

Simulating a DL/AI accelerator in Verilog

(CNN Algorithm)

Course Project

<u>Course :-</u>	EEL3090 (Embedded Systems)
<u>Project code :-</u>	ESE3
<u>Course Instructor :-</u>	Dr. Binod Kumar
<u>Group Number :-</u>	A4
<u>Group Members :-</u>	Patel Aaditya Shvetang (B20EE040) Pratham Chaurasia (B20EE041)

Theory

Introduction:

- Deep learning models called convolutional neural networks (CNNs) are employed for computer vision tasks including object recognition and picture categorization.
- They are created to automatically learn characteristics from photographs and are inspired by how the human brain processes visual information.
- The architecture of CNNs, their applications, as well as their benefits and drawbacks, will all be covered in this study.

Architecture:

- Convolutional, pooling, and fully linked layers are among the layers that make up a standard CNN.
- The network learns different filters in the convolutional layers to extract information from the input picture.
- The spatial size of the feature maps is decreased using the pooling layers.
- The input picture is finally classified using the retrieved features using the fully linked layers.
- CNNs may have their architectures altered by adding or deleting layers and altering their hyperparameters.

Applications:

- It includes determining if a certain item is present in an image or recognising handwritten numbers.
- CNNs are frequently employed in image classification jobs.
- They are also employed in jobs requiring object detection, such as finding people in pictures or recognising cars in video streams.
- CNNs have been effectively used in tasks related to natural language processing, such as text sentiment analysis, and medical image analysis, such as the detection of cancer in mammography pictures.

Advantages:

- Compared to conventional machine learning models, CNNs provide a number of benefits.
- They don't require human feature engineering because they can automatically learn features from photos.
- They are useful for big data applications because they can manage massive volumes of data.
- Additionally, CNNs generalize well to fresh data, making them appropriate for use in practical applications.

Limitations:

- CNNs also have a number of them. For training, they need a lot of data, which may be expensive and time-consuming to gather.
- They may also be expensive to compute, particularly for large-scale applications.
- Finally, if the training data is not representative of the target population, CNNs may be vulnerable to overfitting.

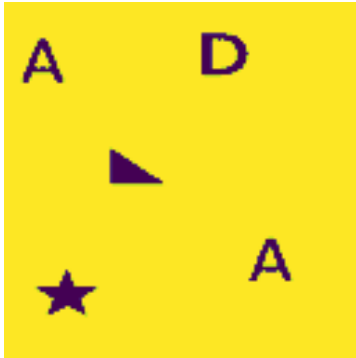
Conclusion:

- In **conclusion**, CNNs are a potent deep learning model that can carry out a variety of computer vision tasks and learn features from pictures.
- They have been effectively used in a variety of domains, including natural language processing and the interpretation of medical images.
- They do, however, have certain drawbacks, such as the requirement for substantial data and processing resources.
- In general, CNNs are a useful tool for resolving challenging image processing issues and will continue to be a key area of machine learning research.

Procedure

1. Converting the given image into a binary text file:-

- Suppose we have an image and pattern, shown below:-



Image



Pattern

- Now, we will use Python to convert the image into text format using the code:-

```
[25] import numpy as np
import pandas as pd
import cv2
import sys
import io
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
```

```
import struct

def int_to_binary(num):
    # Convert the integer to a binary string with '0b' prefix
    binary_str = bin(num)
    # Remove the '0b' prefix and pad the string with zeroes to length 4
    binary_str = binary_str[2:].zfill(4)
    # Return only the last 4 bits
    return binary_str[-4:]
```

```
img = cv2.imread('image.png', 2)
ret, bw_img = cv2.threshold (img, 127, 255, cv2. THRESH_BINARY)
bw = cv2.threshold (img, 127, 255, cv2.THRESH_BINARY)

A = bw[1].reshape([128, 128])

import matplotlib.image
matplotlib.image.imsave('image.png', A)

X=[]
for i in A:
    for j in i:
        X.append(j)

bin_image=[]
img = X
for i in range(len(img)):
    num=img[i]
    binary = int_to_binary(num)
    # temp.append(binary)
    bin_image.append(binary)
    # bin_image.append(binary)

df=pd.DataFrame (bin_image, index=None)
bin_image=np.array(bin_image)
df.to_csv("bin_image.txt",sep=" ",header=None, index=None)
```

```

img = cv2.imread('ptn.png', 2)
ret, bw_img = cv2.threshold (img, 127, 255, cv2. THRESH_BINARY)
bw = cv2.threshold (img, 127, 255, cv2.THRESH_BINARY)

A = bw[1].reshape([8, 8])

import matplotlib.image
matplotlib.image.imsave('ptn.png', A)

X=[]
for i in A:
    for j in i:
        X.append(j)

bin_image=[]
img = X
for i in range(len(img)):
    num=img[i]
    binary = int_to_binary(num)
    # temp.append(binary)
    bin_image.append(binary)
    # bin_image.append(binary)

df=pd.DataFrame (bin_image, index=None)
bin_image=np.array(bin_image)
df.to_csv("bin_ptn.txt",sep=" ",header=None, index=None)

```

- This code snippet helps in the required task.
- The 4 bit binary for image pixels is saved in “bin_image.txt” and “bin_ptn.txt”.

