UiA University of Agder

Master

Hand Gestures Recognition using Thermal Images

DANIEL SKOMEDAL BRELAND

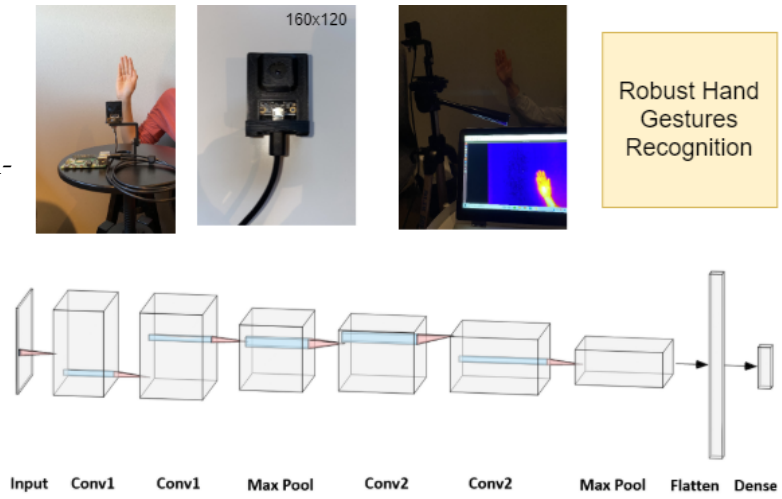
SUPERVISOR
Linga Reddy Cenkeramaddi

University of Agder, 2021
Faculty of Engineering and Science
Department of Engineering Sciences

Acknowledgements

I would like to thank Assoc. Prof. Linga Reddy Cenkeramaddi for the help and guidance through out this project. I would also like to thank Aveen Dayal, for helping me understand and in development of the CNN model. Lastly, I would thank Sreenivas Reddy Yeduri for proof reading and tips to a well formulated report.

Abstract



Hand gesture recognition is important in a variety of applications, including medical systems and assistive technologies, human-computer interaction, human-robot interaction, industrial automation, virtual environment control, sign language translation, crisis and disaster management, entertainment and computer games, and robotics. RGB cameras are usually used for most of these applications. However, their performance is limited especially in low-light conditions. It is challenging to accurately classify the hand gestures in dark conditions. In this thesis, we propose the robust hand gestures recognition based on high resolution thermal imaging. These thermal images are captured using FLIR Lepton 3.5 thermal camera which is a high resolution thermal camera with a resolution of 160×120 pixels. Thereafter, we feed the captured thermal images to a deep CNN model to accurately classify the hand gestures. We evaluate the performance of the proposed model with the benchmark models in terms of accuracy as well as the inference time when deployed on edge computing devices such as Raspberry Pi 4 Model B and NVIDIA JETSON AGX XAVIER. Through extensive results, we observe that the proposed model achieves an accuracy of 98.81% that is compared to MobileNetV3 Large model with 99.98% and MobileNetV3 Small model at 99.72% accuracy. The inference time is 0.075138s on Nvidia Jetson AGX, compared to the inference time on Raspberry Pi 4 that is 0.140968s. Even though the accuracy of the proposed model is comparable to the benchmark models, the size of the proposed model is very less comparable to the benchmark models.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Gesture is a very important part of our communication language, as it may show the importance or engagement between communicators. The normal conversation contains a lot of gestures, either face or hand gestures. Hand gesture recognition is useful for users who are in the need of physical access to spaces unavailable for humans, to communicate with other people or machines. A dynamic gesture recognition named Nintendo Wii has been on the entertainment market for a long time [5]. To develop a hand gesture application like this, it would require a high resolution camera and still be able to operate at live timing speeds. As the quality of the system is dependent on the quality of every part, the introduction of high resolution images with more pixels needs high computational power to operate at the same speed. This in turn increases the cost of the system due to the usage of high quality equipment. To find the best cost/benefit solution the system would need a lot of field testing. This would also discover which areas that needs improvement [30]. Further, in normal RGB images, the lighting and background lighting is a big factor. This factor will make image recognition systems more vulnerable to changes in scenery or time of day. Switching to a thermal camera with Infra Red (IR) lighting will address this issue, as it does not depends on the visible light. An obstacle will be clearly visible in an image, if the background temperature is different from the object. This makes the thermal camera to work even in complete darkness. The thermal camera is then programmed to process these IR rays into human visible colors, of the users choice. As image recognition is hard enough itself, thermal cameras have the upside of mainly reproduce the hottest objects. Meaning the images are often more simple and contains less details [49], [48].

