VIETNAM NATIONAL UNIVERSITY HO CHI MINH

UNIVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF COMPUTER ENGINEERING

VÕ DUY

TRƯƠNG DUY ĐỨC

PROJECT OF DESIGN DIGITAL SYSTEM WITH HDL

DESIGN HARDWARE OF CNN MODEL

HỒ CHÍ MINH CITY, 2024

VIETNAM NATIONAL UNIVERSITY HO CHI MINH

UNIVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF COMPUTER ENGINEERING

VÕ DUY – 21522015

TRƯƠNG DUY ĐỨC - 21521970

PROJECT OF DESIGN DIGITAL SYSTEM WITH HDL

DESIGN HARDWARE OF CNN MODEL

LECTURER

PhD. LÂM ĐỨC KHẢI

HỒ CHÍ MINH CITY, 2024

MỤC LỤC

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

CNN	Convolutional Neural Network
FF	Flipflop
ReLu	Rectified Linear Unit
SVM	Support Vector Machine
ResNet	Residual Neural Network
IP	Intellectual Property
MNIST	Mixed National Institute of Standards and Technology
MSB	Most Significant Bit
FC	Fully Connected
RAM	Random-access memory
FIFO	First In First Out

CHAPTER 1. INTRODUCTION

1.1. What is CNN?

In recent years, along with the explosion of the 4.0 revolution, Machine Learning plays an important role in human life. We can find it in many devices around us and it also is an important thing to build many future devices such as an automated car.

A Convolutional Neural Network (CNN) also known as ConvNet, is one of the most popular models of machine learning, it has many architectures and uses. It is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation. CNNs are employed in a variety of practical scenarios, such as autonomous vehicles, security camera systems, and others.

A CNN for image classification takes an input image, processes it, and classifies it into specific categories (e.g., Dog, Cat, or numbers, etc.). Which means after CNN being trained and tested, you give an input image, after a period of time, a CNN will provide the percentage of how much the input image belongs to each category.

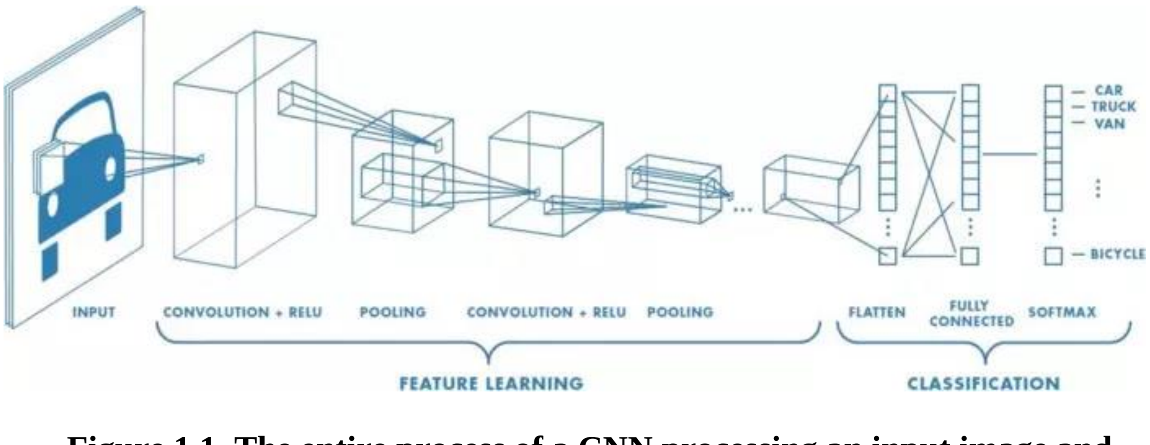


Figure 1.1. The entire process of a CNN processing an input image and classifying it [11]

1.1.1. A basic architecture of a CNN

A Convolutional Neural Network (CNN) is a specialized structure of artificial neural networks that simulate features of the human brain. The network consists of multiple layers and is designed to effectively recognize complex patterns in images. The innovation of the proposed model includes incorporating multiple perceptron layers of convolutional layers into the structure, where each neuron is only connected to a small region of neurons from the previous layer. This functionality allows the detection of initial features in the original image, and in subsequent layers, the network captures more complex features, thereby identifying complex objects in the image. The typical layers of a CNN architecture are as follows.

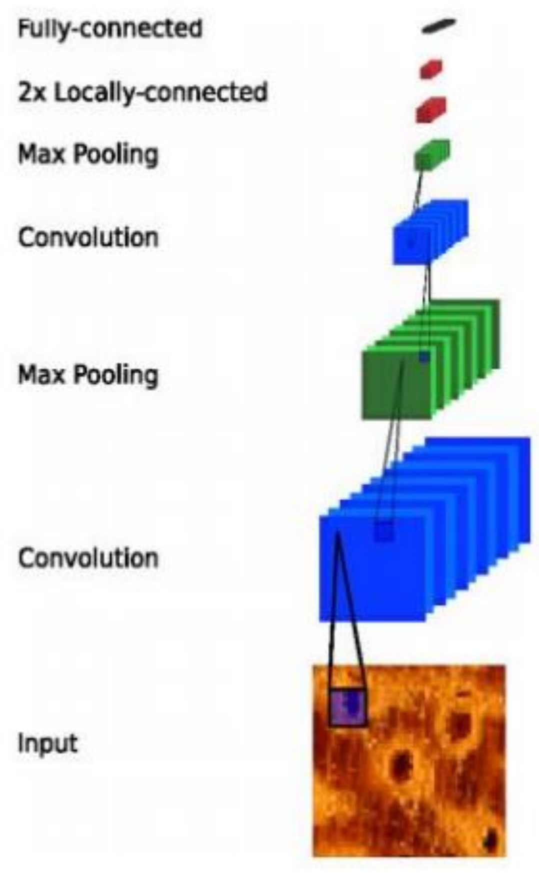


Figure 1.2. Basic structure of CNN [7]

In detail of basic structure CNN:

- *Input layer:* Resolution of the input image.
- *Convolutional layer*: All neurons in the layer, unlike perceptrons, are connected to only a portion of neurons in the previous layer. This layer utilizes an activation function to introduce non-linearity to CNN.
- *Subsampling layer*: Aims to select the most important features from the previous layer, reducing the size of processed information and passing it to the next layer.
- *Fully-connected layer*: A hidden layer of artificial neural network perceptrons. After a convolutional layer and a fully connected layer, the activation function of neurons can be used to transform neuron signals into output signals.

1.1.1.1. Input layer

Regarding the task of object detection in images, the input layer is typically represented as a three-dimensional grid with a size that depends on:

$$I = W * H * D$$

Where I is the size of the input layer, W and H are the dimensions of the image (width and height), and D is the depth or the number of channels in the image (for example, an RGB image has 3 channels).

1.1.1.2. Convolutional layer

The convolutional layer of a neural network is one of the key layers designed to highlight features in images and their transformations. It is used to capture more complex properties and ultimately determine recognition. This type of layer is called a convolutional layer, representing a multidimensional filter that acts as the weight matrix of connections between neurons of the previous layer and neurons of the convolutional layer. The stride may be smaller than the filter size, causing the filters to be applied in overlapping windows. Each neuron signal value from the previous layer, located in a specific region corresponding to the filter, is multiplied by the corresponding value of the filter kernel in CNN. The values of the filter matrix are referred to as the weights of the neuron connection in the convolutional layer.

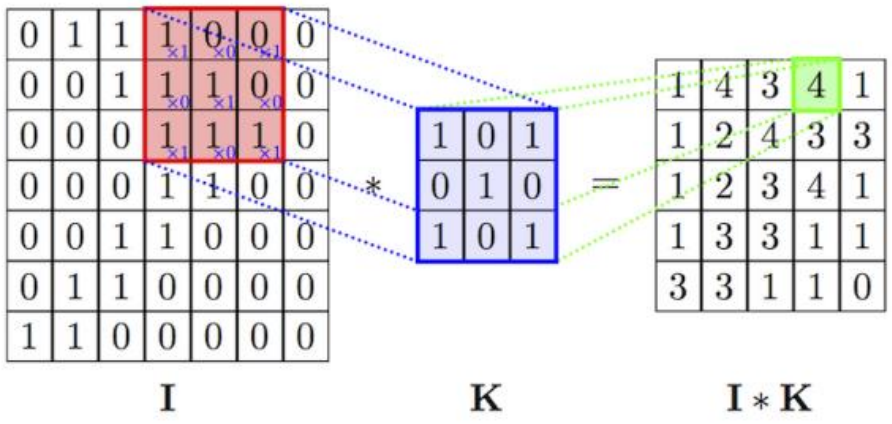


Figure 1.3. How convolutional layer works [9]

The activation function in CNN is the decision-making function for the output of the convolutional layer. Using an activation function is the only way to introduce non-linearity into the processing. Without it, since all processes inside the layers of the CNN are linear transformations, a vast number of new neurons would be required, making such a system inefficient and inflexible.

Essentially, the activation function allows CNN to "learn" after computing the error. This process is called backpropagation. By calculating the error to determine the relative amount of change needed (influenced by the learning rate), it propagates from neurons in the layer closest to the output all the way to the first layers.

Common activation functions used in building CNNs today include the Sigmoid function, ReLU, Leaky ReLU, Parameter ReLU, etc. Properly and appropriately using these functions helps avoid issues such as faster weight updates, network learning halting due to vanishing gradients, and more.

These activation functions play a crucial role in the training process and can significantly impact the performance of CNN.

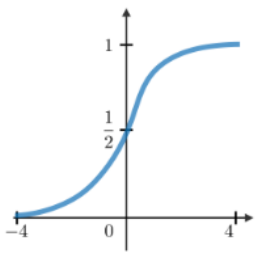
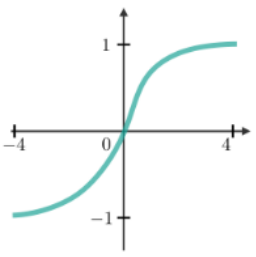
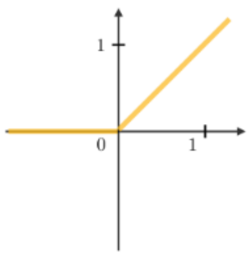
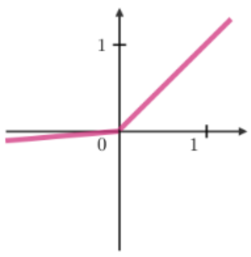
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Figure 1.4. Graph of activation function [6]

1.1.1.3. Subsampling layer

This layer of the CNN performs non-linear down sampling, reducing the size of the data to decrease the likelihood of overfitting quickly. It also aims to reduce computational cost and memory consumption. Typically, this layer is used after performing the convolution operation and transforming signals from complex layers, highlighting the most important features based on certain criteria.

This layer uses a window of a specific size to select data from neurons of the previous layer that are connected to neurons of the pooling layer. Then, according to certain rules, a signal from a pooling layer neuron is selected and computed. The most common filter used to form the signal of this layer is by finding the maximum value among the signals from the previous layer (max pooling) within the pooling window or by computing the average value. An example of a pooling layer with a max pooling filter of size 2x2 and a stride of 2 which is generally being used, shown in figure **1.5**.

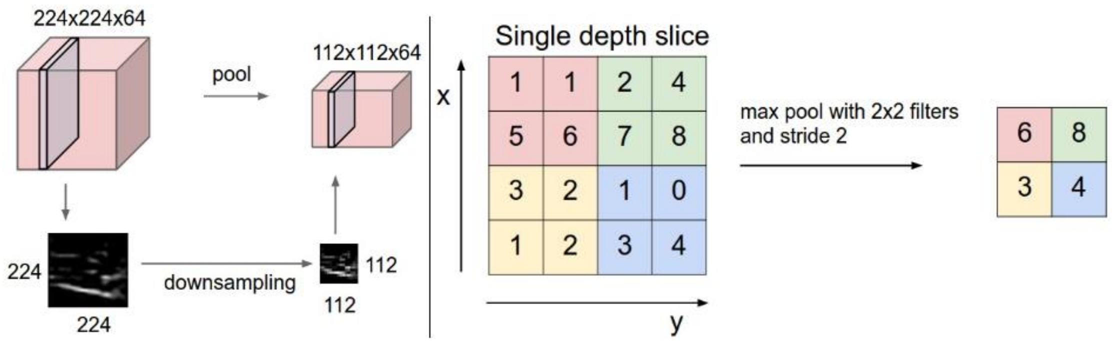


Figure 1.5. Describe how to sample data [2]

1.1.1.4. Fully-connected layer

After processing the image, image features will be pushed into the fully connected layer. This layer has the function of converting the feature matrix in the previous layer into a vector containing the probabilities of the object that need to be predicted.

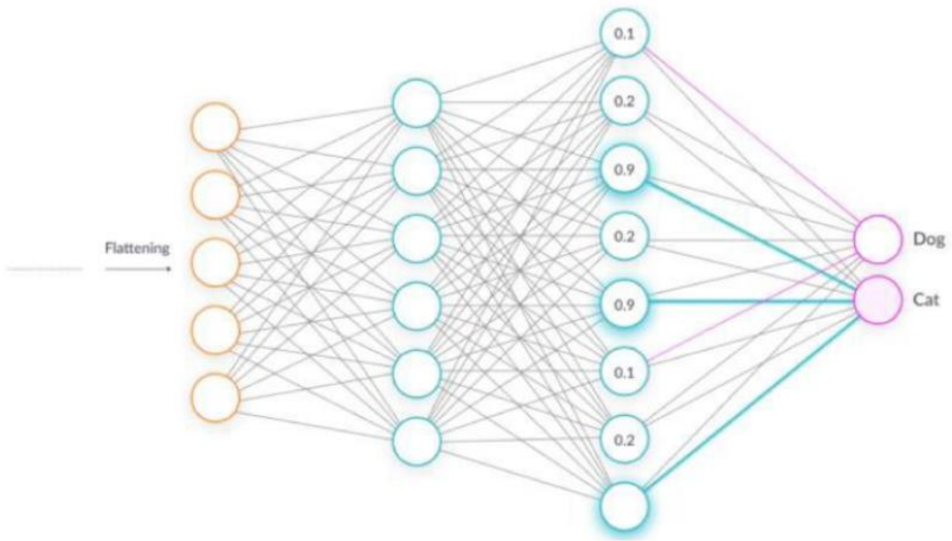


Figure 1.6. Fully-connected layer [1]

For example, in **Figure 1.6**, the fully connected layer covers the characteristic tensor of the previous layer into a 2-dimensional vector representing the probabilities of the 2 similar layers.

