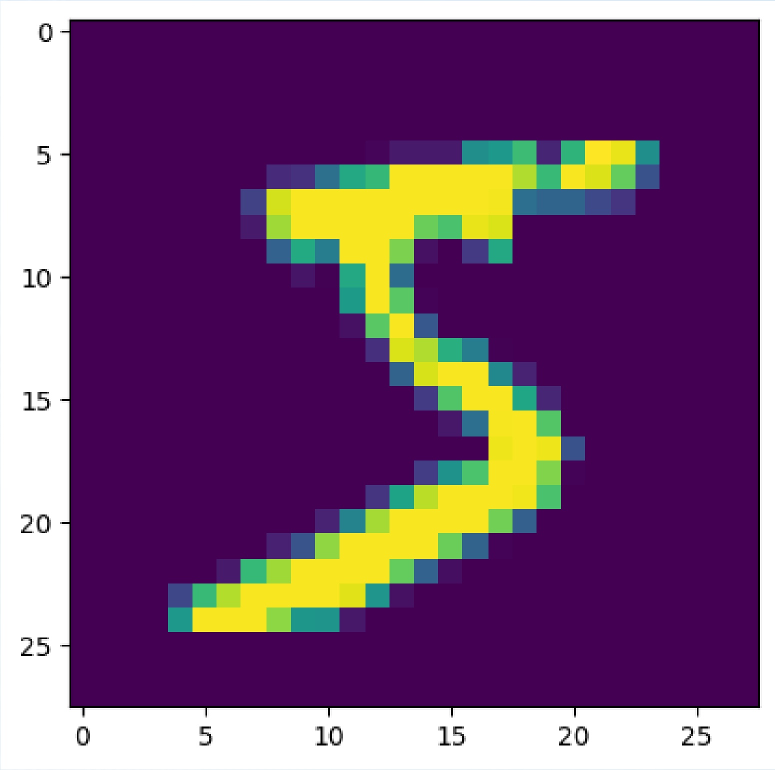
```
int gcd(int Ain, int
Bin)
{
    int a = Ain;
    int b = Bin;
    while (b != 0)
    {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }
    return a;
}
```