Thus, in this thesis, we focus on using thermal images for hand gesture recognition. The images are taken with separate gestures that are supposed to look somewhat alike. This is to increase the difficulty and robustness of the system. We created two datasets. The first dataset is created with a clear background such that the image is as clear as possible. In these images the hand will be in focus without any dots or noise in the frame. With this dataset, it will be easier to create a model for solid classification accuracy. The second dataset is created with background noise as well as by placing the objects in the background. This gives more texture for an algorithm to be considered for classification, as the hand gestures may not be clear in all images. We consider lamps, windows, monitors, and furniture to create the second dataset due to which the hand may be colder than the background. This will make the algorithm to find the patterns and not just colors.

In [2], the authors have created a dataset with thermal camera which is available in [43]. Similar to the dataset in [43], our datasets are created on the same basis, with 24 images per gesture with a total of 10 gestures. The images are captured using FLIR Lepton 3.5 thermal camera module [57] and a Pure-thermal 2 breakout board [23]. All images are captured with a resolution of 160×120 pixels, and about 40cm distance between camera and hand. The first dataset is created from 30 people, all with 240 color scale and 240 gray scale images which results in a total of 14400 images. The gesture controlled appli-

cations are possible to implement in a lot of settings, to assist humans to communicate with other humans or machines. This makes the work relevant within entertainment, health care, gaming and many other possible areas [1], [31], [60].
To be able to do hand gesture recognition, we propose a Convolutional Neural Network (CNN) model to learn and classify hand gestures from the created datasets. In order to increase the learning ability and to make the system usable in practice, the most well known gestures should be used. Then the system should be expandable, either by offline training or online updates with new gestures. This will make it a more adapted system to each users gestures, and how they uses it [30].

1.1 Organization

Name	Role	Company	Email
Daniel S. Breland	Student	UiA	danisb15@uia.no
Linga Reddy Cenkeramaddi	Supervisor	UiA	linga.cenkeramaddi@uia.no

Table 1.1: Project roles

As a weekly support, there was meetings once a week on Teams. This was setup in the beginning to keep all parties updated, as well as keep working in the correct direction. Status updates has been given for each week, to evaluate the progress and guide towards solutions. The meetings has mostly been held online, using Teams. From time to time, it has been physical meetings at Campus in Grimstad. These physical meetings has mostly been extra and when there was work to be done on campus.

Meeting place	Participants	Frequency	Day of week
Online	Student and supervisor	Weekly	Wednesday
Physical	Student and Supervisor	Irregular	Any weekday

Table 1.2: Meetings

1.2 Similar Products

Image recognition is a field of research that has gained a lot of attention in resent years and will be an existing issue for researcher to come. A lot of work has been carried out in image recognition with RGB images [6]. However, less research has been carried out in thermal image recognition. A hand gesture recognition sensor that uses reflected impulses in Ultra-Wideband (UWB) spectrum from a hand [33]. A Convolutional Neural Network (CNN) that is used to identify six hand gestures, using UWB impulse radar sensor alongside an 8×8 pixel thermal sensor [52]. A deep learning based hand gesture recognition approach has been used, where surrounding noise may limit the performance of such a system [52]. A three-axis accelerometer and gyroscope sensor-based continuous hand gesture recognition technique, in a smart device is proposed in this work [24]. The usage of simultaneous pressure sensors for hand gesture recognition has been explored in [4] where these sensors capture the signals related to muscle movement. Here, it has been proposed a work with an extreme learning method where 11 gestures were classified [4]. A long short-term memory (LSTM) model has been proposed in [63] to calssify the different hand gestures, by using data from inertial measurements unit (IMU), electromyographic (EMG), and finger and palm pressure sensor. As these previous works are similar, they are however not fitted to use in many practical environments and applications.

1.3 Background

In medical, entertainment, industrial and many more places, the use for a robust and reliable image recognition system will have many benefits. As the technology is developing and the human-robot interactions are increasing, controlling them using gesture control will be a big assist for many people. For people with certain disabilities, it can be helpful in daily life to open doors or to control indoor temperature. However, it is inefficient in a noisy environment where voice control will be unavailable or it is not possible to speak. If the working environment has changing light, normal RGB images will go from bright to dark depending on external factors. Thus, thermal images will provide a reliable and constant flow of data, that is not dependent on external sound or lightning.

