Handwritten Alphabet Recognition

by

Arun Kumar Rathod (2020-BCS-014), Raj Vardhan Bundela (2020-BCS-062), Rohit Kumar (2020-BCS-065)

A report submitted for VLSI Project

Bachelor of Technology

in

Computer Science and Engineering

विश्वजीवनामृतं ज्ञानम्

ATAL BIHARI VAJPAYEE-

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT

GWALIOR - 474015, MADHYA PRADESH, INDIA

Abstract

This project is an implementation of research work on Hand Written Character classification that is based on Artificial neural networks that operate on the pixles value of the image whose character has to be recognized. This model uses the Artificial Neural Network which mimicks the human brain and how neuron works.

Acknowledgments

I am extremely grateful to Dr. Gaurav Kaushal for giving me the freedom to develop and experiment with new ideas. I would like to take this opportunity to express my sincere gratitude to them for their academic and personal mentoring, their interest in my idea, and the ongoing support, motivation, and confidence-building sessions that were very successful and helped me gain confidence and trust in the growth and development of the current work primarily because of their insightful advice, suggestions, good judgment, and constructive criticism as well as their desire for excellence. My mentors never let me feel like a newbie by always listening to my opinions, respecting and enhancing them, and providing me complete freedom in my project. They always responded to all of my questions with a smile and an abundance of patience. The current work has only progressed to this far because to their intense interest and supportive demeanour.

Contents

Tables

Table

Figures

Figure

Chapter 1

Introduction

In this machine learning project, we will recognize handwritten alphabets, i.e, English alphabets from A-Z. This we are going to achieve by modeling a neural network that will have to be trained over a dataset containing images of alphabets. Also, we would like to know the hardware requirements for doing the classification.

1.1 Context

For the VLSI project at the Indian Institute of Information Technology and Management in Gwalior includes this study/implementation. In order to make this project, We've explored the concepts of Artificial Neural Networks and Image Processing.

1.2 Problem

In recent years, handwriting identification has been one of the most exciting and hard research areas in image processing and pattern recognition. It makes a significant contribution to the advancement of an automation process and can improve the interaction between man and machine in a variety of applications. Several studies have been conducted to develop new strategies and methodologies for reducing processing time while improving recognition accuracy.

1.3 Objectives

This project can be used for recognising all English alphabets in an input image. When a character input image is presented to the proposed system, it will recognise the character in the image. Neural Networks are used to recognise and classify characters. The primary goals of this project are to use the Artificial Neural Network approach to effectively recognise a certain character of type format and get the hardware requirements for doing the same.

1.4 Research work flow

According to the research objectives, the report will describe the work flow as below:

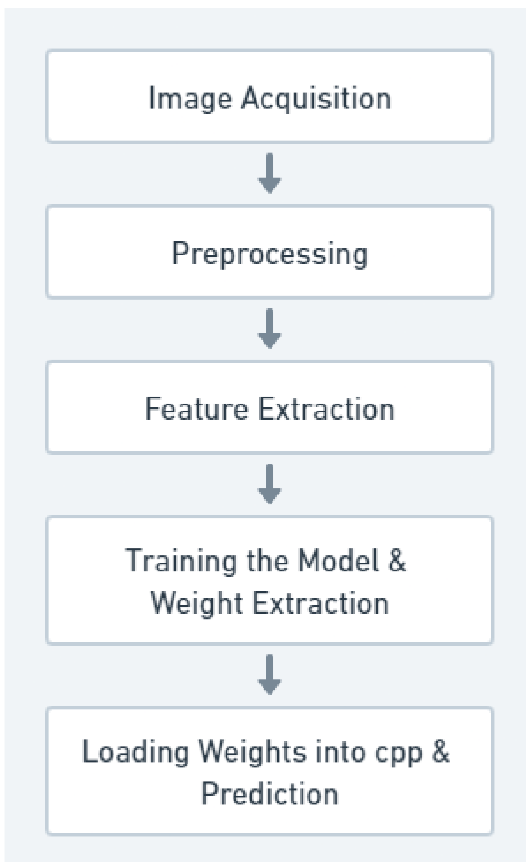

Figure 1.1: Workflow of the project

Step 1 We load the dataset and extract the pixels value of the gray-scale images of the images of the character. Each value lies between 0 to 255. The matrix which represents an image is of the size 28x28 which is, 784 pixels.

Step 2 We extract pixels value and separate the data-set of into three sections, namely test set, validation set, and trainset. In this case, we are aware of the labels for both the trainset and validationset. We use the trainset and validation set for fine-tuning and updating the hyper-parameters for the model with a biased approach. And in the end when the model is ready, we use the test set to check the model's prediction rate in an unbiased approach.