2. Matlab Implementation

The CNN algorithm is implemented on MATLAB to test for results of convolution, maxpooling and thresholding.

a. Convolution of pattern with Laplacian Filter

```
clc;

W = [0 -1 0; -1 4 -1; 0 -1 0];
b = 0;

ptn = imread('ptn.png');
gray_ptn = rgb2Gray2Mat(ptn);
W = [0 -1 0; -1 4 -1; 0 -1 0];
b = 0;
%Convolution Operation
conv_ptn_1 = cnn2dConvolution(gray_ptn,W,b);
imshow(conv_ptn_1);
```

b. Convolution of test image with Laplacian Filter

```
W = [0 -1 0; -1 4 -1; 0 -1 0];
b = 0;
|
image = imread('image.png');
gray_image = rgb2gray(image);
%Convolution Operation
conv_image_1 = cnn2dConvolution(gray_image,W,b);
imshow(conv_image_1);
```

c. Convolution of the two convolved images

```
conv_2 = cnn2dConvolution(conv_image_1,conv_ptn_1,b);
```

d. Applying Max Pooling

```
poolSize = [2, 2];
image = conv_2;
% Determine the size of the output image
outputSize = floor(size(image) ./ poolSize);

% Preallocate the output image
maxPooledImage = zeros(outputSize);

% Perform max pooling on the image
for i = 1:outputSize(1)
    for j = 1:outputSize(2)
        subImage = image((i-1)*poolSize(1)+1:i*poolSize(1), ...
                           (j-1)*poolSize(2)+1:j*poolSize(2));
        maxPooledImage(i,j) = max(subImage(:));
    end
end
% Display the original and pooled images side by side
subplot(1, 2, 1);
imshow(image);
title('Original Image');

subplot(1, 2, 2);
imshow(maxPooledImage);
title('Max Pooled Image');
```


e. Applying thresholding and counting the no. of occurrences

```
% Apply thresholding
img_thresh = maxPooledImage > 0.5;
pattern = conv_ptn_1;

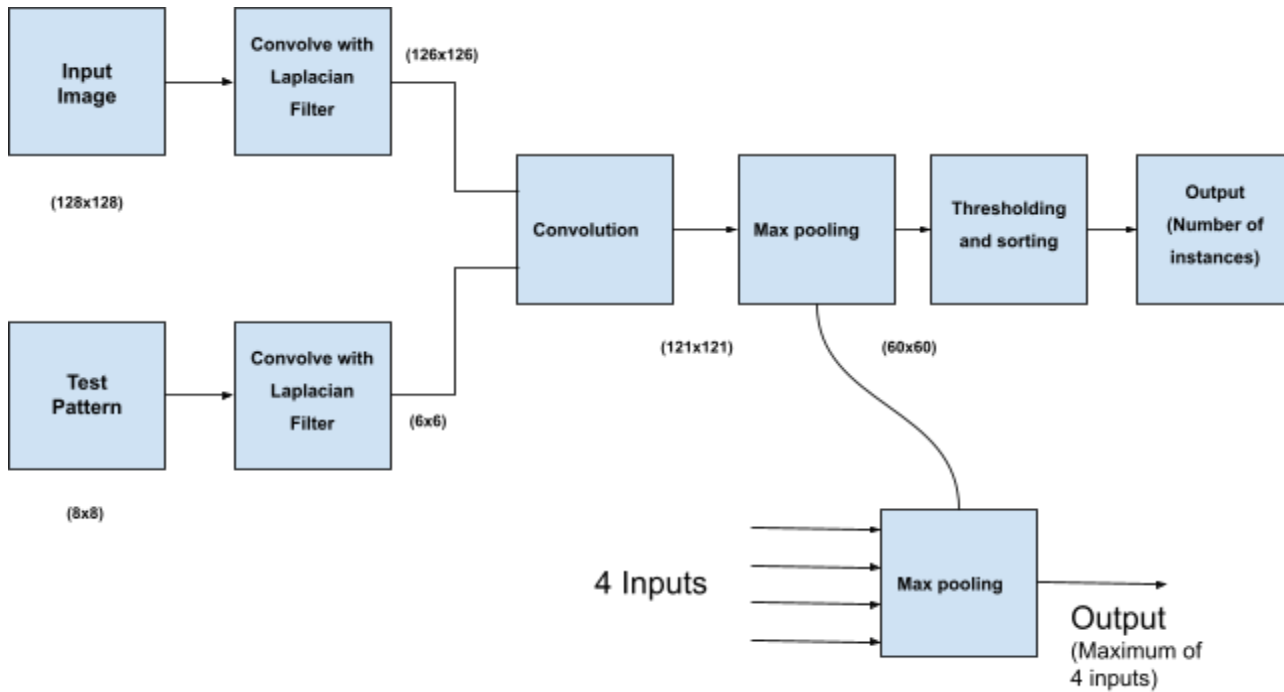
% Calculate the cross-correlation between the pattern and the thresholded image
corr = normxcorr2(pattern, img_thresh);
[row, col] = find(abs(corr) > 0.263);

% Display the thresholded image and the pattern with boxes around the matched regions
figure;
imshow(img_thresh);
hold on;
for i = 1:length(row)
    rectangle('Position', [col(i) - size(pattern, 2)/2, row(i) - size(pattern, 1)/2, size(pattern, 2), ...
        size(pattern, 1)], 'EdgeColor', 'r', 'LineWidth', 2);
end
title('Thresholded Image with Matched Regions Highlighted');

figure;
imshow(pattern);
title('Pattern =>');
disp("Pattern detected")
disp(length(row));
```

3. CNN in Verilog

- We will then have to create a code which can execute CNN and make a model for the required task.
- This code will include numerous parts as shown below:-



○ **Laplacian Filter:-**

- The Laplacian filter is basically a matrix with which the image is convolved to extract the required features.
- This is done for both the input image as well as the test pattern.

