CSE 237C Final Project Report
Relu Optimizer for HLS4ML

Gandhar Deshpande
PID: A59005457
gdeshpande@ucsd.edu

Introduction:

Convolutional Neural Networks are one of the most efficient ways to run algorithms for image processing, signal processing, video processing and image classification. These networks are usually made of multiple layers and each layer does some processing on the data. These can be optimized using FPGAs to provide high performance as compared to a traditional software-based implementations. HLS4ML is a framework that allows automation of a neural network given its parameters into a HLS project which can then be synthesized to a FPGA using Xilinx Vivado. This project aims to add an optimizer to the HLS4ML framework to improve the resource usage of the CNN on the FPGA, which may further allow deeper networks to be implemented on the same FPGA.

Literature:

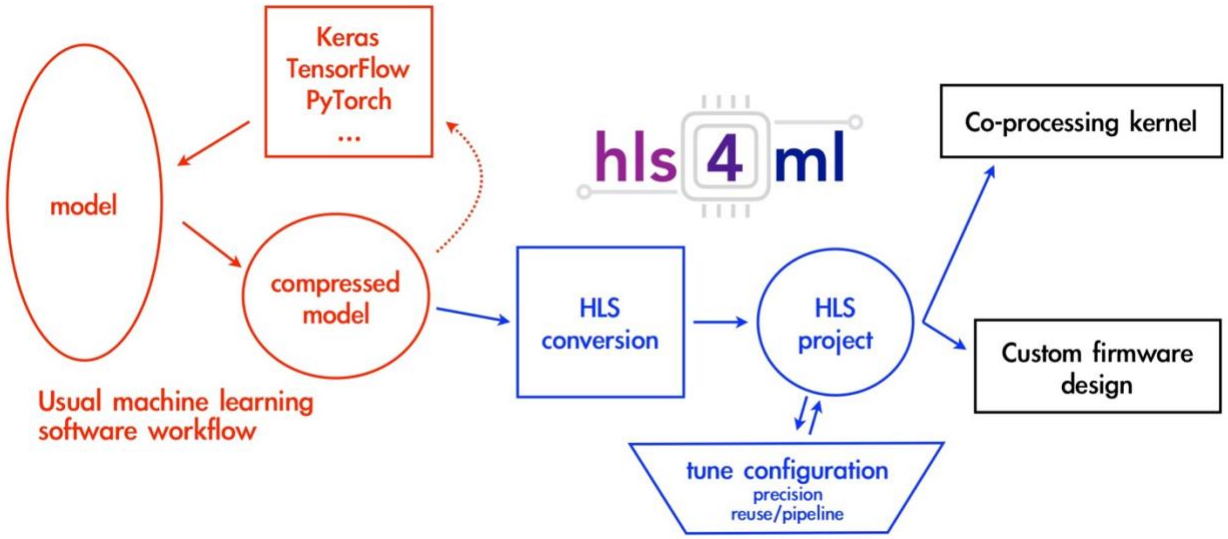
Convolutional Neural Networks:

In Deep Learning, Convolutional Neural Networks is a class of Neural Networks that is largely used for analyzing visual imagery. The larger applications of this come in the field of image and video recognition, recommender systems, image classification and so on. They are a regularized form of a multilayer perceptron artificial neural network. While normal neural networks require significant amount of data processing before it can be fed into the neural networks, CNNs provide an advantage by using the convolution kernels in a manner to automate the learning, requiring very little effort from a developer.
[Information taken from

HLS4ML:

HLS4ML is a Python package used for machine learning inference in FPGAs. The package creates firmware implementations of machine learning algorithms using High Level Synthesis language (HLS). The aim is to translate traditional open-source machine learning models into HLS that can be configured according to use-case. The resulting IP can then be plugged into more complex designs or used for creating a kernel for CPU co-processing.