And finally, the process of training the Convolutional Neural Network Model for the image classification problem is like training other models. We need an error function to calculate the error that keeps the model and label prediction accurate, as well as we use a backpropagation algorithm for the weight update process.

1.2. The importance of CNN

CNNs are distinguished from classic machine learning algorithms such as SVMs and decision trees by their ability to autonomously extract features at a large scale, bypassing the need for manual feature engineering and thereby enhancing efficiency.

The convolutional layers grant CNNs their translation-invariant characteristics, empowering them to identify and extract patterns and features from data irrespective of variations in position, orientation, scale, or translation.

A variety of pre-trained CNN architectures, including VGG-16, ResNet50, Inceptionv3, and EfficientNet, have demonstrated top-tier performance. These models can be adapted to new tasks with relatively little data through a process known as fine-tuning.

Beyond image classification tasks, CNNs are versatile and can be applied to a range of other domains, such as natural language processing, time series analysis, and speech recognition.

1.3. Applications of CNN

Convolutional Neural Networks have revolutionized the field of computer vision, leading to significant advancements in many real-world applications. Below are a few examples of how they are applied.

- *Image classification:* Convolutional neural networks are used for image categorization, where images are assigned to predefined categories. One use of such a scenario is automatic photo organization in social media platforms.
- *Object detection:* CNNs can identify and locate multiple objects within an image. This capability is crucial in multiple scenarios of shelf scanning in retail to identify out-of-stock items.

- *Facial recognition*: this is also one of the main industries of application of CNNs. For instance, this technology can be embedded into security systems for efficient control of access based on facial features.

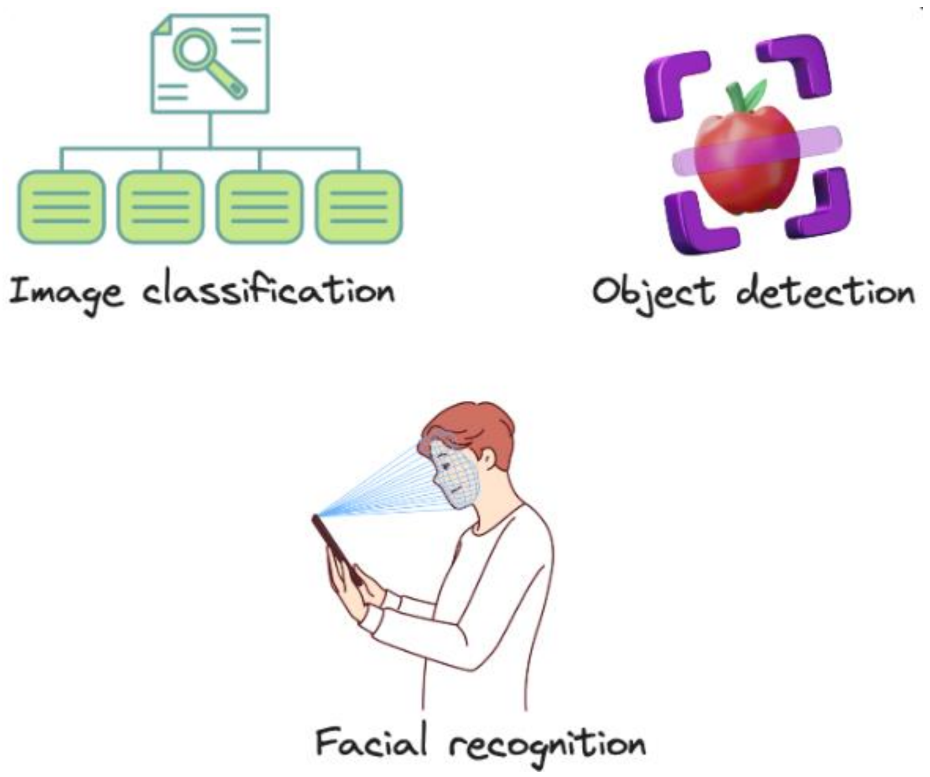


Figure 1.7. Application of CNN [3]

1.4. Algorithms of CNN

1.4.1. Feed-forward algorithm

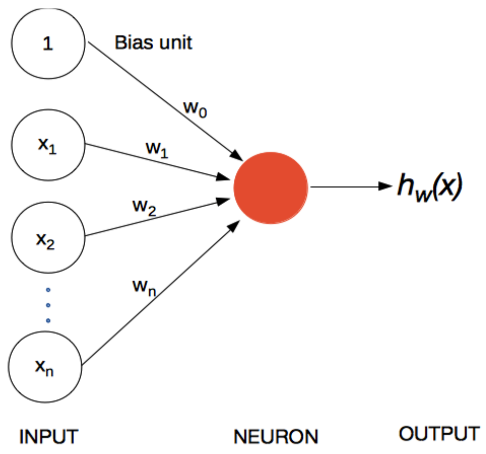


Figure 1.8. Feed-forward algorithm [10]

In Figure 1.8, the inputs x are being fed to the red neuron along with w in their path. The input value of the red Neuron is called Val with the formula:

$$val = \sum_{i=1}^{n} w_i x_i + b$$

In which, the main b value is w0 in Figure 1.8, which is called the bias weight. Each neuron when producing output will have to go through an activation function, the formula is as shown in Figure 1.8.

$$out = f(val)$$

1.4.2. Backpropagation algorithm

Learning in feed-forward networks belongs to the realm of supervised learning, in which pairs of input and output values are fed into the network for many cycles, so that the network 'learns' the relationship between the input and output.

We provide the network with a number of training samples, which consists of an input vector i and its desired output o. For instance, in the classification problem, suppose we have points (1, 2) and (1, 3) belonging to group 0, points (2, 3) and (3, 4) belonging to group 1, (5, 6) and (6, 7) belonging to group 2, then for a feed-forward network with 2 input nodes and 2 output nodes, the training set would be:

{i = (1, 2) , o =(0, 0)

i = (1, 3) , o = (0, 0)

i = (2, 3) , o = (1, 0)

i = (3, 4) , o = (1, 0)

i = (5, 6) , o = (0, 1)

i = (6, 7) , o = (0, 1) }

The basic rule for choosing the number of output nodes depends on the number of different regions. It is advisable to use a unary notation to represent the different regions, i.e. for each output only one node can have value 1. Hence the number of output nodes = number of different regions -1.

In backpropagation learning, every time an input vector of a training sample is presented, the output vector o is compared to the desired value d.

The comparison is done by calculating the squared difference of the two:

$$\boldsymbol{Err = (d - o)^2}$$

The value of Err tells us how far away we are from the desired value for a particular input. The goal of backpropagation is to minimize the sum of Err for all the training samples, so that the network behaves in the most "desirable" way.

$$\textbf{Minimize} \sum \boldsymbol{Err = (d - o)^2}$$

We can express Err in terms of the input vector (i), the weight vectors (w), and the threshold function of the perceptions. Using a continuous function (instead of the step function) as the threshold function, we can express the gradient of Err with respect to the w in terms of w and i.

Given the fact that decreasing the value of w in the direction of the gradient leads to the most rapid decrease in Err, we update the weight vectors every time a sample is presented using the following formula:

$$\boldsymbol{w_{new} = w_{old} - n.\frac{\delta\, Err}{\delta\, w}}$$

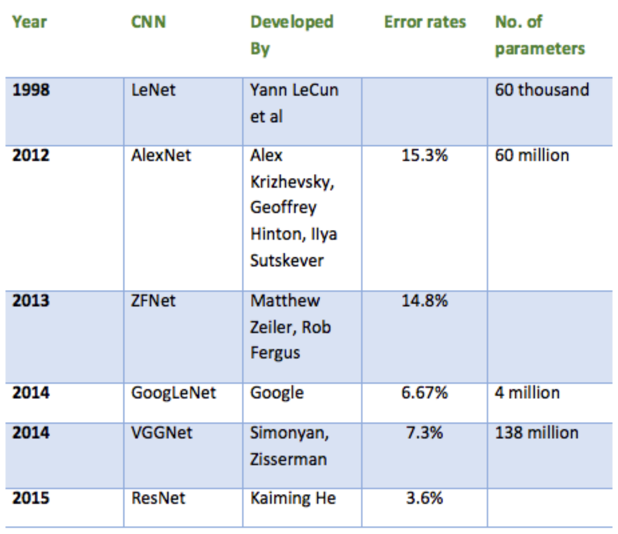
where n is the learning rate (a small number ~ 0.1)

Using this algorithm, the weight vectors are modified so that the value of Err for a particular input sample decreases a little bit every time the sample is presented. When all the samples are presented in turns for many cycles, the sum of Err gradually decreases to a minimum value, which is our goal as mentioned above.

1.5. The neural networks we choose

1.5.1. Reason choose this neural networks

Currently, there are numerous CNN models such as LeNet-5, AlexNet, VGG, etc., each with its own strengths and weaknesses. These are specifically described in [Table 1.9].



Year	CNN	Developed By	Error rates	No. of parameters
1998	LeNet	Yann LeCun et al		60 thousand
2012	AlexNet	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	15.3%	60 million
2013	ZFNet	Matthew Zeiler, Rob Fergus	14.8%	
2014	GoogLeNet	Google	6.67%	4 million
2014	VGGNet	Simonyan, Zisserman	7.3%	138 million
2015	ResNet	Kaiming He	3.6%	

Figure 1.9. Compare CNN model [12]

Each model is optimized for either minimizing error rates or reducing the number of parameters. Specifically, in this report, we aim to optimize the number of parameters to create the least complex hardware. Therefore, we have chosen to utilize LeNet-5.

Furthermore, in terms of hardware optimization, Lennet also optimizes certain components such as the number of Flip-Flops (FF), Look-Up Tables (LUT), Block RAM (BRAM), and DSP, as shown in the table.

Name	FF	LUT	BRAM	DSP	CLOCK
Vgg16[13]	127653	182616	486	780	150MHz
Lenet5[14]	42618	38136	144	206	100MHz
Alexnet[15]	437200	51378	303	808	303 MHz

Table 1. Compare CNN model

1.5.2. Architecture of Lenet-5

This is a architecture we referenced from a research paper "Gradient-based learning applied to document recognition [5]".

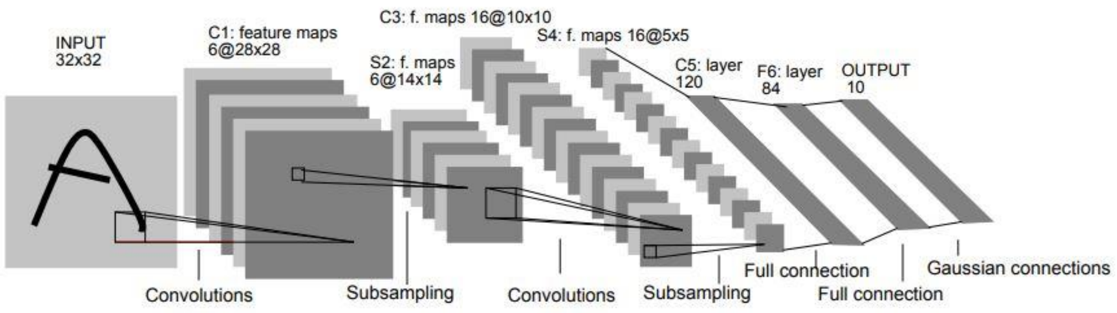


Figure 1.10. Architecture of Lenet-5 [5]

The architecture in detail

Layer	Filter	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32x32x1	-
Conv1	6	5x5	1	28x28x6	ReLu
Max pooling 1	-	2x2	2	14x14x6	-
Conv2	16	5x5	1	10x10x16	ReLu

Max pooling 2	-	2x2	2	5x5x16	-
Fully-connected1	-	-	-	120	ReLu
Fully-connected 2	-	-	-	84	ReLu
Fully-connected 3	-	-	-	10	Softmax

Table 2. The architecture detail

This architecture has 5 layers with learnable parameters and hence named Lenet-5. It has two sets of convolutional layers with a combination of max pooling. After the convolution and max pooling layers we have 3 fully connected layers. At last, a Softmax classifier which classfies the image into respective class.

The input to this model is 32x32 grayscale image hence the number of channels is one

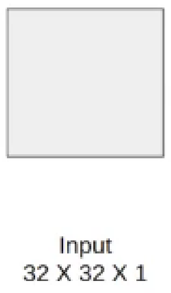


Figure 1.11. Input of Lenet-5 [8]

Then apply the first convolution operation with the fitler size 5x5, and we have 6 filters. As the result we get the feature map of size 28x28x6. Here the number of channel is equal to the number of filter applied.