○ **Convolution:-**

- In this step, the output of both the filters are convolved with one another.
- Implemented as **ConvolutionStage_1.v** and **ConvolutionStage_2.v**
- Consists of **adderstages** for adding up to give the final output

- **Max Pooling:-**

- In this step, the entire matrix containing the convolved output is pooled as per the parameters and the maximum value from each pool is chosen and a new matrix is created.
- For instance, in the diagram above, the max pooling is done in a 4:1 fashion.
- Implemented as **maxPooling.v**

- **Thresholding and Sorting:-**

Now, this new matrix is applied with some constraints such as thresholding each element's value and then all these are sorted to get the desired output.

- Now, this step is for using the created CNN model to count the number of instances the test pattern is detected.

- **Loading Data:-**

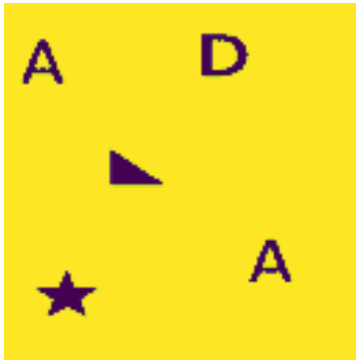
This binary text file is then uploaded to the memory of the Verilog module and then the code is tested.

Note:- The steps mentioned above can be repeated multiple times in a pair form too, but keeping in mind the time it will take to do this task.

Results

1. Matlab Results

a. Input image and test pattern

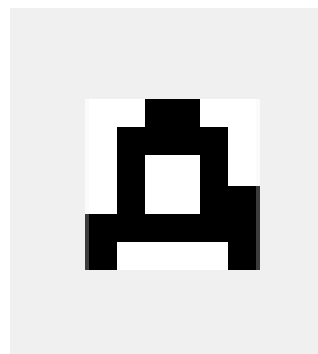
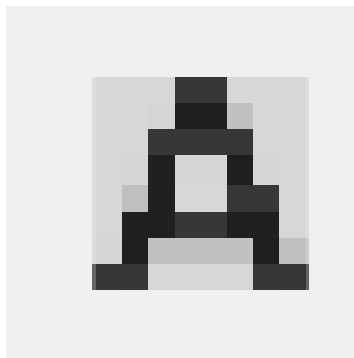


Image

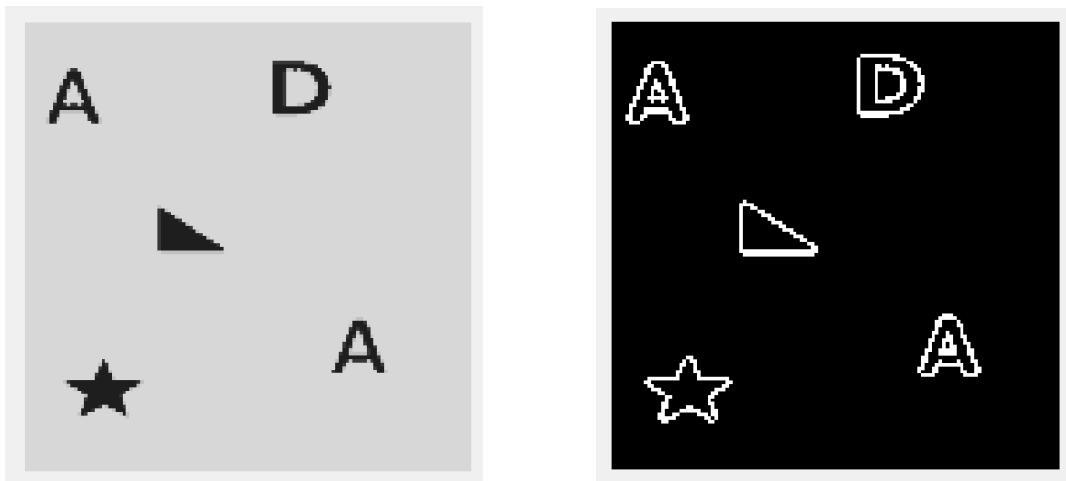


Pattern

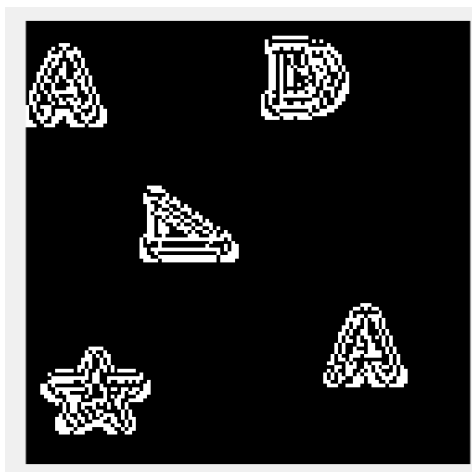
b. Convolution of pattern with Laplacian filter