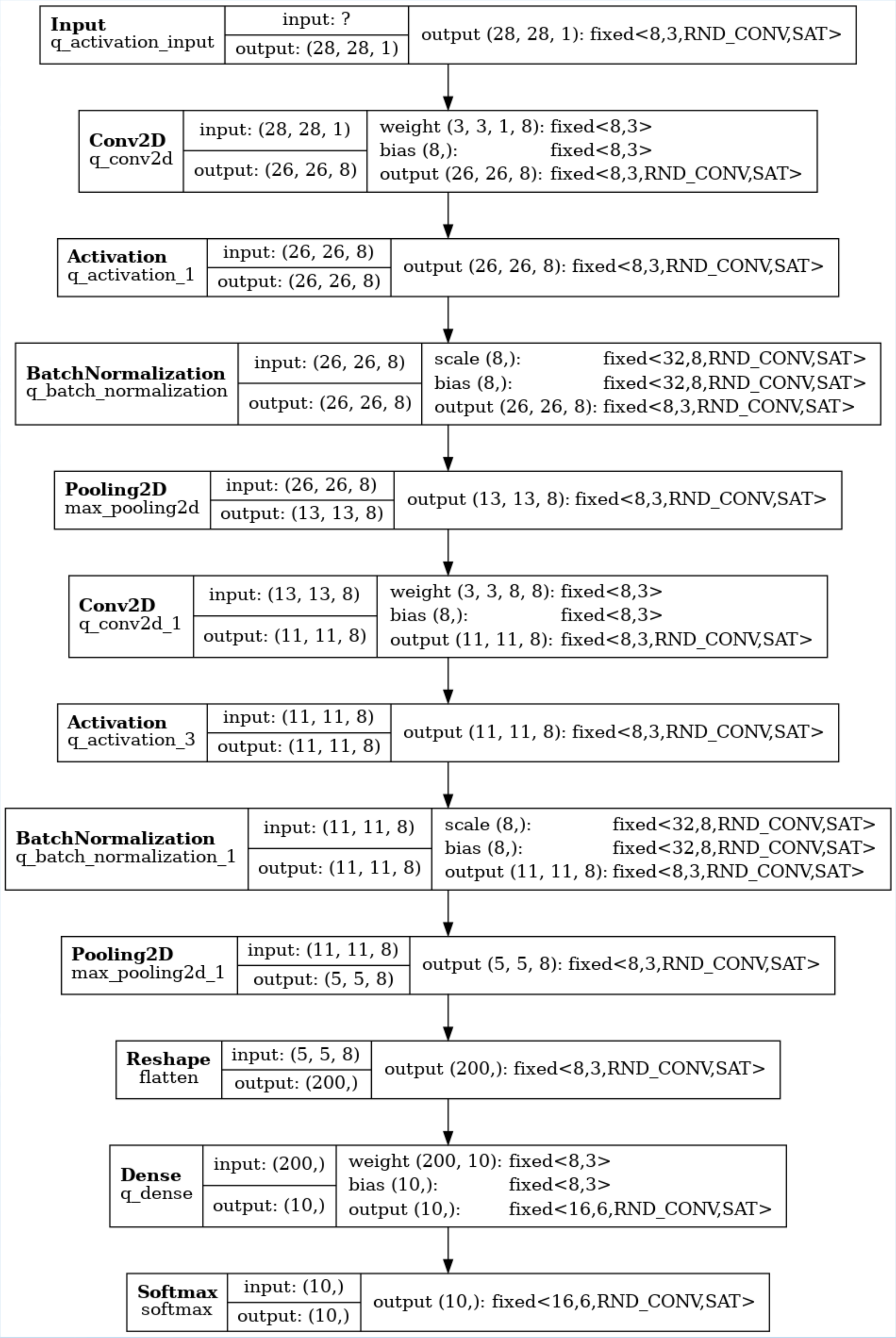
Euclidean algorithm

Verilog

```
module gcd (clk, start, Ain,
Bin, out, done);

input clk, start;
Input [31:0] Ain, Bin;
output reg [31:0] out;
output reg done;

reg [31:0] a, b;

always @ (posedge clk)
    begin
        if (start)
            begin
                a <= Ain; b <=
Bin; done <= 0;
            end
        else if (b == 0)
            begin
                out <= a;done <=
1;
            end
        else if (a > b)
            a <= a – b;
        else
            b <= b – a;
        end
endmodule
```

Test on mnist dataset





Qkeras

```
model = keras.Sequential([
	QActivation(activation=quantized_bits(8, 2), input_shape=(28, 28, 1)),
	QConv2D(8, 3, activation=quantized_bits(8, 2),
kernel_quantizer=quantized_bits(8,2,alpha=1),
bias_quantizer=quantized_bits(8,2,alpha=1), kernel_initializer='lecun_uniform'),
	QActivation(activation=quantized_relu(8, 2)),
	QBatchNormalization(beta_quantizer=quantized_bits(32, 8),
gamma_quantizer=quantized_bits(32, 8), mean_quantizer=quantized_bits(32, 8),
variance_quantizer=quantized_bits(32, 8)),
	QActivation(activation=quantized_bits(8, 2)),
	MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'),
	QConv2D(8, 3, activation=quantized_bits(8, 2),
kernel_quantizer=quantized_bits(8,2,alpha=1),
bias_quantizer=quantized_bits(8,2,alpha=1), kernel_initializer='lecun_uniform'),
	QActivation(activation=quantized_relu(8, 2)),
	QBatchNormalization(beta_quantizer=quantized_bits(32, 8),
gamma_quantizer=quantized_bits(32, 8), mean_quantizer=quantized_bits(32, 8),
variance_quantizer=quantized_bits(32, 8)),
	QActivation(activation=quantized_bits(8, 2)),
	MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'),
	Flatten(),
	QDense(10, kernel_quantizer=quantized_bits(8,2,alpha=1),
bias_quantizer=quantized_bits(8,2,alpha=1), kernel_initializer='lecun_uniform'),
	Softmax()
])
```

