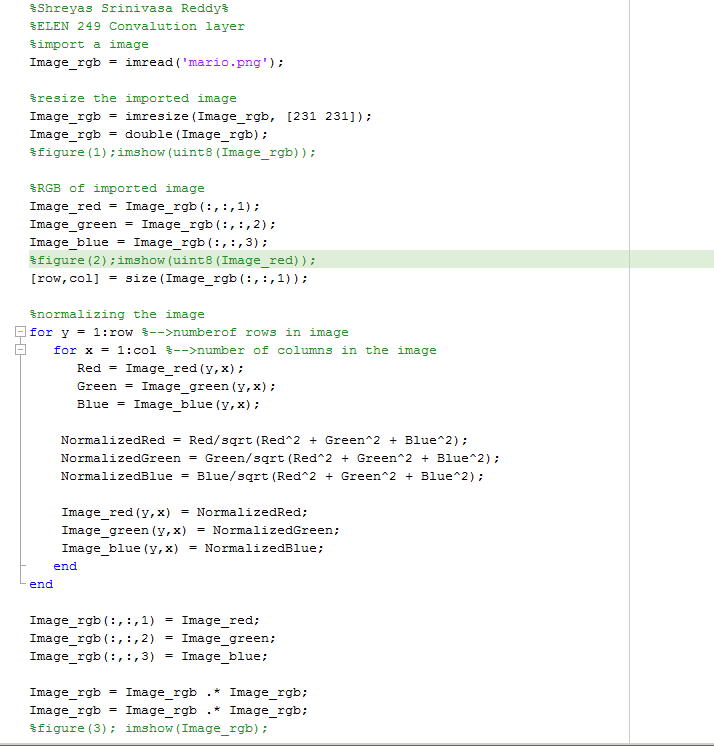
**SHREYAS SRINIVASA REDDY**

**ELEN249**



Input image (Imported image)

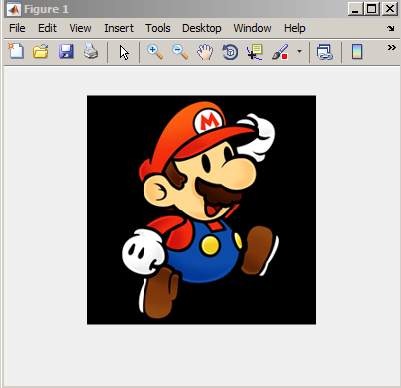
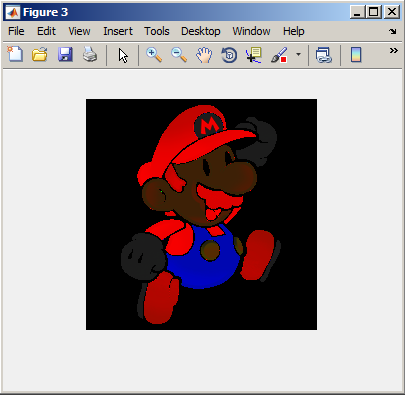
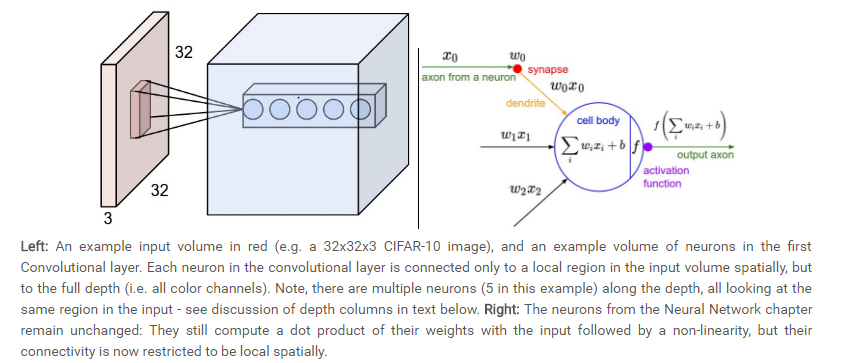


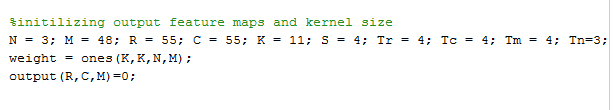
Image resized and normalized, RGB (Input to Convolution layer)