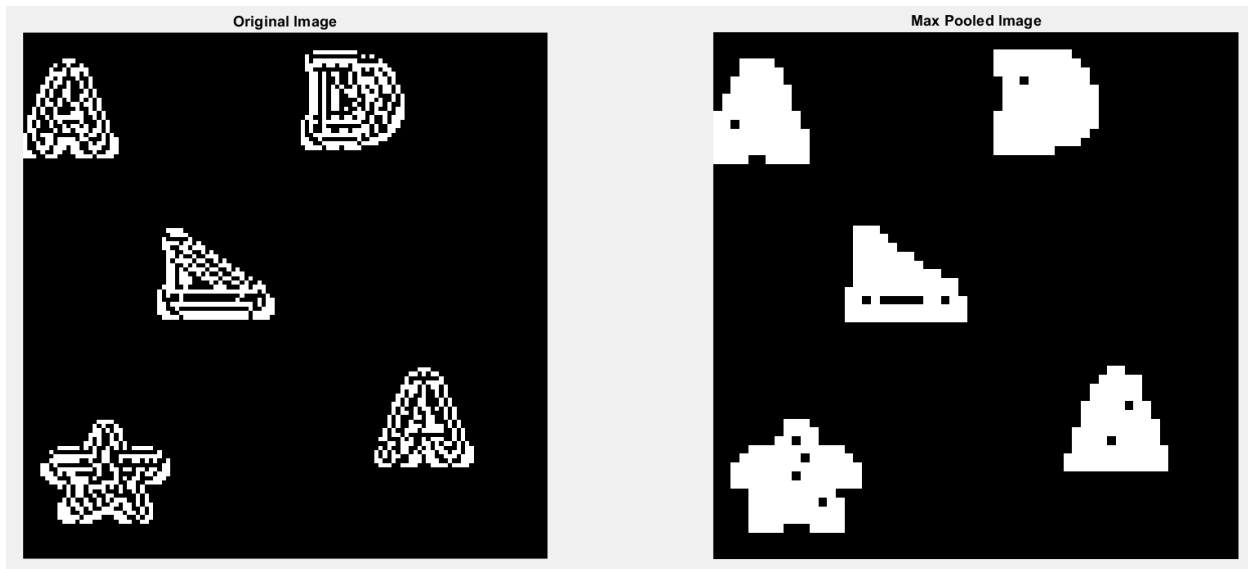
c. Convolution of image with Laplacian filter