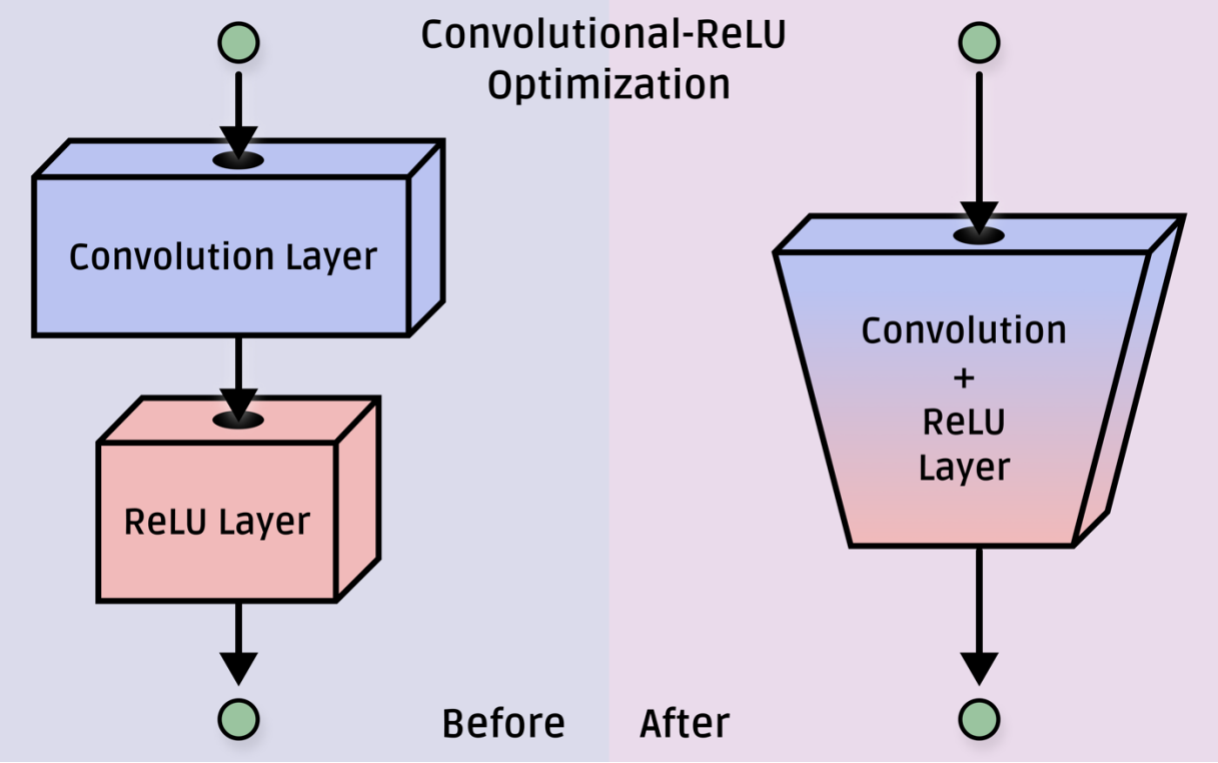
This solution was inspired by data analysis problems in the Large Hadron Collider in Particle Physics. FPGAs allow data filtering at the source, which cannot be managed with a traditional inferencing model, which is not able to match the live speed of events. This package allows user to define various parameters like precision, reuse factor and data quantization aware training. It expects that the input network is simplified for the best improvement in results. All of this together results in microsecond-latency in predictions with FPGAs. The below shows a broad workflow model of HLS4ML.



[Image and documentation with reference from hls4ml website https://fastmachinelearning.org/hls4ml/index.html]

Aim:

A convolutional neural network typically contains multiple layers of convolution followed by layers of activation. One of the most popular activation techniques used is called ReLU, Rectified Linear Unit. When HLS4ML converts a given network model into an HLS project in C++, it essentially converts each layer into a function. These functions are pipelined using Dataflow pragmas, which bring optimization in the hardware. Each layer connects to the next with a BRAM, which is costly in FPGA resources. Since the implementation of the ReLU is simple, this project aims to essentially merge the ReLU function into the previous convolution layer, which should reduce the number of BRAMS that are being used. The aim is to reduce resource usage while still gaining the same hardware acceleration. This was verified by manually modifying the HLS project to merge the Relu and convolution layers with better results.



