T.C.
DOKUZ EYLÜL UNIVERSITY
ENGINEERING FACULTY
ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT

Smart Mirror Controller

Final Project

by

Rabia DOĞAN

Advisor

Ph.D. Özgür TAMER

January,2021

IZMIR

THESIS EVALUATION FORM

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and qualify as an undergraduate thesis, based on the result of the oral examination taken place on ___/___/_____

Ph.D. Özgür TAMER
(ADVISOR)

Prof. Dr. Gülay TOHUMOĞLU
(COMMITTEE MEMBER)

Ph.D. Abdül BALIKCI
(COMMITTEE MEMBER)

Prof. Dr. Mehmet KUNTALP
(CHAIRPERSON)

ABSTRACT

In the project, general purpose is to give the Smart Mirror controllable features with hand gestures. In this way, the product will be able to appeal to the upper customer segments and increase its market share. The main goal here is to enable users to use most of the smart mirror applications with hand movements, to eliminate the need to touch the mirror, thus to eliminate the contamination in the touched parts of the mirrors and therefore to possible dissatisfaction with the product and to ensure that the product can be used even in cases where the users cannot touch the mirror with their hands for various reasons, such as the bathroom. Although camera-based systems are generally used for gesture recognition, it will not be welcomed by many users to have a camera in a personal use area such as the bathroom.Therefore, passive infrared sensor arrays will be preferred for gesture recognition in our project. They can be preferred in application areas where cameras are relatively weak because they work with the infrared radiation emitted by living creatures. We aim to detect hand gesture and then the movement of the hand gesture. In this way, gesture distinction can be made with a simpler electronic design with less processing load and lower cost than a standard camera. The project is carried out and supported by Vestel Elektronik A.Ş. with the code TEYDEB 3170688.

ÖZET

Projede genel amaç Akıllı Aynaya el jestleri ile kontrol edilebilir özellikler kazandır- maktır.Bu sayede ürün üst müşteri segmentlerine de hitap edebilecek, pazar payını artırabilecektir.Burada temel hedef kullanıcıların akıllı ayna uygulamalarının birçoğunu el hareketleri ile kullanabilmesini sağlayarak, aynaya dokunma ihtiyacını ortadan kaldır- mak böylece aynaların dokunulan bölgelerindeki kirlenmeyi ve dolayısıyla da ürün ile ilgili olası memnuniyetsizliği ortadan kaldırmak ve banyo gibi kullanıcıların çeşitli neden- ler ile elleri ile aynaya dokunamayacakları durumlarda dahi ürünün kullanılabilmesini sağlamaktır.Jest tanıma için genellikle kamera temelli sistemler kullanılmakla beraber, banyo gibi kişisel kullanıma dönük bir alanda kamera bulunması birçok kullanıcı tarafın- dan hoş karşılanmayacaktır.Bu nedenle projemizde jest tanımlama amacıyla pasif kızılötesi sensör dizileri tercih edilecektir.Bu tip sensörler oldukça düşük çözünürlüklerde görev yapmasına karşın, nesne ya da canlıların yaydığı kızılötesi ışıma ile çalıştıkları için kameraların görece zayıf kaldığı uygulama alanlarında tercih edilebilmektedirler.Pasif kızılötesi sensörler ile öncelikle kullanıcının jestini ve sonrasında da jestin hareketini al- gılamayı amaçlamaktayız.Bu sayede standart bir kameraya göre daha az işlem yükü ve daha düşük maliyet ve daha basit bir elektronik tasarım ile jest ayrımı yapılabilecektir. Proje Vestel Elektronik A.Ş.taarafından TEYDEB 3170688 kodu ile yürütülmekte ve desteklenmektedir.

Contents

List of Tables

List of Figures

1. INTRODUCTION

2. TECHNICAL BACKGROUND

2.1 Introduction to Convolutional Neural Networks

In deep learning, CNNs are the most common networks used with image classification. CNNs were inspired by the human visual system proposed by Fukushima [5] and LeCun et al. [6] State-of-the-art approaches to pattern recognition, object detection, and many other image applications. It was a deep CNN solution by Krizhevsky et al. [7] CNNs are very different from other pattern recognition algorithms because CNNs combine both feature extraction and classification [6]. The simple network model consists of five different layers: an input layer, a convolution layer, a pooling layer, a fully connected layer, and an output layer. These layers are divided into two parts: feature extraction and classification. Feature extraction consists of an input layer, a convolution layer, and a pool layer, while classification consists of a fully connected layer and an output layer. The input layer specifies a fixed size for input images, which are resized as needed. The image is then convoluted with multiple learned kernels using the weights shared by the convolution layer. Then, the repository layer reduces the image size while trying to preserve the information it contains. The outputs of feature extraction are known as feature maps. Classification combines extracted features into fully connected layers. Finally, there is one output neuron for each object category in the output layer. The output of the classification section is the classification result.

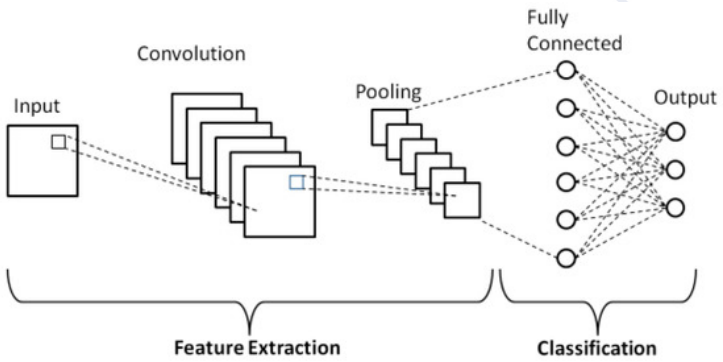

Figure 2.1: Schematic diagram of a basic convolutional neural network (CNN) architecture [11]

Deep learning techniques have emerged recently and advances in convolutional neural networks (CNN) surpass the classical approach to hand gesture recognition as it eliminates the

need to derive complex handcrafted features from images [8]. CNN's automate the feature extraction process by learning high-level abstractions in images and capturing the most distinguishing feature values using the hierarchical architecture. Thus, it solves the disadvantage of obtaining inconsistent property descriptors when working with large numbers of motion classes with very small cross-class variations [12].

2.2 Model Architecture for LE-NET5

LeNet-5 was one of the earliest convolutional neural networks to support the deep learning event. After countless years of analysis and numerous compelling iterations, the final result was named LeNet-5 in 1988.

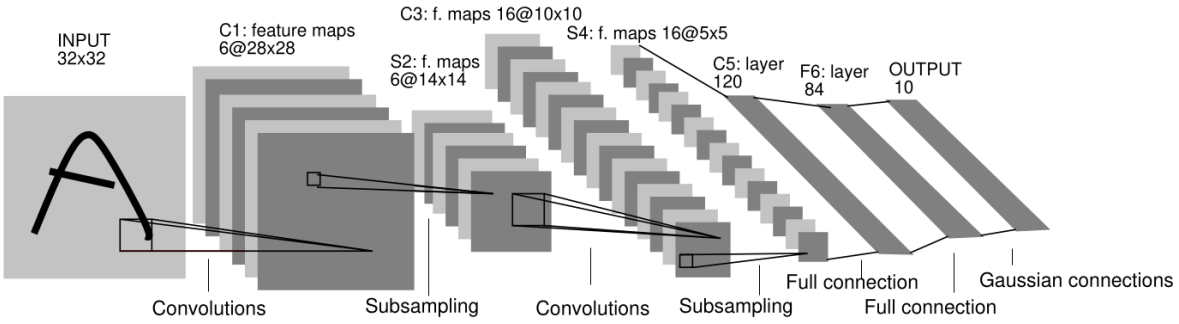

Figure 2.2: Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical. [6]

LeNet-5 A total of seven layers, each with no input with trainable parameters; each layer has multiple FeatureMap, which is a property of each of the FeatureMap inputs extracted via a convolution filter, and then each FeatureMap has multiple neurons.