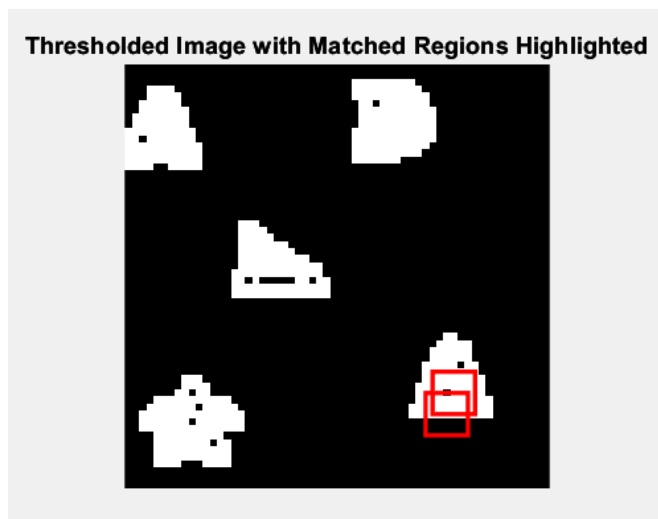
d. Convolution of both the images



e. Max Pooling

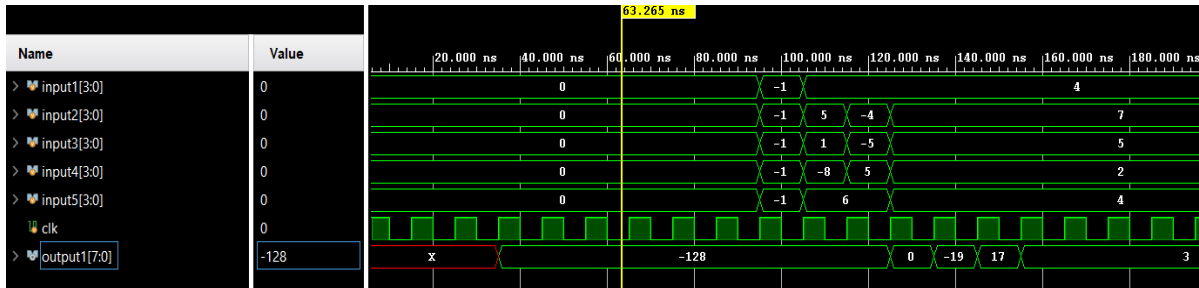


f. Thresholding and pattern detection

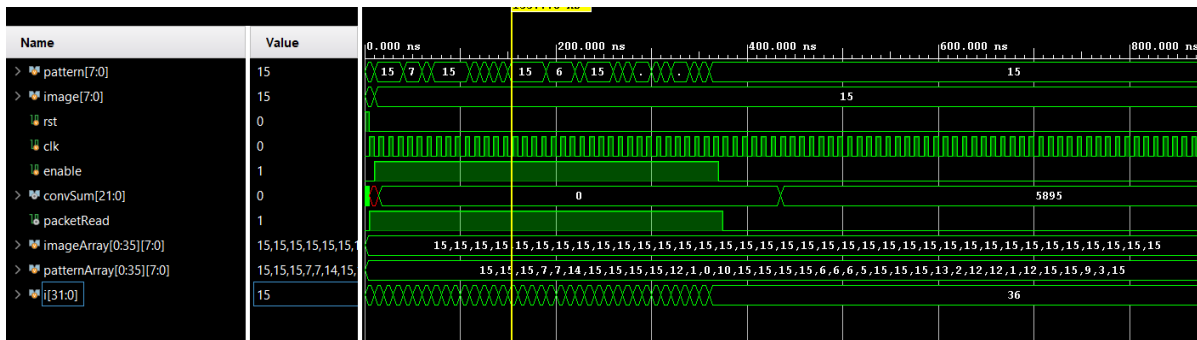


Verilog Results

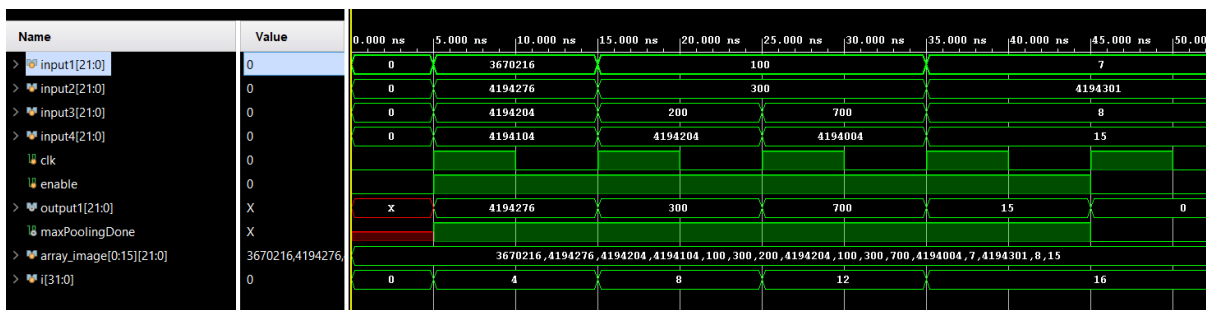
a. Convolution Stage 1



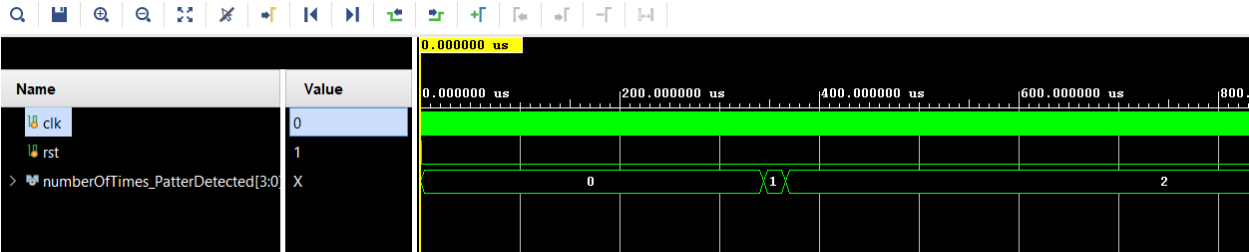
b. Convolution Stage 2



c. Max Pooling

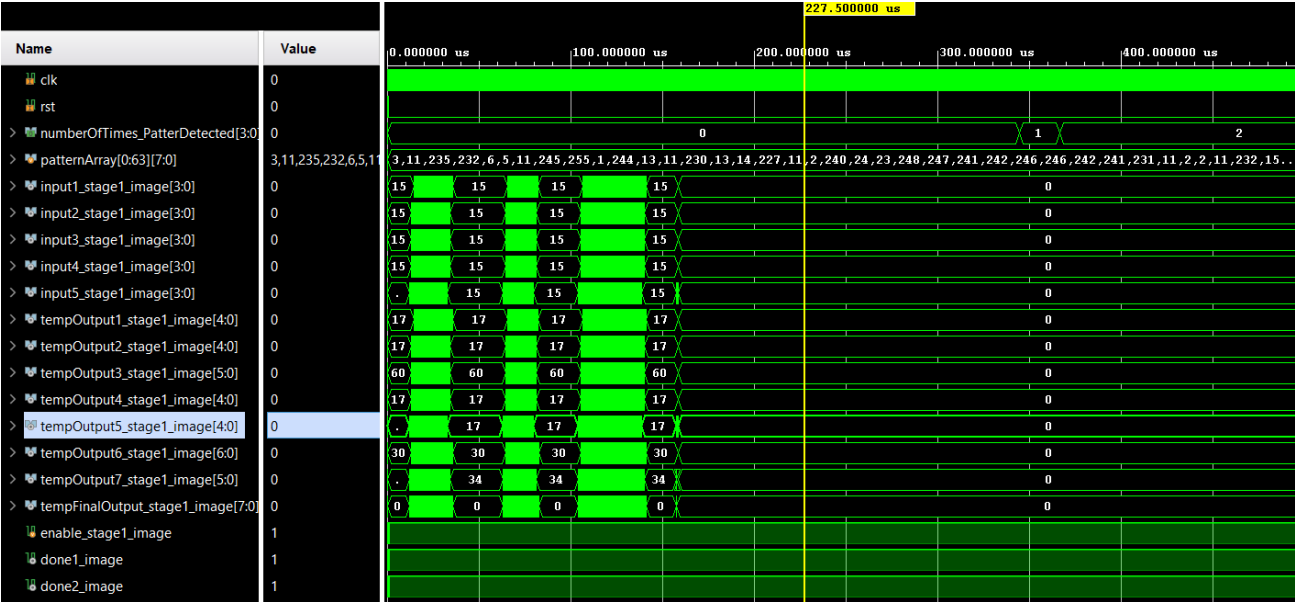


d. Top level output



The final output '2' is received after **3687** cycles

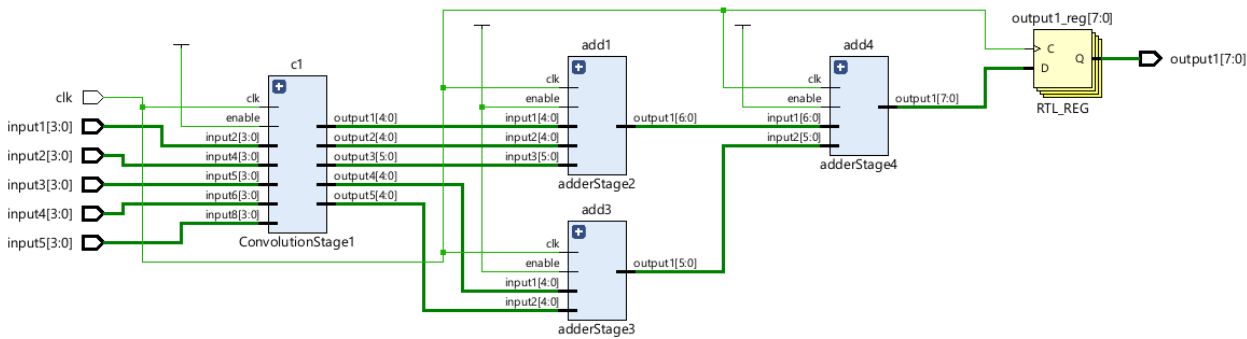
e. Sub module internal signals



RTL Analysis

1. Convolution stage 1

a. Schematic



b. Implementation

i. Register analysis