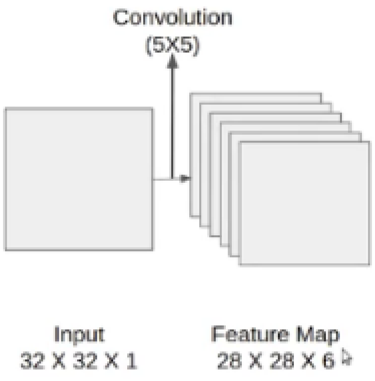


Figure 1.12. First convolution layer [8]

After the first convolution layers, we apply the max polling and the size of feature is reduced by half.

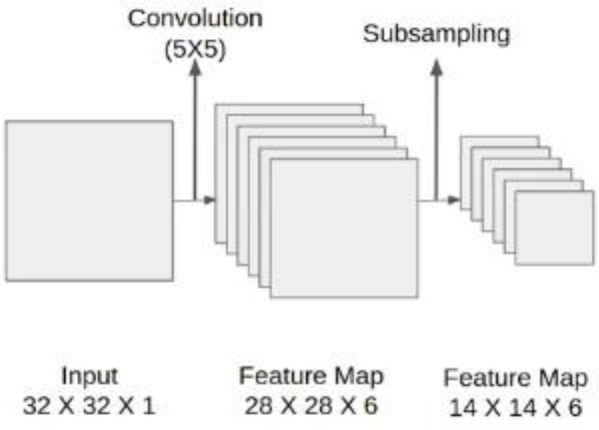


Figure 1.13. First max pooling [8]

Next, we apply the next convolution layers with sixteen filters of size 5x5. Again the feature map changed it is 10x10x16. The output size is calculated in a simliar manner. After this, we again applied an max pooling layer, which again reduce the size of feature map by half is 5x5x16

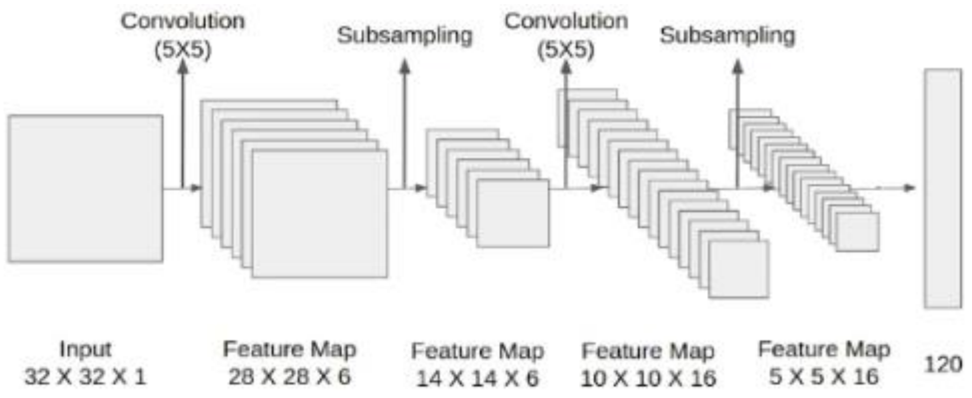


Figure 1.14. After second max pooling and convolution [8]

Then we have a final convolution layer of size 5X5 with 120 filters. As shown in the above image. Leaving the feature map size 1X1X120. After which flatten result is 120 values.

After these convolution layers, we have a fully connected layer with eighty-four neurons. At last, we have an output layer with ten neurons since the data have ten classes.

Here is the final architecture of the Lenet-5 model.

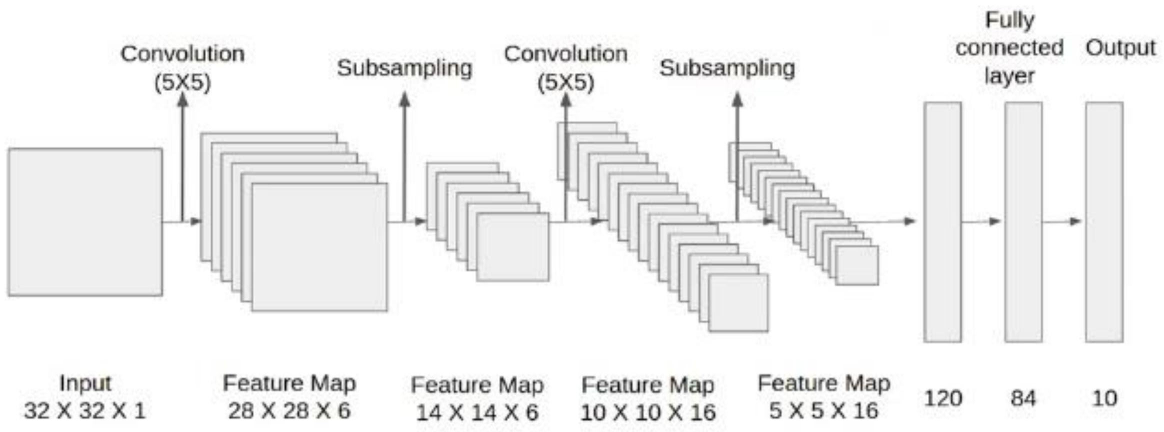


Figure 1.15. Final architecture of Lenet-5 model [8]

Chương 2. SOFTWARE IMPLEMENTATION

2.1. Dataset MNIST

The MNIST [4] database (Figure 2.1) is a large database of handwritten digits commonly used in training various image processing systems. This database is also widely used for training and testing in the field of machine learning. The database was created by “remixing” samples from the original NIST dataset. The database creators felt that because the NIST training dataset was obtained from the US Census Bureau, while the test dataset was obtained from US high school students because, so it is not suitable for machine learning experiments. Furthermore, the black and white images from NIST normalized to fit the 28x28 pixel.

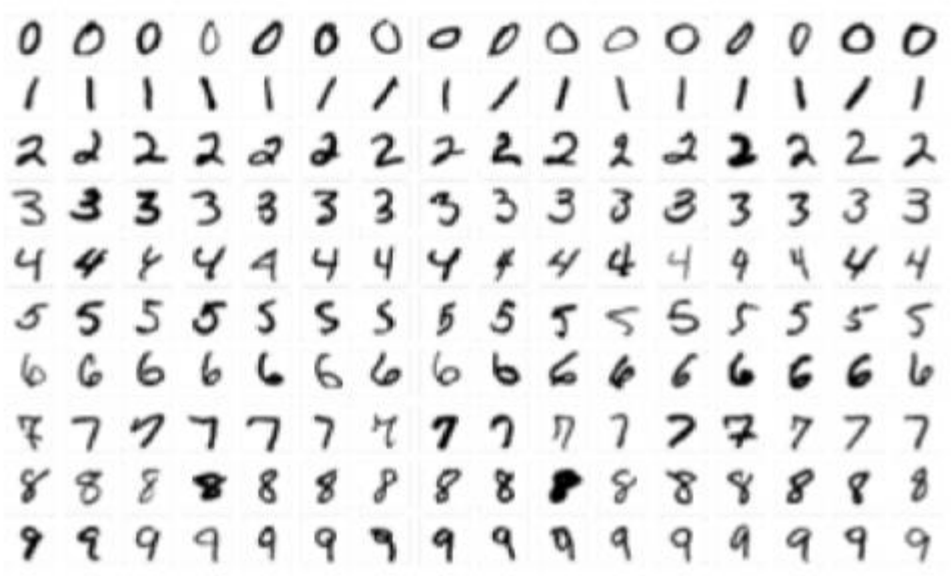


Figure 2.1 MNIST dataset

2.2. Training model LeNet-5

2.2.1. Processing Data

Because the dataset MNIST has size for each image is 28x28, but our architecture we choose has a input is 32x32, so we need to resize image from the dataset MNIST.

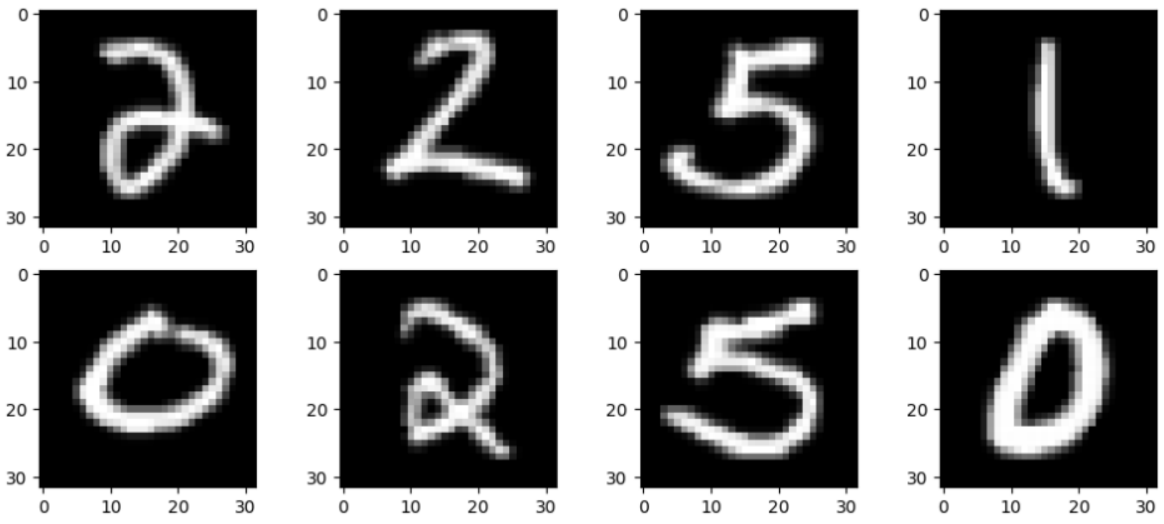


Figure 2.2. Images after resize

After that, we covert image values from integer to floats. Then normalize the data by dividing two training set and testing set for value maximum. Next, we use one hot encoding the labels for the training set and the testing set.

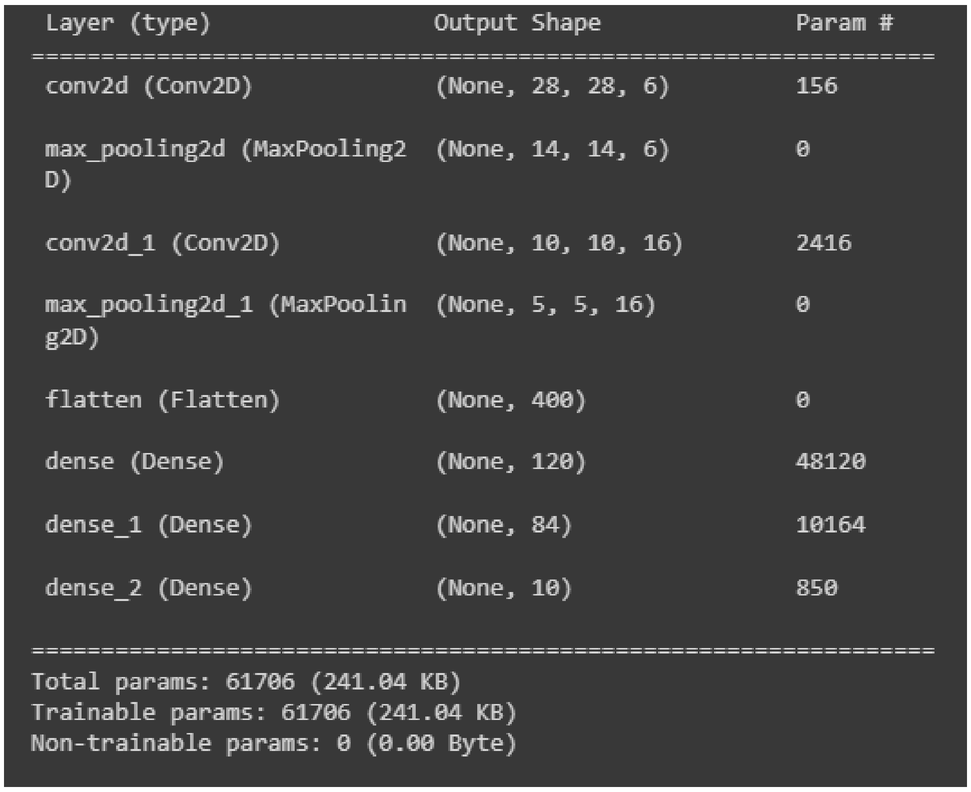
In the beginning we choose the training set is 60000 images and the testing set is 10000 images. But how to know the quality of model with unseen data. In order to evaluate model with unseen data, we use a easy method is take a mount of data from training set and evaluate the model in this set. This set is called validation test. In this model, we take 20% of the training set to make a validation set which means validation set has 12000 images and the training set has 60000 images.

2.2.2. Build model.

2.2.2.1. Create a model LeNet

We built this model based on the architecture that we discuss above.