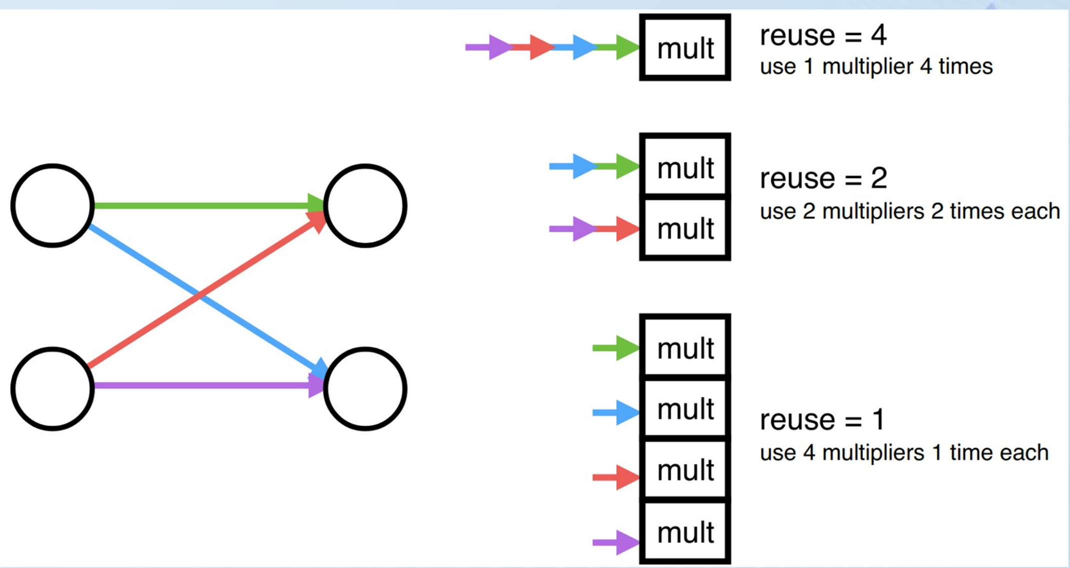
Yaml Configuration Example

```
OutputDir: model_1
ProjectName: myproject
XilinxPart: xcu250-figd2104-2L-e
ClockPeriod: 5
Backend: Vivado

IOType: io_stream
Model:
  Precision: ap_fixed<8,3,AP_RND_CONV,AP_SAT>
  ReuseFactor: 1
  Strategy: latency
LayerType:
  Softmax:
    Strategy: Stable
    Precision: ap_fixed<16,6,AP_RND_CONV,AP_SA
  QConv2D:
    Precision:
      weight: ap_fixed<8,3,AP_RND_CONV,AP_SAT>
      bias: ap_fixed<8,3,AP_RND_CONV,AP_SAT>
  QDense:
    Precision:
      weight: ap_fixed<8,3,AP_RND_CONV,AP_SAT>
      bias: ap_fixed<8,3,AP_RND_CONV,AP_SAT>
```

```
LayerName:
  q_batch_normalization:
    Precision:
      scale:
ap_fixed<32,8,AP_RND_CONV,AP_SAT>
      bias:
ap_fixed<32,8,AP_RND_CONV,AP_SAT>
  q_batch_normalization_1:
    Precision:
      scale:
ap_fixed<32,8,AP_RND_CONV,AP_SAT>
      bias:
ap_fixed<32,8,AP_RND_CONV,AP_SAT>
  q_conv2d:
    Precision:
      accum:
ap_fixed<26,16,AP_RND_CONV,AP_SAT>
  q_dense:
    Precision:
      accum:
ap_fixed<26,16,AP_RND_CONV,AP_SAT>
      result:
ap_fixed<16,6,AP_RND_CONV,AP_SAT>
```

Reuse factor

CONFIG_T::reuse_factor



nnet_conv2d_stream.h

```
    for (unsigned i_ih = 0; i_ih < CONFIG_T::in_height; i_ih++) {
    ReadInputWidth:
        for (unsigned i_iw = 0; i_iw < CONFIG_T::in_width; i_iw++) {
            #pragma HLS LOOP_FLATTEN
            if (CONFIG_T::strategy == nnet::latency) {
                #pragma HLS PIPELINE II=CONFIG_T::reuse_factor
            }
            if (CONFIG_T::filt_height > 1) {
                compute_output_buffer_2d<data_T, res_T, CONFIG_T>(data.read(), line_buffer, res,
weights, biases);
            } else {
                compute_output_buffer_1d<data_T, res_T, CONFIG_T>(data.read(), res, weights,
biases);
            }
        }
    }
```

