Optimizing Hardware Accelerator For CNN(Image Classification) on FPGA’s.

VL-901 PROJECT ELECTIVE

Presented By

PRAFFUL CHOUDHARY (&) ARCHIE MISHRA
MT2021523 MT2021508

WORK FLOW

Architecture Based on CNN implementation on Basys3 FPGA Board

(BASIC MODEL FOR PRACTICE)

HLS4ML

VS

Keras

Vs

TensorFlow

Vs

1-Bit INN

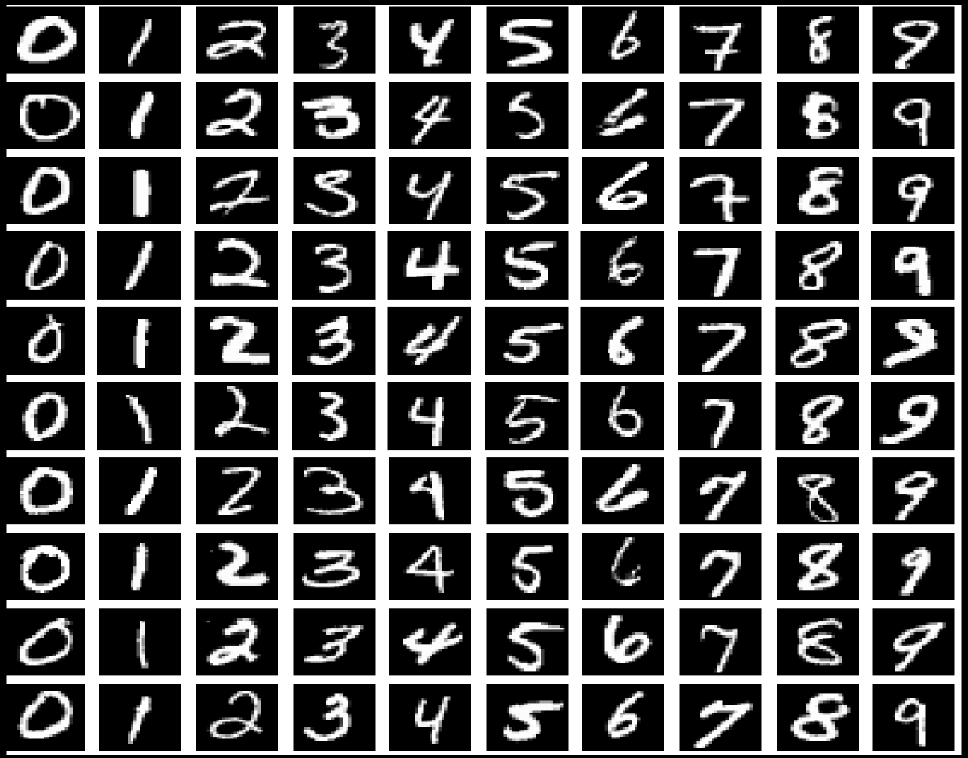
ACCURACY

(1-Bit) Integer Neural Network Architecture for Faster Calculations, Minimizing Hardware Utilization and with HLS pragmas for example By using Pipelining , Loop Unrolling, Resource sharing Methods to get the Optimized Hardware for Our proposed Design on HLS

Comparing the Hardware Utilization on FPGA of our model with existing Fixed Point CNN model which used Line-Buffers and Large Feature maps calculations.