Implementation:

First, I modified the python conversion script which parses the trained model and writes it with user defined precision. I added a new flag variable called 'MergedRelu', which can be set to 1 or 0 depending on whether one wants to merge the Relu layer with the convolution layer. The convert script parses this configuration and based on the value of the flag, excludes the optimizer which would do the merging.

```
convert:
  RemoveSoftmax: 1
  MergedRelu: 1
  OutputDir: my-hls-tiny2-auto-merged-relu
  XilinxPart: xc7z020clg400-1
  Backend: Vivado
  IOType: io_stream
  Interface: axi_stream
  Driver: python
  Board: pynq-z2
```

Fig 1.3 YML with MergedRelu Flag set

```
    config['Model']['MergedRelu'] = our_config['convert']['MergedRelu']
    if bool(config['Model']['MergedRelu']):
        config['SkipOptimizers'] = ['reshape_stream']
    else:
        config['SkipOptimizers'] = ['reshape_stream', 'relu_merge']
```

Fig 1.4 Script snippet that add the flag to the config and decides whether the optimizer will be called.

The way the model work is that first a model is created with all the layers in the network, and then optimizers are applied to it to modify the model in an optimal fashion. SkipOptimizers from the snippet above indicates that the optimizers in that array would not be applied.

The next step was to write an actual optimizer pass. The optimizer has two parts, one where it matches that the optimizer is applicable, and where it transforms the model for optimization. The matching part is done by comparing the name of the layer and the previous layer, which should be Activation and Convolution respectively. The transformation does a few things. First, it sets the output of the previous layer to be same as the output for the activation layer. Then it removes the activation layer and rewires the previous and the next node together, so that the layers are connected. Since the activation and the convolution have the same shapes for the output, reconnecting does not require any other effort. This optimizer is registered during initialization.

```
class MergeRelu(OptimizerPass):
    def match(self, node):
        is_match = node.__class__.__name__ == 'Activation' and \
            (node.get_input_node().__class__.__name__ == 'Conv2D' or
            node.get_input_node().__class__.__name__ == 'Conv2DBatchnorm')
        return is_match

    def transform(self, model, node):
        #Merge ReLU and Convolution layer if needed
        previous_node = node.get_input_node()
        previous_node.index = node.index
        if previous_node.get_attr('data_format') == 'channels_last':
            shape = [previous_node.attributes['out_height'], previous_node.attributes['out_width'], previous_node.attributes['n_filt']]
            dims = ['OUT_HEIGHT_{}'.format(previous_node.index), 'OUT_WIDTH_{}'.format(previous_node.index), 'N_FILT_{}'.format(previous_node.index)]
        else:
            shape = [previous_node.attributes['n_filt'], previous_node.attributes['out_height'], previous_node.attributes['out_width']]
            dims = ['N_FILT_{}'.format(previous_node.index), 'OUT_HEIGHT_{}'.format(previous_node.index), 'OUT_WIDTH_{}'.format(previous_node.index)]
        previous_node.add_output_variable(shape, dims)
        if not node.get_output_nodes():
            print("WARNING: {} is the output layer! No rewiring performed.".format(node.name))
            model.remove_node(node, rewire=False)
        else:
            model.remove_node(node, rewire=True)
        return True
```

Fig 1.5: Operation of the MergeRelu Optimizer pass

The code up until now creates a merged ReLU layer in an HLS model in python. However, the same changes need to be reflected in the actual firmware code that gets created. To implement this, there are 2 major areas where changes are required. The first is the HLS Layers, where we need to read the flag from the model and add it into the configuration of the convolution layer. The second is make changes in the templates which will read the changes in configuration of the layers. The C++ functions are written in an object-oriented fashion, so after going through the original convolution function, we find that the actual computation is happening in a class

called nnet_dense_resource.h. So, I add new compute functions that do the computation as well as the activation in a single function. I also modify the main function call to check if the configuration has a merged_relu flag, and decides which compute function to call based on the flag.