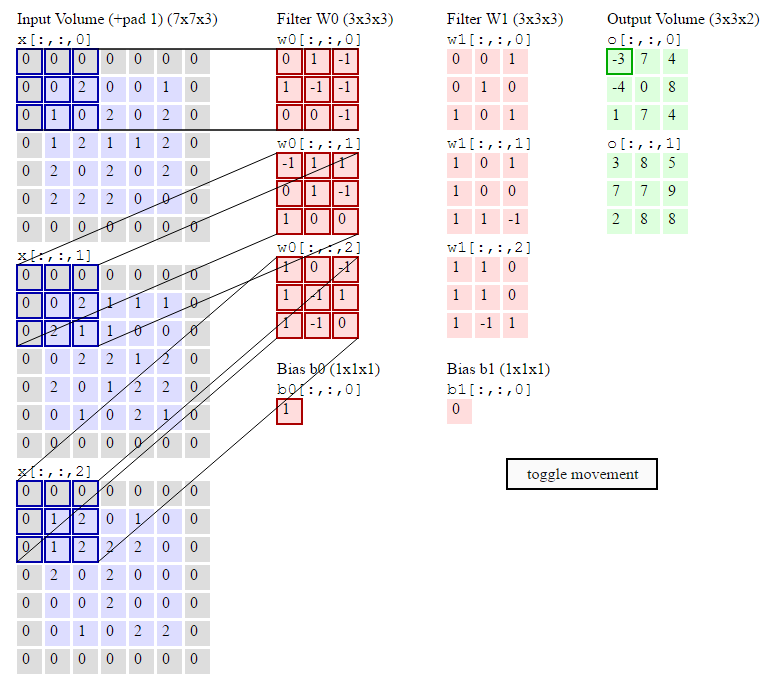


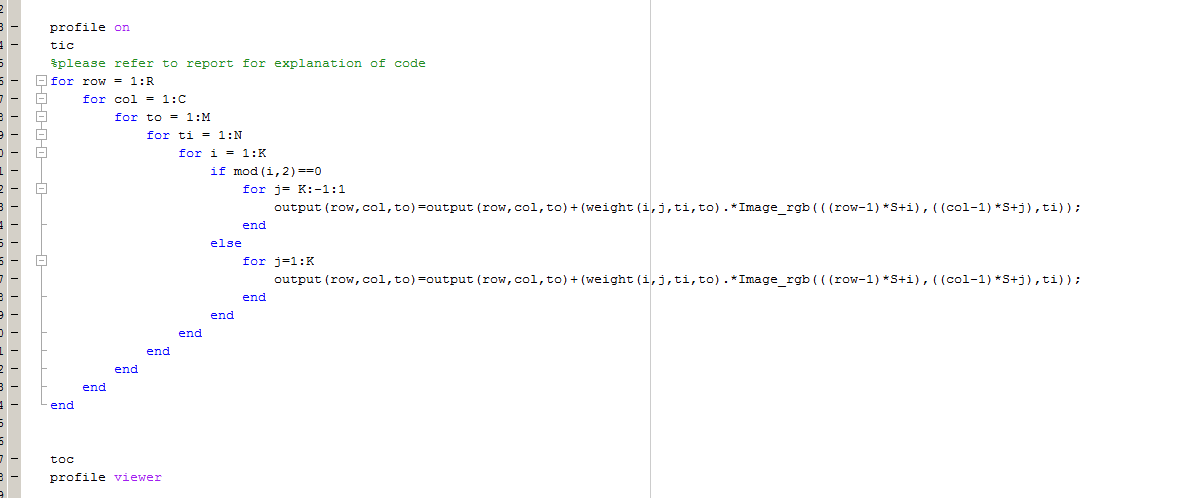


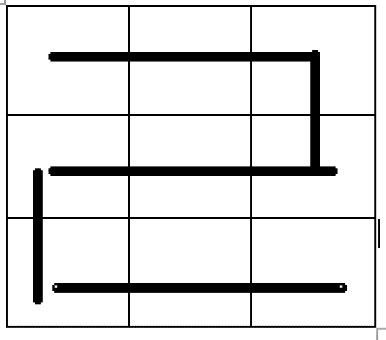
According to my assumption, there are 55\*55\*48 = 145,200 neurons in the first Convolution Layer, and each has 11\*11\*3 = 363 weights. Together, this adds up to 145200 \* 363 = 52707600 parameters on the first layer of the ConvNet alone. Clearly, this number is very high.