C1 layer-convolutional layer:

The first convolution operation is performed on the input image (using 6 convolution kernels of size 5*5) to obtain 6 feature maps of C1 (6 feature maps of size 28 28, 32-5 + 1 = 28). The size of the convolution kernel is 5, and there are 6 (5 * 5 + 1) = 156 parameters in total, where +1 indicates that a kernel has a bias. For convolution layer C1, each pixel in C1 is dependent on 5 of 5 pixels and 1 aberration in the input image, so there are 156 28 * 28 = 122304 links in total. [6]

S2 layer-pooling layer (downsampling layer):

The pooling is done immediately after the first convolution. Pooling is done using 2 cores and S2, 14x14 (28/2 = 14) 6 feature maps are obtained. S2's pooling layer is a weighting coefficient plus an offset multiplied by the sum of the pixels in the 2*2 area in C1, and then the result is remapped. So each pooling core has two training parameters i.e. 2x6 = 12 training parameters but there are 5x14x14x6 = 5880 connections. [6]

C3 layer-convolutional layer:

After the first pooling, the second convolution, the output of the second convolution is C3, 16 pieces of 10x10 feature maps, and the size of the convolution kernel is 5. The first 6 feature maps of C3 (corresponding to column 6 of the first). red box) connects to 3 feature maps connected to S2 layer and next 6 feature maps are connected to S2 layer 4 feature maps are connected, next 3 feature maps are connected to 4 feature maps are unconnected in S2 layer and last one is linked to all feature maps in S2 layer. The convolution kernel size is still 5x5, so there are 6 (3*5*5 + 1) + 6 (4*5*5 + 1) + 3 (4*5*5 + 1) +1 (6*5*5 + 1) = 1516 parameters. The image size is 10 10 so there are 151600 connections. [6]

S4 layer-pooling layer (downsampling layer):

S4 is the pooling layer, the window size is still 2*2, a total of 16 feature maps and 16 10x10 maps of the C3 layer are pooled in units of 2x2 to get 16 5x5 feature maps. This layer has a total of 32 training parameters, 2x16, 5x5x5x16 = 2000 connections. [6]

C5 layer-convolution layer:

The C5 layer is a convolution layer. Since the size of the 16 images of the S4 layer is 5x5, the size of the image formed after convolution is 1x1, which is the same as the size of the convolution kernel. This results in 120 convolution results. Each is linked to 16 maps from

the previous level. So there are (5x5x16 + 1) x120 = 48120 parameters and there are also 48120 connections. [6]

F6 layer-fully connected layer:

Layer 6 is a fully connected layer. The F6 layer has 84 nodes corresponding to a 7x12 bitmap, -1 means white, 1 means black, so the black and white of each symbol's bitmap corresponds to a code. The training parameters and number of connections for this layer is (120 + 1) x84 = 10164. [6]

2.3 Python Libraries for Project

2.3.1 Python-Peripherial

2.3.2 Tensorflow-Keras

3. MATERIALS AND METHODS

3.1 Htpa32x32d Thermopile Infrared Array

The sensor chosen to be used in the project is HTPA32x32d thermopile array sensor. HTPA32x32d thermopile array sensor 32x32 pixel, operates between -10 and 70 degrees, provides I2C communication, has an internal EEPROM and provides an 8-bit data set. EEPROM data contains calibration data for each pixel of the sensor.

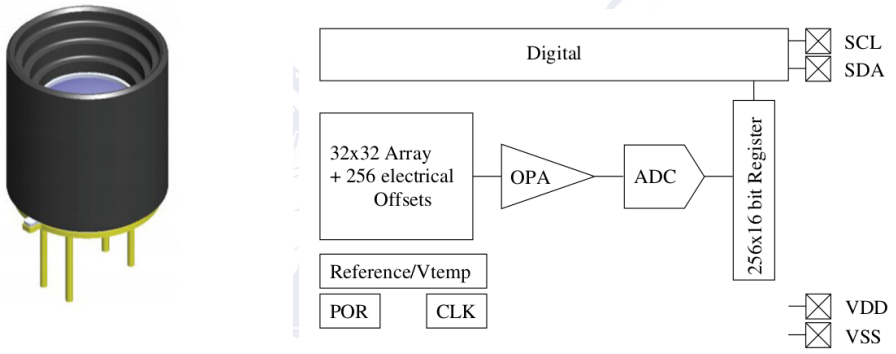

Figure 3.1: Schematic for HTPA32x32d

Table 3.1: Genaral Features HTPA32x32d

Features\{}	Sensitivity	Therm. Pix. Time Const.	Digital Interface	EEPROM Size
	450V/W	$< 4ms$	I2C	64kBit
Features\{}	Max Frame	Field of View	Selectable Clock	Storage Temperature
	60 Hz	33*33 deg	1 to 13 Mhz	-40/85 Deg.C

3.2 Sensor and MCU Communication PCB Design

A mini card has been designed between the sensor and the Raspberry pi card to communicate. However, the card could not be printed. The card that will connect the sensor and the motherboard has been designed completely according to the standards recommended by Heimann company.

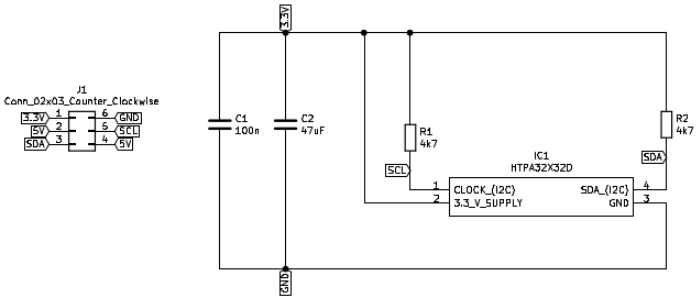

Figure 3.2: Schematic Design of Communication PCB

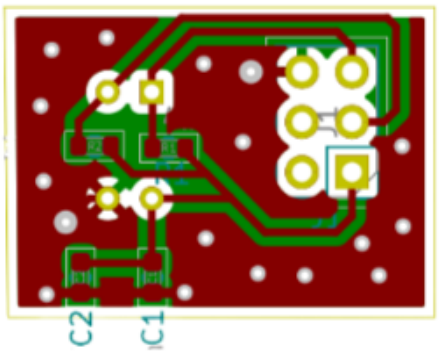

Figure 3.3: PCB Design of Communication PCB

In addition to this, we have produced a product suitable for a schematic on a mini perforated plate for use in communication with the card. In addition, we designed a protective cage from a 3d printer to protect the sensor.

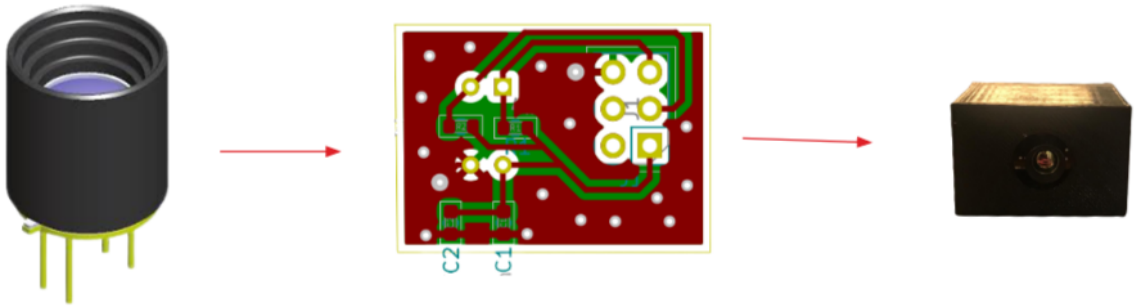

Figure 3.4: Mini Sensor Board

3.3 Dataset

3.3.1 Multi-Modal Hand Gesture Dataset for Hand Gesture Recognition

This dataset was created to validate a hand-gesture recognition system for Human-Machine Interaction (HMI). It is composed of 15 different hand-gestures (4 dynamic and 11 static) that

are split into 16 different hand-poses, acquired by the Leap Motion device. Hand-gestures were performed by 25 different subjects (8 women and 17 men). Every gesture has 20 instances (repetitions) per subject, performed in different locations in the image. [2]
for static and dynamic gestures:
This set contains 16 hand-poses, used for both static and dynamic hand-gestures:
A: L B: fist moved C: index D: ok E: C F: heavy G: hang H: two I: three J: four K: five L: palm M: down N: palm moved O: palm up P: up