1.4 Limitations

The time aspect is a very limiting factor as any new device requires time to learn. Further, new parts in a device increases the learning time as they need much time for learning. On the other end, mmWave radar is a complex that was dropped, due to the complexity of learning and utilizing it. The amount of new components in the proposed system are already high with a FLIR Lepton module, CNN model creation and using it on the Nvidia Jetson AGX Xavier. With the local restrictions of Covid-19 rules, the cooperation with others have also been hampered due to which I had limited possibilities to have physical meetings, as most people are working from home.
Thermal cameras have some limitations, as they are not perfect in any situation. When thermal camera acts as a standalone sensor, it will not cover all forms of noise or disturbances. As mentioned earlier, it is not affected by surrounding light and will function even in complete darkness. However, it will most be affected if pointed straight towards a light source. Depending on the distance, size, and temperature of the source, the light sources might be the dominant temperature, making the object in focus harder to get detected and recognized. Thus, the thermal camera may not be efficient in a fire place or any heat source.
Similar to the Omron D6T thermal camera used in [6], the resolution of thermal cameras like FLIR Lepton 3.5, is high compared to other similar devices. However, the thermal cameras are expensive as compared to RGB cameras.

1.5 Structure of the thesis

1. Chapter 1

    - At the start of the thesis, introduction to the project is described. How is the state of the art, what is supposed to be developed and other important factors for the solution. This chapter will also describe the cooperation and working environment during this time.

2. Chapter 2

    - The theory chapter is used to give a certain knowledge about the used elements. To give any reader the basic knowledge to understand the technical terms. It will describe the most important hardware and software components, to understand how and why they are used.

3. Chapter 3

    - In this chapter, the way things have been solved is explained. It explain the way different parts have been developed, before they were put together with other

objects to create a product. With the main focus on data gathering and model creation.

4. Chapter 4

    - The results are found and evaluated. The comparison between models and how the performance in different aspect is, along with some theoretical alternative solutions.

5. Chapter 5

    - This chapter will take on the discussion on how the project has evolved, how the end product is, challenges and what could be done to further develop the system.

6. Chapter 6

    - The conclusion will summarize the previous mentioned chapters, collect the key parts and how the end system is. It answers whether if the results is a solution to the problem statement and if it is a solid results.

Apart from the chapters, the content will be listed as in the table of contents (ToC). With Acknowledgments, Abstract, Figures and Tables. After all chapters it will be the appendices and references in the end.

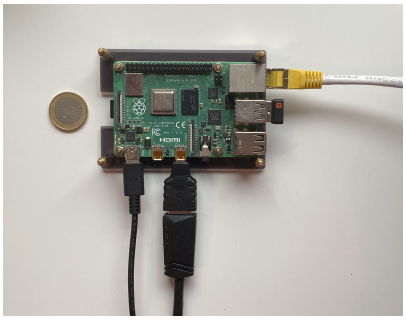
Chapter 2

Theory

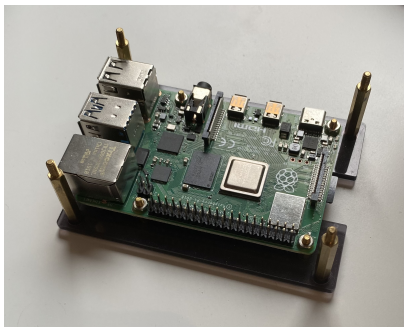
In this chapter, we describe the details of the hardware components and software tools used in this project.

2.1 Raspberry Pi 4B

Raspberry Pi is a powerful minicomputer that is available in affordable price which is used for development purpose. The Raspberry Pi Foundation has its own operating system (OS) called Raspberry Pi operating system, very similar to other Linux based OS. In RPi software web page [45], it is also mentioned that it is possible to run a RPi with another OS. Thus, as part of this project, in addition to Raspberry Pi operating system [45], Linux based operating systems that include Ubuntu [12]. These operating systems support most of the programming languages and software development tools. Thus, it has many use cases due to its ability of determining the operating system. There are many versions of the Raspberry Pi with Raspberry Pi 4 Model B as the latest one released from Raspberry Pi Foundation.