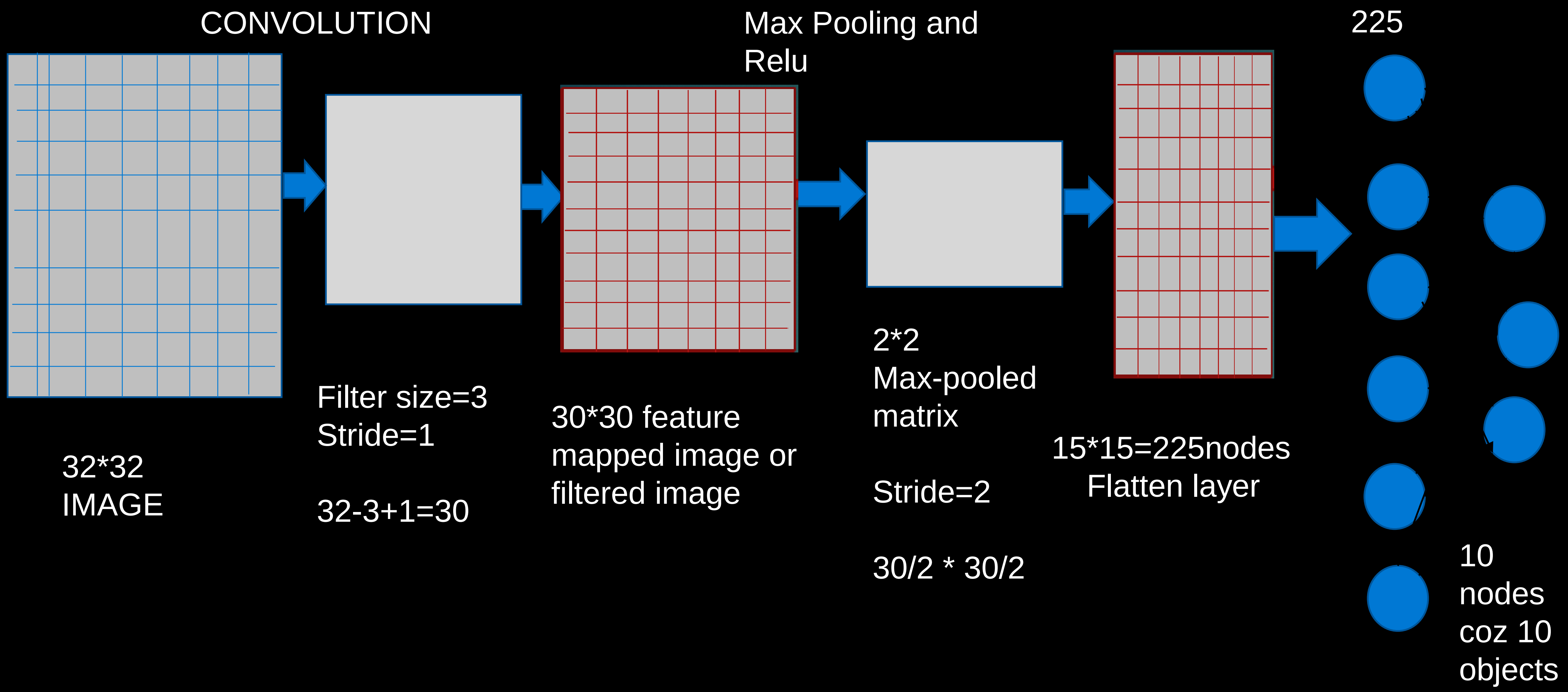
DATASET

MNIST: dataset of hand-written digits with 60k training images and 10k test images. Images are grayscale (one channel), 32x32 pixels. The number of classes is 10 (0 to 9 digits).

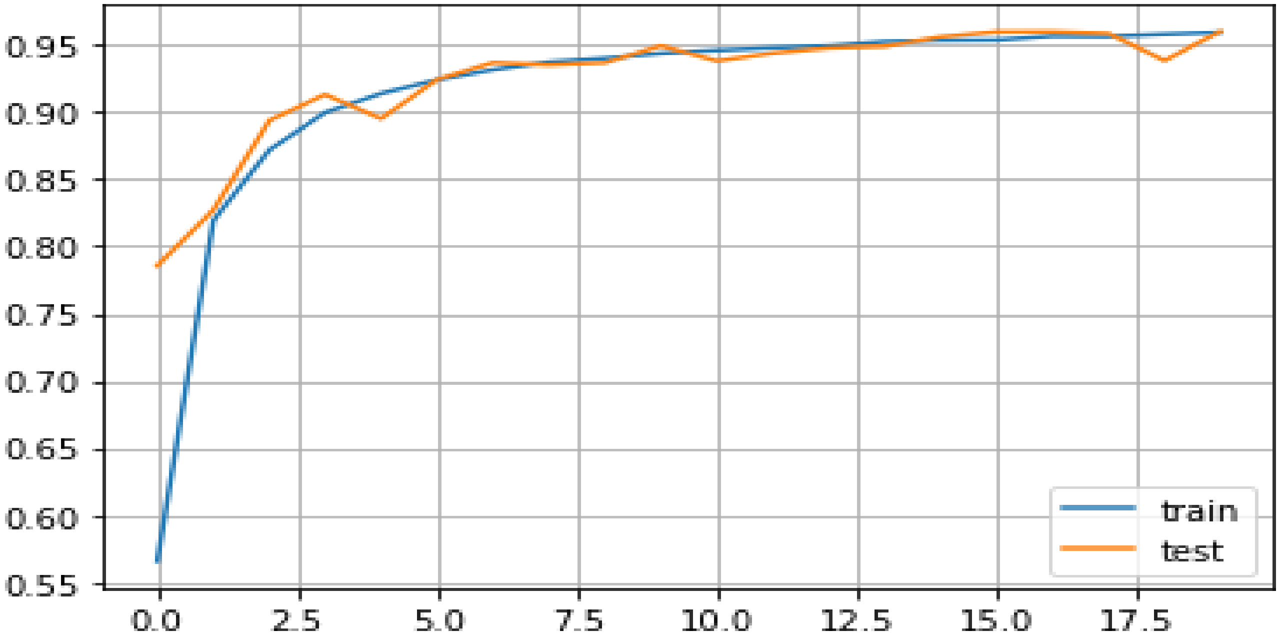


Reference: https://www.researchgate.net/figure/Example-images-from-the-MNIST-dataset_fig1_306056875