Quantization

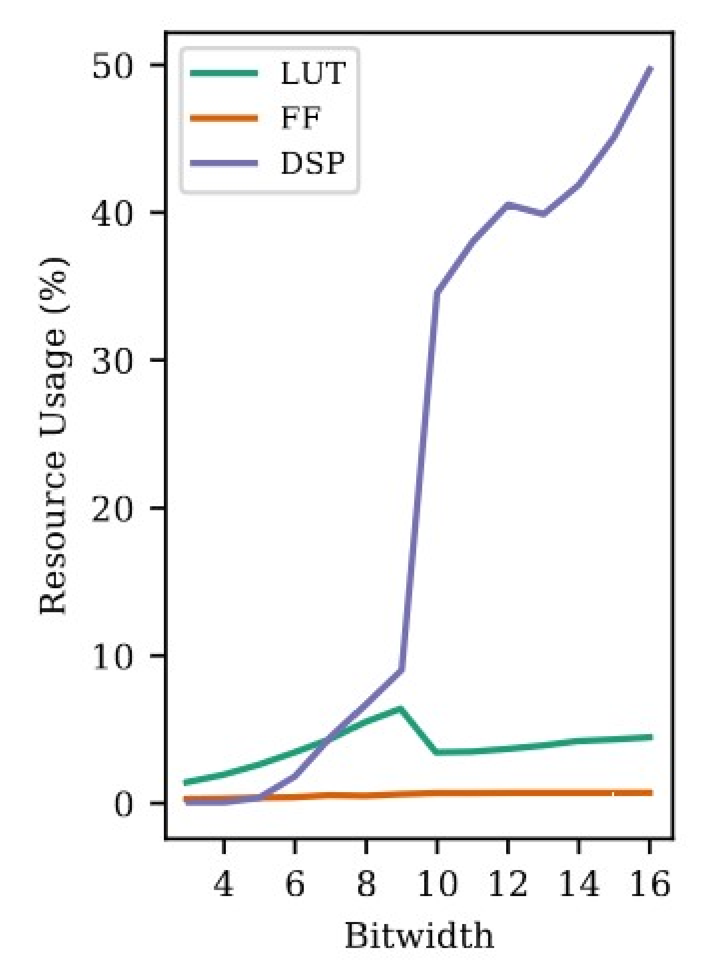
AutoQKeras

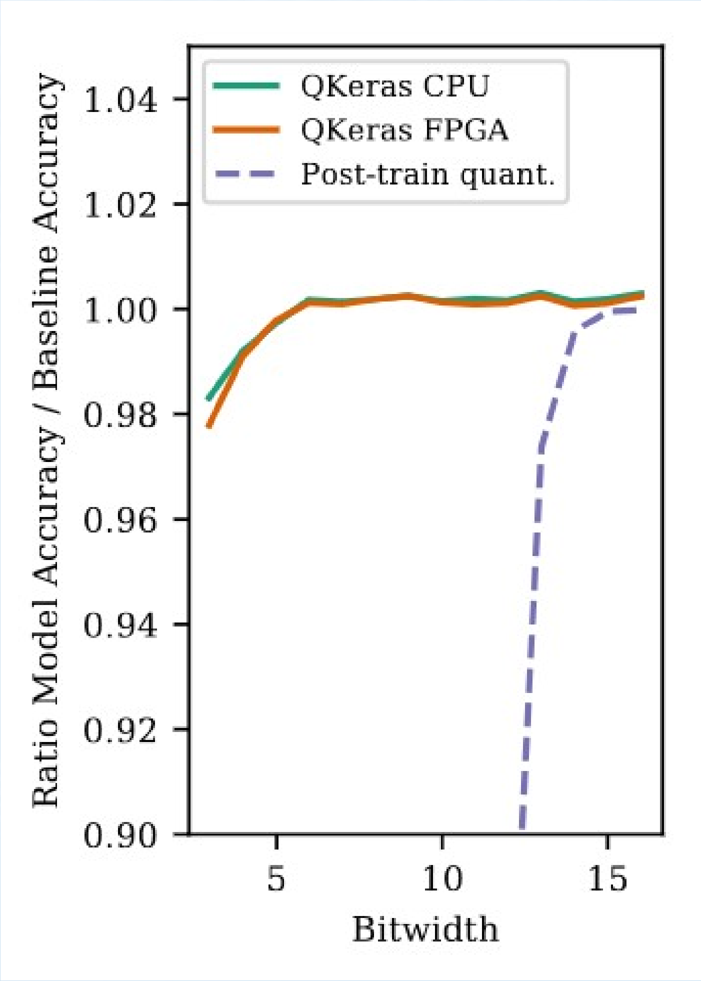
Post-training quantization (PTQ)

vs

Quantization aware training (QAT)

Quantization can significantly reduce resource usage!





Claudionor N. Coelho Jr. et al. 2021; Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors

Qkeras model vs hls model

- Qkeras: quantized_bits(8,2,alpha=1)
- Hls4ml: ap_fixed<8, 3, AP_RND_CONV, AP_SAT>

	Rounding mode (default)	Saturation mode (default)	accum_t (default)	Input / output quantization
Qkeras	AP_RND_CONV	AP_SAT	float	Qactivation layer
HLS	AP_TRN	AP_WRAP	Same as input	User-defined type

Pruning

Model complexity is reduced!