Step 3 We design the ANN model and feed the vector containing the pixel value of images as the input to train our model. Before giving the input values to the input layer, we first flatten the grey-scale matrix into 1-D vector of size 784. Then this vector is given as an input to the input layer if the model. The input layer also contains 784 neurons, hence 1 value per neuron.

Step 4 After getting the ideal weights, extract the weights of model and save the weights and bias for each layer in a excel file.

Step 5 We read the csv file and load the bias and weights into the cpp file and we define function in cpp file to use those weights to do prediction on the input data.

Chapter 2

Methodology

This section introduces the hypothesis and the analytical validation of the proposed solution.

2.1 Proposed hypothesis

This work explores two aspects. Machine learning and image processing The complete machine learning idea was not utilised here, but we did deal with a machine learning building element called a neural network. These two subjects are appropriate research topics, and many academics and professors work with them every day to improve approaches or algorithms or to develop new algorithms. The project's extension can be utilised on a broad scale to recognise written characters in photos and extract them quickly. In banking, for example, signature recognition, licence plate verification, real-time image chaining or filtering, object identification, and so on.

2.2 Artificial Neural Network

One of the most important tools in machine learning is artificial neural networks. They are brain-inspired systems designed to mimic how humans learn, as the ”neural” portion of their name suggests. Neural networks are made up of input and output layers, as well as a hidden layer made up of units that turn the input into something usable by the output layer. They're great for detecting patterns that are far too complex or numerous for a human programmer to extract and train the computer to recognise. The concept of ANNs is founded on the premise that by building the proper connections, the workings of the human brain may be replicated using silicon and wires as real neurons and dendrites. ANNs are made up of numerous nodes that mimic biological neurons in the human brain. The neurons are linked together and interact with one another. The nodes can accept input data and conduct basic operations on it. The outcome of these activities is sent to other neurons. Each node's output is referred to as its activation or node value. Each link has a weight connected with it. ANNs have the ability to learn by changing their weight values. The diagram below depicts a basic ANN.

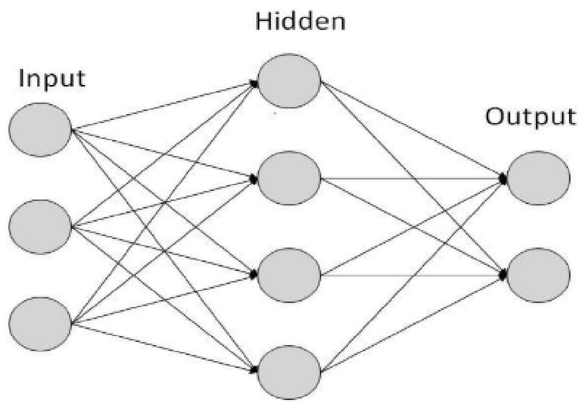

Figure 2.1: Artificial Neural Network

The information flow in this ANN is unidirectional. A unit delivers information to a unit from which it receives no information. There are no feedback loops present. They are employed in the generation, recognition, and classification of patterns. They have predetermined inputs and outputs.

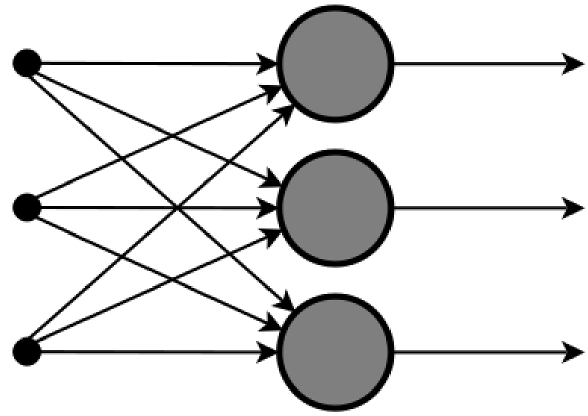

Figure 2.2: FeedForward Neural Network

2.2.1 Activation Function

An Activation Function determines whether or not a neuron should be activated. This means that it will use simpler mathematical operations to determine whether the neuron's input to the network is essential or not throughout the prediction phase. Sigmoid Activation Function: The Sigmoid Function curve looks like an S-shape. The key reason why we employ the sigmoid function is that it occurs between (0 to 1). As a result, it is particularly useful for models that require us to anticipate the probability as an output. Because the probability of anything occurs only between 0 and 1, the sigmoid is the best choice. The function can be differentiated. That is, we can calculate the slope of the sigmoid curve between any two points. The function is monotonic, but its derivative is not.