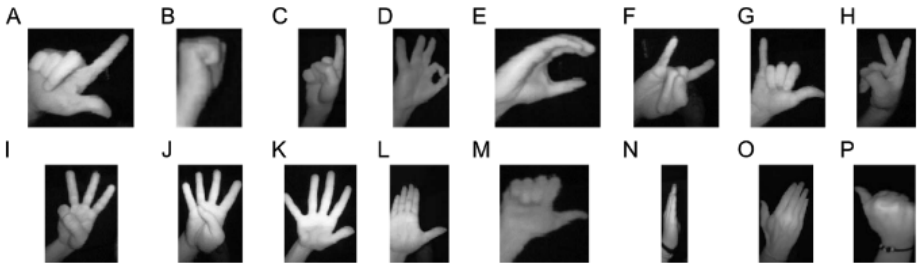

Figure 3.5: Hand Gesture Dataset

3.4 Data Augmentation

Using this dataset, a new train and validation set was created for the static gestures in the project. A total of 8000 and 2000 train and validation sets were created with randomly selected images.However, both resizing and data augmentation were done in order to make the data set suitable for the project. [9]

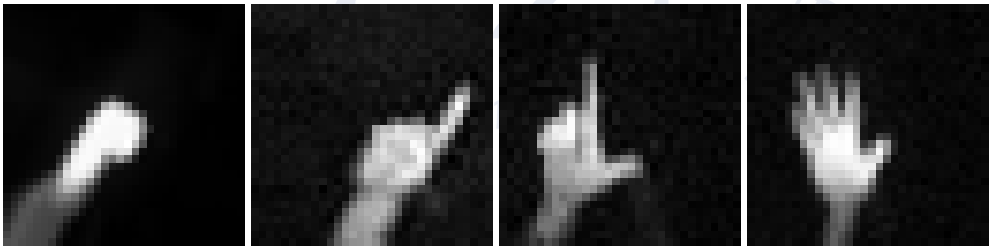

Figure 3.6: Close-Index-Last Page-Open Gestures

4. PROGRESS AND RESULT

In this section, the methods to be applied in the project are included.

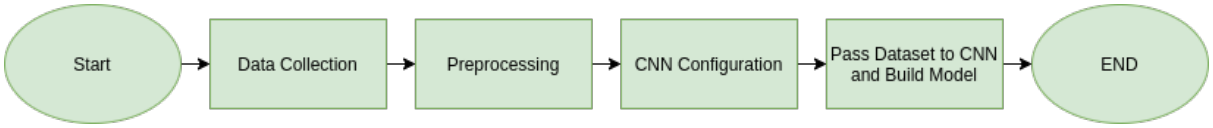

Figure 4.1: System Framework for Static Gestures

4.1 Capture Thermopiles Image

4.1.1 Heimann Thermopile Array Sensor communication

First, I provided the connections between the sensor and the Raspberry Pi in order to receive the image. I provided the communication with the mini card I made in section 4.2 using the I2C protocol.