SIMPLE CNN ARCHITECTURE



```python
model = Sequential()


# conv1
model.add(BinaryConv2D(32, (3,3), name='conv1', input_shape=X_train.shape[1:]))
model.add(BatchNormalization(epsilon=epsilon, momentum=momentum, name='bn1'))
model.add(Activation(binary_tanh, name='act1'))

# conv2
model.add(BinaryConv2D(32, (3,3), name='conv2'))
model.add(BatchNormalization(epsilon=epsilon, momentum=momentum, name='bn2'))
model.add(Activation(binary_tanh, name='act2'))
model.add(MaxPooling2D(name='pool2'))

# conv3
model.add(BinaryConv2D(64, (3,3), name='conv3'))
model.add(BatchNormalization(epsilon=epsilon, momentum=momentum, name='bn3'))
model.add(Activation(binary_tanh, name='act3'))
model.add(MaxPooling2D(name='pool3'))
# conv4
model.add(BinaryConv2D(64, (3,3), name='conv4'))
model.add(BatchNormalization(epsilon=epsilon, momentum=momentum, name='bn4'))
model.add(Activation(binary_tanh, name='act4'))
model.add(MaxPooling2D(name='pool4'))


model.add(Flatten())
# dense1
model.add(BinaryDense(128, name='dense5'))
model.add(BatchNormalization(epsilon=epsilon, momentum=momentum, name='bn5'))
model.add(Activation(binary_tanh, name='act5'))
# dense2
model.add(BinaryDense(classes, name='dense6'))
model.add(BatchNormalization(epsilon=epsilon, momentum=momentum, name='bn6'))

if train =='softmax':
    model.add(Activation('softmax'))
```



Convolution

Batch Normalization

binary_tanh

LAYERS PARAMETERS

This is the test network in summary:

- Conv 3x3, 32 filters
- Conv 3x3, 32 filters
- Max Pooling 2x2
- Conv 3x3, 64 filters
- Conv 3x3, 64 filters
- Max Pooling 2x2
- Dense, 128 outputs
- Dense, 10 outputs (classes)

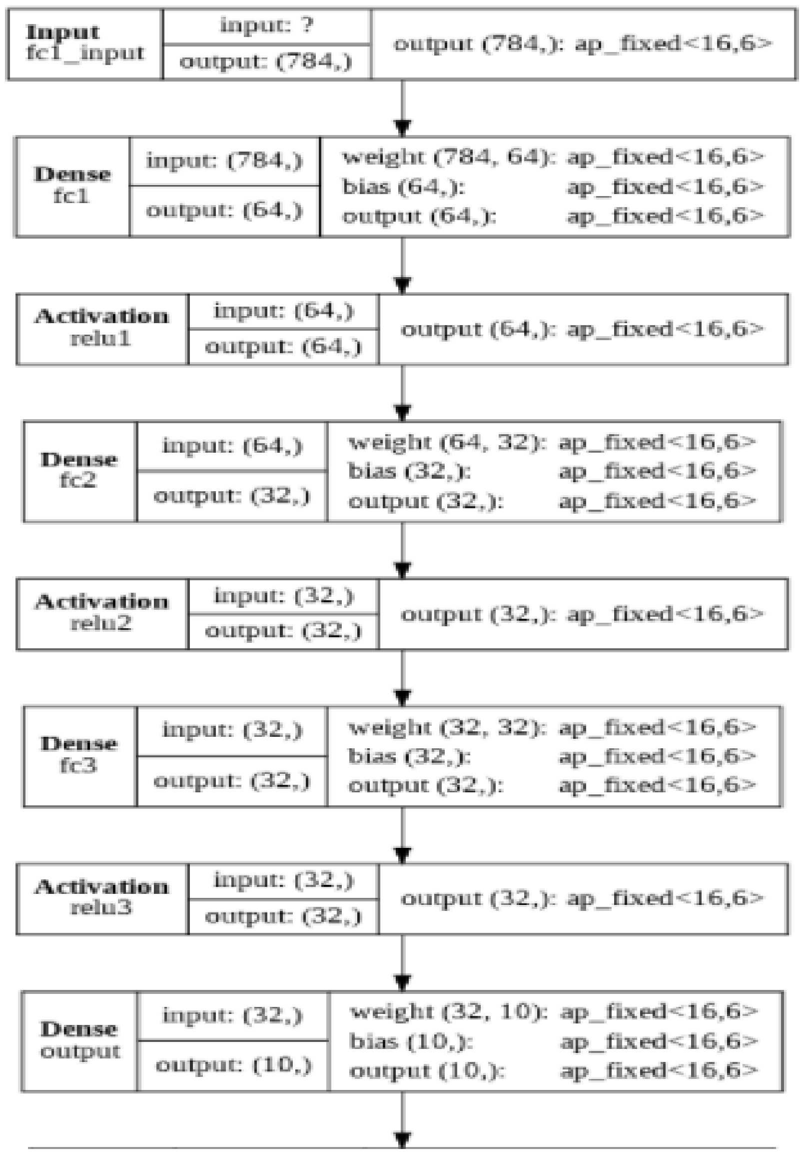
Total number of weights: 468k. About 85% of the total weights belong to the first fully connected layer