It turns out that we can dramatically reduce the number of parameters by making one reasonable assumption: That if one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2). In other words, denoting a single 2-dimensional slice of depth as a **depth slice** (e.g. a volume of size [55x55x48] has 48 depth slices, each of size [55x55]), we are going to constrain the neurons in each depth slice to use the same weights. With this parameter sharing scheme, the first Convolution Layer in our assumption would now have only 48 unique set of weights (one for each depth slice), for a total of 48\*11\*11\*3 = 17424 unique weights, or 17424 parameters. Alternatively, all 55\*55 neurons in each depth slice will now be using the same parameters. In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

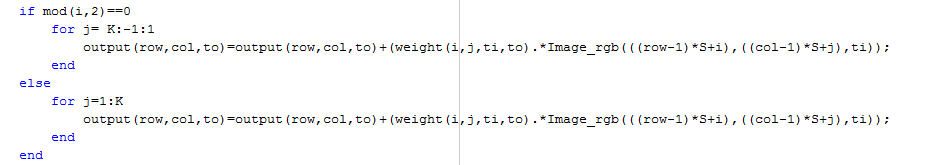




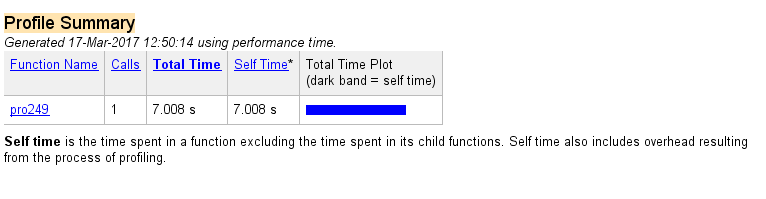
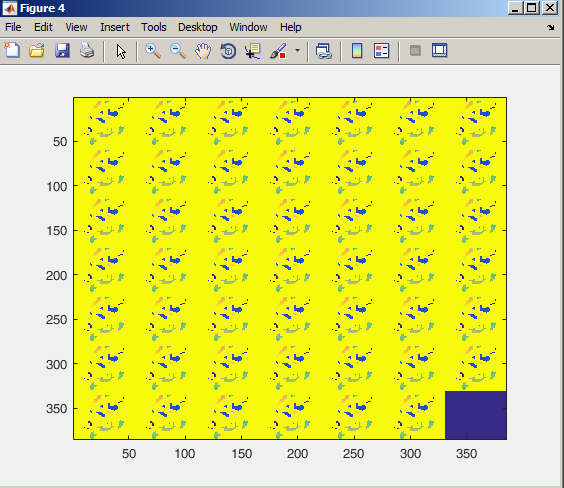
Using this above figure as a reference to Code1 %% Implementation of pesudo code\_1 in pro249.m%%