With the device connected to a Raspberry Pi, and with the Pi configured. [1] correctly for I2C, I was able to see the devices connected with the i2cdetect command.

```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- 1a -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 4.2: Thermopile Infrared Array Device and EEPROM Addresses

In order to be able to read the data properly, the python-periphery [4] library was used. The sensor is divided into two parts (Top and Bottom Half), which are also divided into 4 blocks. The reading order is shown below for different blocks. When a conversion is initiated, the X Block of the upper and lower half are measured simultaneously. Each block consists of 128

Pixels sampled entirely in parallel. The reading order in the lower half is mirrored compared to the upper half so the center lines are always read last.

Table 4.1: Read Data 1 Command (Top Half of Array)

Addr/CMD	0x1A (7 Bit!) / 0x0A							
Read Data	7	6	5	4	3	2	1	0
1. Byte / 2. Byte	PTAT 1 MSB / LSB or Vdd 1 MSB / LSB							
3. Byte / 4. Byte	Pixel (0+BLOCK*128) MSB / LSB							
5. Byte / 6. Byte	Pixel (1+BLOCK*128) MSB / LSB							
...								
257. Byte / 258. Byte	Pixel (127+BLOCK*128) MSB / LSB							

Table 4.2: Read Data 2 Command (Bottom Half of Array)

Addr/CMD	0x1A (7 Bit!) / 0x0B							
Read Data	7	6	5	4	3	2	1	0
1. Byte / 2. Byte	PTAT 2 MSB / LSB or Vdd 2 MSB / LSB							
3. Byte / 4. Byte	Pixel (992-BLOCK*128) MSB / LSB							
5. Byte / 6. Byte	Pixel (993-BLOCK*128) MSB / LSB							
...								
65. Byte / 66. Byte	Pixel (1023-BLOCK*128) MSB / LSB							
65. Byte / 66. Byte	Pixel (1023-BLOCK*128) MSB / LSB							
67. Byte / 68. Byte	Pixel (960-BLOCK*128) MSB / LSB							
69. Byte / 70. Byte	Pixel (961-BLOCK*128) MSB / LSB							
...								
129. Byte / 130. Byte	Pixel (991-BLOCK*128) MSB / LSB							
131. Byte / 132. Byte	Pixel (928-BLOCK*128) MSB / LSB							
...								
257. Byte / 258. Bytes	Pixel (927-BLOCK*128) MSB / LSB							

Each block is checked before it is read. The python-opencv [3] library was used to visualize the obtained result.

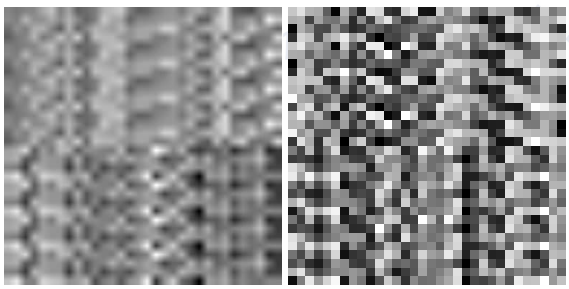

Figure 4.3: Thermal Image

Each pixel (or each analog-to-digital converter, given the repeating structure corresponding to each "block" of the sensor) has its own offset and sensitivity to incident light. Without calibrating it, this constant "noise" suppresses the signal from changing IR/temperature conditions. By subtracting the two frames in quick succession, this common noise signal is removed.

However, it is still quite noisy, as this frame subtraction increases random noise (since we now have contributions from two frames) and does not correct pixel-dependent sensitivity. Only fabrication calibration will be done with EEPROM data in the next step.

4.1.2 Calibrating images from Heimann Thermopile Array Sensor

After reading an image off a Heimann thermopile array, the pixel values can be converted to temperature readings through the use of calibration parameters stored on the device. To extract the calibration parameters, it is easiest to first read off the entire EEPROM on the thermopile array.

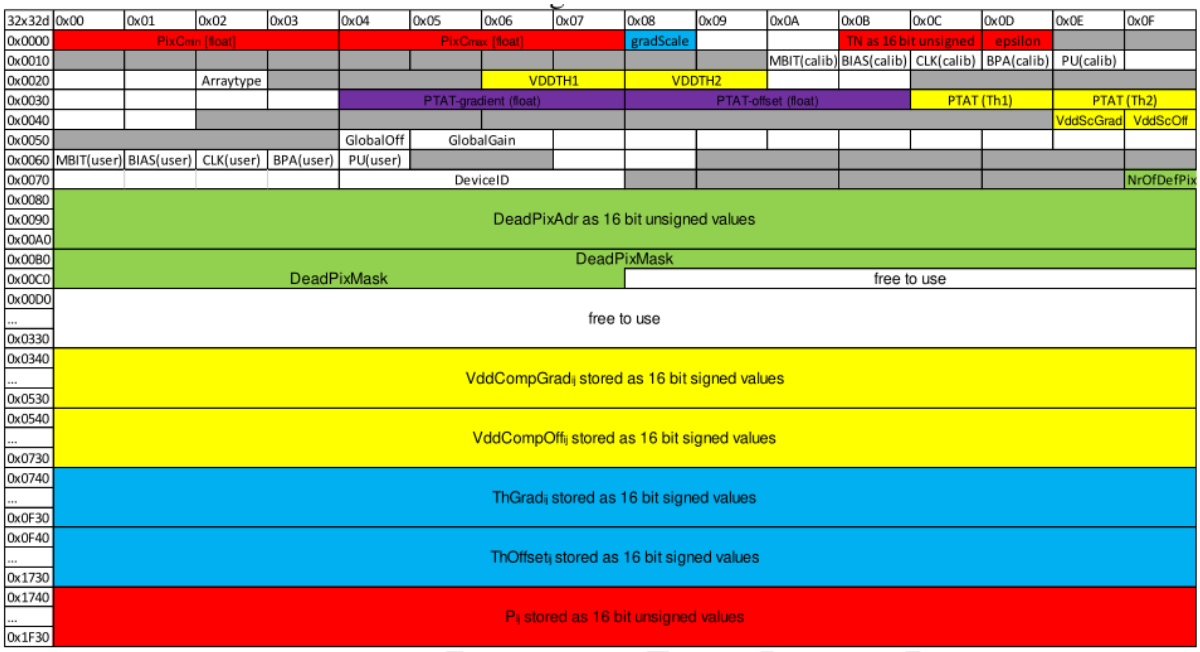

Figure 4.4: EEPROM overview 32x32d [10]

Then, parameters and calibration values can be extracted from this array, as described in the Heimann datasheet. [10]

Calibration for only one pixel is done as follows.

$PTAT_{av} = \frac{\sum_{i=0}^{7} PTAT_i}{8} = 38152 Digits$

$PTAT_{gradient} = 0.0211 dK/Digit$ and $PTAT_{offset} = 2195.0 dK$

$V_{00} = 34435 Digits$

$elOffset[0] = 34240$

$gradScale = 24$

$ThGrad_{00} = 11137$

$ThOffset_{00} = 65506$

$VDD_{av} = 35000$

$VDD_{TH1} = 33942$

$VDD_{TH2} = 36942$

$PTAT_{TH1} = 30000$

$PTAT_{TH_2} = 42000$

$VddCompGrad[0] = 10356$

$VddCompOff[0] = 51390$

$VddScGrad = 16$

$VddScOff = 23$

$PixC_{00} = 1 \cdot 087 \cdot 10^8$

$PCSCALEVAL = 1 \cdot 10^8$

Calculation of ambient temperature:

$T_a = PTAT_{av} \cdot PTAT_{gradient} + PTAT_{offset} = 38152 \cdot 0.0211 + 2195.0dK = 3000dK$

Compensation of thermal offset:

$V_{00\_Comp} = V_{00} - \frac{Th_{Grad00} \cdot T_a}{2^{gradScale}} - Th_{Offset_{00}} = 34439$

Compensation of electrical offset:

$V^*_{00\_Comp} = V_{00\_Comp} - elOffset[0] = 199$

Compensation of supply voltage:

$V_{00-VDD_{Comp}} = V^*_{00\_Comp} - \frac{\frac{Vdd_{CompGrad}[0] \cdot PTAT_{av}}{2^{Vdd_{ScGrad}}} + V_{Vdd_{Compoff}}[0]}{2^{Vdd_{ScOff}}}$

$\cdot (VDD_{av} - VDD_{TH1} - (\frac{VDD_{TH2} - VDD_{TH1}}{PTAT_{TH2} - PTAT_{TH1}}) \cdot (PTAT_{av} - PTAT_{TH1})) = 199 - 1 = 198$ The sensitivity coefficients (PixC ij) are calculated:

$PixC_{00} = (\frac{P_{00} \cdot (PixC_{Max} - PixC_{Min})}{65535} + PixC_{Min}) \cdot \frac{epsilon}{100} \cdot \frac{GlobalGain}{100000} = 1 \cdot 087 \cdot 10^8$

Leading to a compensation of the pixel voltage:

$V_{00PixC} = \frac{V_{00-VDD_{Comp}} \cdot PCSCALEVAL}{PixC_0} = 182$

All operations are applied for 1024 pixels. Application result images are as in figure 4.4.

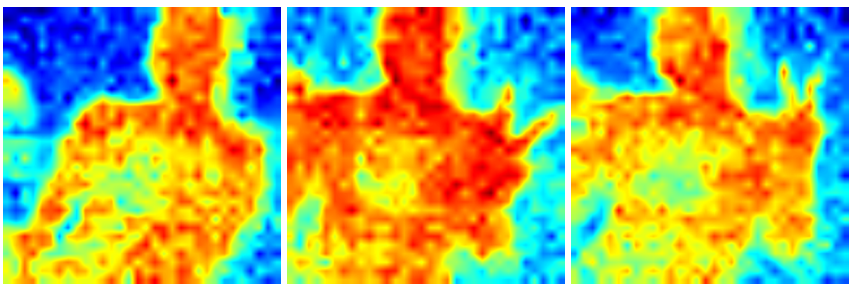

Figure 4.5: Thermal Images with EEPROM Calibration Data

NOTE:All steps to acquired the image are made with reference to the datasheet [10].

4.2 Hand Thermal Image Isolation

The hand was isolated from the background without using any image processing method. For this, it has been arranged in a way that can remove the ambient temperature of the device from the image before giving a command. First, the average of 10 images was taken and given to all images. Thus, the background temperature was isolated.

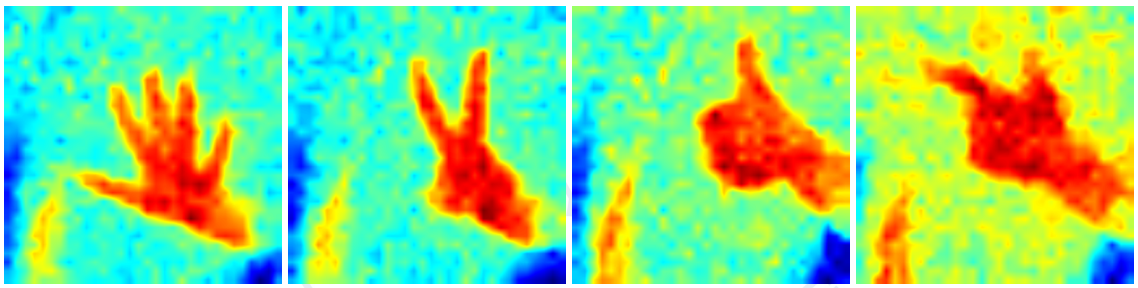

Figure 4.6: Thermal Images with Background

4.3 Hand Gesture Recognition