(a)



(b)

Figure 2.1: Images of Raspberry Pi 4B: (a) Top view of RPi 4B compared to a 1 Euro coin; (b) Sideways view of the Raspberry Pi 4B

In this project, we used Raspberry Pi 4 Model B as an edge computing device for the performance evaluation of the proposed model. Raspberry Pi 4 Model B comes with a 64-bit Quad core Cortex-A72 (ARMv8) System on Chip (SoC) processor, with 1.5GHz clock speed. This model has a customized LPDDR4-3200 SDRAM memory of 2, 4 or 8GB. It has different wireless connectivity such as Bluetooth 5.0 and BLE (Bluetooth Low Energy) and IEEE 802.11ac operating at 5GHZ and 2.4GHz connections for better range and bandwidth. The processor is supported with OpenGL ES 3.0 graphics that enhances the speed of any graphical interface. As the whole computer is one socket (SoC), it also has a micro-SD card holder for inserting SD card with operating system. This micro-SD card can also be used for storage [47].

The Raspberry Pi 4 Model B is a small and powerful computer, powered by a USB-C port. To run its processors and other components, it requires a voltage of 5.1V with 3A current. It is then able to run two 4K resolution monitors through its micro-HDMI ports. It has four USB ports among which two ports are USB v2.0 and two are USB v3.0 ports that gives 10 times transfer rate in comparison to USB v2.0. In addition to wireless network connectivity, it also comes with a Gigabit Ethernet port [46].

2.2 Ubuntu 18.04 LTS

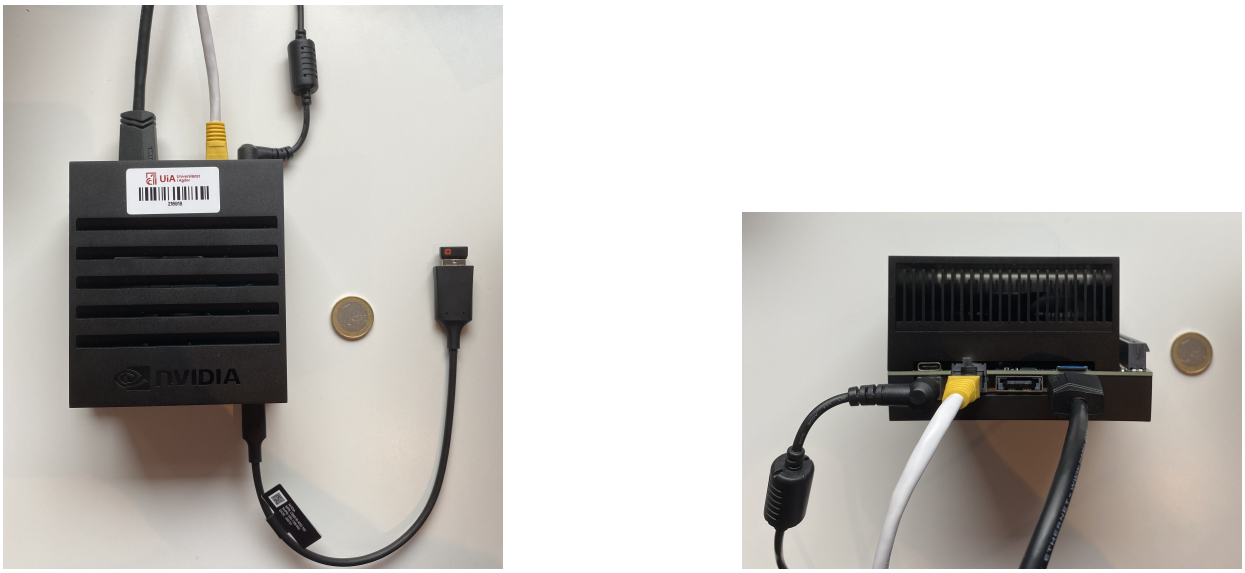
In order to work with the machine learning models, we installed tensorflow on Ubuntu 18.04, 64-bit version. Ubuntu is a well established OS for use in many cases, from lightweight computers to more heavy machinery. The adaptability and availability of this free OS makes it very useful for developers. It is observed in [14] that the Ubuntu can be deployed directly or can be used on Virtual Machine (VM) for deploying machine learning models.