I used this method to traverse the matrix

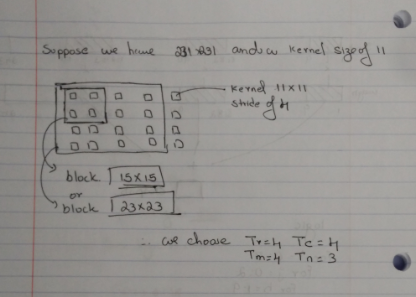


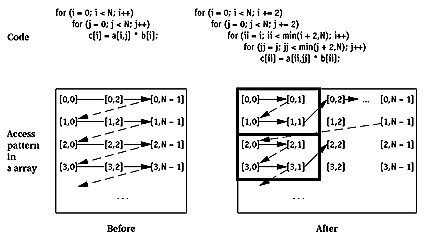
Output and Profile

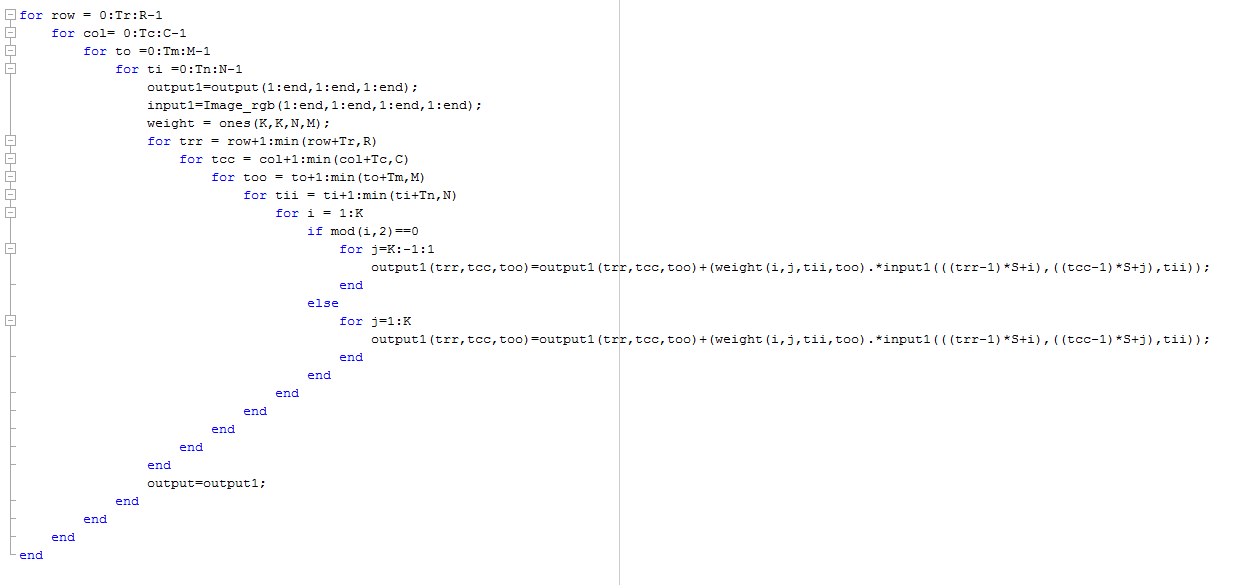


%% Implementation of code 2%%

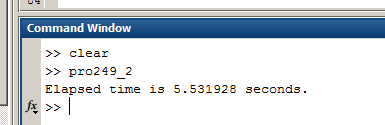
Loop Tiling

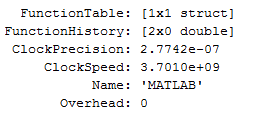


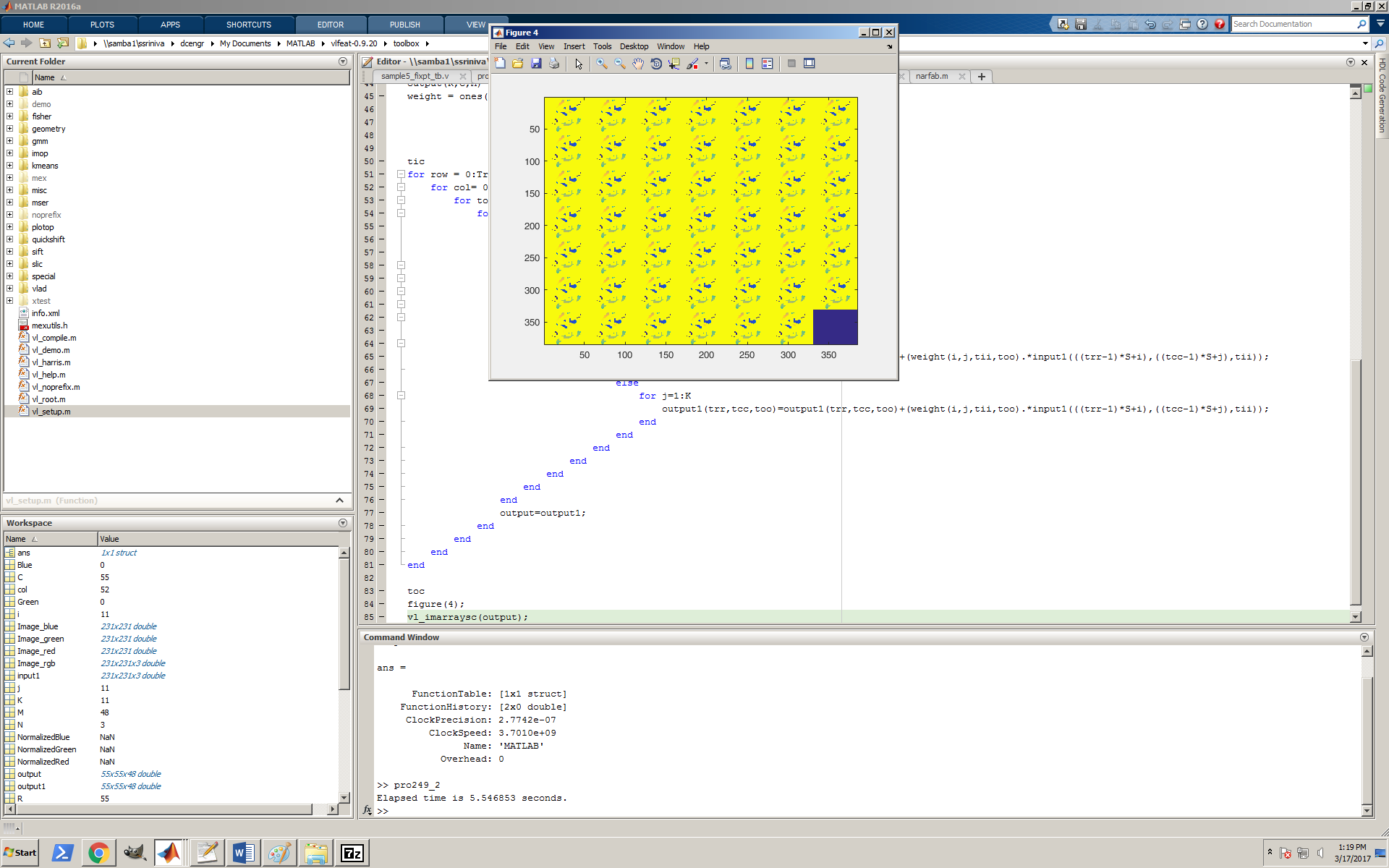




Output and Profile







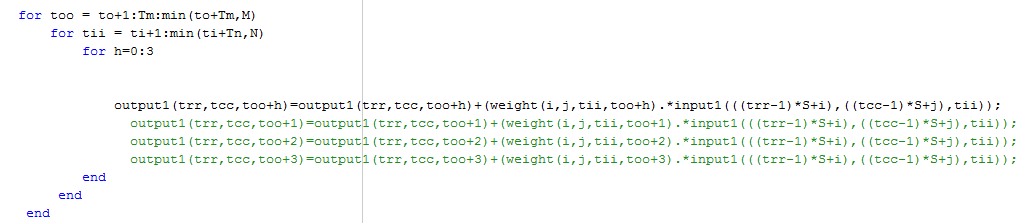
%% Implementation of Code 3 %%

NOTE ERRORs: I did try to do implementation of pipeline using parfor but matlab though a error bcoz I’m using parfor in side nested for loop which is not permitted. I did try creating a separate function and using a parfor

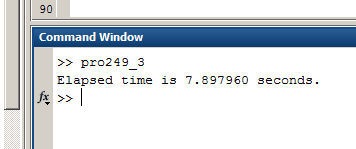
Below is the code to implemented for unroll