This is the detail of the architecture model we built.



```
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 6)         156       
                                                                 
 max_pooling2d (MaxPooling2  (None, 14, 14, 6)         0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 10, 10, 16)        2416      
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 16)          0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (None, 400)               0         
                                                                 
 dense (Dense)               (None, 120)               48120     
                                                                 
 dense_1 (Dense)             (None, 84)                10164     
                                                                 
 dense_2 (Dense)             (None, 10)                850       
                                                                 
=================================================================
Total params: 61706 (241.04 KB)
Trainable params: 61706 (241.04 KB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 2.3. Detail architecture of Lenet-5

2.1.1.1. Optimizer Adam

Beside that we also use the optimizer Adam for the loss function, to explain the optimizer Adam, it likes heavy ball with friction, where the ball with mass overshoots the local minimum $\theta^+$and settles at the flat minimum θ^*

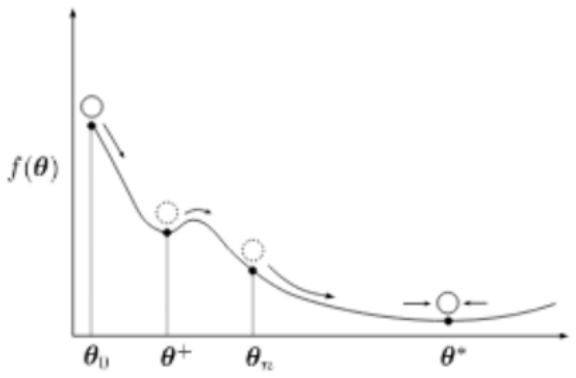


Figure 2.4. Adam optimizer

This optimizer is popular being used in deep learning because it does not take much time to oscillate around the target because it has friction, making it easier to come to a stop.

2.2.2.3. Filter we use in this model

In this model, at first convolution layers there are 6 filter like figure below to extract the feature of image.

Figure 2.5. Filters at first convolutional layer

After this layer, the feature maps in the output is like figure below

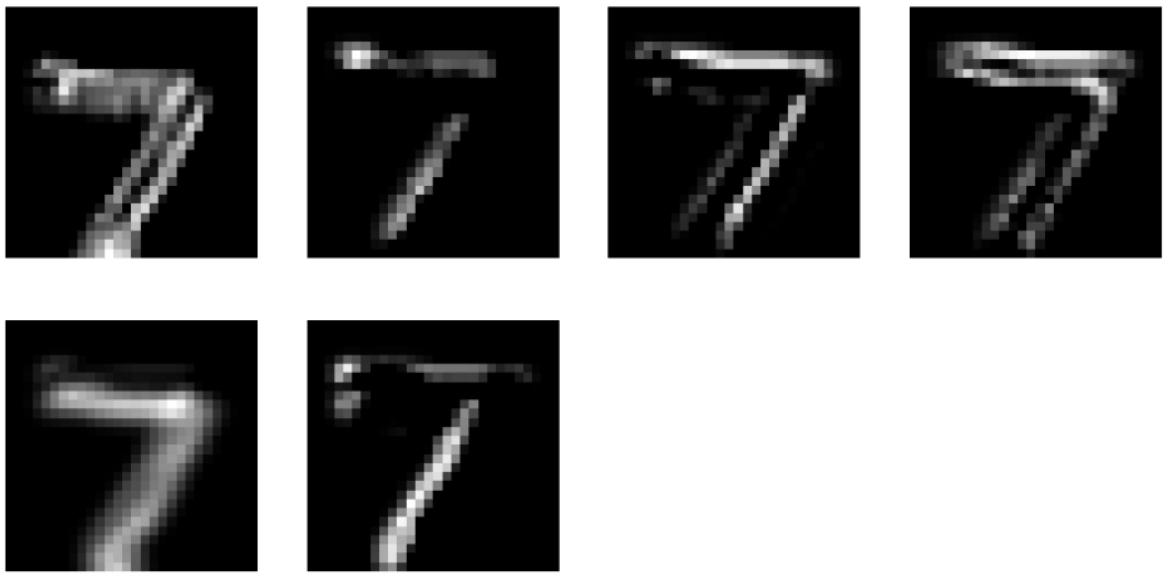


Figure 2.6. Output of first convolutional layer

As we can see, the image is being extracted to 6 feature maps that include the baisc information necessary such as the image vertically, horizontally, or corners, ... By this way, after this convolution layers it take the basic feature of image.

After first pooling, we apply the second convolution layers which has 16 filters to extract more complex feature of image like figure below.

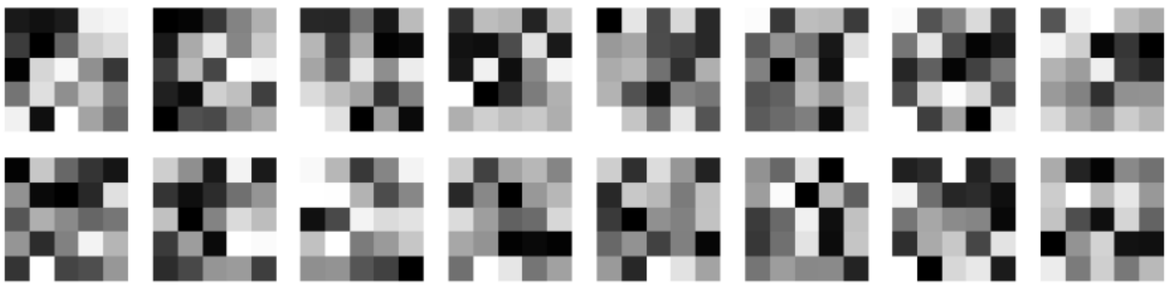


Figure 2.7. Filter at second convolutional layer

After this layer, the feature maps in the output is like figure below

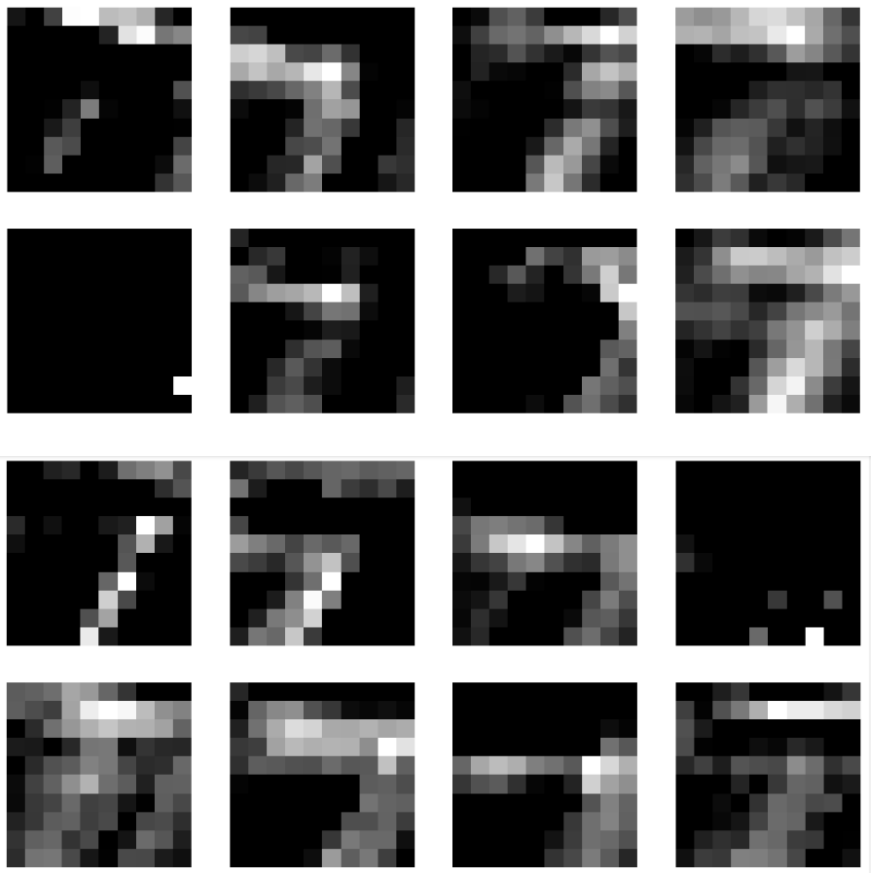


Figure 2.8. Outputs of second convolutional layer

As we can see, the image is being extracted to 16 feature maps that include the complex information such as the combined of vertical and horizontal, and also the other simple shapes from the previous feature maps, ... By this way, after this convolution layers it take the more complex feature of image.

2.2.3. Training & evaluate model

Like we discuss above, there are training set, validation set and testing set and the validation test is 20% of the training set. The data of training, we always shuffle after each epoch.

The result of training model:

Epoch	Time training	Batch size	Accuracy
50	21,167 minutes	64	98.63 %
75	29,45 minutes	128	98,9 %
100	38,05 minutes	128	99,24 %

Table 3. Result of training model

As we can see, the result at epoch 100 has the most accuracy, so we choose this model.