4.3.1 Static Gesture Recognition

A number of scenarios have been prepared for command matching for the smart mirror. The scenarios prepared are as follows;

For Static Movements:

There are 4 fixed movements. It is the ability to Open, Close, Tap and Return to Home.

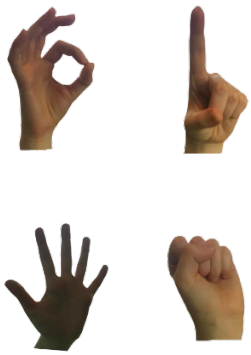

Figure 4.7: Gestures of Open,Close,Touch,Return to the Home Page

We tried to create our own data set using the data set we mentioned in Section 3.3. We used images that are similar to each of the four identified gestures. It was found correct to use K: Five set for Open, B: First Moved set for Close, C: Index set for Touch, A:L set for ,Return to the Home Page. [2] First, the images were cropped and resized that are chosen mixed. After these processes, the data reproduced by data augmentation were divided into two as validation and training data. The number of trains defined for each movement is approximately 9200, and the number of validation images is around 2300.

Model Architecture for LE-NET5

In 1989, Yann LeCun presented a convolutional neural network called LeNet. Generally, LeNet refers to LeNet-5 and is a simple convolutional neural network. [6]

The fact that our Input Image sizes are 32X32 was the biggest factor pushing us to use this model. Since we could not use a ready dataset, the change in the number of layers in order to train our model well, resulted in good results.

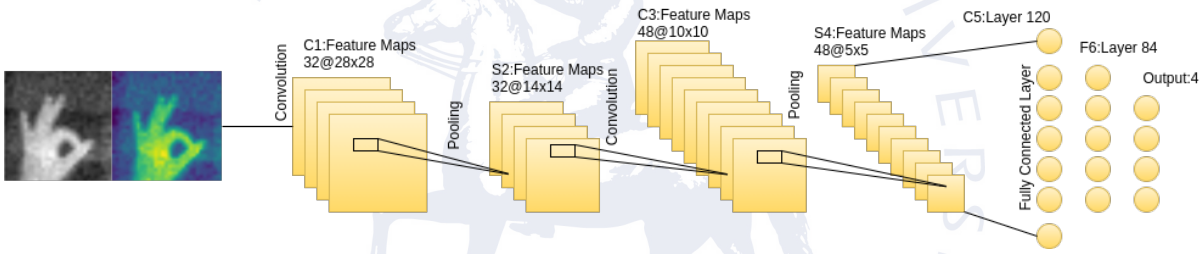

Figure 4.8: A Structural Diagram of the Proposed LENET5 for Static Gesture

The general architecture for LeNet-5 is as given in Figure 3. The input layer C1 acts like the retina, which receives centered and size-normalized gesture images (otherwise, some images may not fit in the input layer). The next layer, S2, consists of several feature maps that have the same role to gestures as their simple cells. In practice, a feature map is a square. The weights in a feature map need to be the same so they can detect the same local feature in the input image. The weights between feature maps are different so they can detect different local features. Each unit in a feature map has a receiver field.

Table 4.3: The structural layers and number of parameters of the LENET5.

Model:"Static_Gesture_Model		
Layer(type)	Output Shape	Param#
conv2d_8 (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d_8 (MaxPooling2)	(None, 14, 14, 32)	0
conv2d_9 (Conv2D)	(None, 10, 10, 48)	38448
max_pooling2d_9 (MaxPooling2)	(None, 5, 5, 48)	0
flatten_4 (Flatten)	(None, 1200)	0
dense_16 (Dense)	(None,120)	307456
dense_17 (Dense)	(None, 84)	21588
dense_18 (Dense)	(None, 10)	850
dense_19 (Dense)	(None, 4)	44
Total params: 369,218 Trainable params: 369,218 Non-trainable params: 0		

If to summarize the model we outlined in the table, we use two convolutional layers, then twice a pooling layer (32-filters and 48-filters, respectively), and finally three fully connected layers with 4-class softmax units.

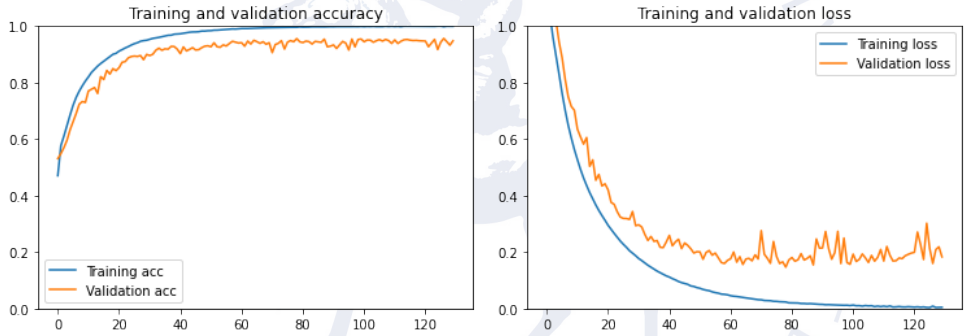

Figure 4.9: Loss and Accuracy Curves (Training and Validation Set)

And the result is close to perfect. Data augmentations worked. The learning algorithm, which reached 95% validation accuracy, gave very successful results. In the next step, images will be given to the model again for testing and the results will be observed.

Test Sets and Results

4.3.2 Dynamic Gesture Recognition

Table 4.4: Trim Register 1 (write only) [10]

Addr / CMD	0x1A (7 Bit!) / 0x03							
Trim Reg 1	7	6	5	4	3	2	1	0
Name	RFU		REF_CAL		MBIT TRIM			

REF_CAL: selectable amplification

MBIT_TRIM: m = 4 to 12 (m+4) bit as ADC resolution

In order to get the maximum efficiency from the sensor, to set the ADC Resolution to 16 bits, m of 12 was determined according to the data in the table. Since the framerate was too low, it was decided to speed up by sacrificing quality, but the quality was too low, revealing that ADC resolution should not be compromised. According to the resolutions, the images are as in figure 2.

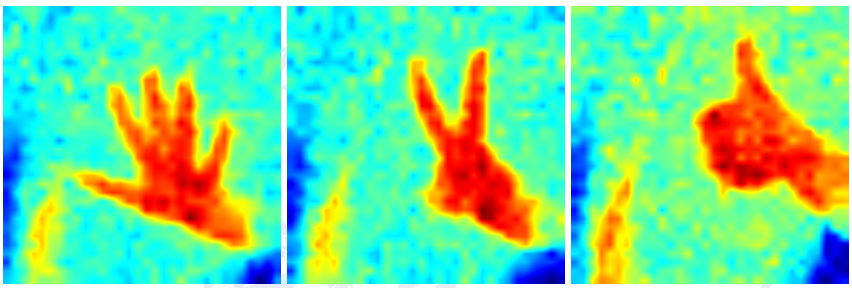

Figure 4.10: Thermal Images 8Bit-12Bit-16Bit ADC Resolution)

Table 4.5: Trim Register 2 (write only) [10]

Addr / CMD	0x1A (7 Bit!) / 0x04							
Trim Reg 1	7	6	5	4	3	2	1	0
Name	RFU			BIAS TRIM TOP				

BIAS_TRIM_TOP:0 to 31 $\longrightarrow$ $1\mu A$ to $13\mu A$

Table 4.6: Trim Register 3 (write only) [10]

Addr / CMD	0x1A (7 Bit!) / 0x05							
Trim Reg 1	7	6	5	4	3	2	1	0
Name	RFU			BIAS TRIM BOT				

BIAS_TRIM_BOT: 0 to 31 $\longrightarrow$ $1\mu A$ to $13\mu A$

This setting is used to adjust the bias current of the ADC. A faster clock frequency requires a higher bias current setting. [10]

Having ADC resolution 16 made us think that it did not affect us much. Afterward, it was said that if the BIAS current value is set to the maximum, its speed may increase. However, as a result of this experiment, almost no change was observed in the framerate.

Table 4.7: Trim Register 4 (write only) [10]

Addr / CMD	0x1A (7 Bit!) / 0x06							
Trim Reg 1	7	6	5	4	3	2	1	0
Name	RFU		CLK TRIM					

CLK_TRIM:0 to 63 $\longrightarrow$ 1MHz to 13MHz

The point where we could most easily observe the increase in Frame Rate was the Clock Trim set. Although I set the maximum value, I could not get the desired result here.

4.4 Matching Gesture to Commands of Smart Mirror

In the project carried out by Vestel, friends who are interested in the smart mirror part are expected to define the gestures for the android side of the smart mirror.

5. COST ANALYSIS

6. CONCLUSION

7. APPENDIX

7.1 Sensor Configuration

Sensor Configuration

HTPA32x32d Library

[ ]:
```
from periphery import I2C
import time
import numpy as np
import copy
import struct