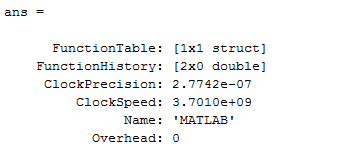


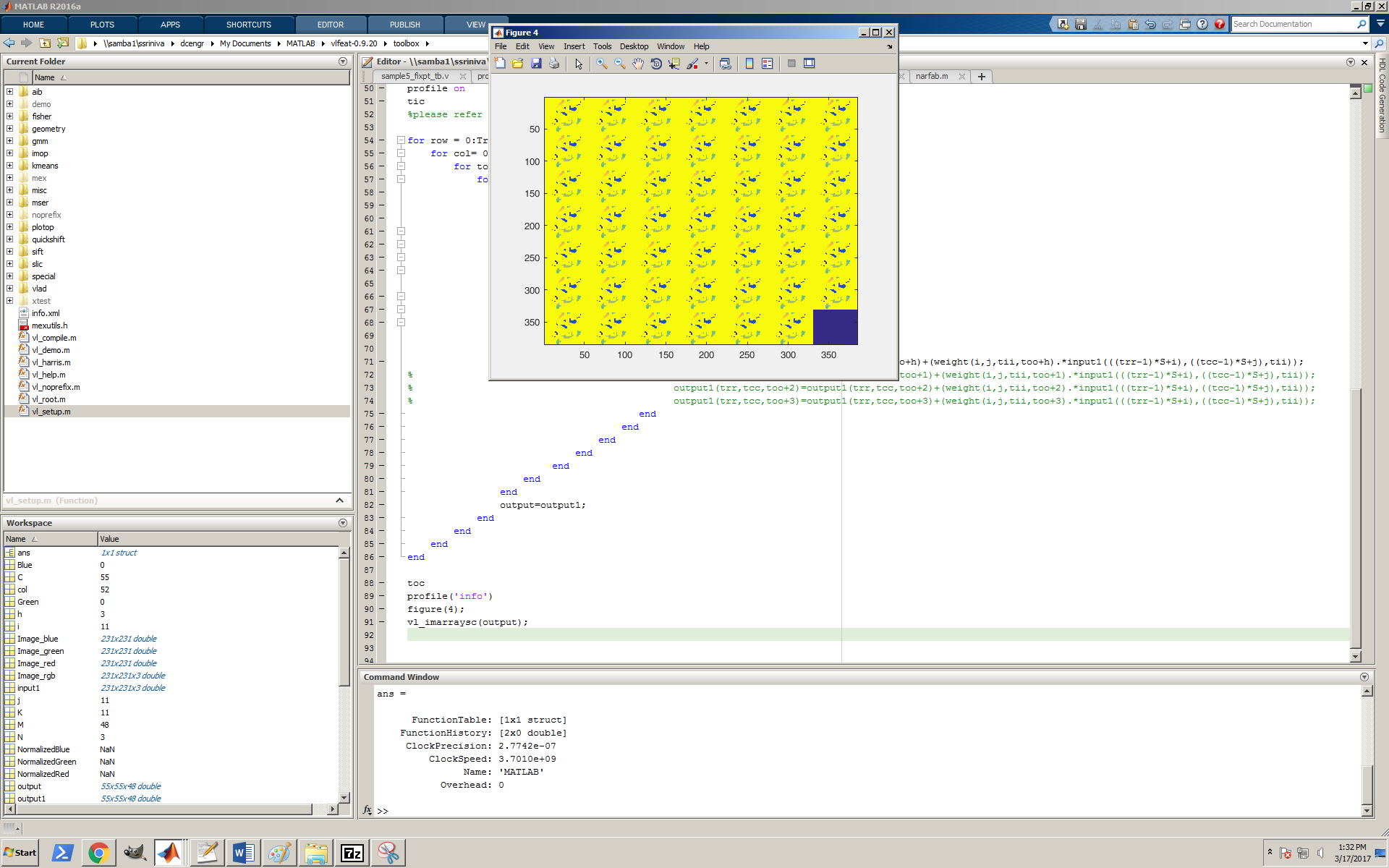
Logic used for Unroll



Output and Profile

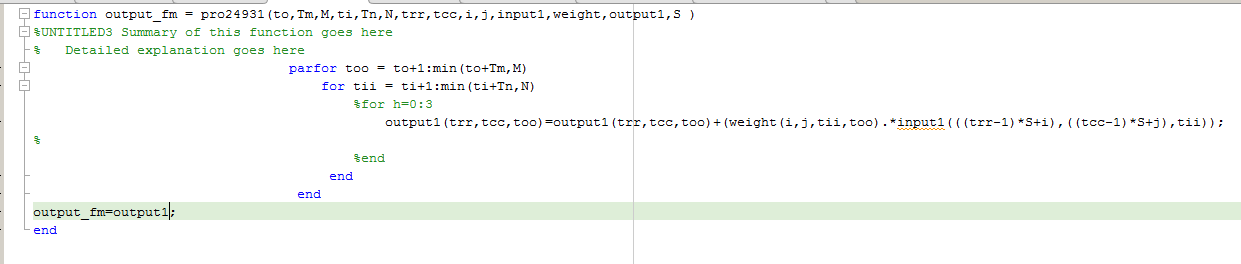






Below is the code using parfor





Converting Matlab to Verilog using HDL CODER

W=7;c = [0,0,0];

Image\_rgb = imread('mario.png');

%%resizing to input image to 7\*7

Image\_rgb = imresize(Image\_rgb, [W W]);

Image\_rgb = double(Image\_rgb);

%figure(1);imshow(uint8(Image\_rgb));

Image\_red = Image\_rgb(:,:,1);

Image\_green = Image\_rgb(:,:,2);

Image\_blue = Image\_rgb(:,:,3);

%figure(2);imshow(uint8(Image\_red));

[row,col] = size(Image\_rgb(:,:,1));

for y = 1:row %-->numberof rows in image

for x = 1:col %-->number of columns in the image

Red = Image\_red(y,x);

Green = Image\_green(y,x);

Blue = Image\_blue(y,x);

NormalizedRed = Red/sqrt(Red^2 + Green^2 + Blue^2);

NormalizedGreen = Green/sqrt(Red^2 + Green^2 + Blue^2);

NormalizedBlue = Blue/sqrt(Red^2 + Green^2 + Blue^2);

Image\_red(y,x) = NormalizedRed;

Image\_green(y,x) = NormalizedGreen;

Image\_blue(y,x) = NormalizedBlue;

end

end

Image\_rgb(:,:,1) = Image\_red;

Image\_rgb(:,:,2) = Image\_green;

Image\_rgb(:,:,3) = Image\_blue;

Image\_rgb = Image\_rgb .\* Image\_rgb;

Image\_rgb = Image\_rgb .\* Image\_rgb;

N = 3; M = 1; R = 3; C = 3; K = 3; S = 2;Z=W-K;

%initializing weight in a liner array

weight = ones(1,243);

weight2= ones(K,K,N,M);

%converting input matrix to linear array 7\*7 would have 9 kernels with

%srtide of 2 so each kernel is appended to the previous kernel 7\*7=49

%with kernel of 3\*3 we have 9 elements and 9 kernels 9\*9=81 elements

%input image has 3 features, therefore 81\*3=234

for a=1:3

input(:,:,a)=reshape(Image\_rgb(:,:,a),[1 W\*W]);

%2=stride incremented by 2 9kernels\*2=18

for j=1:2:18

%each kernel has 3 rows

for i=0:2

%K=kernel size Z=length of row input matrix - kernel

b=input(j+(i\*(K+Z)):j+(i\*(K+Z))+K-1);