Results:

The project creates a full project where the main project file is now created without the activation layers. The final merged project is compiled with Vivado and passes the C Simulation. It gives slightly different results compared to the manually merged project. The code is synthesized into RTL and gives results.

```
//hls-fpga-machine-learning insert layers

hls::stream<layer2_t> layer2_out("layer2_out");
#pragma HLS STREAM variable=layer2_out depth=1024
nnet::conv_2d_cl<input_t, layer2_t, config2>(input_1, layer2_out, w2, b2); // q_conv2d_batchnorm

hls::stream<layer4_t> layer4_out("layer4_out");
#pragma HLS STREAM variable=layer4_out depth=1024
nnet::relu<layer2_t, layer4_t, relu_config4>(layer2_out, layer4_out); // q_activation

hls::stream<layer20_t> layer20_out("layer20_out");
#pragma HLS STREAM variable=layer20_out depth=1225
nnet::zeropad2d_cl<layer4_t, layer20_t, config20>(layer4_out, layer20_out); // zp2d_q_conv2d_batchnorm_1

hls::stream<layer5_t> layer5_out("layer5_out");
#pragma HLS STREAM variable=layer5_out depth=1024
nnet::conv_2d_cl<layer20_t, layer5_t, config5>(layer20_out, layer5_out, w5, b5); // q_conv2d_batchnorm_1

hls::stream<layer7_t> layer7_out("layer7_out");
#pragma HLS STREAM variable=layer7_out depth=1024
nnet::relu<layer5_t, layer7_t, relu_config7>(layer5_out, layer7_out); // q_activation_1

hls::stream<layer21_t> layer21_out("layer21_out");
#pragma HLS STREAM variable=layer21_out depth=1225
nnet::zeropad2d_cl<layer7_t, layer21_t, config21>(layer7_out, layer21_out); // zp2d_q_conv2d_batchnorm_2

hls::stream<layer8_t> layer8_out("layer8_out");
#pragma HLS STREAM variable=layer8_out depth=1024
nnet::conv_2d_cl<layer21_t, layer8_t, config8>(layer21_out, layer8_out, w8, b8); // q_conv2d_batchnorm_2

hls::stream<layer10_t> layer10_out("layer10_out");
#pragma HLS STREAM variable=layer10_out depth=1024
nnet::relu<layer8_t, layer10_t, relu_config10>(layer8_out, layer10_out); // q_activation_2
```