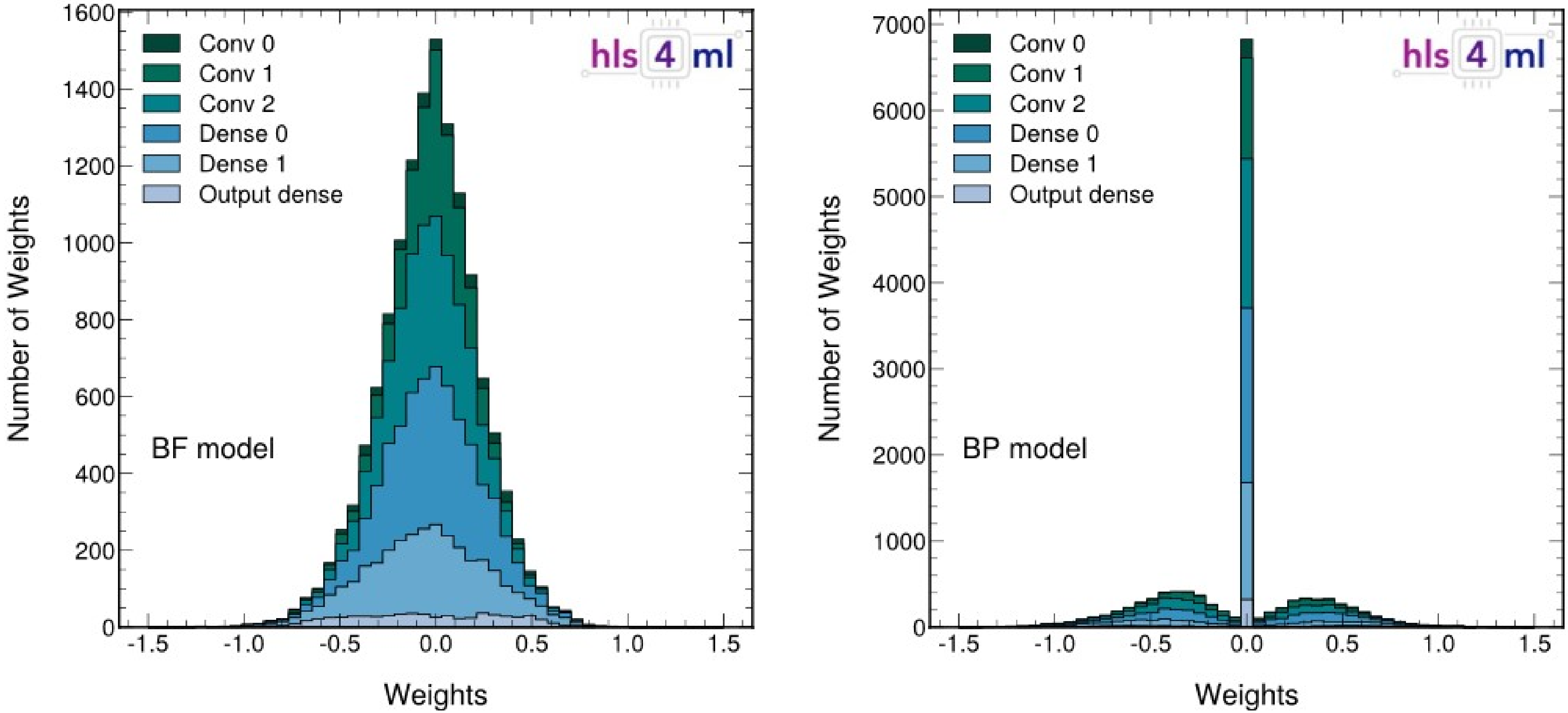


Figure 5. The weights per layer for the the BF model (left) and the baseline pruned (BP) model (right). The BP model is derived by starting from the BF model and repeating the training while applying a pruning procedure with a target sparsity of 50% for each layer.