%concatrinate the input to linear array

c=horzcat(c,b);

end

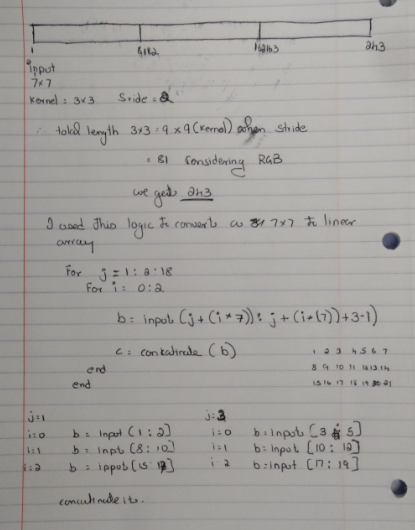
end

end

%I was getting an error so i initialized [0 0 0] to c

%now I'm removing it by just eliminating the first 3 elements

input1=c(4:end);



%initializing output to 0 as linear array

%since we have taken M=1 we just have 3\*3 matrix as output

p=[0 0 0 0 0 0 0 0 0];

output2(R,C,M)=0;

profile on

tic

%calling a function

output=sample5(input1,weight,p);

toc

p = profile('info')

%function

function output\_fm = sample5( input,weight,p)

out=0;

for i=0:8

for j=0:2

for h=1:9

output=(input(1,((i\*9)+(j\*81)+h)).\*weight(1,((i\*9)+(j\*81)+h)));

out=out+output;

end

end

p(i+1)=out;

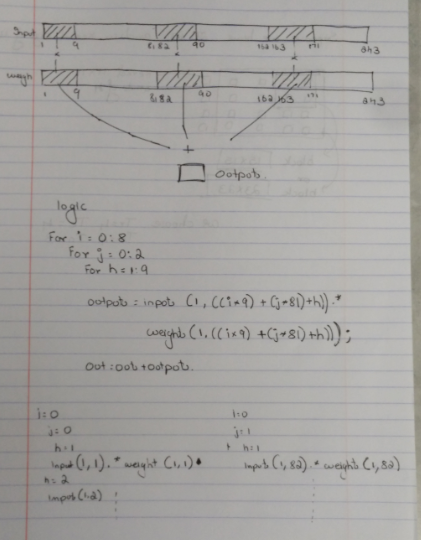
output=0;

out=0;

end

output\_fm=p;

end



Verilog code was generated using HDL coder in matlab I would upload the code to GitHub.

With the help of my friend I got this Verilog code here are they

//'timescale 1ns / 1ns

//‘timescale 1 ns / 100 ps

module Loop\_cnn(input [431:0]input\_fm ,output [699:0]output\_fm , input clk);

int row=0;

int col=0;

int to=0;

int ti=0;

int i=0;

int j=0;

int S=1;

//int trr, tcc, too,tii;

int Tr=0;

int Tc=0;

int tm=0;

reg [83:0]weights= 84'he\_e9e9\_e9e9\_e9e9\_e9e9\_e9e0; // initializing weights

reg[699:0]output\_fm;

reg[699:0]temp;

always @ (posedge clk)begin

output\_fm=0;

temp=0;

repeat (7) begin //to 48 (output)

repeat (10) begin //row 55

repeat (10) begin //col 55

repeat (3) begin // input ti 3

repeat (6) begin // kernal i 11

repeat (6) begin // kernal j 11

//PE logic part

temp[(to\*99)+(row\*9)+(col)]=weights[(to\*11)+(ti\*3)+(i)+(j)]\*input\_fm[(ti\*143)+((S\*row+i)\*11)+(S\*col+j)];

output\_fm[(to\*99)+(row\*9)+(col)]= weights[(to\*11)+(ti\*3)+(i)+(j)]\*input\_fm[(ti\*143)+((S\*row+i)\*11)+(S\*col+j)];

j=j+1;

end

j=0;

i=i+1;

end

i=0;

ti=ti+1;

end

ti=0;

col=col+1;

end

col=0;

row=row+1;

end

row=0;

to=to+1;

end

to=0;

end

endmodule

CODE 2

//'timescale 1ns / 1ns

//‘timescale 1 ns / 100 ps

module Loop\_cnn(input [431:0]input\_fm ,output [699:0]output\_fm , input clk);

int row=0;

int col=0;

int to=0;

int ti=0;

int i=0;

int j=0;

int S=1;

int trr=0;

int tcc=0;

int trr\_temp=0;

int tcc\_temp=0;

reg [83:0]weights= 84'he\_e9e9\_e9e9\_e9e9\_e9e9\_e9e9; // initializing weights

reg[699:0]output\_fm;

reg[699:0]temp;

always @ (posedge clk)begin

output\_fm=0;

temp=0;

repeat (7) begin //to 48 (output)

repeat (10) begin //row 55

repeat (10) begin //col 55

tcc=row;if (row+2 < 10) trr\_temp=col+2; else trr\_temp=10;

repeat (trr\_temp) begin

tcc=col;if (col+2 < 10) tcc\_temp=col+2; else tcc\_temp=10; //looptiling only for column and row