class HTPA:


        def __init__(self, address):
                self.address = address
                self.i2c = I2C("/dev/i2c-1")
                print("Grabbing EEPROM data")
                eeprom = self.get_eeprom()
                self.extract_eeprom_parameters(eeprom)
                self.eeprom = eeprom
                wakeup_and_blind = self.generate_command(0x01, 0x01)  # wake␣
↪up the device
                # set ADC resolution to 16 bits
                adc_res = self.generate_command(0x03, self.mbit_value)# set␣
↪ADC resolution in eeprom
                pull_ups = self.generate_command(0x09, self.pu_value)#pu␣
↪value in eeprom

                print("Initializing capture settings")

                self.send_command(wakeup_and_blind)
                self.send_command(adc_res)
                self.send_command(pull_ups)

                self.set_bias_current(self.bias_value)  # bias value on eeprom
                self.set_clock_speed(0x050)  # clk value on eeprom self.
↪clk_value
                self.set_cm_current(self.bpa_value)  # BPA value in eeprom

                # initialize offset to zero
                self.offset = np.zeros((32, 32))
```

```
    def get_eeprom(self, eeprom_address=0x50):#Talking EEPROM
            query = [I2C.Message([0x00, 0x00]), I2C.Message(
                [0x00]*8000, read=True)]  # 8 Kbit Data from EEPROM
            self.i2c.transfer(eeprom_address, query)
            return np.array(query[1].data)


    def extract_eeprom_parameters(self, eeprom):#EEPROM Data
            self.VddCompgrad = eeprom[0x0340:0x0540:2] + (eeprom[0x0341:
↪0x0540:2] << 8)
            self.VddCompoff = eeprom[0x0540:0x0740:2] + (eeprom[0x0541:
↪0x0740:2] << 8)

            ThGrad = eeprom[0x0740:0x0F40:2] + (eeprom[0x0741:0x0F40:2]␣
↪<< 8)
            ThGrad = [tg - 65536 if tg >= 32768 else tg for tg in ThGrad]
            ThGrad = np.reshape(ThGrad, (32, 32))
            ThGrad[16:, :] = np.flipud(ThGrad[16:, :])
            self.ThGrad = ThGrad

            ThOffset = eeprom[0x0F40:0x1740:2] + (eeprom[0x0F41:0x1740:2]␣
↪<< 8)
            ThOffset = np.reshape(ThOffset, (32, 32))
            ThOffset[16:, :] = np.flipud(ThOffset[16:, :])
            self.ThOffset = ThOffset

            P = eeprom[0x1740::2] + (eeprom[0x1741::2] << 8)
            P = np.reshape(P, (32, 32))
            P[16:, :] = np.flipud(P[16:, :])
            self.P = P

            epsilon = float(eeprom[0x000D])
            GlobalGain = eeprom[0x0055] + (eeprom[0x0056] << 8)
            Pmin = eeprom[0x0000:0x0004]
            Pmax = eeprom[0x0004:0x0008]
            Pmin = struct.unpack('f', reduce(
                lambda a, b: a+b, [chr(p) for p in Pmin]))[0]
            Pmax = struct.unpack('f', reduce(
                lambda a, b: a+b, [chr(p) for p in Pmax]))[0]
            self.PixC = (P * (Pmax - Pmin) / 65535. + Pmin) * \
                        (epsilon / 100) * float(GlobalGain) / 100
            self.gradScale = eeprom[0x0008]
            self.VddCalib1 = eeprom[0x0046] + (eeprom[0x0047] << 8)
            self.VddCalib = eeprom[0x0046] + (eeprom[0x0047] << 8)
            self.VddCalib2 = eeprom[0x0048] + (eeprom[0x0049] << 8)
            self.Vdd = 3000.0
            self.VddScaling = eeprom[0x004E]
```

```
                self.Vddoff = eeprom[0x004F]

                self.PtatCalib1 = eeprom[0x003C] + (eeprom[0x003D] << 8)
                self.PtatCalib2 = eeprom[0x003E] + (eeprom[0x003F] << 8)
                PTATgradient = eeprom[0x0034:0x0038]
                self.PTATgradient = struct.unpack('f', reduce(
                    lambda a, b: a+b, [chr(p) for p in PTATgradient]))[0]
                PTAToffset = eeprom[0x0038:0x003c]
                self.PTAToffset = struct.unpack('f', reduce(
                    lambda a, b: a+b, [chr(p) for p in PTAToffset]))[0]
                self.clk_value = eeprom[0x001C]
                self.bias_value = eeprom[0x001B]
                self.pu_value = eeprom[0x001E]
                self.mbit_value = eeprom[0x001A]
                self.bpa_value = eeprom[0x001D]
                self.subt = np.zeros((32, 32))


        def set_clock_speed(self, clk):#set clock speed
                if clk > 63:  # Max 64 Hz
                        clk = 63
                if clk < 0:
                        clk = 0
                clk = int(clk)
                print(clk)
                # The measure time depends on the clock frequency settings.
↪(optimal value)
                clk_speed = self.generate_command(0x06, clk)
                self.send_command(clk_speed)  # send clock data

        # This setting is used to adjust the common mode voltage of the␣
↪preamplifier.


        def set_cm_current(self, cm):
                cm = int(cm)
                cm_top = self.generate_command(0x07, cm)
                cm_bottom = self.generate_command(0x08, cm)

                self.send_command(cm_top)
                self.send_command(cm_bottom)


        def set_bias_current(self, bias):
                bias = int(bias)
                # This setting is used to adjust the bias current of the ADC.␣
↪A faster clock frequency requires a higher bias current setting.
                bias_top = self.generate_command(0x04, bias)
                # This setting is used to adjust the bias current of the ADC.␣
↪A faster clock frequency requires a higher bias current setting.
```

```
                bias_bottom = self.generate_command(0x05, bias)
                self.send_command(bias_top)  # send bias top data
                self.send_command(bias_bottom)  # send bias bottom data


    def temperature_compensation(self, im, ptat):#Thermal Offset Calculate
            comp = np.zeros((32,32))
            Ta = np.mean(ptat) * self.PTATgradient + self.PTAToffset
                #     temperature compensated voltage
            comp = ((self.ThGrad * Ta) / pow(2, self.gradScale)) + self.
↪ThOffset
            Vcomp = np.reshape(im,(32, 32)) - comp
            return Vcomp

        def offset_compensation(self, im):#general environment offset send␣
↪offset data
                return im-self.offset

        def sensitivity_compensation(self, im):#object temperature
                return (im*100000000)/self.PixC

        def measure_observed_offset(self):#Measuring observed offsets
                mean_offset = np.zeros((32, 32))
                for i in range(10):
                        print("    frame " + str(i))
                        (p, pt) = self.capture_image()
                        im = self.temperature_compensation(p, pt)
                        mean_offset += (im-10)/10.0
                self.offset = mean_offset


        def Vdd_Comperasition(self,im,ptat):#Vdd Comperasition calculate
                VVddComp=[]
                for i in range(16):
                        for j in range(32):
                                VVddComp.append((((self.
↪VddCompgrad[(j+i*32)%128]*np.mean(ptat))/pow(2, self.VddScaling)+self.
↪VddCompoff[(j+i*32)%128])/pow(2, self.Vddoff))*(self.Vdd-self.
↪VddCalib1-((self.VddCalib2-self.VddCalib1)/(self.PtatCalib2-self.
↪PtatCalib1))*(np.mean(ptat)-self.PtatCalib1)))
                for i in range(16,32):
                        for j in range(32):
                                VVddComp.append((((self.
↪VddCompgrad[(j+i*32)%128+128]*np.mean(ptat))/pow(2, self.VddScaling)+self.
↪VddCompoff[(j+i*32)%128+128])/pow(2, self.Vddoff))*(self.Vdd-self.
↪VddCalib1-((self.VddCalib2-self.VddCalib1)/(self.PtatCalib2-self.
↪PtatCalib1))*(np.mean(ptat)-self.PtatCalib1)))
                self.VVddComp=VVddComp
                return im-np.reshape(self.VVddComp,(32, 32))
```

```
        def measure_electrical_offset(self, blind=True):
↪#measure_electrical_offset
                pixel_values = np.zeros(256)
                ptats = np.zeros(8)

        self.send_command(self.generate_expose_block_command(0, blind=blind),␣
↪wait=False)

                query = [I2C.Message([0x02]), I2C.Message([0x00], read=True)]

                read_block = [I2C.Message([0x0A]), I2C.Message([0x00]*258,␣
↪read=True)]
                self.i2c.transfer(self.address, read_block)
                top_data = np.array(copy.copy(read_block[1].data))

                read_block = [I2C.Message([0x0B]), I2C.Message([0x00]*258,␣
↪read=True)]
                self.i2c.transfer(self.address, read_block)
                bottom_data = np.array(copy.copy(read_block[1].data))

                top_data = top_data[1::2] + (top_data[0::2] << 8)
                bottom_data = bottom_data[1::2] + (bottom_data[0::2] << 8)
        # bottom data is in a weird shape
                pixel_values[0:128] = top_data[1:]
                # bottom data is in a weird shape
                pixel_values[224:256] = bottom_data[1:33]
                pixel_values[192:224] = bottom_data[33:65]
                pixel_values[160:192] = bottom_data[65:97]
                pixel_values[128:160] = bottom_data[97:]
                ptats[block] = top_data[0]
                ptats[7-block] = bottom_data[0]

        self.elloff=pixel_values;


        def electrical_offset(self,im):#electrical offset calculate
        V_new = np.zeros((32,32))
            for i in range(16):
                for j in range(32):
                V_new[i,j]=self.elloff[(j+i*32)%128]
            for i in range(16,32):
                for j in range(32):
                V_new[i,j]=self.elloff[(j+i*32)%128+128]
        self.V_new=V_new
                return im - self.V_new
        def capture_image(self, blind=False):
                pixel_values = np.zeros(1024)
                ptats = np.zeros(8)

                for block in range(4):
```

```
                        print("Exposing block " + str(block))
                        self.send_command(self.
↪generate_expose_block_command(block, blind=blind), wait=False)