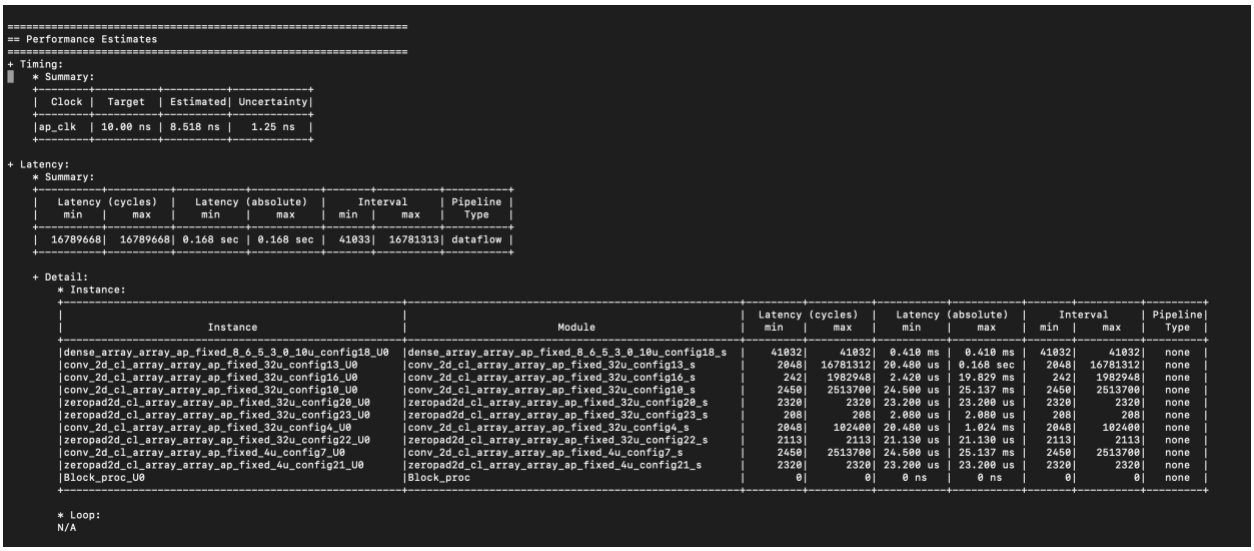
Fig 1.6: Code generated without MergeRelu flag

```
    //hls-fpga-machine-learning insert layers

    hls::stream<layer4_t> layer4_out("layer4_out");
    #pragma HLS STREAM variable=layer4_out depth=1024
    nnet::conv_2d_cl<input_t, layer4_t, config4>(input_1, layer4_out, w2, b2); // q_conv2d_batchnorm

    hls::stream<layer20_t> layer20_out("layer20_out");
    #pragma HLS STREAM variable=layer20_out depth=1225
    nnet::zeropad2d_cl<layer4_t, layer20_t, config20>(layer4_out, layer20_out); // zp2d_q_conv2d_batchnorm_1

    hls::stream<layer7_t> layer7_out("layer7_out");
    #pragma HLS STREAM variable=layer7_out depth=1024
    nnet::conv_2d_cl<layer20_t, layer7_t, config7>(layer20_out, layer7_out, w5, b5); // q_conv2d_batchnorm_1

    hls::stream<layer21_t> layer21_out("layer21_out");
    #pragma HLS STREAM variable=layer21_out depth=1225
    nnet::zeropad2d_cl<layer7_t, layer21_t, config21>(layer7_out, layer21_out); // zp2d_q_conv2d_batchnorm_2

    hls::stream<layer10_t> layer10_out("layer10_out");
    #pragma HLS STREAM variable=layer10_out depth=1024
    nnet::conv_2d_cl<layer21_t, layer10_t, config10>(layer21_out, layer10_out, w8, b8); // q_conv2d_batchnorm_2

    hls::stream<layer22_t> layer22_out("layer22_out");
    #pragma HLS STREAM variable=layer22_out depth=1024
    nnet::zeropad2d_cl<layer10_t, layer22_t, config22>(layer10_out, layer22_out); // zp2d_q_conv2d_batchnorm_3

    hls::stream<layer13_t> layer13_out("layer13_out");
    #pragma HLS STREAM variable=layer13_out depth=64
    nnet::conv_2d_cl<layer22_t, layer13_t, config13>(layer22_out, layer13_out, w11, b11); // q_conv2d_batchnorm_3

    hls::stream<layer23_t> layer23_out("layer23_out");
    #pragma HLS STREAM variable=layer23_out depth=121
    nnet::zeropad2d_cl<layer13_t, layer23_t, config23>(layer13_out, layer23_out); // zp2d_q_conv2d_batchnorm_4

    hls::stream<layer16_t> layer16_out("layer16_out");
    #pragma HLS STREAM variable=layer16_out depth=64
    nnet::conv_2d_cl<layer23_t, layer16_t, config16>(layer23_out, layer16_out, w14, b14); // q_conv2d_batchnorm_4

    nnet::dense<layer16_t, layer18_t, config18>(layer16_out, layer18_out, w18, b18); // q_dense
```