repeat (tcc\_temp) begin //col 55

repeat (3) begin // input ti

repeat (6) begin // kernal i

repeat (6) begin // kernal j

//PE logic part

temp[(to\*99)+(trr\*9)+(tcc)]=weights[(to\*11)+(ti\*3)+(i)+(j)]\*input\_fm[(ti\*143)+((S\*trr+i)\*11)+(S\*tcc+j)];

output\_fm[(to\*99)+(trr\*9)+(tcc)]= temp[(to\*99)+(trr\*9)+(tcc)];

j=j+1;

end

j=0;

i=i+1;

end

i=0;

ti=ti+1;

end

ti=0;

tcc=tcc+1;

end

tcc=0;

trr=trr+1;

end

trr=0;

col=col+2;

end

col=0;

row=row+2;

end

row=0;

to=to+1;

end

to=0;

end

endmodule

code3

//'timescale 1ns / 1ns

//‘timescale 1 ns / 100 ps

module Loop\_cnn(input [431:0]input\_fm ,output [699:0]output\_fm , input clk);

int row=0;

int col=0;

int to=0;

int ti=0;

int i=0;

int j=0;

int S=1;

int trr=0;

int tcc=0;

int trr\_temp=0;

int tcc\_temp=0;

reg [83:0]weights= 84'he\_e9e9\_e9e9\_e9e9\_e9e9\_e9e9; // initializing weights

reg[699:0]output\_fm;

reg[699:0]temp;

always @ (posedge clk)begin

output\_fm=0;

temp=0;

repeat (7) begin //to 48 (output)

repeat (10) begin //row 55

repeat (10) begin //col 55

tcc=row;if (row+2 < 10) trr\_temp=col+2; else trr\_temp=10;

repeat (trr\_temp) begin

tcc=col;if (col+2 < 10) tcc\_temp=col+2; else tcc\_temp=10; //looptiling only for column and row

repeat (tcc\_temp) begin //col 55

repeat (3) begin // input ti

repeat (6) begin // kernal i

repeat (1) begin // kernal j

//PE logic part

temp[(to\*99)+(trr\*9)+(tcc)]=weights[(to\*11)+(ti\*3)+(i)+(j)]\*input\_fm[(ti\*143)+((S\*trr+i)\*11)+(S\*tcc+j)];

temp[(to\*99)+(trr\*9)+(tcc+1)]=weights[(to\*11)+(ti\*3)+(i)+(j+1)]\*input\_fm[(ti\*143)+((S\*trr+i)\*11)+(S\*tcc+j+1)];

temp[(to\*99)+(trr\*9)+(tcc+2)]=weights[(to\*11)+(ti\*3)+(i)+(j+2)]\*input\_fm[(ti\*143)+((S\*trr+i)\*11)+(S\*tcc+j+2)];

temp[(to\*99)+(trr\*9)+(tcc+3)]=weights[(to\*11)+(ti\*3)+(i)+(j+3)]\*input\_fm[(ti\*143)+((S\*trr+i)\*11)+(S\*tcc+j+3)];

temp[(to\*99)+(trr\*9)+(tcc+4)]=weights[(to\*11)+(ti\*3)+(i)+(j+4)]\*input\_fm[(ti\*143)+((S\*trr+i)\*11)+(S\*tcc+j+4)];

temp[(to\*99)+(trr\*9)+(tcc+5)]=weights[(to\*11)+(ti\*3)+(i)+(j+5)]\*input\_fm[(ti\*143)+((S\*trr+i)\*11)+(S\*tcc+j+5)];

output\_fm[(to\*99)+(trr\*9)+(tcc)]= temp[(to\*99)+(trr\*9)+(tcc)];

output\_fm[(to\*99)+(trr\*9)+(tcc+1)]= temp[(to\*99)+(trr\*9)+(tcc+1)];

output\_fm[(to\*99)+(trr\*9)+(tcc+2)]= temp[(to\*99)+(trr\*9)+(tcc+2)];

output\_fm[(to\*99)+(trr\*9)+(tcc+3)]= temp[(to\*99)+(trr\*9)+(tcc+3)];

output\_fm[(to\*99)+(trr\*9)+(tcc+4)]= temp[(to\*99)+(trr\*9)+(tcc+4)];

output\_fm[(to\*99)+(trr\*9)+(tcc+5)]= temp[(to\*99)+(trr\*9)+(tcc+5)];

j=j+5;

end

j=0;

i=i+1;

end

i=0;

ti=ti+1;

end

ti=0;

tcc=tcc+1;

end

tcc=0;

trr=trr+1;

end

trr=0;

col=col+2;

end

col=0;

row=row+2;

end

row=0;

to=to+1;

end

to=0;

end

endmodule