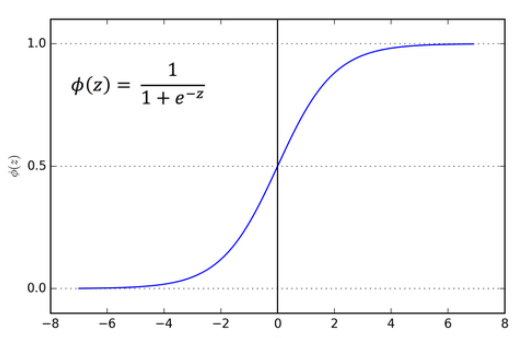

Figure 2.3: Sigmoid Activation Function

2.3 Python Code:

```
#Mounting The Google Drive
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Importing the libraries into the file.
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.model_selection import train_test_split
from keras.utils import np_utils
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from mlxtend.evaluate import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from matplotlib.pyplot import subplots
from keras.callbacks import *

#Reading the csv file which has stored the pixel value of all the images of datase
dataset = pd.read_csv("/content/drive/MyDrive/A_Z_Handwritten_Data.csv").astype('fl
dataset.rename(columns={'0':'label'}, inplace=True)

#Pixel values of Images
```

```
x= dataset.drop('label',axis = 1)

#Seperating the column which has labels of all the data items
y = dataset['label']

# We are spliting the dataset into train,test and validation dataset.

x_train,x_test,y_train,y_test = train_test_split(x,y)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
x_train,inputs_validation,y_train,targets_validation=train_test_split(x_train,y_tra

#Scaling the values of the pixels in the different dataset.
standard_scaler = MinMaxScaler()
standard_scaler.fit(x_train)

x_train = standard_scaler.transform(x_train)
x_test = standard_scaler.transform(x_test)

y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

#Defining the Architecture Of the ANN Model to be used.

model = keras.Sequential()
model.add(layers.Dense(100, activation="relu" , input_dim = x_train.shape[1]))
model.add(layers.Dense(len(y.unique()), activation="softmax"))
```

```
adam = keras.optimizers.Adam(learning_rate=0.001,decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=adam , metrics = ['Accurac

#Training the model with the training dataset

model.fit(x_train,y_train,epochs=5)

#Evaluating the Model on Test Dataset
model.evaluate(x_test,y_test)

#Storing the predicted values of data items in test dataset.
y_pred=model.predict(x_test)

#Function To get appropriate Alphabet based on the predictions of the Model
#The model returns an array of 26 size describing the probabilities of each charac
#We take the character with max probability as the predicted result for that sampl

def get_alphabet(y):
  s="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  val=[0]*len(y)
  for j in range (len(y)):
    val[j]=s[np.argmax(y[j])]
  return val

#Saving the model in the drive to later extract weight from it.

model.save_weights('gfgModelWeights.h5')
```

```
print('Model Saved!')

#Calling the get_alphabet() to get the respective value of alphabets.
y_test=get_alphabet(y_test)
y_pred=get_alphabet(y_pred)

#Displaying the confusion matrix based upon the values of Predicted value and actu

disp = metrics.ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
fig = disp.ax_.get_figure()
fig.set_figwidth(10)
fig.set_figheight(10)

loc = "./" # save location
shape_dict = {} # (layer name:shape) save dictionary

#To print the value of weights in each layer of model.
for layer in model.layers:
  print(layer.get_weights())
  print("layer finished")

for layer in model.layers:
    if layer.get_weights() != []:
        shape_dict[layer.name] = np.shape(layer.get_weights()[2]) # No bias, only 
        np.savetxt(loc + layer.name +".csv", layer.get_weights()[2].flatten(), deli
```

Chapter 3

Experiments and results

In this chapter we have tested the accuracy of the ANN model that were discussed in previous chapters and for carrying out the Alphabet classification we have used the NIST dataset which has gray-scale images 26 capital alphabet . The collection has around 0.4 million images in total. The characters are evenly distributed images of all the characters as shown in figure 3.1, with 0.4 million images in total. [?] .

NIST Dataset Link

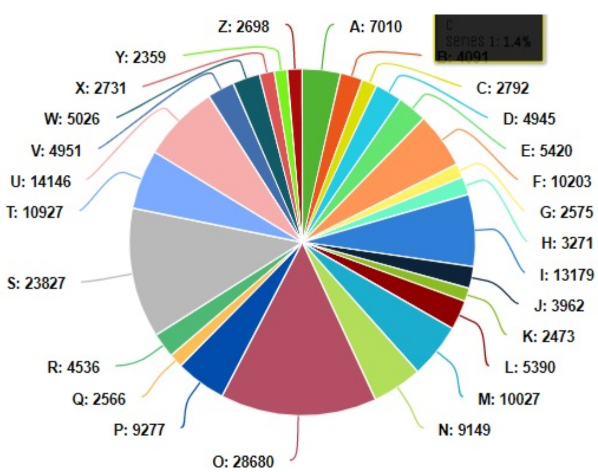

Figure 3.1: Class Distribution Of Various Characters in the data-set

3.1 Artificial Neural Network-

The artificial neural network we have designed consists of a input layer, a densely connected hidden layer and a output layer consisting of 26 neuron as there are 26 alphabets in which a value can be classified into.The activation function we have used is softmax.We used the nist dataset for character classification.The images used is of dimensions 28x28 resolution. And after flattening we get an array of 784 pixel values.We feed this layer into the ANN model and we train our model.