Fig 1.7: Merged layers generated by HLS4ML

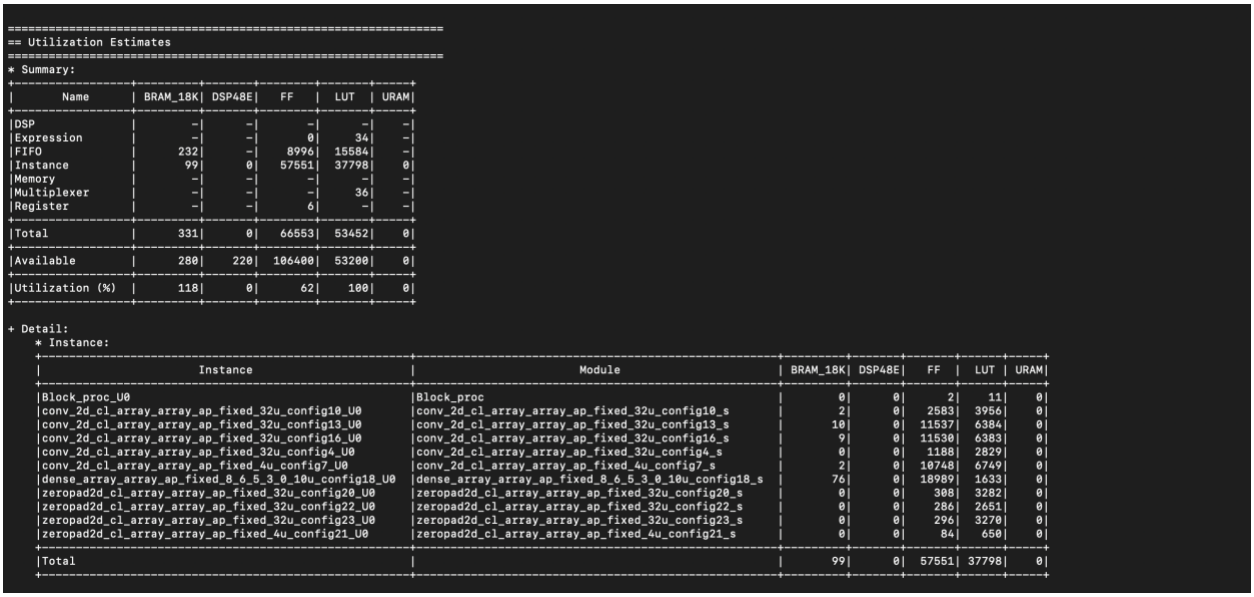
```
Processing input 0:
    Ground Truth Predictions (gt):
        0 0 0 1 0 0 0 0 0 0
    hls_py Predictions (hp):
        -15.5 -15.75 -13 -8.75 -16.25 -10.25 -11 -16.75 -11.75 -14.25
    keras Predictions (k):
        -16.2554 -15.3726 -12.2603 -8.08398 -16.9204 -7.89014 -11.2266 -16.5654 -13.1304 -14.7075
    hls_csim Predictions (hc):
        -15.5 -16 -12.5 -8.5 -16.25 -9.75 -11 -16.75 -12 -14.75
Summary of Test:
    Total # of Test Inputs = 100
    Total # of Matching Predictions = 80 (0.8)

| 1| 0.83| 0.83| 0.8 |
| 0.83| 1| 0.93| 0.97 |
| 0.83| 0.93| 1| 0.9 |
| 0.8| 0.97| 0.9| 1 |
INFO: Saved inference results to file: tb_data/csim_results.log
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Fig 1.8: C Simulation Results



```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  | 10.00 ns | 8.518 ns |   1.25 ns  |
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+----------+-----------+-----------+-------+---------+----------+
    |   Latency (cycles)  |   Latency (absolute)  |     Interval     | Pipeline |
    |    min  |    max   |    min    |    max    |  min  |   max   |   Type   |
    +---------+----------+-----------+-----------+-------+---------+----------+
    | 16789668| 16789668| 0.168 sec | 0.168 sec |  41033|  16781313| dataflow |
    +---------+----------+-----------+-----------+-------+---------+----------+