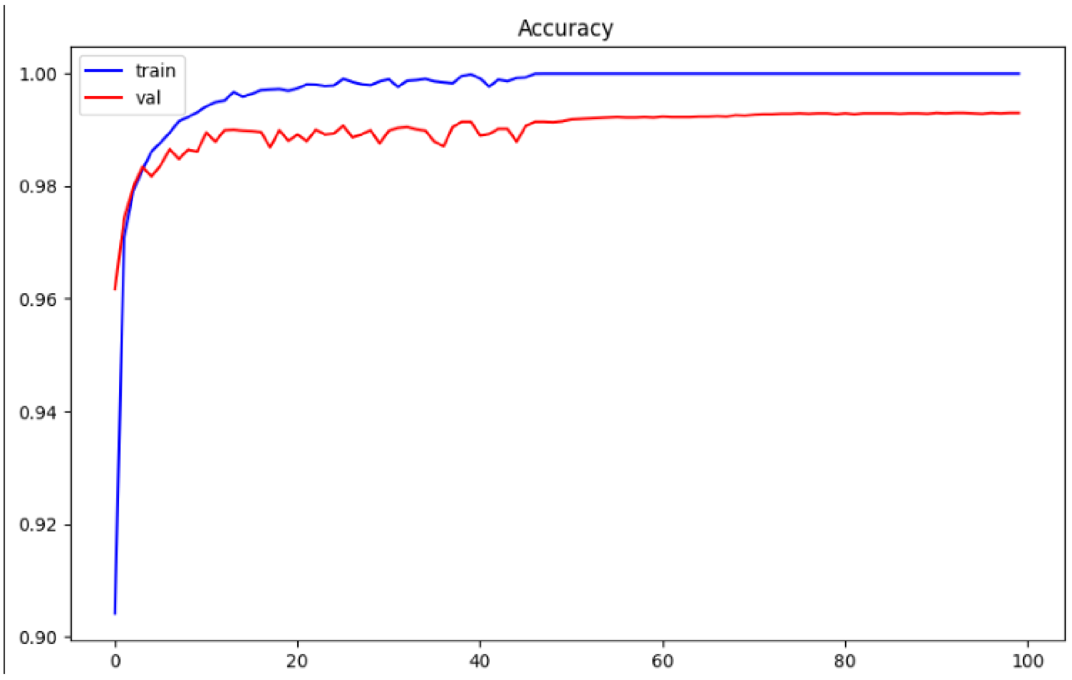


Figure 2.9. The graph of training model epoch 100

Look at graph above, we can see this model is either overfitting or underfitting so this model is quite good.

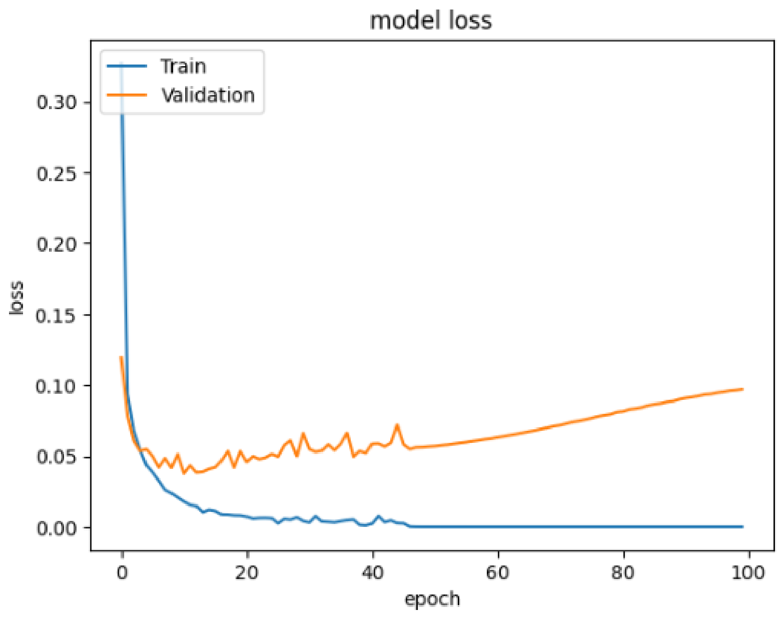
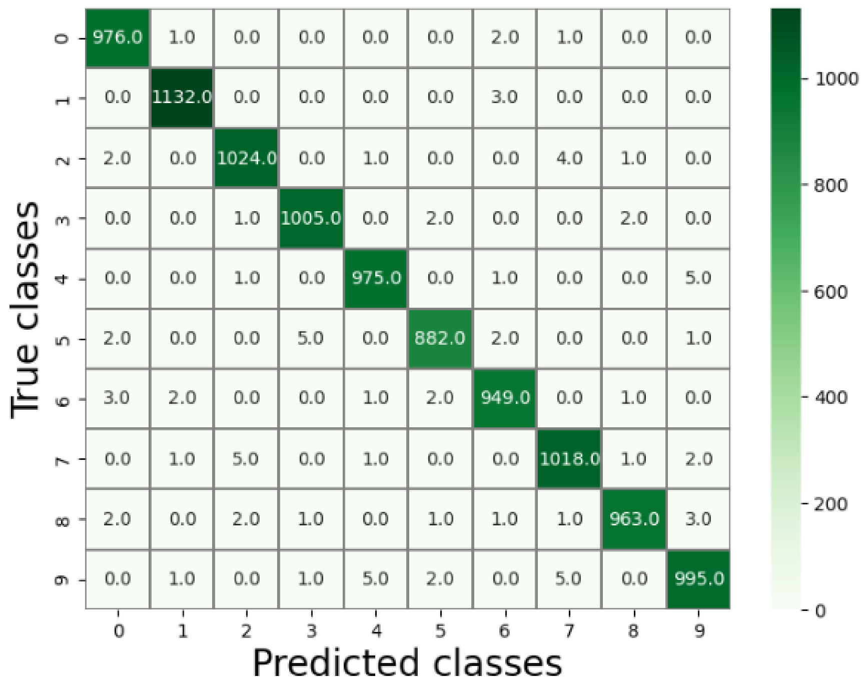


Figure 2.10. The graph of loss function

As we can see, the loss function is proportional with epoch which means when the epoch is increasing so the loss is also increasing. But it is increasing slowly so we can accept that.

We also use the confusion matrix the evaluate the model



	0	1	2	3	4	5	6	7	8	9
0	976.0	1.0	0.0	0.0	0.0	0.0	2.0	1.0	0.0	0.0
1	0.0	1132.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0
2	2.0	0.0	1024.0	0.0	1.0	0.0	0.0	4.0	1.0	0.0
3	0.0	0.0	1.0	1005.0	0.0	2.0	0.0	0.0	2.0	0.0
4	0.0	0.0	1.0	0.0	975.0	0.0	1.0	0.0	0.0	5.0
5	2.0	0.0	0.0	5.0	0.0	882.0	2.0	0.0	0.0	1.0
6	3.0	2.0	0.0	0.0	1.0	2.0	949.0	0.0	1.0	0.0
7	0.0	1.0	5.0	0.0	1.0	0.0	0.0	1018.0	1.0	2.0
8	2.0	0.0	2.0	1.0	0.0	1.0	1.0	1.0	963.0	3.0
9	0.0	1.0	0.0	1.0	5.0	2.0	0.0	5.0	0.0	995.0

Figure 2.11. Confusion matrix of model

By the confusion matrix, we can calculate the precision, f1 score and recall for the model. And there are 3 values we got:

- Precision: 0.9919
- Recall: 0.9919
- F1 score: 0.9919

Which means the model is good.

2.2.4. Test model

We will use the model to classify some data in the testing set and the result quite good.

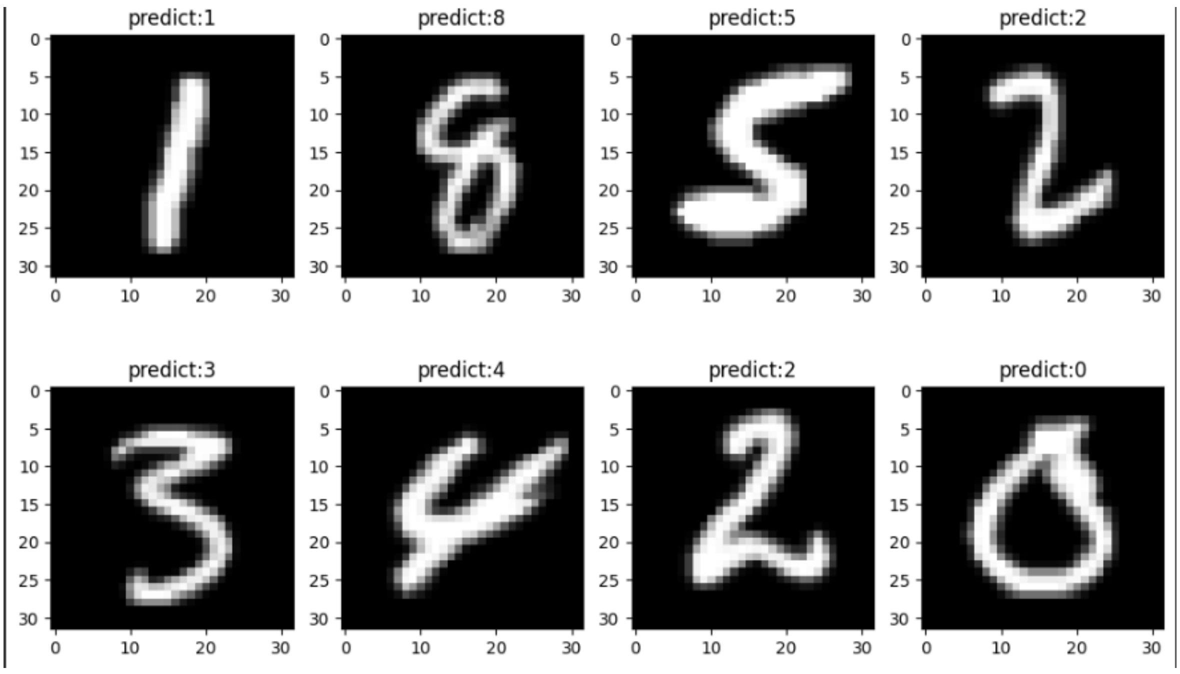


Figure 2.12.Test the data from testing set

Chương 3. HARDWARE IMPLEMENTATION

3.1. CNN Datapath IP architecture

The CNN Core IP is designed to perform image recognition, taking pixel values of an image as input and producing predicted labels for the image in one-hot encoded format as output.

3.1.1. Interface description

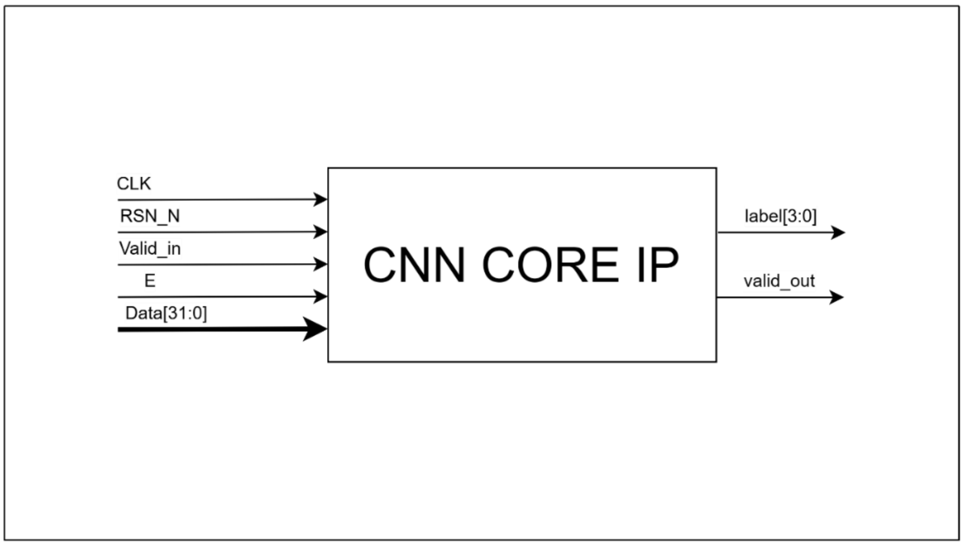


Figure 3. 1. CNN Core IP interface

Signal	I/O	Bits	Description
CLK	I	1	The system clock ticks, rising edge triggered.
RSN_N	I	1	Asynchronous reset, active low level.
Valid_in	I	1	Enable data input.
E	I	1	Enable weight loading.
Valid_out	O	32	Procedure completed.
Label	O	4	The predicted output label is what number.

Table 4. Describe the signal in the CNN Core IP block

3.1.2. Desgin description

In theory, within a CNN (Convolutional Neural Network), there are different functional blocks such as the "Convolutional layer", "Max pooling layer", "Fully connected layer" and "One-hot encoding layer." Additionally, there are supporting blocks such as the addition, multiplication, and comparison blocks.

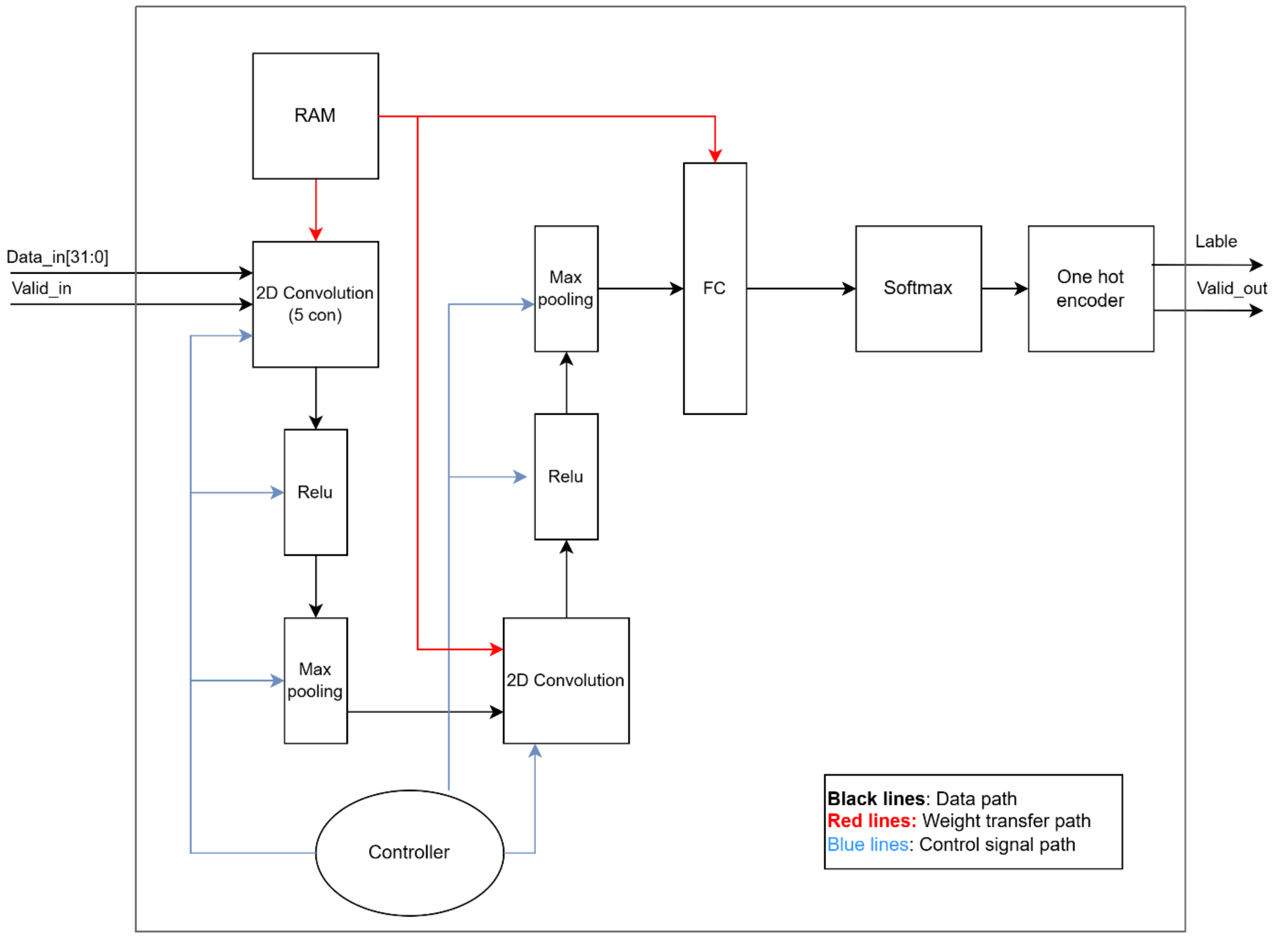


Figure 3.2. CNN Core IP block diagram

Block	Description
RAM	Memory serves the purpose of storing parameters
2D Convolution	Convolutional block is a 2-dimensional array.
Relu	ReLU block.
Max pooling	Block to find the maximum value within a sliding window of size K x K with a step size of Stride.
FC	Fully connected block performs the flatten function and performs one-dimensional matrix multiplication.
Softmax	Softmax block is responsible for calculating ratios.
One hot encoder	This module is primarily responsible for returning the predicted label of the input image.
Controller	Signal control block.

Table 5. Describe the blocks in the CNN Core IP block

3.2. Convolution Layer

3.2.1. Interface description

The Convolution 2D module functions to perform image convolution, taking input as pixels or features from the previous layer and producing convolved features as output.