Layer (type)	Output Shape	Param #
conv1 (BinaryConv2D)	(None, 28, 28, 32)	288
bn1 (BatchNormalization)	(None, 28, 28, 32)	128
act1 (Activation)	(None, 28, 28, 32)	0
conv2 (BinaryConv2D)	(None, 28, 28, 32)	9216
bn2 (BatchNormalization)	(None, 28, 28, 32)	128
act2 (Activation)	(None, 28, 28, 32)	0
pool2 (MaxPooling2D)	(None, 14, 14, 32)	0
conv3 (BinaryConv2D)	(None, 14, 14, 64)	18432
bn3 (BatchNormalization)	(None, 14, 14, 64)	256
act3 (Activation)	(None, 14, 14, 64)	0
pool3 (MaxPooling2D)	(None, 7, 7, 64)	0
conv4 (BinaryConv2D)	(None, 7, 7, 64)	36864
bn4 (BatchNormalization)	(None, 7, 7, 64)	256
act4 (Activation)	(None, 7, 7, 64)	0
pool4 (MaxPooling2D)	(None, 3, 3, 64)	0
flatten_1 (Flatten)	(None, 576)	0
dense5 (BinaryDense)	(None, 128)	73728
bn5 (BatchNormalization)	(None, 128)	512
act5 (Activation)	(None, 128)	0
dense6 (BinaryDense)	(None, 10)	1280
bn6 (BatchNormalization)	(None, 10)	40
activation_1 (Activation)	(None, 10)	0

Total params: 141,128
Trainable params: 140,468
Non-trainable params: 660

HLS4ML MODEL

```python
hls4ml.utils.plot_model(hls_model, show_shapes=True, show_precision=True, to_file=None)
```




```python
import hls4ml
config = hls4ml.utils.config_from_keras_model(model, granularity='model')
print("-----------------------------------")
print("Configuration")

print("-----------------------------------")
hls_model = hls4ml.converters.convert_from_keras_model(model,
                                                       hls_config=config,
                                                       output_dir='model_1/hls4ml_prj',
                                                       part='xcu250-figd2104-2L-e')
```

```
Interpreting Sequential
Topology:
Layer name: fc1_input, layer type: Input
Layer name: fc1, layer type: Dense
  -> Activation (linear), layer name: fc1
Layer name: relu1, layer type: Activation
Layer name: fc2, layer type: Dense
  -> Activation (linear), layer name: fc2
Layer name: relu2, layer type: Activation
Layer name: fc3, layer type: Dense
  -> Activation (linear), layer name: fc3
Layer name: relu3, layer type: Activation
Layer name: output, layer type: Dense
  -> Activation (linear), layer name: output
Layer name: softmax, layer type: Activation
```