Name	Constraints	Status	Total Power	Failed Routes	Methodology	...	LUT	FF	BRAM	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!					28	53	0	0	0
✓ impl_1	constrs_1	route_design Complete!	8.630	0	53 CW		28	53	0	0	0

Detailed RTL Component Info :

+---Adders :

2 Input	8 Bit	Adders := 1
3 Input	7 Bit	Adders := 1
2 Input	6 Bit	Adders := 1
2 Input	5 Bit	Adders := 4

+---Registers :

8 Bit	Registers := 2
7 Bit	Registers := 1
6 Bit	Registers := 2
5 Bit	Registers := 4

ii. Power analysis

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 8.63 W (Junction temp exceeded!)

Design Power Budget: Not Specified

Power Budget Margin: N/A

Junction Temperature: 124.5°C

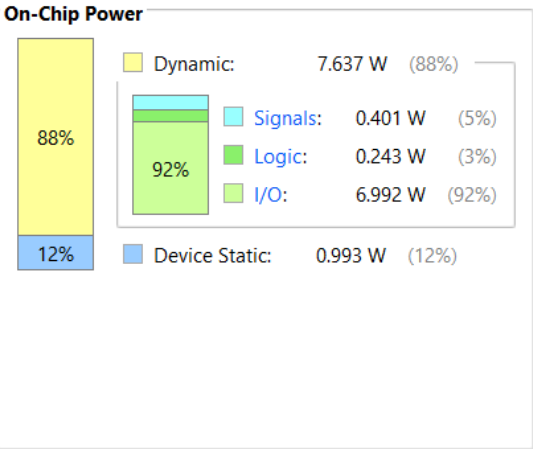
Thermal Margin: -39.5°C (-2.7 W)