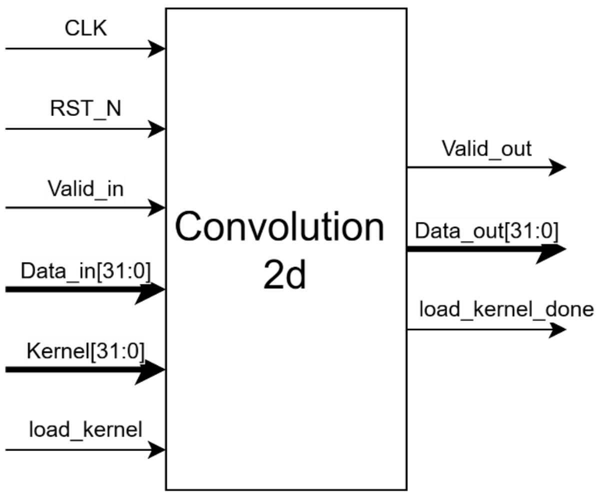


Figure 3.3. Interface of Convolution 2D

Signal	I/O	Bits	Description
Clk	I	1	The system clock ticks, rising edge triggered.
RST_N	I	1	Asynchronous reset, active low level.
Valid_in	I	1	Enable data input.
data_in	I	32	Feature data.
kernel	I	32	Weight data.
Data_out	O	32	Feature data after performing convolution.
Valid_out	O	1	Procedure completed.
Load_kernel	I	1	Signal enabling weight loading.
Load_kernel_done	O	1	Signal indicating that the weight loading process is complete.

Table 6. Describe the signal of Convolution 2D

3.2.2. Design description

In this algorithm, the convolutional block uses a 5x5 kernel size. It employs a line buffer for window sliding. Image features experience multiplication and addition through two multiplier and adder blocks.The RAM block serves the purpose of storing weights.

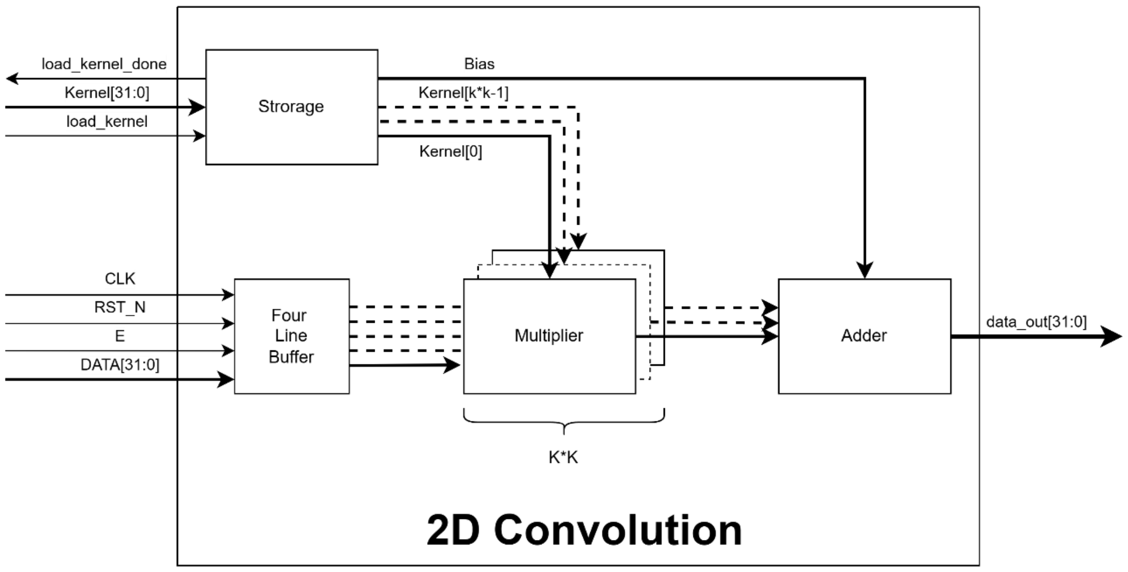


Figure 3.4. Block diagram of 2D Convolution

Block	Description
Storage	Weights loaded into this block.
K-1 Lines buffer	Used to extract matrix features, it consists of K-1 shift register blocks with K being the filter size.
Multiplier	Multiplier Block
Adders	Adder Block

Table 7. Describe the Block in 2D Convolution

3.2.2.1. Design of Line Buffer

A Line Buffer is a technique used in pipeline design for extracting matrices in convolution operations. Essentially, a Line Buffer is either a FIFO or simply a set of consecutive D Flip-flops, with a size equal to the width of the feature. Valid_out will be activated after (W*(K-1)+5) cycles.

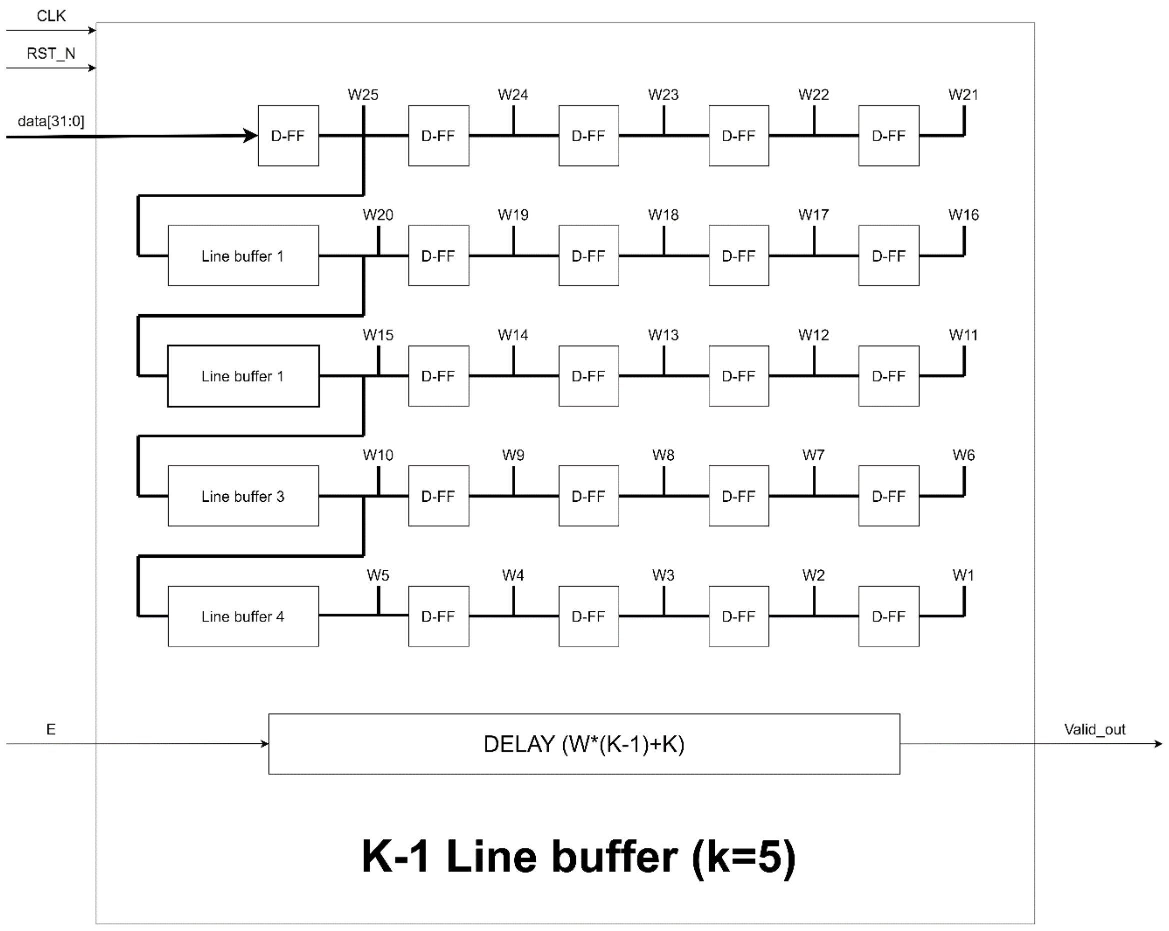


Figure 3.5. Block diagram of Line Buffer

3.3. Max pooling Layer

3.3.1. Interface description

Pooling layers are often used between convolutional layers to reduce the data size while preserving important features, helping to decrease computational load in the network. Assuming the pooling layer has a size of KxK and the input data has dimensions HxW, after pooling, it will produce features with dimensions (H/K) x

(W/K). Max pooling is typically used, meaning it identifies the most important feature in each pooling region of the input.

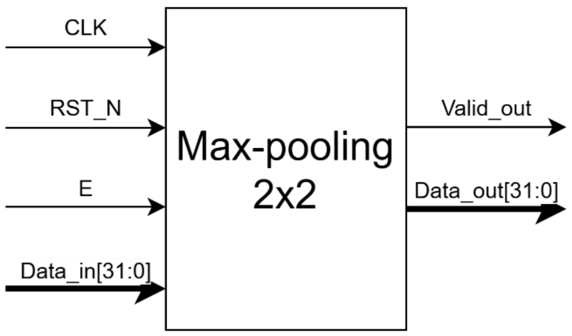


Figure 3.6. Interface of Max pooling

Signal	I/O	Bit	Description
clk	I	1	The system clock ticks, rising edge triggered.
RST_N	I	1	Asynchronous reset, active low level.
E	I	1	Enable data input.
Data_in	I	32	Input data.
data_out	O	32	Output data.
valid_out	O	1	Procedure completed.

Table 8. Describe the signal of Max pooling

3.3.2. Design description

Similar to Convolution, Max-pooling 2x2 also extracts matrices. Utilizing the Line Buffer technique, we can extract 2x2 matrices. However, the difference with Max-pooling 2x2 compared to Convolution is that the extracted matrices do not "overlap." Therefore, the module needs to check the current position of the sliding window to ensure no overlap. It then feeds the inputs into the Max module to find the maximum value among the 4 inputs.

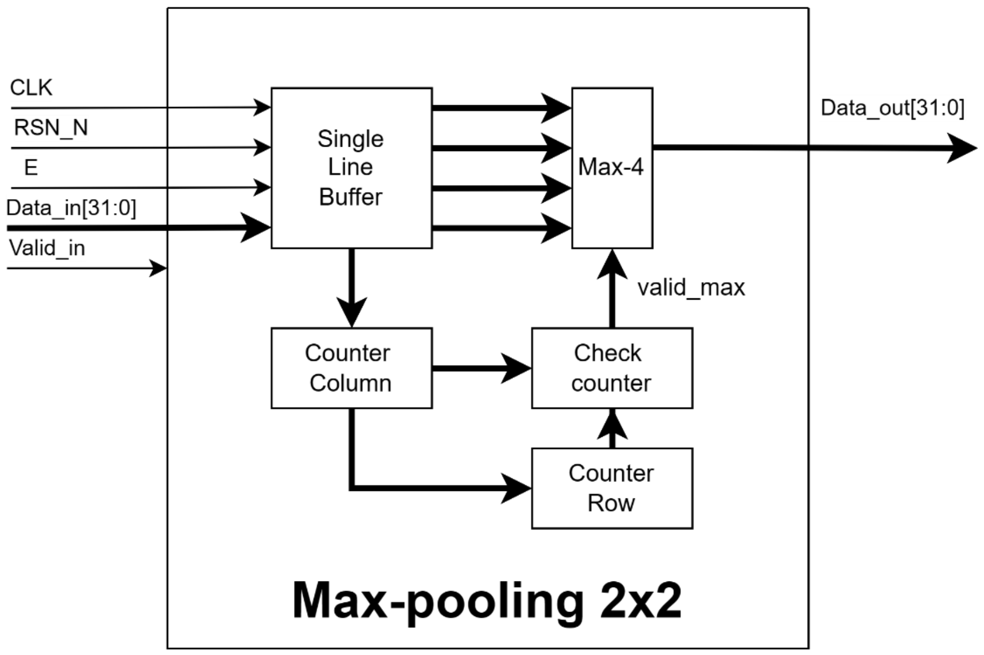


Figure 3.7. Block diagram of max pooling

Block	Desciption
Single Lines buffer	The line buffer block is responsible for extracting matrix features.
Max-4	Finding the maximum value among 4 numbers.
Counter column	Counting the position along the column of the sliding window.
Counter row	Counting the position along the row of the sliding window.
Check counter	Checking the row and column positions to ensure that sliding windows do not overlap.

Table 9. Describe the block in max pooling