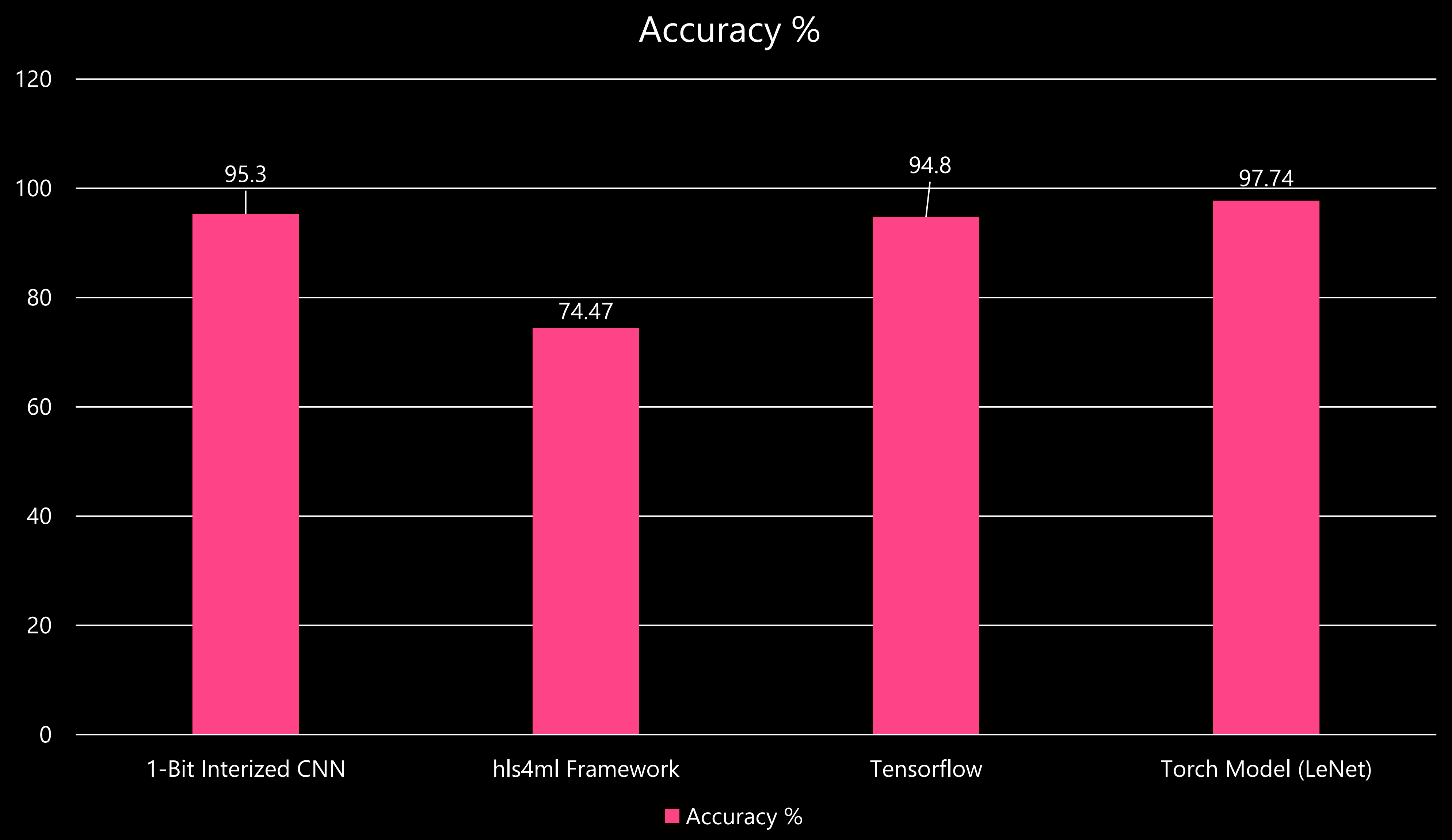
KERAS Vs HLS4ML

```
hls_model.compile()
X_test = np.ascontiguousarray(X_test)
y_hls = hls_model.predict(X_test)
```

```
Writing HLS project
Done
```