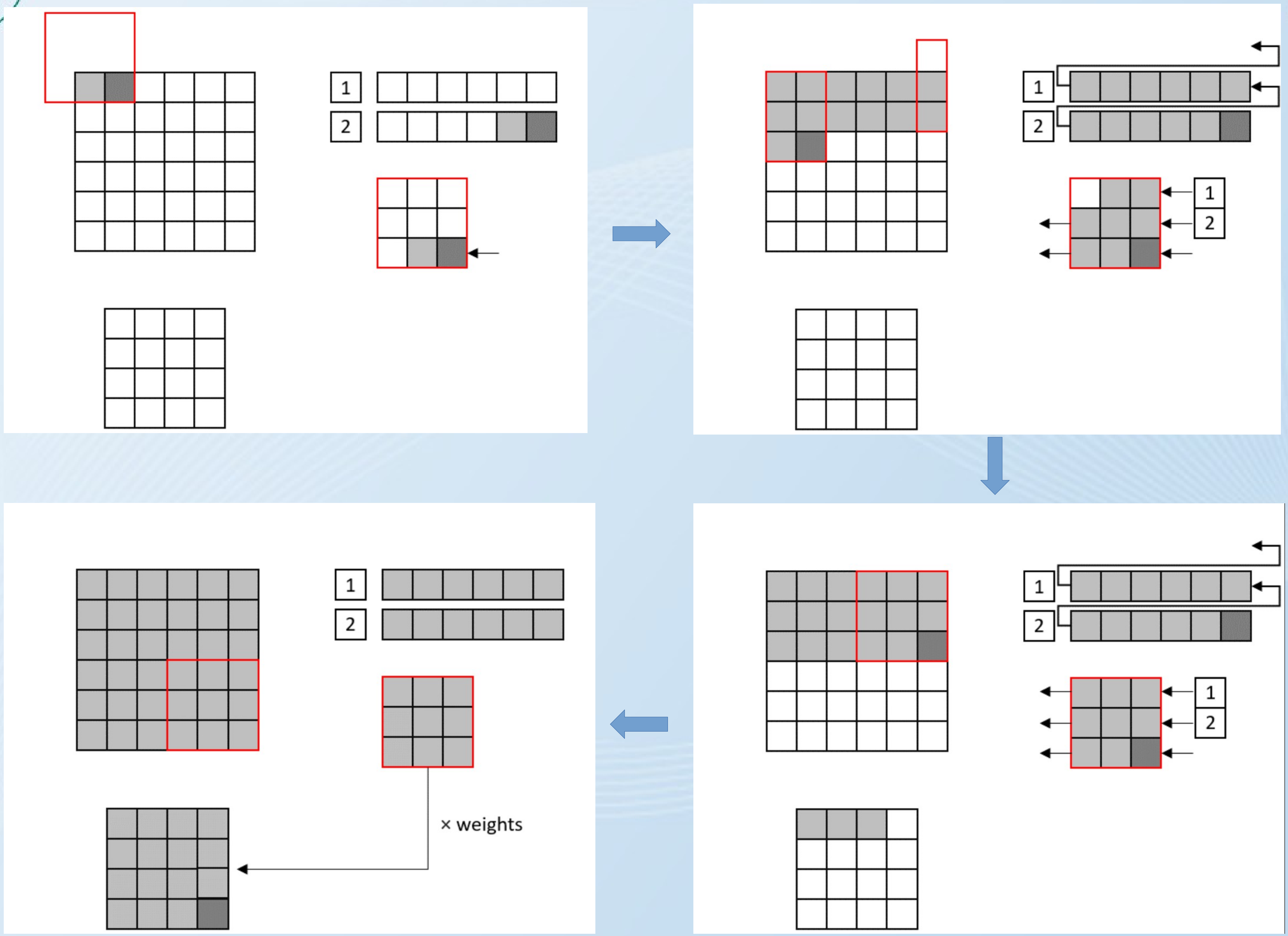
Aarrestad, Thea et al. 2021; Fast convolutional neural networks on FPGAs with hls4ml

Zero weights will be removed during hls synthesis!

Model pruning

```
pruning_params = {
    'pruning_schedule': tfmot.sparsity.keras.ConstantSparsity(0.9, 0),
    'block_size': (1, 1),
    'block_pooling_type': 'AVG'
}

def apply_pruning(layer):
  if isinstance(layer, QDense):
    return tfmot.sparsity.keras.prune_low_magnitude(layer, **pruning_params)
  elif isinstance(layer, QConv2D):
    return tfmot.sparsity.keras.prune_low_magnitude(layer, **pruning_params)
  return layer

model_for_pruning = tf.keras.models.clone_model(
    model,
    clone_function=apply_pruning,
)
```

hls4ml convolution algorithm



Aarrestad, Thea et al. 2021; Fast convolutional neural networks on FPGAs with hls4ml

hls4ml

Conv2D layer:

- Latency = in_h * in_w * reused factor
- Multiplier usage = filt_h * filt_w * n_ch * n_filt / reuse_factor
- Limitation: A model with many filters in one layer cannot be synthesized! (filt_h * filt_w * n_ch * n_filt < 4096)

nnet_dense_latency.h

```
  // Do the matrix-multiply
Product1:
   for (int ii = 0; ii < CONFIG_T::n_in; ii++) {
      cache = data[ii];
   Product2:
      for (int jj = 0; jj < CONFIG_T::n_out; jj++) {
         int index = ii * CONFIG_T::n_out + jj;
         mult[index] = CONFIG_T::template product<data_T, typename
CONFIG_T::weight_t>::product(cache, weights[index]);
      }
   }
```