                        query = [I2C.Message([0x02]), I2C.Message([0x00],␣
↪read=True)]
                        expected = 1 + (block << 4)

                        done = False

                        while not done:
                                self.i2c.transfer(self.address, query)

                                if not (query[1].data[0] == expected):
                                else:
                                        done = True

                        read_block = [I2C.Message([0x0A]), I2C.
↪Message([0x00]*258, read=True)]
                        self.i2c.transfer(self.address, read_block)
                        top_data = np.array(copy.copy(read_block[1].data))

                        read_block = [I2C.Message([0x0B]), I2C.
↪Message([0x00]*258, read=True)]
                        self.i2c.transfer(self.address, read_block)
                        bottom_data = np.array(copy.copy(read_block[1].data))

                        top_data = top_data[1::2] + (top_data[0::2] << 8)
                        bottom_data = bottom_data[1::2] + (bottom_data[0::2]␣
↪<< 8)

                        pixel_values[(0+block*128):(128+block*128)] =␣
↪top_data[1:]
                        # bottom data is in a weird shape
                        pixel_values[(992-block*128):(1024-block*128)] =␣
↪bottom_data[1:33]
                        pixel_values[(960-block*128):(992-block*128)] =␣
↪bottom_data[33:65]
                        pixel_values[(928-block*128):(960-block*128)] =␣
↪bottom_data[65:97]
                        pixel_values[(896-block*128):(928-block*128)] =␣
↪bottom_data[97:]

                        ptats[block] = top_data[0]
                        ptats[7-block] = bottom_data[0]

                pixel_values = np.reshape(pixel_values, (32, 32))

                return (pixel_values, ptats)
```

```
        def generate_command(self, register, value):#periphery library␣
↪register activate
                return [I2C.Message([register, value])]


        def generate_expose_block_command(self, block, blind=False):#read␣
↪data command
                if blind:
                        return self.generate_command(0x01, 0x0B)
                else:
                        return self.generate_command(0x01, 0x09 + (block <<␣
↪4))


        def send_command(self, cmd, wait=True):#send data to registers
                self.i2c.transfer(self.address, cmd)
                if wait:
                        time.sleep(0.005) # sleep for 5 ms


        def close(self):#closed device
                sleep = self.generate_command(0x01, 0x00)
                self.send_command(sleep)
```

7.1.1 Capture Image

[ ]:
```
import numpy as np
import cv2
from htpa import *
import pickle
i = 0
k = 0
dev = HTPA(0x1A)

while(True):
    if (i == 5):
    dev.measure_observed_offset()
    dev.measure_electrical_offset()

    pixel_values, ptats) = dev.capture_image()  # Capture Image
    im = dev.temperature_compensation(pixel_values, ptats)  # thermal offset
    im = dev.offset_compensation(im)  # general offset
    if(k>5):
    im=dev.electrical_offset(im)#electrical offset
    im=dev.Vdd_Comperasition()#Vdd Comperasition
    im = dev.sensitivity_compensation(im)#Sensitivity

    # resize and scale image to make it more viewable on raspberry pi screen
    im = cv2.resize(im, None, fx=12, fy=12)
```

```
    im -= np.min(im)
    im /= np.max(im)
    imcolor=cv2.applyColorMap(im,cv2.COLORMAP_JET)

    cv2.imshow('frame', im)
    cv2.imshow('frame1', imcolor)

    i += 1

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

dev.close()

cv2.destroyAllWindows()
```

7.2 Creating Dataset

7.2.1 Resize Dataset

[ ]:
```
import cv2
import os
from PIL import Image
import numpy as np

src='/open_train/five/'
filenames_train=os.listdir(src)

print(len(filenames_train))
for f_name in filenames_train:
    im=Image.open(src+f_name)
    # Size of the image in pixels (size of orginal image)
    # (This is not mandatory)
#    width, height = im.size

    # Setting the points for cropped image
    # Setting the points for cropped image
    left = 120
    top = 45
    right = 390
    bottom = 240

    # Cropped image of above dimension
    # (It will not change orginal image)
    #im1 = im.crop((left, top, right, bottom))
    im1=im1.resize((32, 32))
    im1.save('/open_train/five_new/'+f_name)
```

7.2.2 Dataset Augmentation

[ ]:
```
# example of images augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
import os
# Passing the path of the image directory
src='/home/rabikkk/Desktop/final_project/last_dataset/close/'
path1='/home/rabikkk/Desktop/final_project/last_dataset/train/';
filenames_train=os.listdir(src)

print(len(filenames_train))
for f_name in filenames_train:
    # load the image
    img = load_img(src+f_name)
# convert to numpy array
    data = img_to_array(img)
# expand dimension to one sample
    samples = expand_dims(data, 0)
# create image data augmentation generator
    datagen1 = ImageDataGenerator(zoom_range=[0.8,1])
# create image data augmentation generator
    datagen = ImageDataGenerator(brightness_range=[0.6,1.0])
# create image data augmentation generator
    datagen2 = ImageDataGenerator(horizontal_flip=True)
# prepare iterator
    it = datagen.flow(samples, batch_size=1)
    it1 = datagen1.flow(samples, batch_size=1)
    it2 = datagen2.flow(samples, batch_size=1)
    it = datagen.flow(samples, batch_size=4, save_to_dir=path1,␣
 ↪save_prefix='index_test03', save_format='png')
    it1 = datagen1.flow(samples, batch_size=5, save_to_dir=path1,␣
 ↪save_prefix='index_test4', save_format='png')
    it2= datagen2.flow(samples, batch_size=4, save_to_dir=path1,␣
 ↪save_prefix='index_test5', save_format='png')
```

7.3 Static Gesture Learning Model

[ ]:
```
import numpy as np
import pandas as pd
import keras
from keras.preprocessing.image import ImageDataGenerator,load_img
#from keras.utils import to_categorical
from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
```

```
import os
import cv2
from PIL import Image,ImageOps
from numpy import asarray
```

7.3.1 Adding Train_set and Test_set

[ ]:
```
#Training Set
src='/home/rabikkk/Desktop/final_project/last_dataset/train/'
filenames_train=os.listdir(src)

categories_train=[]
image_train=[]
print(len(filenames_train))
close=0
index=0
last=0
open1=0
for f_name in filenames_train:
    image=Image.open(src+f_name).convert('RGB')
    image=ImageOps.grayscale(image)
    #image=cv2.imread(src+f_name)
    #image=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    numpydata = asarray(image)
    image_train.append(numpydata)
    #print(len(image_train))
    category=f_name.split('_')[0]
    if category=='close':
        categories_train.append(0)
        close+=1
    elif category=='index':
        categories_train.append(1)
        index+=1
    elif category=='last':
        categories_train.append(2)
        last+=1
    else:
        categories_train.append(3)
        open1+=1

df=pd.DataFrame({
    'filename':filenames_train,
    'category':categories_train
})

image_train=np.asarray(image_train)
image_train = image_train.reshape((image_train.shape[0],32, 32,1))
image_train = image_train.astype("float32") / 255.0
categories_train=np.asarray(categories_train)
categories_train=categories_train.reshape(len(filenames_train),1)
```