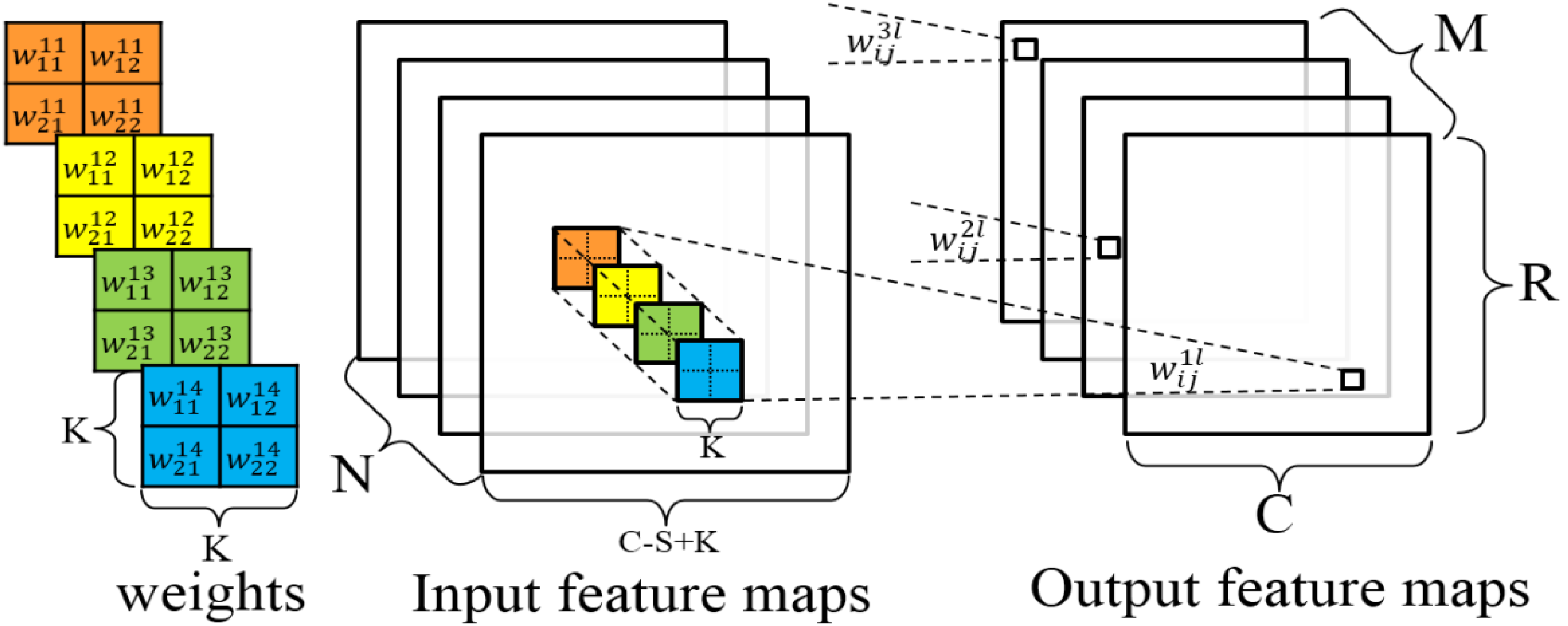
```
print("Keras  Accuracy: {}".format(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_keras, axis=1))))
print("hls4ml Accuracy: {}".format(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_hls, axis=1))))
```

```
Keras  Accuracy: 0.9482857142857143
hls4ml Accuracy: 0.7447857142857143
```



Accuracy %
120
100
80
60
40
20
0
95.3
74.47
94.8
97.74
1-Bit Interized CNN
hls4ml Framework
Tensorflow
Torch Model (LeNet)
Accuracy %

TILED CNN ARECHITECTURE FOR MULTIPLE IMAGES



Reference: J. -W. Chang and S. -J. Kang, "Optimizing FPGA-based convolutional neural networks accelerator for image super-resolution," *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 343-348.

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
   L:   output_fm[to][row][col] +=
          weights[to][ti][i][j]*
          input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

- The convolutional layer receives N feature maps as input.
- Each input feature map is convolved by a shifting window with a K ×K kernel to generate one pixel in one output feature map.
- The stride of the shifting window is S, which is normally smaller than K.
- A total of M output feature maps will form the set of input feature maps for the next convolutional layer

For this project we are going to use Xilinx’s High-Level Synthesis Tool, Vivado HLS, that allows us to write C++ code that will be translated into RTL code (VHDL, Verilog);

The major benefits of using Vivado HLS are its pragams, that allow the programmer to control resource usage, timing requirements and the architecture implementation .

The most important pragmas allow for example:

- To infer **pipeling** into our modules
- To **unroll loops** and achieve parallelism
- To **reshape arrays** in order to maximize memory usage
- To control input and output ports via standard protocols (**AXI, AXI-Stream**)

Inline, Dependence , Partitioning pragmas

By exploring different configurations of pragmas we can evaluate multiple architectures and find the optimal one for our purposes (e.g. trade-off timing/resources

BINARY CNN

```
for (int k=0; k < K2; k++)

c[k] = (input_image[k] == weights[k]);
            for(int k=0; k < Kernal_size; k++)
            {
                   ap_uint<K_BITS> storing_variable= 0;
                        for(int h=0; h < K; h++)
          if(c[k*Kernal_size + h] == true)
             storing_variable++;
        accumulator +=storing_variable;
             }
```

FIXED POINT CNN

```
for(int i=0; i < kernel_size; i++)
{
    fixed_point     storing_variable= 0;
            for(int j=0; j < Kernel_size; j++)
                        if(weights[i*Kernel_size +j] == 0)
                           storing_variable-= input_image[i][j];
                        else
                           storing_variable+= input_image[i][j];
              accumulate += storing_variable ;
}
```