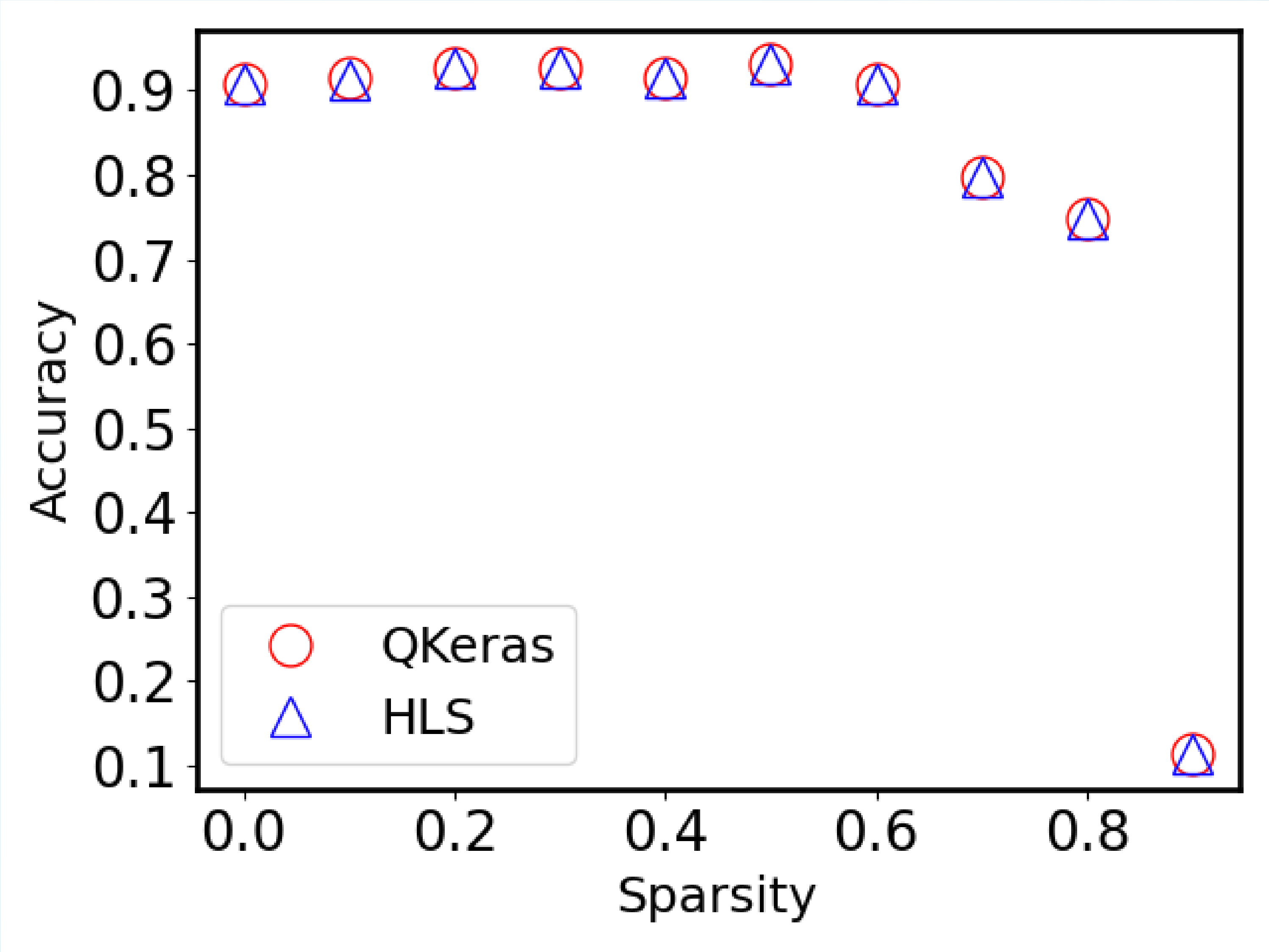
HLS code

```
void myproject(
    hls::stream<input_t> &q_activation_input,
    hls::stream<result_t> &layer18_out
) {

    // hls-fpga-machine-learning insert IO
    #pragma HLS INTERFACE axis port=q_activation_input,layer18_out
    #pragma HLS DATAFLOW

#ifndef __SYNTHESIS__
    static bool loaded_weights = false;
    if (!loaded_weights) {
        // hls-fpga-machine-learning insert load weights
        nnet::load_weights_from_txt<weight3_t, 72>(w3, "w3.txt");
        nnet::load_weights_from_txt<bias3_t, 8>(b3, "b3.txt");
        nnet::load_weights_from_txt<q_batch_normalization_scale_t, 8>(s6, "s6.txt");
        nnet::load_weights_from_txt<q_batch_normalization_bias_t, 8>(b6, "b6.txt");
        nnet::load_weights_from_txt<weight9_t, 576>(w9, "w9.txt");
        nnet::load_weights_from_txt<bias9_t, 8>(b9, "b9.txt");
        nnet::load_weights_from_txt<q_batch_normalization_1_scale_t, 8>(s12, "s12.txt");
        nnet::load_weights_from_txt<q_batch_normalization_1_bias_t, 8>(b12, "b12.txt");
        nnet::load_weights_from_txt<weight16_t, 2000>(w16, "w16.txt");
        nnet::load_weights_from_txt<bias16_t, 10>(b16, "b16.txt");
        loaded_weights = true;
    }
#endif
```