3.2 Results of ANN Model-

Our ANN was able to recognise the never seen before alphabet from the test set with an accuracy of 97 percent and was successfully able to recognize 72000 data samples.And as the number of epochs were increasing we can see the anticipated decrease in the loss values and the increase in the accuracy of the model as shown in fig 3.2 and fig 3.3. The fig 3.4 depicts the confusion matrix of the model to summarize the predictions of the model. Through fig 3.4 we saw that our model misinterpreted the character "A" with characters "B","H","K","N", "R" and "K" and variation of the other alphabets were recorded as listed in the fig 3.5.

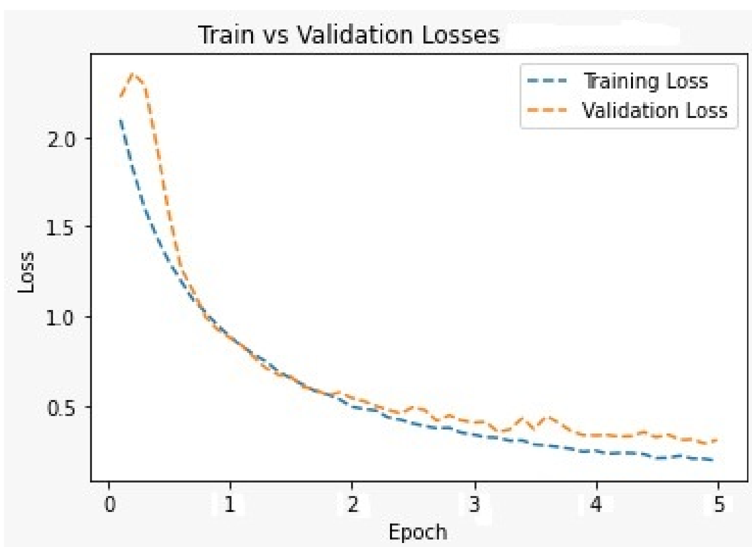

Figure 3.2: Loss comparison Of ANN Model

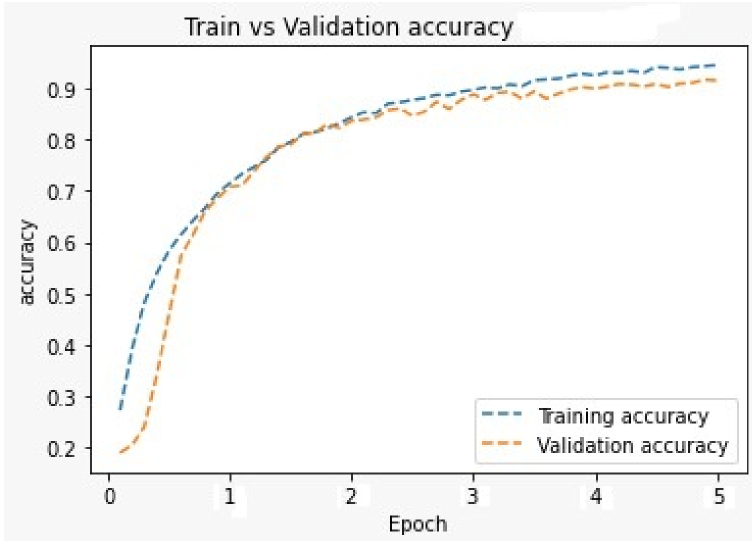

Figure 3.3: Accuracy comparison Of ANN Model

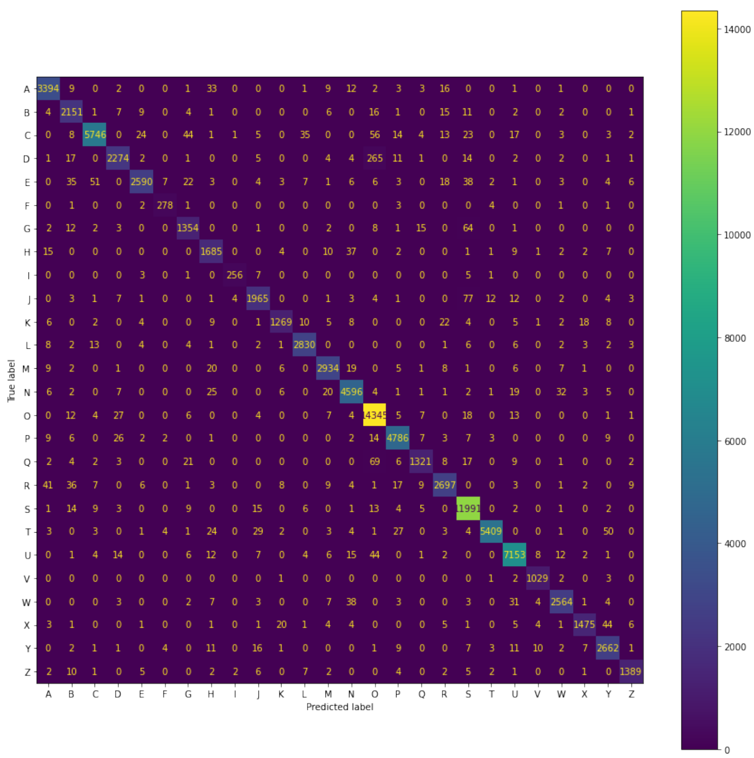

Figure 3.4: Confusion Matrix of Predicted Result

Character	Attempts	Correctly Classified	Accuracy	Misclassified With
A	2,774	2,696	97.2%	'B', 'H', 'K', 'N', 'R', 'X'
B	1,734	1,572	90.7%	'A', 'D', 'E', 'G', 'H', 'R', 'S'
C	4,682	4,544	97.1%	'E', 'G', 'L', 'O'
D	2,027	1,776	87.6%	'B', 'O', 'P', 'Q'
E	2,288	2,115	92.4%	'B', 'C', 'F', 'G', 'K', 'S'
F	233	210	90.4%	'E', 'P', 'T'
G	1,152	1,050	91.1%	'B', 'C', 'E', 'O', 'Q'
H	1,444	1,278	88.5%	'A', 'B', 'K', 'N', 'R'
I	224	196	87.4%	'J', 'L', 'T', 'Z'
J	1,699	1,589	93.5%	'I', 'T', 'Z'
K	1,121	1,008	90.0%	'A', 'E', 'H', 'M', 'N', 'R', 'X', Y
L	2,317	2,255	97.3%	'C', 'I'