SLIDING WINDOW & LINE BUFFER

```
// shift columns of processing window
    for(int i = 0; i < Kernel_size; i++)
        for(int j = 0; j < Kernel_size-1; j++)
            sliding_window[i][j] = sliding_window[i][j+1];

//line_buffer
    for(int i = 0; i < Kernel_size - 1; i++)
        sliding_window[i][Kernel_size - 1] = line_buffer[i][col];


//shift row of line buffer
    for(int i = 0; i < Kernel_size-2; i++)
        line_buffer[i][col] = line_buffer[i+1][col];

//storing inputs
        input temp = input_stream.read();
        sliding_window[Kernel_size-1][Kernel_size-1]  = temp;
        line_buffer[Kernel_size-2][col] = temp;
```

Binary Convolution Utilization Report

IMG_H=32, IMG_W=32

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	5091	-
FIFO	-	-	-	-	-
Instance	0	-	37	87	-
Memory	10	-	18	5	0
Multiplexer	-	-	-	437	-
Register	-	-	1298	-	-
Total	10	0	1353	5620	0
Available	280	220	106400	53200	0
Utilization (%)	3	0	1	10	0

IMG_H=64, IMG_W=64

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	17298	-
FIFO	-	-	-	-	-
Instance	0	-	37	187	-
Memory	41	-	0	0	0
Multiplexer	-	-	-	797	-
Register	-	-	4345	-	-
Total	41	0	4382	18282	0
Available	280	220	106400	53200	0
Utilization (%)	14	0	4	34	0

Max Pooling Utilization Report

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	217	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	0	0	0
Multiplexer	-	-	-	141	-
Register	-	-	275	-	-
Total	1	0	275	358	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	~0	~0	0

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	505	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	4	-	0	0	0
Multiplexer	-	-	-	141	-
Register	-	-	947	-	-
Total	4	0	947	646	0
Available	280	220	106400	53200	0
Utilization (%)	1	0	~0	1	0

Dense Layer Report (1st Implementation)

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6884	-
FIFO	-	-	-	-	-
Instance	0	-	37	1404	-
Memory	9	-	0	0	0
Multiplexer	-	-	-	2573	-
Register	-	-	2903	-	-
Total	9	0	2940	10861	0
Available	280	220	106400	53200	0
Utilization (%)	3	0	2	20	0

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	3426	-
FIFO	-	-	-	-	-
Instance	0	-	37	715	-
Memory	5	-	0	0	0
Multiplexer	-	-	-	1421	-
Register	-	-	1486	-	-
Total	5	0	1523	5562	0
Available	280	220	106400	53200	0
Utilization (%)	1	0	1	10	0

INP_DIM = 1024

Padding

INP_DIM = 256

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	201	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	171	-
Register	-	-	196	-	-
Total	0	0	196	372	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	201	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	171	-
Register	-	-	187	-	-
Total	0	0	187	372	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0

Resource Utilization Report of the LENET 5 architecture using BNN

	B-CNN	Max Pooling	Padding	B-CNN	Padding	B-CNN	Padding	Max Pooling	Fully Connected	Fully Connected
Input Parameters	Image size 32x32 K = 5 FAN_IN = 32 FAN_OUT = 32 P_IN = 32 P_OUT = 1	Image size 28x28 K=2 FMAPS=32	Image size 14x14 PAD=1 FMAPS=32	Image size 15x15 K = 3 FAN_IN = 32 FAN_OUT = 64 P_IN = 32 P_OUT = 1	Image size 13x13 PAD=2 FMAPS=32	Image size 15x15 K = 3 FAN_IN = 64 FAN_OUT = 64 P_IN = 32 P_OUT = 1	Image size 13x13 PAD=1 FMAPS=32	Image size 14x14 K=2 FMAPS=32	INP_DIM 3136 OUT_DIM 128 P_OUT =32	INP_DIM 128 OUT_DIM 10 P_OUT =32
BRAM	41	4	0	10	0	10	0	1	9	5
FF	4321	947	196	1071	191	1179	187	275	2940	1823
FF %	4	0.89	0.184	1	0.179	1	0.175	0.258	2	1
LUT	18293	646	372	1826	345	3455	327	358	10861	5562
LUT %	34	1.21	0.7	3	0.659	6	0.614	0.67	20	10