Effective θ_{JA} : 11.5°C/W

Power supplied to off-chip devices: 0 W

Confidence level: Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

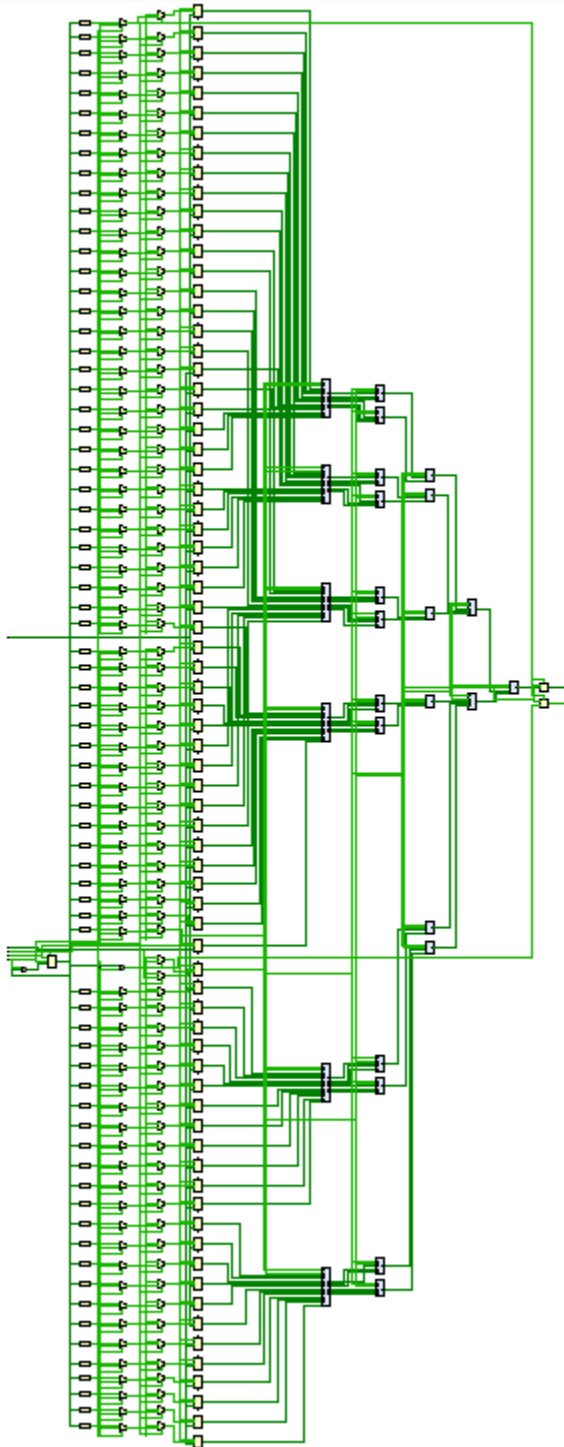


iii. Noise

I/O Bank Details								
Name	Port	I/O Std	Vcco	Slew	Drive Strength (...)	Off-Chip Termina...	Remaining Margin...	Notes
I/O Bank 0 (0)								
I/O Bank 13 (8)		LVCMOS18	1.80	SLOW	12	FP_VTT_50		
Y8	output1[0]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	20.18	
Y9	output1[1]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	33.75	
Y6	output1[2]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	23.45	
Y7	output1[3]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	20.00	
U10	output1[4]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	20.00	
T9	output1[5]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	37.92	
V7	output1[6]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	47.80	
V5	output1[7]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	90.46	
I/O Bank 34 (0)								
I/O Bank 35 (0)								

2. Convolution stage 2

a. Schematic



b. Implementation

i. Register Analysis

Name	Total Power	Failed R... ¹	Methodology	LUT	FF	BRAM	URAM	DSP
✓ synth_1				2628	1576	0	0	0
✓ impl_1	22.166	0	1000 CW, 432 Warn	2625	1576	0	0	0

Detailed RTL Component Info :

----Adders :

2 Input	22 Bit	Adders := 1
3 Input	21 Bit	Adders := 2
2 Input	19 Bit	Adders := 6
3 Input	18 Bit	Adders := 12
2 Input	6 Bit	Adders := 1

----Registers :

22 Bit	Registers := 2
21 Bit	Registers := 2
19 Bit	Registers := 6
18 Bit	Registers := 12
8 Bit	Registers := 72
6 Bit	Registers := 1
1 Bit	Registers := 2

----Muxes :

2 Input	1 Bit	Muxes := 36
---------	-------	-------------

ii. Power Analysis

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: **22.166 W (Junction temp exceeded!)**