M	2,467	2,337	94.7%	'K', 'N', 'W'
N	3,802	3,606	94.9%	'A', 'H', 'K', 'M', 'R'
O	11,565	11,338	98.0%	'C', 'D', 'G', 'Q'
P	3,868	3,778	97.7%	'D', 'F', 'R'
Q	1,162	1,009	86.8%	'D', 'G', 'O'
R	2,313	2,144	92.7%	'A', 'B', 'H', 'K', 'N', 'P'
S	9,684	9,481	97.9%	'B', 'E'
T	4,499	4,430	98.5%	'F', 'I', 'J'
U	5,802	5,658	97.5%	'V', 'W'
V	836	810	96.8%	'U', 'W', 'Y'
W	2,157	2,017	93.5%	'M', 'U', 'V'
X	1,254	1,163	92.7%	'A', 'K', 'Y'
Y	2,172	2,055	94.6%	'K', 'X'
Z	1,215	1,162	95.6%	'I', 'J'

Figure 3.5: Variation Of Results

3.3 Vivado-

In C/C++ code, all input and output operations are performed, in zero time, through formal function arguments. In a RTL design, these same input and output operations must be performed through a port in the design interface and typically operate using a specific input/output (I/O) protocol. In C++, a function starts to process data when the function is called from a parent function. The function call is pushed onto the stack when called, and removed from the stack when processing is complete to return control to the calling function. This process ensures the parent knows the status of the child.

Since the host and kernel occupy two separate compute spaces in the Vitis kernel flow, the "stack" is managed by the Xilinx Run Time (XRT), and communication is managed through the s_axilite interface. The kernel is software controlled through XRT by reading and writing the control registers of an s_axilite interface as described in S_AXILITE Control Register Map.