3.3.2.1. Design Max

This block functions to compare the four input numbers and select the highest value as the output. In the Max module, it identifies the larger number among two inputs. In the A-B comparison, if the result is negative, then B is greater. if the result is positive, then A is greater. Therefore, the module relies on the MSB: if the MSB is 0, A is greater, and if the MSB is 1, B is greater.

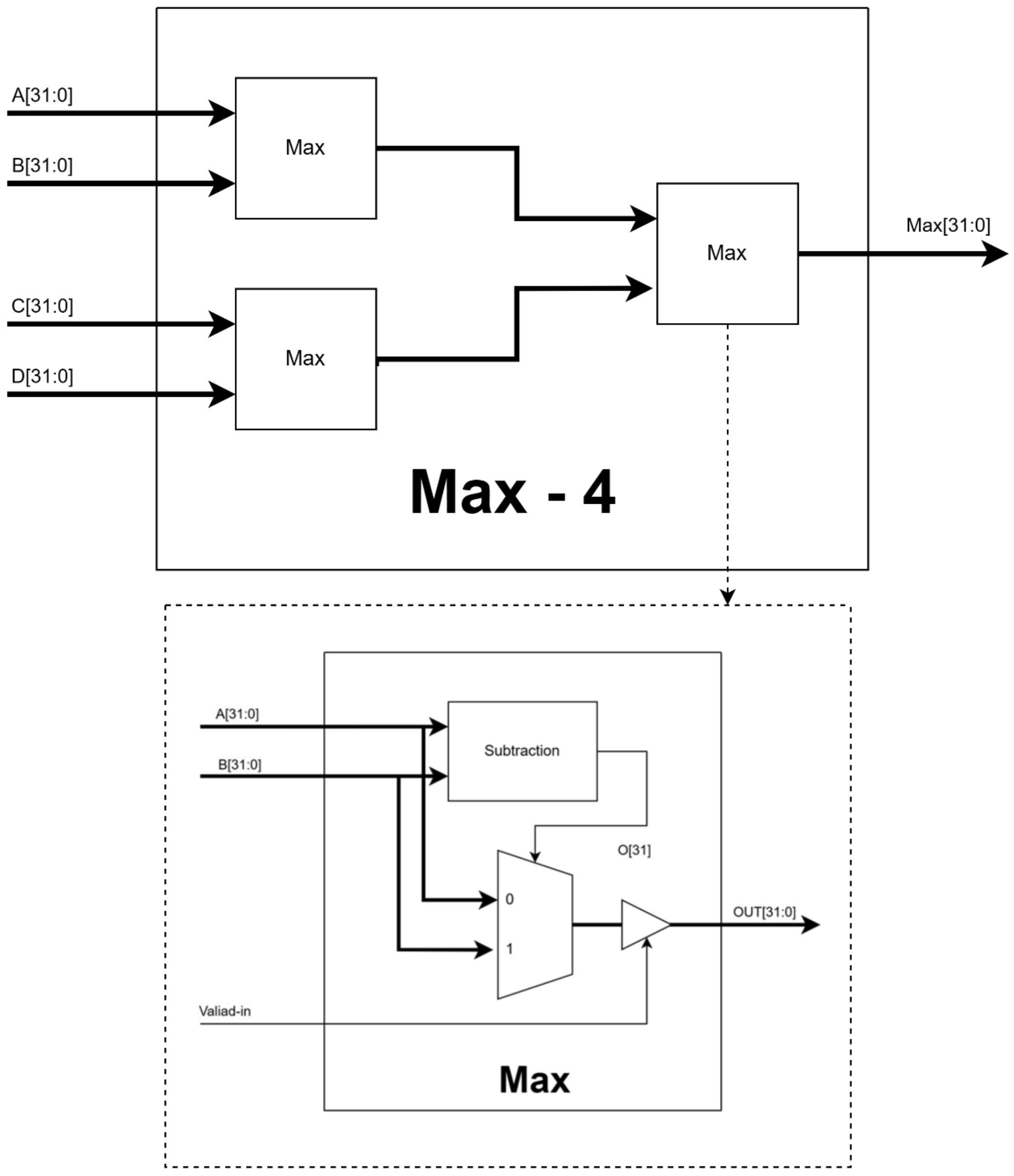


Figure 3.8. Block diagram of Max

3.4. Relu Layer

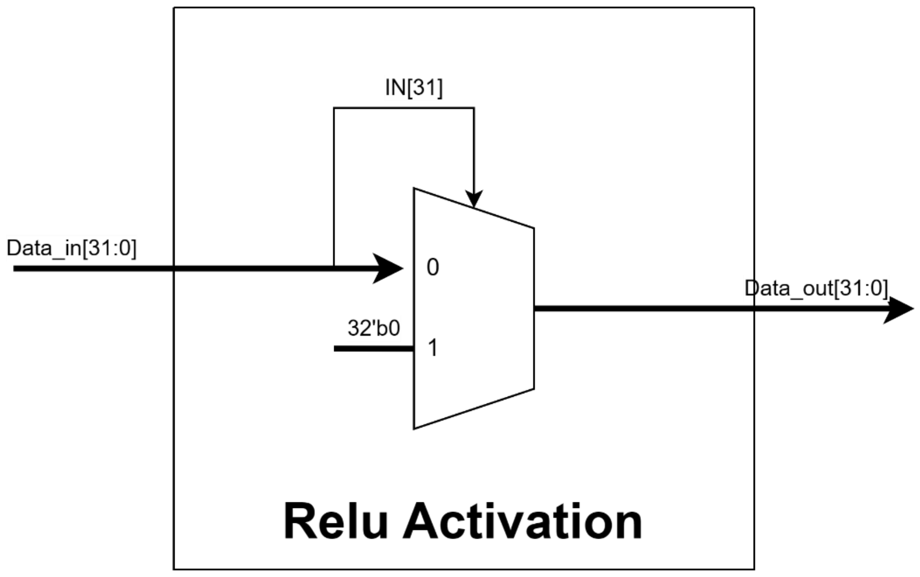


Figure 3.9. Block Diagram of Relu

The Relu activation block functions similarly to [reference to section abc xyz] and operates like a multiplexer (mux). Based on the highest weight of the input (MSB), if the MSB is 1 (indicating a negative number), the output is 0. Conversely, if the MSB is 0 (indicating a positive number), the output is the input itself.

3.5. Fully connected Layer

3.5.1. Interface description

The module performs the function of multiplying the input feature by N one-dimensional matrices, where N is the number of output nodes. The output of the module is the data of the N nodes.

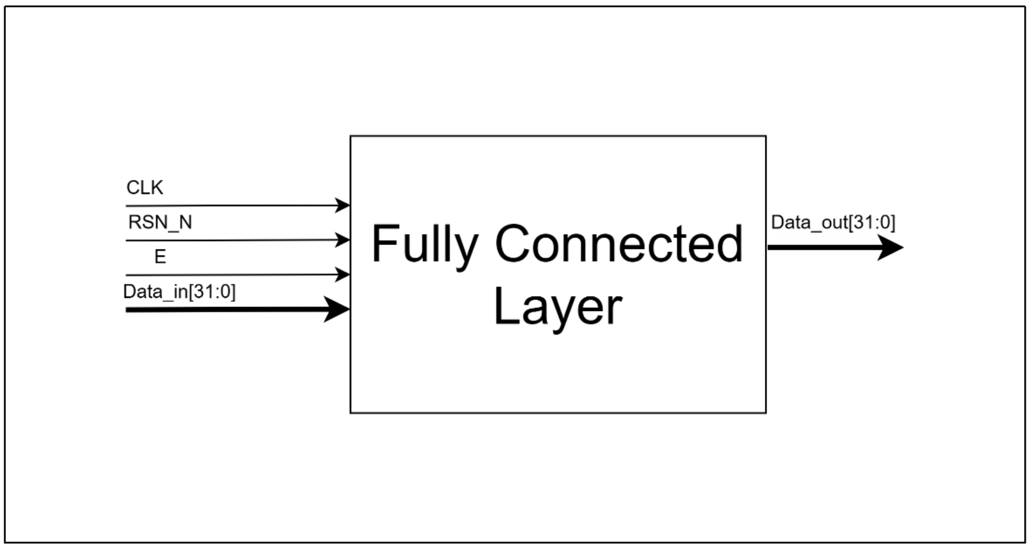


Figure 3.10. Interface of FC

Signal	I/O	Bits	Description
CLK	I	1	The system clock ticks, rising edge triggered.
RST_N	I	1	Asynchronous reset, active low level.
E	I	1	Enable data input.
Data_in	I	32	Input data
Data_out	O	32	Output data

Table 10. Descibe the signal in FC Block

3.5.2. Design description

The Fully Connected Layer is typically the last layer in a CNN. With input x of size H x W and N output nodes, this layer flattens the input, creating a one-dimensional matrix, and performs matrix multiplication with N one-dimensional matrices w, each of size H x W.

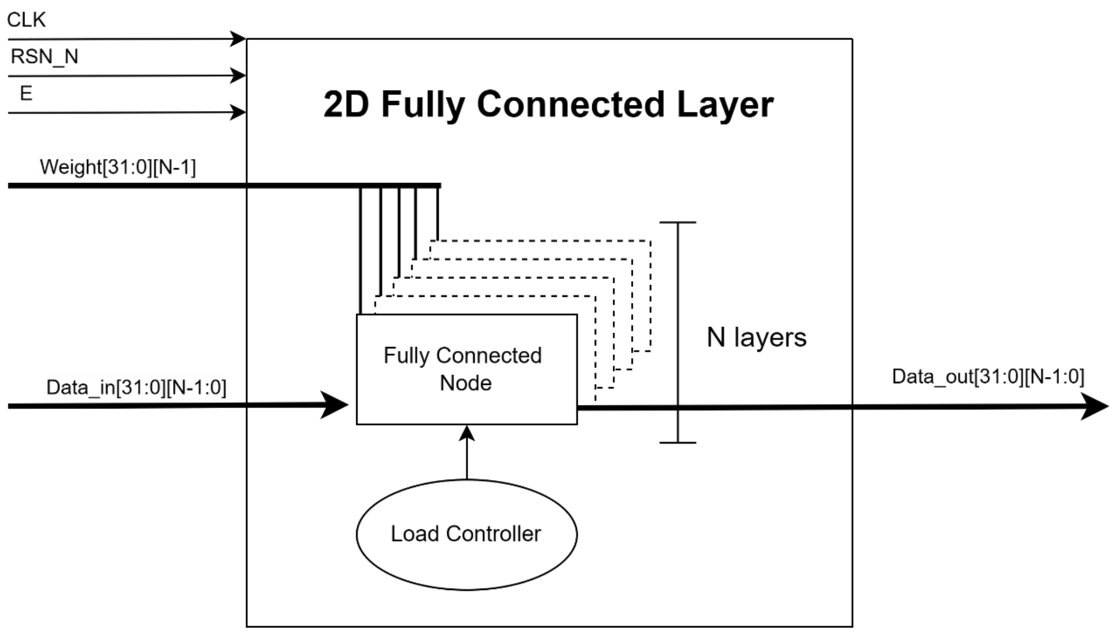


Figure 3.11. Block Diagram of FC layer

3.5.3. Fully Connected Node

The fully connected nodes will include RAM to store weights. These weights will be multiplied with the input features, and the results will be summed up.

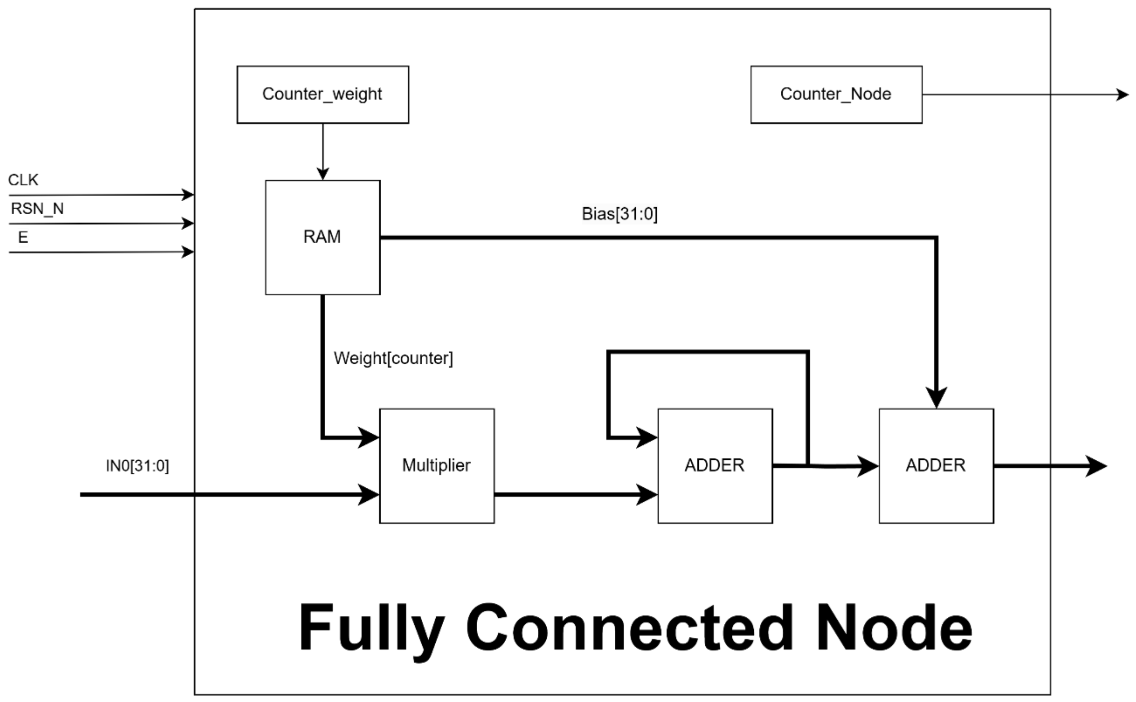


Figure 3.12. Block Diagram of FC node

Block	Description
Counter_weight	Counter for weight indexing in desired calculations.
Adder	Adder block.
Multiplier	Multiplier block.
Ram	Weight storage.

Table 11. Describe the Block in FC node

3.6. Softmax10 Layer

This block simply performs exponentiation, addition, and division operations to calculate the ratio for each label.

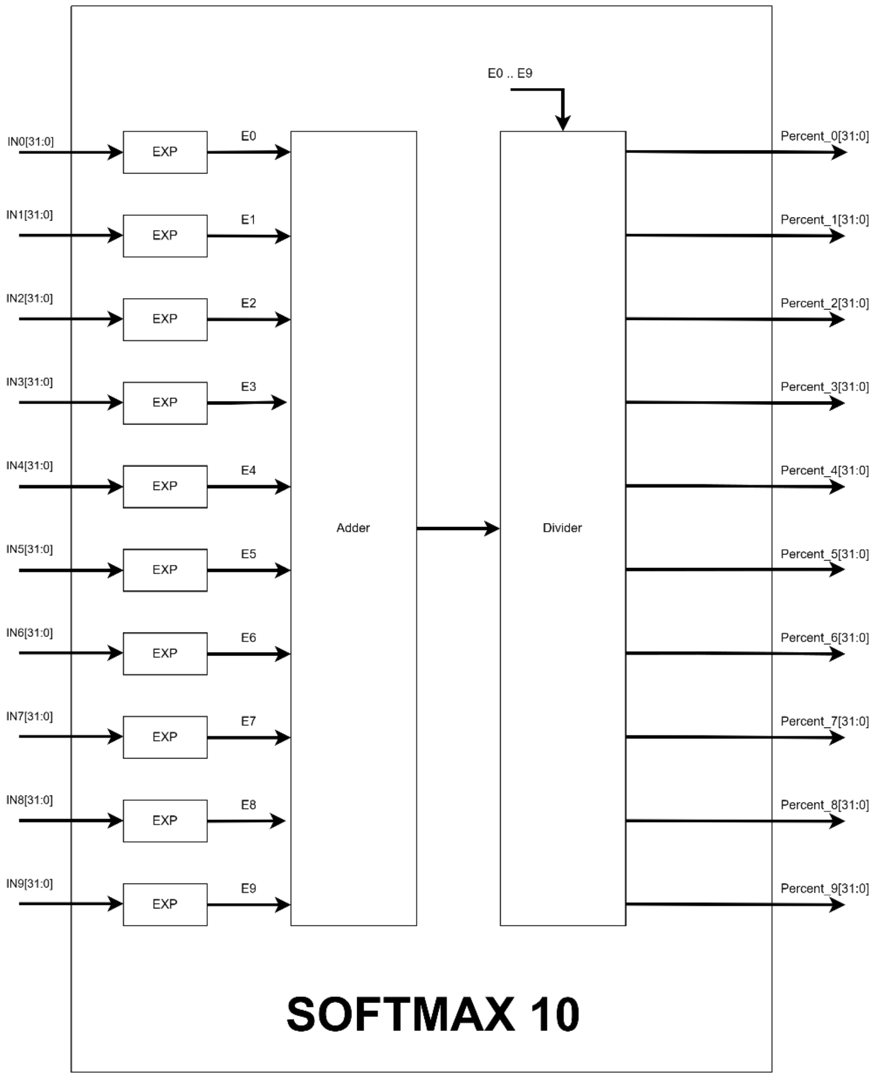


Figure 3.13. Block Diagram of Softmax

Block	Description
Exp	Exponential function is used to calculate ratios.
Adder	Block for adding N integers.
Divider	Division block.

Table 12. Descibe Block in Softmax

3.7. One-hot encoder

This block is responsible for finding the label with the highest ratio to serve as the final output, predicting the label of the input imageSimply finding the maximum among 10 numbers.

3.8. Controller (Finite State Machine):

3.8.1. Controller of Convolution

Initially, the system will be in the IDLE (waiting for startup) state, then proceed to load weights for the filters and input features. After that, it executes tasks including convolution, max pooling, fully connected (FC), softmax, and so on.

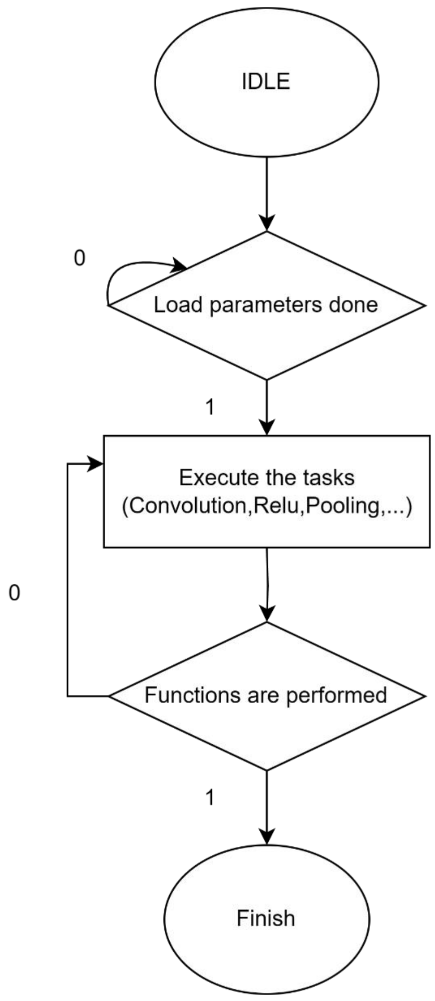


Figure 3.14. Flowchart of Convolution controller

We can look at the FSM below to better understand the operating mechanism

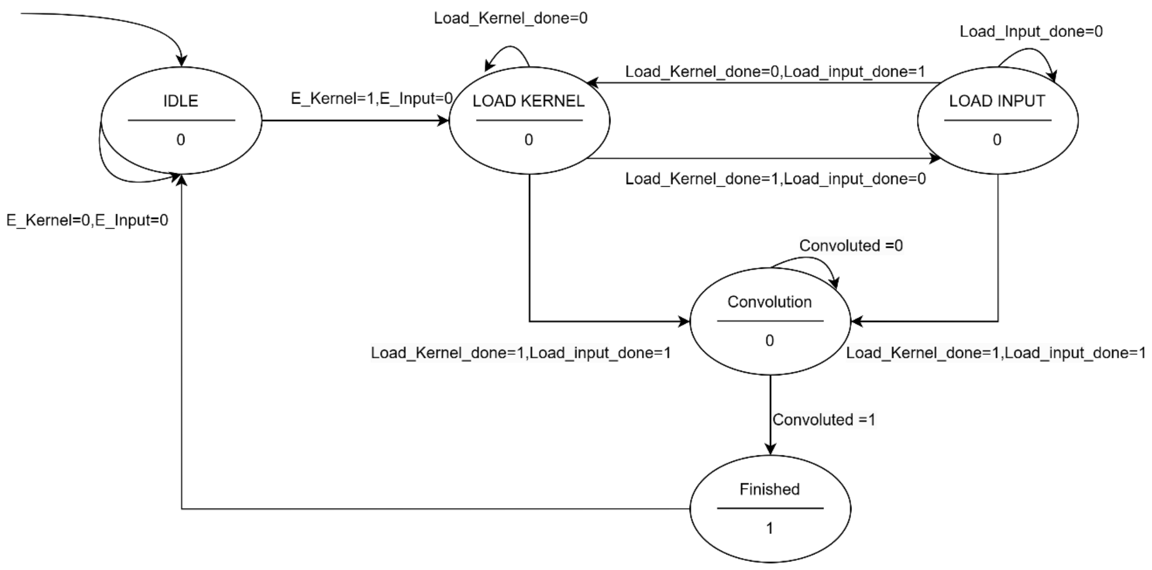
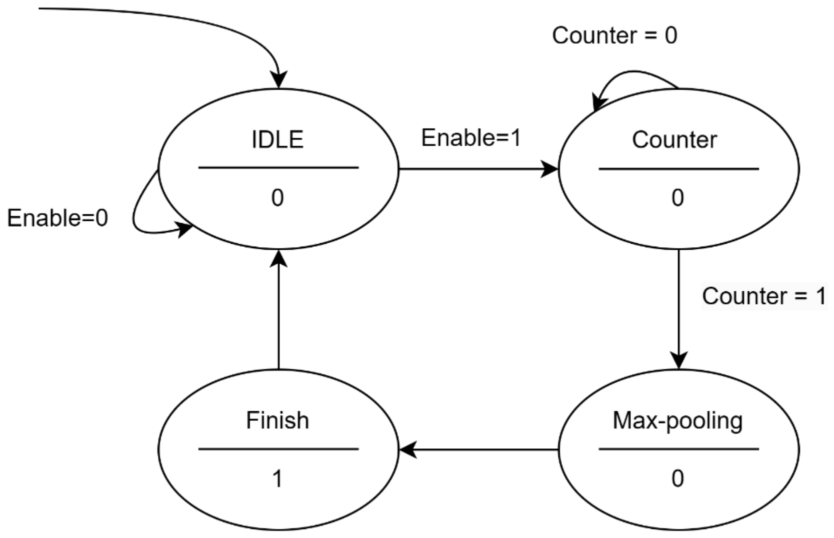