Design Power Budget: **Not Specified**

Power Budget Margin: **N/A**

Junction Temperature: **125.0°C**

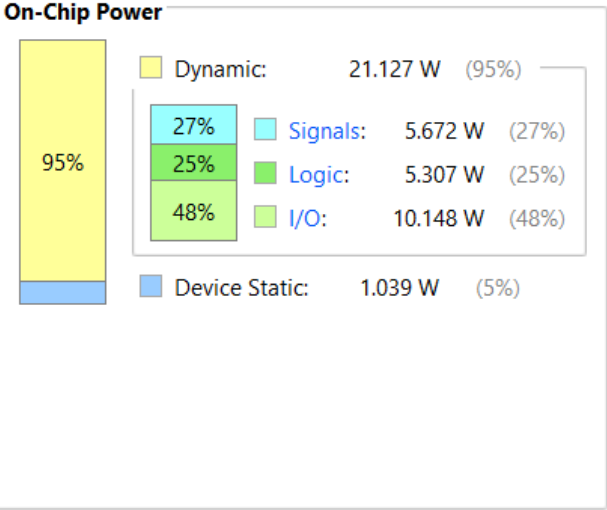
Thermal Margin: **-195.6°C (-16.2 W)**

Effective θ_{JA} : **11.5°C/W**

Power supplied to off-chip devices: **0 W**

Confidence level: **Low**

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

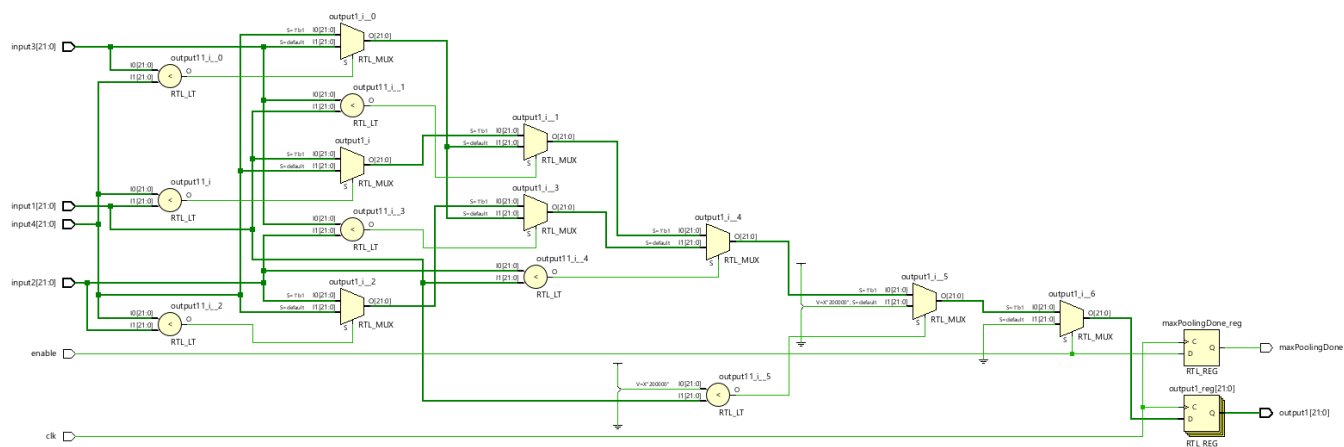


iii. Noise

I/O Bank Details								
Name	Port	I/O Std	Vcco	Slew	Drive Strength (...)	Off-Chip Termina...	Remaining Margin...	Notes
V6	convSum[1]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-31.30	
V10	convSum[2]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-23.75	
V11	convSum[3]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-44.18	
Y13	convSum[4]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-66.86	
Y12	convSum[5]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-90.78	
U5	convSum[6]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	67.77	
T5	convSum[7]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	2.72	
Y11	convSum[8]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-46.31	
W11	convSum[9]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-44.11	
U8	convSum[10]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-22.94	
U9	convSum[11]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-54.55	
W9	convSum[12]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-81.35	
W10	convSum[13]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-55.46	
W8	convSum[14]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-74.29	
V8	convSum[15]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-85.15	
Y8	convSum[16]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-21.81	
Y9	convSum[17]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-38.03	
Y6	convSum[18]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-86.00	
Y7	convSum[19]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-70.13	
U10	convSum[20]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-59.74	
T9	convSum[21]	LVCMOS18	1.80	SLOW	12	FP_VTT_50	-5.85	
V7	packetRead	LVCMOS18	1.80	SLOW	12	FP_VTT_50	7.68	
I/O Bank 34 (0)								

3. MaxPooling

a. Schematic



b. Implementation

i. Register Analysis

Name	Total Power	Failed Routes	Methodology	LUT	FF	BRAM	URAM	DSP	S
✓ synth_1				144	23	0	0	0	
✓ impl_1	16.443	0	23 CW	144	23	0	0	0	

```
Detailed RTL Component Info :
+---Registers :
                22 Bit    Registers := 1
                1 Bit     Registers := 1
+---Muxes :
        2 Input    22 Bit    Muxes := 8
```

ii. Power Analysis

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 16.443 W (Junction temp exceeded!)

Design Power Budget: Not Specified

Power Budget Margin: N/A

Junction Temperature: 125.0°C

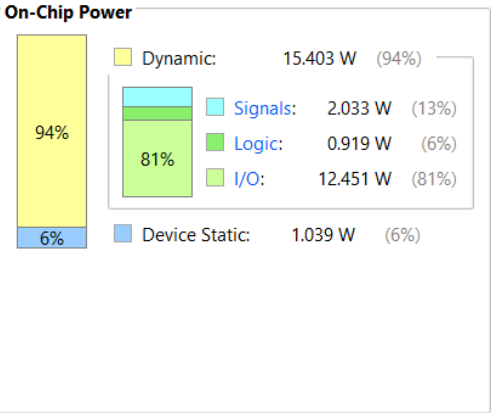
Thermal Margin: -129.6°C (-10.5 W)

Effective θ_{JA} : 11.5°C/W


























Power supplied to off-chip devices: 0 W

Confidence level: Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity



iii. Noise

I/O Bank Details								
Name	Port	I/O Std	Vcco	Slew	Drive Strength (...)	Off-Chip Termina...	Remaining Margin...	Noi
 I/O Bank 0 (0)								
  I/O Bank 13 (22)		LVC MOS18	1.80	SLOW	12	FP_VTT_50		
 W6	output1[0]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-32.71	
 V6	output1[1]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-29.51	
 V10	output1[2]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-21.97	
 V11	output1[3]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-42.94	
 Y13	output1[4]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-65.34	
 Y12	output1[5]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-90.17	
 U5	output1[6]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	68.37	
 T5	output1[7]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	3.74	
 Y11	output1[8]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-45.86	
 W11	output1[9]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-43.35	
 U8	output1[10]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-21.52	
 U9	output1[11]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-52.58	
 W9	output1[12]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-78.46	
 W10	output1[13]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-48.08	
 W8	output1[14]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-66.43	
 V8	output1[15]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-79.03	
 Y8	output1[16]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	3.92	
 Y9	output1[17]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-27.18	
 Y6	output1[18]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-83.45	
 Y7	output1[19]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-68.48	
 U10	output1[20]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	-56.48	
 T9	output1[21]	LVC MOS18	1.80	SLOW	12	FP_VTT_50	0.39	

Conclusions

- The results shown above verifies that using CNN we can detect the number of times a particular pattern is occurring in a given image.
- We can also verify results from Matlab using Laplacian filter matlab code.
- Comparison of outputs with results MATLAB implementation and RTL implementation (with example of convolution of Pattern with Laplacian Filter) shows the function of RTL blocks in Verilog is matching.
- Tuning of thresholds of correlations can lead to a better performance of the algorithm.

References

- <https://colab.research.google.com/drive/1q5-MZjkux5TTn2gIuui3iI04gVSPyWP2#scrollTo=3Keo88WCzG8G>
- https://drive.google.com/drive/folders/1mbTyqJxzbHl-QZqhAkSy8xY7CeMnomOT?usp=share_link
- [‘An FPGA implementation for Convolutional Neural Network’](#) by Guilherme Dos Santos Korol, 2019.
- <https://docs.python.org/3/>
- <https://in.mathworks.com/help/matlab/>
- <https://github.com/padhi499/Image-Classification-using-CNN-on-FPGA>
- [CNN Algorithm on MATLAB](#)