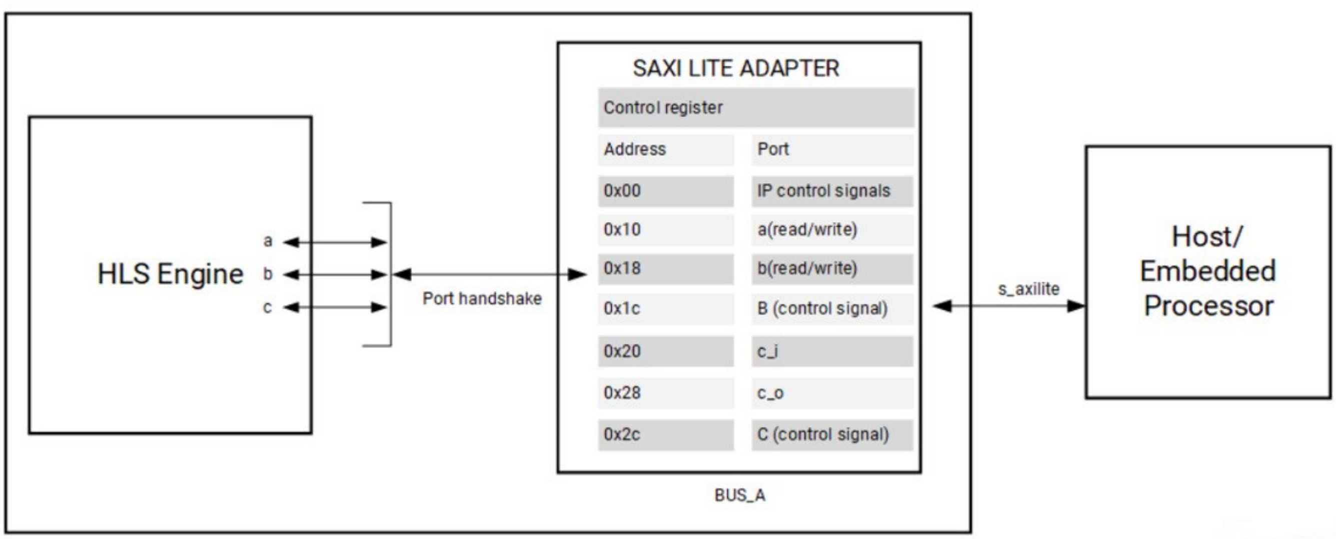

Figure 3.6: Interface

3.4 Vivado Results-

Performance Estimates

- **Timing (ns)**
  - **Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.451	1.25

- **Latency (clock cycles)**
  - **Summary**

Latency		Interval		Type
min	max	min	max	
1134163	1134163	1134163	1134163	none

  - **Detail**
    - **Instance**

N/A

    - **Loop**

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- memcpy_lay1	156899	156899	1569	-	-	100	no
+ memcpy_lay1	1567	1567	2	-	-	784	no
- Loop 2	200	200	2	-	-	100	no
- Loop 3	946900	946900	9469	-	-	100	no
+ Loop 3.1	9408	9408	12	-	-	784	no
- Loop 4	30160	30160	1160	-	-	26	no
+ Loop 4.1	1100	1100	11	-	-	100	no

Figure 3.7: Performance Estimate.

- **Summary**

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	573
FIFO	-	-	-	-
Instance	0	34	5993	8898
Memory	522	-	0	0
Multiplexer	-	-	-	895
Register	-	-	960	-
Total	522	34	6953	10366
Available	40	40	16000	8000
Utilization (%)	1305	85	43	129

- **Detail**
  - **Instance**

Instance	Module	BRAM_18K	DSP48E	FF	LUT
hand_chrc_nn_CRTL_BUS_s_axi_U	hand_chrc_nn_CRTL_BUS_s_axi	0	0	82	120
hand_chrc_nn_daddibs_U6	hand_chrc_nn_daddibs	0	3	509	1165
hand_chrc_nn_ddivjbC_U7	hand_chrc_nn_ddivjbC	0	0	3211	3644
hand_chrc_nn_dexpkbM_U8	hand_chrc_nn_dexpkbM	0	26	1549	2597
hand_chrc_nn_fadddEe_U1	hand_chrc_nn_fadddEe	0	2	205	390
hand_chrc_nn_fcmphbi_U5	hand_chrc_nn_fcmphbi	0	0	66	239
hand_chrc_nn_fmuleOg_U2	hand_chrc_nn_fmuleOg	0	3	143	322
hand_chrc_nn_fpexg8j_U4	hand_chrc_nn_fpexg8j	0	0	100	137
hand_chrc_nn_fptrfYi_U3	hand_chrc_nn_fptrfYi	0	0	128	284
Total	9	0	34	5993	8898

Figure 3.8: Utilization Estimate.

  - **Memory**

Memory	Module	BRAM_18K	FF	LUT	Words	Bits	Banks	W*Bits*Banks
bias1_0_U	hand_chrc_nn_biascud	1	0	0	100	32	1	3200
h1_U	hand_chrc_nn_h1	1	0	0	100	32	1	3200
hand_mulchrc_nn_float_s_U	hand_chrc_nn_handbkb	256	0	0	78400	32	1	2508800
lay1_U	hand_chrc_nn_lay1	256	0	0	78400	32	1	2508800
lay21_U	hand_chrc_nn_lay21	8	0	0	2600	32	1	83200
Total	5	522	0	0	159600	160	5	5107200

Figure 3.9: Memory Estimate.