    + Detail:
        * Instance:
        +-----------------------------------------------------+-----------------------------------------------------+---------+---------+-----------+-----------+-------+-------+----------+
        |                                                     |                                                     | Latency (cycles) |   Latency (absolute)  |     Interval     | Pipeline |
        |                       Instance                      |                        Module                       |   min   |   max   |    min    |    max    |  min  |  max  |   Type   |
        +-----------------------------------------------------+-----------------------------------------------------+---------+---------+-----------+-----------+-------+-------+----------+
        |dense_array_array_ap_fixed_8_6_5_3_0_10u_config18_U0 |dense_array_array_ap_fixed_8_6_5_3_0_10u_config18_s |    41032|    41032| 0.410 ms  | 0.410 ms  |  41032|  41032|   none   |
        |conv_2d_cl_array_array_ap_fixed_32u_config13_U0      |conv_2d_cl_array_array_ap_fixed_32u_config13_s      |     2048| 16781312| 20.480 us | 0.168 sec |   2048| 16781312|   none   |
        |conv_2d_cl_array_array_ap_fixed_32u_config16_U0      |conv_2d_cl_array_array_ap_fixed_32u_config16_s      |      242|  1982948|  2.420 us | 19.829 ms |    242|  1982948|   none   |
        |conv_2d_cl_array_array_ap_fixed_32u_config10_U0      |conv_2d_cl_array_array_ap_fixed_32u_config10_s      |     2450| 2513700| 24.500 us | 25.137 ms |   2450| 2513700|   none   |
        |zeropad2d_cl_array_array_ap_fixed_32u_config20_U0    |zeropad2d_cl_array_array_ap_fixed_32u_config20_s    |     2320|     2320| 23.200 us | 23.200 us |   2320|     2320|   none   |
        |zeropad2d_cl_array_array_ap_fixed_32u_config23_U0    |zeropad2d_cl_array_array_ap_fixed_32u_config23_s    |      208|      208|  2.080 us |  2.080 us |    208|      208|   none   |
        |conv_2d_cl_array_array_ap_fixed_32u_config4_U0       |conv_2d_cl_array_array_ap_fixed_32u_config4_s       |     2048|   102400| 20.480 us |  1.024 ms |   2048|   102400|   none   |
        |zeropad2d_cl_array_array_ap_fixed_32u_config22_U0    |zeropad2d_cl_array_array_ap_fixed_32u_config22_s    |     2113|     2113| 21.130 us | 21.130 us |   2113|     2113|   none   |
        |conv_2d_cl_array_array_ap_fixed_4u_config7_U0        |conv_2d_cl_array_array_ap_fixed_4u_config7_s        |     2450| 2513700| 24.500 us | 25.137 ms |   2450| 2513700|   none   |
        |zeropad2d_cl_array_array_ap_fixed_4u_config21_U0     |zeropad2d_cl_array_array_ap_fixed_4u_config21_s     |     2320|     2320| 23.200 us | 23.200 us |   2320|     2320|   none   |
        |Block_proc_U0                                        |Block_proc                                          |        0|        0|   0 ns    |   0 ns    |      0|        0|   none   |
        +-----------------------------------------------------+-----------------------------------------------------+---------+---------+-----------+-----------+-------+-------+----------+

        * Loop:
        N/A
```

Fig 1.9: Synthesis performance results



```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+--------+-------+-----+
|       Name      | BRAM_18K| DSP48E|   FF   |  LUT  | URAM|
+-----------------+---------+-------+--------+-------+-----+
|DSP              |        -|      -|       -|      -|    -|
|Expression       |        -|      -|       0|     34|    -|
|FIFO             |      232|      -|    8996|  15584|    -|
|Instance         |       99|      0|   57551|  37798|    0|
|Memory           |        -|      -|       -|      -|    -|
|Multiplexer      |        -|      -|       -|     36|    -|
|Register         |        -|      -|       6|      -|    -|
+-----------------+---------+-------+--------+-------+-----+
|Total            |      331|      0|   66553|  53452|    0|
+-----------------+---------+-------+--------+-------+-----+
|Available        |      280|    220|  106400|  53200|    0|
+-----------------+---------+-------+--------+-------+-----+
|Utilization (%)  |      118|      0|      62|    100|    0|
+-----------------+---------+-------+--------+-------+-----+