Fixed Convolution Utilization Report

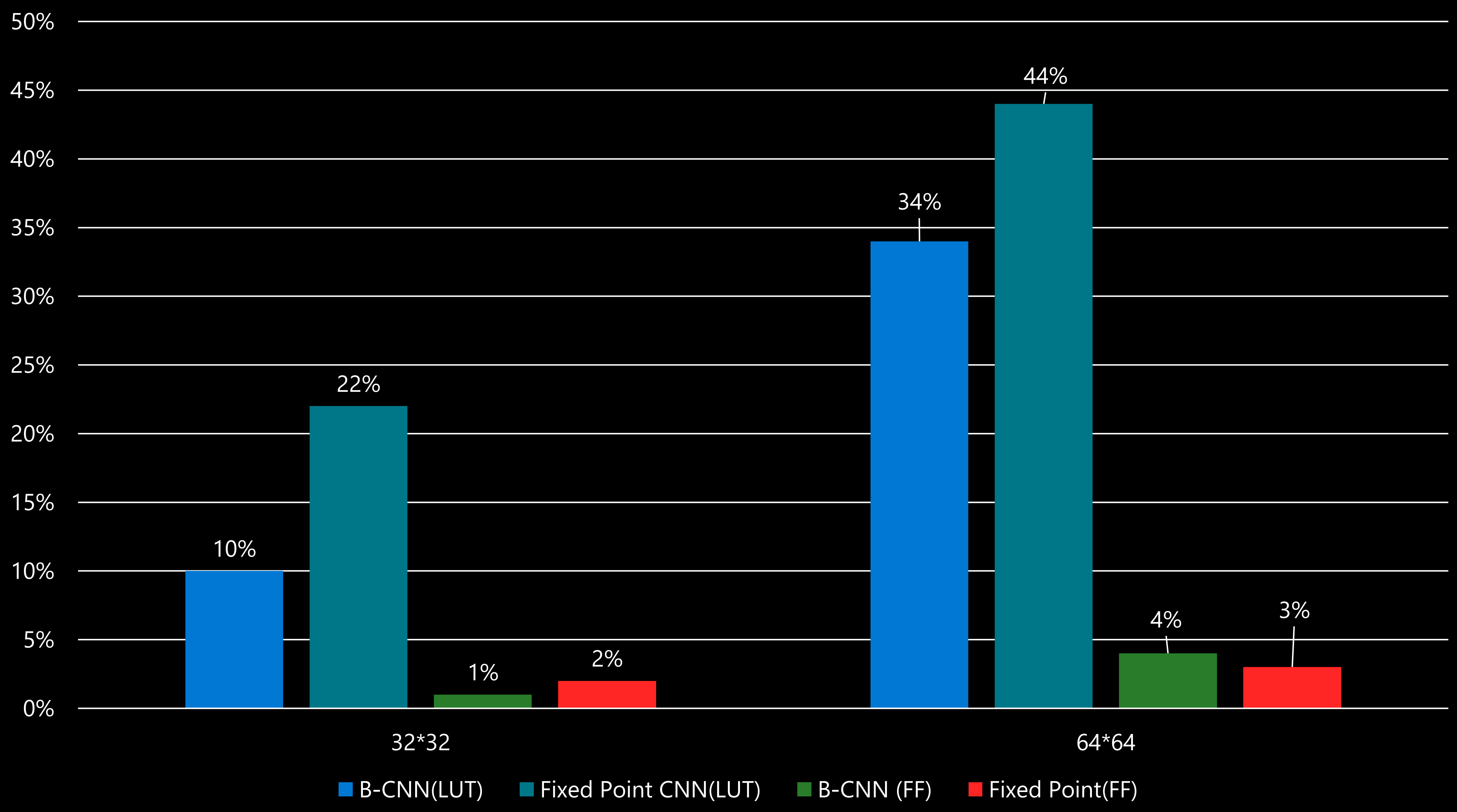
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	11861	-
FIFO	-	-	-	-	-
Instance	0	-	37	42	-
Memory	0	-	52	14	0
Multiplexer	-	-	-	203	-
Register	0	-	2222	64	-
Total	0	0	2311	12184	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	2	22	0

IMG_H=32, IMG_W=32

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	23451	-
FIFO	-	-	-	-	-
Instance	0	-	37	42	-
Memory	0	-	52	26	0
Multiplexer	-	-	-	203	-
Register	0	-	3992	64	-
Total	0	0	4081	23786	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	3	44	0

IMG_H=64, IMG_W=64

BINARY CNN vs FIXED POINT CNN



TIME PLAN

Months / Tasks	Jan-1 22-Jan-5 22	Jan- 5,22-Jan -10 22	Jan-11 22– Jan-12 22	Jan 12, 22– Jan 18 22	Jan19 ,22 - Feb 22	Feb-10 ,22 – Feb-19 22	Feb 20 ,22– Mar 22	Mar 22- Mar 22	Apr 22- May 22
	1	2	3	4	5	6	7	8	9
Paper Selection									
Study and Review of Literature Survey									
Problem Finding									
Objective 1 Completion **(Accuracy Comparison (Tensor Flow, Keras, HLS4ML)**									
Objective 2 Completion **(HLS4ML OUTPUT)**									
Objective 3 Completion **(Binary CNN)**									
Objective 4 Completion **Full LENET ARCHITECTURE**									
Objective 5 Completion **Fixed Point CNN Vs Binary CNN**									
Documentation									