Interface

- **Summary**

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_CRTL_BUS_AWVALID	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_AWREADY	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_AWADDR	in	5	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_WVALID	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_WREADY	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_WDATA	in	32	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_WSTRB	in	4	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_ARVALID	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_ARREADY	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_ARADDR	in	5	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_RVALID	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_RREADY	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_RDATA	out	32	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_RRESP	out	2	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_BVALID	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_BREADY	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_BRESP	out	2	s_axi	CRTL_BUS	scalar
ap_clk	in	1	ap_ctrl_hs	hand_chrc_nn	return value
ap_rst_n	in	1	ap_ctrl_hs	hand_chrc_nn	return value
interrupt	out	1	ap_ctrl_hs	hand_chrc_nn	return value
X_Addr_A	out	32	bram	X	array
X_EN_A	out	1	bram	X	array
X_WEN_A	out	4	bram	X	array
X_Din_A	out	32	bram	X	array
X_Dout_A	in	32	bram	X	array
X_Clk_A	out	1	bram	X	array
X_Rst_A	out	1	bram	X	array

Figure 3.10: Interface Summary.

- **Expression**

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
ap_return	+	0	0	15	7	8
i_1_fu_506_p2	+	0	0	15	7	1
i_2_fu_580_p2	+	0	0	15	5	1
i_fu_483_p2	+	0	0	15	7	1
indvarinc1_fu_444_p2	+	0	0	17	10	1
indvarinc_fu_438_p2	+	0	0	15	7	1
j_2_fu_523_p2	+	0	0	17	10	1
j_3_fu_602_p2	+	0	0	15	7	1
next_mul2_fu_494_p2	+	0	0	24	17	10
next_mul4_fu_564_p2	+	0	0	19	12	7
next_mul_fu_432_p2	+	0	0	24	17	10
tmp_15_fu_538_p2	+	0	0	24	17	17
tmp_1_fu_454_p2	+	0	0	24	17	17
tmp_34_fu_617_p2	+	0	0	19	12	12
tmp_31_fu_715_p2	and	0	0	8	1	1
tmp_33_fu_721_p2	and	0	0	8	1	1
exitcond1_fu_574_p2	icmp	0	0	11	5	4
exitcond3_fu_517_p2	icmp	0	0	13	10	9
exitcond4_fu_500_p2	icmp	0	0	11	7	6
exitcond5_fu_477_p2	icmp	0	0	11	7	6
exitcond_fu_596_p2	icmp	0	0	11	7	6
notlhs8_fu_697_p2	icmp	0	0	11	8	2
notlhs_fu_679_p2	icmp	0	0	11	8	2
notrhs9_fu_703_p2	icmp	0	0	18	23	1
notrhs_fu_685_p2	icmp	0	0	18	23	1
tmp_2_fu_465_p2	icmp	0	0	13	10	9
tmp_3_fu_471_p2	icmp	0	0	11	7	6
tmp_29_fu_691_p2	or	0	0	8	1	1
tmp_30_fu_709_p2	or	0	0	8	1	1
mm_1_fu_733_p3	select	0	0	32	1	32
num_1_fu_726_p3	select	0	0	32	1	32
tmp_19_neg_fu_632_p2	xor	0	0	40	32	33
tmp_9_neg_fu_553_p2	xor	0	0	40	32	33
Total	33	0	0	573	337	274

Figure 3.11: Expression Summary.

- **Multiplexer**

Name	LUT	Input Size	Bits	Total Bits
ap_NS_fsm	661	149	1	149
grp_fu_354_p0	15	3	32	96
grp_fu_359_p0	15	3	32	96
grp_fu_359_p1	15	3	32	96
grp_fu_366_p0	15	3	32	96
h1_address0	21	4	7	28
h1_d0	21	4	32	128
i1_reg_239	9	2	7	14
i2_reg_250	9	2	7	14
invdar1_reg_228	9	2	10	20
invdar_reg_204	9	2	7	14
j_1_reg_343	9	2	7	14
j_reg_273	9	2	10	20
lay1_address0	15	3	17	51
mm_reg_307	9	2	32	64
num_2_reg_296	9	2	5	10
num_reg_284	9	2	32	64
phi_mul1_reg_261	9	2	17	34
phi_mul3_reg_319	9	2	12	24
phi_mul_reg_216	9	2	17	34
tmp_16_reg_331	9	2	32	64
Total	895	198	380	1130

Figure 3.12: Multiplexer Summary.