+ Detail:
    * Instance:
    +-----------------------------------------------------+-----------------------------------------------------+---------+-------+--------+-------+-----+
    |                       Instance                      |                        Module                       | BRAM_18K| DSP48E|   FF   |  LUT  | URAM|
    +-----------------------------------------------------+-----------------------------------------------------+---------+-------+--------+-------+-----+
    |Block_proc_U0                                        |Block_proc                                          |        0|      0|       2|     11|    0|
    |conv_2d_cl_array_array_ap_fixed_32u_config10_U0      |conv_2d_cl_array_array_ap_fixed_32u_config10_s      |        2|      0|    2583|   3956|    0|
    |conv_2d_cl_array_array_ap_fixed_32u_config13_U0      |conv_2d_cl_array_array_ap_fixed_32u_config13_s      |       10|      0|   11537|   6384|    0|
    |conv_2d_cl_array_array_ap_fixed_32u_config16_U0      |conv_2d_cl_array_array_ap_fixed_32u_config16_s      |        9|      0|   11530|   6383|    0|
    |conv_2d_cl_array_array_ap_fixed_32u_config4_U0       |conv_2d_cl_array_array_ap_fixed_32u_config4_s       |        0|      0|    1188|   2829|    0|
    |conv_2d_cl_array_array_ap_fixed_4u_config7_U0        |conv_2d_cl_array_array_ap_fixed_4u_config7_s        |        2|      0|   10748|   6749|    0|
    |dense_array_array_ap_fixed_8_6_5_3_0_10u_config18_U0 |dense_array_array_ap_fixed_8_6_5_3_0_10u_config18_s |       76|      0|   18989|   1633|    0|
    |zeropad2d_cl_array_array_ap_fixed_32u_config20_U0    |zeropad2d_cl_array_array_ap_fixed_32u_config20_s    |        0|      0|     308|   3282|    0|
    |zeropad2d_cl_array_array_ap_fixed_32u_config22_U0    |zeropad2d_cl_array_array_ap_fixed_32u_config22_s    |        0|      0|     286|   2651|    0|
    |zeropad2d_cl_array_array_ap_fixed_32u_config23_U0    |zeropad2d_cl_array_array_ap_fixed_32u_config23_s    |        0|      0|     296|   3270|    0|
    |zeropad2d_cl_array_array_ap_fixed_4u_config21_U0     |zeropad2d_cl_array_array_ap_fixed_4u_config21_s     |        0|      0|      84|    650|    0|
    +-----------------------------------------------------+-----------------------------------------------------+---------+-------+--------+-------+-----+
    |Total                                                |                                                    |       99|      0|   57551|  37798|    0|
    +-----------------------------------------------------+-----------------------------------------------------+---------+-------+--------+-------+-----+
```

Fig 1.10: Synthesis Utilization Results

Conclusion and Future Work:

The project creates an enhancement in a Python package that allows an implementation with a reduced resource usage. With just a single user flag, it creates a complete firmware which is optimized for resource usage. While the logic is in place, there is possibility for some more changes to be done in future. The code is currently created for 2D convolution, but it can be extended to 1D convolution. There are also some issues with co-simulation results, and need some more work to get output from co-simulation. The codebase is added on Github(added in references)

References:

Convolutional Neural Network: https://en.wikipedia.org/wiki/Convolutional_neural_network

HLS4ML:
Documentation: https://fastmachinelearning.org/hls4ml/concepts.html
Codebase: https://github.com/fastmachinelearning/hls4ml

Codebase Inspiration:
https://github.com/anmeza/cse237c_fa21_relu_optimizer

Implemented HLS4ML with MergedRelu:
https://github.com/gdpande97/hls4ml

Implemented codebase with working convert script:
https://github.com/gdpande97/cse237c_fa21_relu_optimizer