Figure 3.15. FSM of Convolution Controller

3.8.2. Controller of Max pooling

Differing from the Convolution Controller, the max pooling controller requires an additional counter to accurately tally positions for executing the 2x2 stride. The counter mechanism has been elucidated in [3.3.2].



Chương 4. VERIFICATION

4.1. Pre-synthesis verification

4.1.1. Convolution 2D

The input data is an image with dimensions HxW (in the test case mentioned, H=W=32), and the filter is of size KxK (where K=5). The valid_out signal activates after Wx(K-1)+K=133 cycles, while the load_kernel_done signal will be output after loading the complete filter, which is 25 cycles.

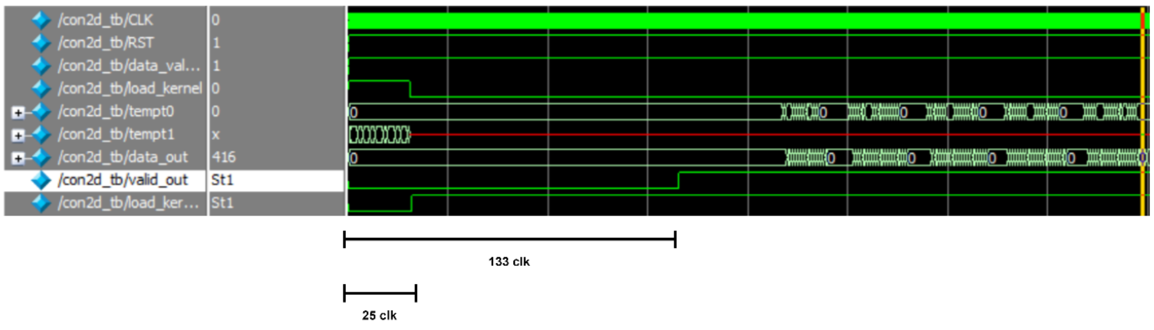


Figure 4.1. Waveform of Convolution 2D

4.1.2. Max pooling

This block will use a line buffer to extract data, resulting in a delay of Wx(K-1)+K cycles, where K is the max pooling size. Therefore, the delay will be 34 cycles. Subsequently, every time the window slides by 2 units, it takes one cycle to produce one output. If the entire row is traversed, an additional W+1 cycles are needed since it requires sliding 2x2.

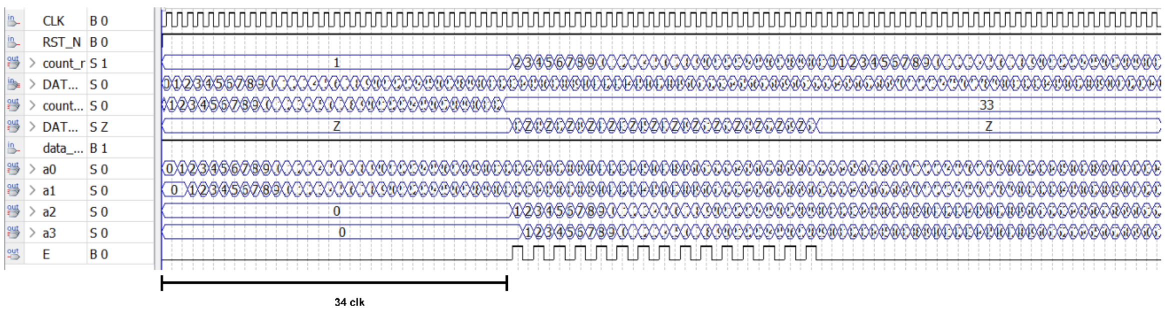


Figure 4.2. Waveform of Max pooling

4.1.3. Relu

As mentioned in section [3.4], if the input is greater than 0, the output will be the input itself. However, if it is less than or equal to 0, the output will be 0.

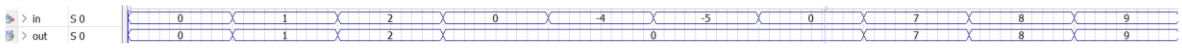


Figure 4.3. Waveform of Relu

4.1.4. Softmax

The softmax function depends on the input of the 10 labels to calculate the probabilities for these 10 labels.

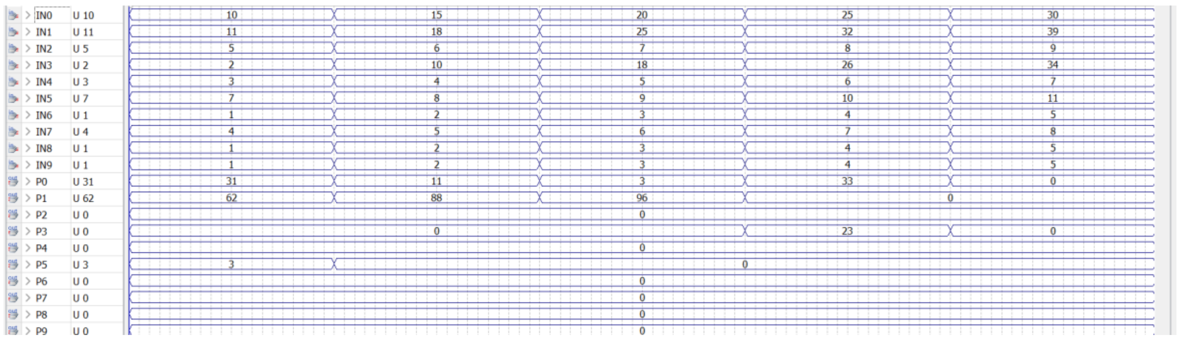


Figure 4.4. Waveform of Softmax

REFERENCES

[1] Lê Minh Đức, Bùi Hữu Trí, (2021), “RESEARCH AND EVALUATE HARDWARE IMPLEMENTATION OF CNN MODEL”, University of Information Technology.

[2] https://cs231n.github.io/convolutional-networks/

[3] https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns

[4] MNIST database Wikipedia, https://en.wikipedia.org/wiki/MNIST_database

[5] Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). “*Gradient-based learning applied to document recognition*”, 86(11), 2278–2324. DOI:10.1109/5.726791

[6] https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning

[7] Luiz G. Hafemann; Luiz S. Oliveira; Paulo Cavalin, (2014) “*Forest Species Recognition Using Deep Convolutional Neural Networks*”, 2014 22nd International Conference on Pattern Recognitio. DOI: 10.1109/ICPR.2014.199

[8] https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

[9] Julien Bringer, Hervé Chabanne, Linda Guiga (May 2020), “Premium Access to Convolutional Neural Networks”,

URL:https://www.researchgate.net/publication/341615524_Premium_Access_to_Convolutional_Neural_Networks

[10] https://dlapplications.github.io/2018-06-11-perceptron/

[11] https://engineering.admicro.vn/coatnet/

[12] Illinois University,lecture cnn architectures

[13] IEEE 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) - Pittsburgh, PA, USA (2016.7.11-2016.7.13)] 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) - Angel-Eye: A Complete Design Flow for Mapping CNN onto Customized Hardware | 10.1109/ISVLSI.2016.129

[14] DASH: Design Automation for Synthesis and Hardware Generation for CNN | 10.1109/icfpt51103.2020.00019

[15] ACM Press the International Conference - San Diego, California (2018.11.05-2018.11.08)] Proceedings of the International Conference on Computer-Aided Design - ICCAD '18 - DNNBuilder | 10.1145/3240765.3240801