- **Register**

Name	FF	LUT	Bits	Const Bits
X_load_reg_827	32	0	32	0
ap_CS_fsm	148	0	148	0
h1_addr_1_reg_804	7	0	7	0
i1_reg_239	7	0	7	0
i2_reg_250	7	0	7	0
i_1_reg_799	7	0	7	0
i_2_reg_850	5	0	5	0
i_reg_776	7	0	7	0
indvarinc1_reg_751	10	0	10	0
indvarinc_reg_746	7	0	7	0
invdar1_reg_228	10	0	10	0
invdar_reg_204	7	0	7	0
j_1_reg_343	7	0	7	0
j_2_reg_812	10	0	10	0
j_3_reg_858	7	0	7	0
j_reg_273	10	0	10	0
lay1_load_reg_832	32	0	32	0
lay21_load_reg_873	32	0	32	0
mm_reg_307	32	0	32	0
next_mul2_reg_791	17	0	17	0
next_mul4_reg_837	12	0	12	0
next_mul_reg_741	17	0	17	0
num_2_cast2_reg_842	5	0	32	27
num_2_reg_296	5	0	5	0
num_reg_284	32	0	32	0
phi_mul1_reg_261	17	0	17	0
phi_mul3_reg_319	12	0	12	0
phi_mul_reg_216	17	0	17	0
reg_389	32	0	32	0
reg_394	32	0	32	0
reg_400	32	0	32	0
reg_406	64	0	64	0
reg_411	64	0	64	0
reg_416	64	0	64	0
reg_421	64	0	64	0
reg_426	32	0	32	0
tmp_16_reg_331	32	0	32	0
tmp_2_reg_766	1	0	1	0
tmp_32_reg_878	1	0	1	0
tmp_36_cast_reg_756	17	0	64	47
tmp_4_reg_781	7	0	64	57
Total	960	0	1091	131

Figure 3.13: Register Summary.

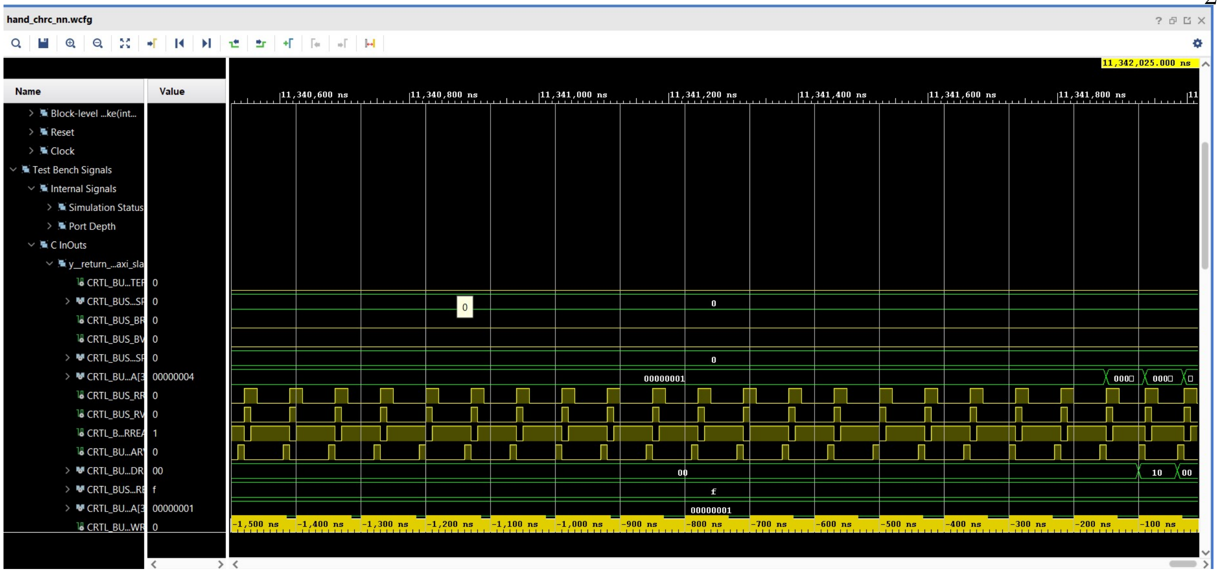

Figure 3.14: Waveform.

Chapter 4

Discussions and conclusion

In this chapter, the work is concluded and future plan is presented. Next, the research contribution are presented. Finally, limitation of the work and possible future extensions are described respectively.

4.1 Conclusion

Artificial neural networks have been used to recognise visual characters in a straightforward manner. The benefits of neural computing over conventional techniques have been described. Artificial neural networks, in the sense of somewhat simulating adaptive human intelligence, offer significant advantages in pattern detection and classification despite the computational complexity involved.

4.2 Performance Issues

The neural system has certain immediate advantages. The approach is quite adaptable; recognition is forgiving of little mistakes and pattern modifications. By teaching it newer characters or new variations of older characters, the system's knowledge base can be changed. The system is extremely generic and size- and aspect-ratio-invariant. The system can be made user-specific by maintaining character user profiles and programming it to recognise characters based on the user's orientation. This can be done by using larger datasets, more training data, and better training resources to train the model with more features and training datasets.

4.3 Future scope

We could try to improve the model by studying other neural network architectures like the graph neural networks and the convolutional neural network to try to capture the relationships and variations of various features of an image.We may also try using better and more realistic datasets and features.