HLS code

```
    hls::stream<layer3_t> layer3_out("layer3_out");
    #pragma HLS STREAM variable=layer3_out depth=676
    nnet::conv_2d_cl<input_t, layer3_t, config3>(q_activation_input, layer3_out, w3, b3);

    hls::stream<layer5_t> layer5_out("layer5_out");
    #pragma HLS STREAM variable=layer5_out depth=676
    nnet::relu<layer3_t, layer5_t, relu_config5>(layer3_out, layer5_out);

    hls::stream<layer6_t> layer6_out("layer6_out");
    #pragma HLS STREAM variable=layer6_out depth=676
    nnet::normalize<layer5_t, layer6_t, config6>(layer5_out, layer6_out, s6, b6);

    hls::stream<layer8_t> layer8_out("layer8_out");
    #pragma HLS STREAM variable=layer8_out depth=169
    nnet::pooling2d_cl<layer6_t, layer8_t, config8>(layer6_out, layer8_out);
    ......

    auto& layer15_out = layer14_out;
    hls::stream<layer16_t> layer16_out("layer16_out");
    #pragma HLS STREAM variable=layer16_out depth=1
    nnet::dense<layer14_t, layer16_t, config16>(layer15_out, layer16_out, w16, b16);

    nnet::softmax<layer16_t, result_t, Softmax_config18>(layer16_out, layer18_out);
}
```

Accuracy-Sparsity



Latency and Resource usage

No pruning

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
4818	4818	24.090 us	24.090 us	4707	4707	dataflow

== Utilization Estimates

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	56	-	2338	3272	-
Instance	2	33	78294	458584	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	58	33	80632	461858	0
Available SLR	1344	3072	864000	432000	320
Utilization SLR (%)	4	1	9	106	0
Available	5376	12288	3456000	1728000	1280
Utilization (%)	1	~0	2	26	0

Pruning with sparsity=0.5

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
3991	3991	19.955 us	19.955 us	3923	3923	dataflow

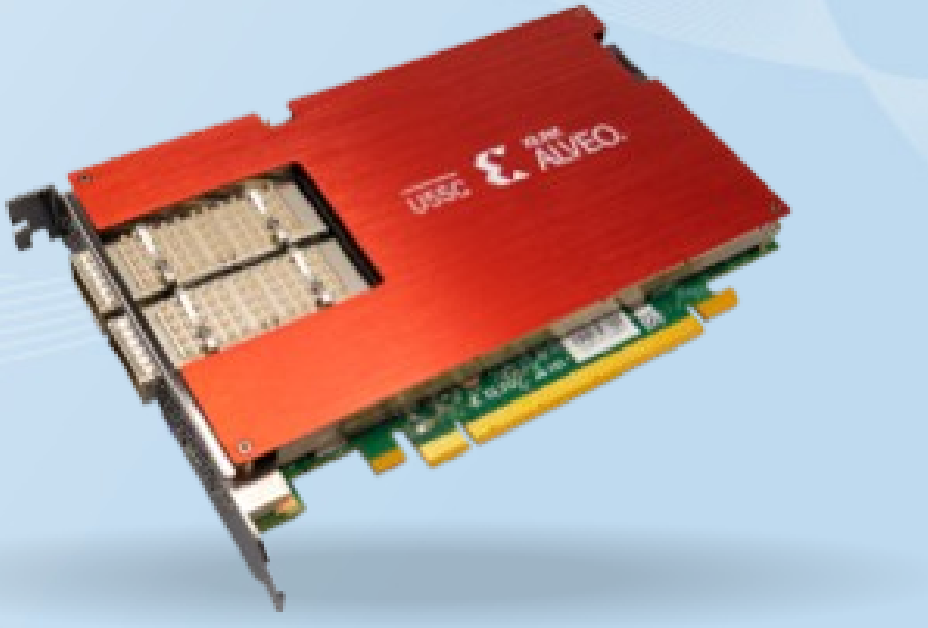
== Utilization Estimates

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	56	-	2338	3272	-
Instance	2	32	59882	249838	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	58	32	62220	253112	0
Available SLR	1344	3072	864000	432000	320
Utilization SLR (%)	4	1	7	58	0
Available	5376	12288	3456000	1728000	1280
Utilization (%)	1	~0	1	14	0

Future Work

- Understand AutoQKeras
- Convert our model with hls4ml
- Deploy the model on Alveo card (some vitis kernel needed)
- Test on real data



Thank you for your attention!