```
print(image_train.shape)
print(categories_train.shape)
print(close)
print(last)
print(open1)
print(index)
#print(categories_train)
```

[ ]:
```
#Test Set
src='/home/rabikkk/Desktop/final_project/last_dataset/test/'
filenames_test=os.listdir(src)

categories_test=[]
image_test=[]
close=0
index=0
last=0
open1=0
print(len(filenames_test))
for f_name in filenames_test:
    image=Image.open(src+f_name).convert('RGB')
    image=ImageOps.grayscale(image)
     #image=cv2.imread(src+f_name)
   #image=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    numpydata = asarray(image)
    image_test.append(numpydata)
   # print(len(image_test))
    category=f_name.split('_')[0]
    if category=='close':
        categories_test.append(0)
        close+=1
    elif category=='index':
        categories_test.append(1)
        index+=1
    elif category=='last':
        categories_test.append(2)
        last+=1
    else:
        categories_test.append(3)
        open1+=1


df=pd.DataFrame({
    'filename':filenames_test,
    'category':categories_test
})
image_test=np.asarray(image_test)
image_test = image_test.reshape((image_test.shape[0], 32, 32,1))
image_test = image_test.astype("float32") / 255.0
#image_test =image_test.reshape(len(image_test),(32,32))
```

```
#image_test= np.ndarray(shape=(2050, 32, 32, 1))
#image_test=image_test/255.0
#image_test=image_test.reshape(2008,32,32)
categories_test=np.asarray(categories_test)
categories_test=categories_test.reshape(len(filenames_test),1)

print(image_test.shape)
print(categories_test.shape)
print(close)
print(last)
print(open1)
print(index)
```

7.3.2 Model

[ ]:
```
import keras
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D,AveragePooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout,LeakyReLU
from tensorflow.keras.utils import plot_model
#Instantieate an empty model
model = Sequential(name="Static_Gesture_Model")

#C1 Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu',␣
 ↪input_shape=(32, 32,1)))
#S2 Pooling Layer
model.add(MaxPooling2D(strides=2))
#C3 Convolutional Layer
model.add(Conv2D(filters=48, kernel_size=(5,5), padding='valid',␣
 ↪activation='relu'))
#S4 Pooling Layer
model.add(MaxPooling2D(strides=2))
#Flatten the CNN output so that we can connect it with fully connected layers
model.add(Flatten())
#Fully Connected Layer
model.add(Dense(256, activation='tanh'))
#Fully Connected Layer
model.add(Dense(84, activation='tanh'))
#Fully Connected Layer
model.add(Dense(10, activation='tanh'))
#Output Layer with Softmax Activation
model.add(Dense(4, activation='softmax'))
model.summary()
```

7.3.3 Learning

```
[ ]: import numpy as np
import keras
from keras.callbacks import ModelCheckpoint
from keras.callbacks import EarlyStopping
import tensorflow as tf
print("The length of list is: ", len(image_train))
print("The length of list is: ", len(image_test))

# early stopping
early_stop = EarlyStopping(patience=30, monitor='val_loss')
opt = keras.optimizers.Adam(learning_rate=0.000025)
model.compile(optimizer=opt, #'adam'
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model_name = "model"
filepath='/home/rabikkk/Desktop/final_project/learning/' + model_name + '.
 ↪hdf5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,␣
 ↪save_best_only=True, mode='auto')
logpath = '/home/rabikkk/Desktop/final_project/learning' + model_name + '.log'
csv_logger = keras.callbacks.CSVLogger(logpath)
callbacks_list = [checkpoint,csv_logger]
history = model.fit(image_train, categories_train,␣
 ↪epochs=100,validation_data=(image_test, categories_test),␣
 ↪callbacks=[callbacks_list,early_stop])#callbacks=[callbacks_list,early_stop]
```

7.3.4 Model Learning Plot

```
[ ]: import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs = range(len(acc))

plt.plot(epochs, acc, label='Training acc')
plt.plot(epochs, val_acc, label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.ylim(0.9,1)
plt.show()

test_loss, test_acc = model.evaluate(image_test, categories_test, verbose=2)
print(test_loss)
print(test_acc)
```

7.3.5 Model Learning Plot

[ ]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(epochs, loss, label='Training loss')
plt.plot(epochs, val_loss, label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.ylim(0,0.5)
plt.show()
```

7.3.6 Testing Models

[ ]:
```
from keras.models import load_model
categories_valtest=[]
model = tf.keras.models.load_model('/home/rabikkk/Desktop/final_project/
 ↪learning/deneme.hdf5')
images = []
# Test edeceğin datanı preprocess et trainingde verdiğin input haline getir.
main_folder0 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set1/'
#main_folder1 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set2/
 ↪'
#main_folder2 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set3/
 ↪'
#main_folder3 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set4/
 ↪'
#main_folder4 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set5/
 ↪'
for f_name in sorted(os.listdir(main_folder0)):
    #image = Image.open(main_folder + f_name)
    image = Image.open(main_folder0 + f_name)
    image=ImageOps.grayscale(image)
    #image_array = asarray(image)
    image_array=np.array(image)
    images.append(image_array)
    category=f_name.split('_')[0]
    if category=='close':
      categories_valtest.append(0)
    elif category=='index':
      categories_valtest.append(1)
    elif category=='last':
      categories_valtest.append(2)
    else:
      categories_valtest.append(3)

df=pd.DataFrame({
    'filename':images,
    'category':categories_valtest
})
```

```
categories_valtest=np.asarray(categories_valtest)
categories_valtest=categories_valtest.reshape(len(images),1)
images=np.asarray(images)
images = images.reshape((images.shape[0],32, 32,1))
images = images.astype("float32") / 255.0
print(images.shape)
yhat = model.predict([images])
#print('Predicted: %.3f' % yhat[0])
predictions = model.predict_classes(images)

print(predictions)
# summarize the first 5 cases
count=0;
for i in range(len(images)):
    if predictions[i]==categories_valtest[i]:
        count+=1
    #print('%s%d=> %d (expected %d)' % (f_name,i,predictions[i],␣
 ↪categories_valtest[i]))
print('Result:%f' % ((count/len(categories_valtest))*100))
```

Bibliography

[1] Adafruit's Raspberry Pi Lesson 4. GPIO Setup.
URL https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c

[2] MultiModalHandGesture_dataset.
URL $http://www.gti.ssr.upm.es/data/MultiModalHandGesture_dataset$

[3] OpenCV: OpenCV-Python Tutorials.
URL $https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html$

[4] vsergeev. python-periphery: A pure Python 2/3 library for peripheral I/O (GPIO, LED, PWM, SPI, I2C, MMIO, Serial) in Linux.
URL https://github.com/vsergeev/python-periphery

[5] Fukushima, K. and Miyake, S. "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position". Pattern Recognition, volume 15, no. 6, pages 455–469, 1982.
URL https://linkinghub.elsevier.com/retrieve/pii/0031320382900243

[6] Lecun, Y., Bottou, L., et al. "Gradient-based learning applied to document recognition". Proceedings of the IEEE, volume 86, no. 11, pages 2278–2324, 1998.
URL http://ieeexplore.ieee.org/document/726791/

[7] Krizhevsky, A., Sutskever, I., and Hinton, G. E. "ImageNet classification with deep convolutional neural networks". Communications of the ACM, volume 60, no. 6, pages 84–90, 2017.
URL https://dl.acm.org/doi/10.1145/3065386

[8] Li, G., Tang, H., et al. "Hand gesture recognition based on convolution neural network". Cluster Computing, volume 22, no. S2, pages 2719–2729, 2017. Publisher: Springer Science and Business Media LLC.

[9] Himblot, T. Data augmentation : boost your image dataset with few lines of Python, 2018. Publication Title: Medium.
URL https://medium.com/@thimblot/data-augmentation-boost-your-image-dataset-with-few-lin

[10] Lupp, S. "HTPA32x32dR2L5.0/0.85F7.7eHiC Thermopile Array With Lens Optics Rev3.0". Technical report, Heimann, 2018.

[11] Phung, V. H. and Rhee, E. J. "A Deep Learning Approach for Classification of Cloud Image Patches on Small Datasets". Journal of Information and Communication Convergence Engineering, volume 16, no. 3, pages 173–178, 2018.
URL https://doi.org/10.6109/JICCE.2018.16.3.173

[12] V, A. and R, R. "A Deep Convolutional Neural Network Approach for Static Hand Gesture Recognition". Procedia Computer Science, volume 171, pages 2353–2361, 2020. Publisher: Elsevier BV.