Ubuntu is available in a lot of different versions and is continuously updated and developed. In Ubuntu version a.b, a represents the year of release and b represents the month. So Ubuntu 18.04 LTS was released in April 2018 [13]. It has been updated quite a lot since the release and currently fifth version i.e., Ubuntu 18.04.5 is available. Further, ubuntu 18.04 LTS version has been released in April 2018 which will be supported and updated for five years from the time of release. Ubuntu version can be downloaded from its original website [11] or through a google search for Ubuntu downloads. Ubuntu can be installed in either a PC or other device with 32-bit and 64-bit versions of processor. Normally, the Raspberry Pi OS will be downloaded at 32-bit, but a 64-bit OS will be faster and may support the programs and software that a normal PC support.

2.3 Nvidia Jetson AGX Xavier DK

NVIDIA company has developed Jetson AGX Xavier [38] which acts as an edge computing device for deploying machine learning and power artificial intelligence models. The AGX Xavier is a high performance module with low power consumption for deep learning and computer vision [40]. The AGX Xavier works as a Linux machine, when installed with an SDK, called JetPack. JetPack provides an operating system as well as the most important software for machine learning. The JetPack bundle installs many packages that include TensorRT, CUDA toolkit, VisionWorks, and OpenCV [40]. Alternative download options are available in NVIDIA website [39]. With JetPack installed, the machine could also be used as a normal Ubuntu machine for other purposes without modifications. By using the USB-C to USB adapter, it can be connected to a mouse and keyboard. For Internet connection, it is required to connect using Ethernet port. The HDMI port can be used to connect this module to monitor and micro-SD card slot is available for inserting a micro-SD card for storage.

NVIDIA Jetson AGX Xavier is built with a 512-core Volta GPU, with Tensor Cores. The CPU is an 8-core ARM v8.2, 64-bit with 8MB L2 + 4MB L3 cache. The AI focus requires a lot of memory and the AGX Xavier has 32GB RAM with 256-bit LPDDR4x that handles 137GB/s. The internal storage is 32GB eMMC 5.1, which can be expanded using a micro-SD card. To make the AGX Xavier a solid platform for AI, it is installed two NVDLA Engines to accelerate deep learning and a 7-way VLIW Vision Processor for accelerating vision tasks. It measures 105mm x 105mm x 65mm in width, length, and height. It comes with two USB-C ports among which one is used for flashing and one is free to use. It is also embedded with UART, SPI, I2C, and more technologies [38, 37].



(a) (b)

Figure 2.2: Images of Jetson AGX Xavier: (a) Top view of AGX compared with a 1 Euro coin; (b) Sideview of the different ports compared to a 1 Euro coin;

2.4 FLIR Lepton

FLIR Lepton 3.5 is a thermal camera from FLIR with the highest resolution Long Wavelength Infrared (LWIR) micro camera. It has a resolution of 160×120 pixel and a radiometric calibrated array of 19200 pixels [55].
Lepton 3.5 requires an external power supply ranging from 2.5V to 3.1V and has a nominal power consumption of 160mW. It can reach a peak power of 650mW for about one second during shutter events, while the low power mode consumes only 5mW. As a result it has many applications that includes IoT or small battery-powered objects. The Field Of View (FOV) is 57° (horizontaly). The thermal sensitivity is 50mK and each pixel is measured individually with a pixel pitch at 12 μm. Sensing temperature range is between -10°C to +140°C, while operating temperature is from -10°C to +65°C. To name a few integrated functions, there is a digital thermal image processing function, automatic environment compensation, noise filters, and gain control. Images are exported at<9Hz with video transmitted via SPI and the module is controlled via I2C [56].
Flat-Field Correction (FFC) is used by a camera module to accurately display the heated areas. The FFC will correct the temperatures that the camera detects in order to produce a more clear and uniform image. Here, FFC will re-calibrate to provide the best image quality [56]. When taking photos, the difference in image quality is visible which will be discussed in later sections. An FFC is performed during the startup sequence, in automatic mode, at regular intervals of 3 min. When turned on, it has some default settings such as auto FFC mode with an interval of 180 000 ms, delta temperature of 150°C, gain mode set to high, color Fusion with Raw 14 video output, and radiometry control enabled in addition to the other settings [55].

2.4.1 Lepton User App

A computer with windows OS will allow the use of the Lepton user app [54]. Developed by FLIR, to use the Lepton as a webcam with different settings and possibility to capture images. The Lepton image is displayed in a large area as a live feed, with possibilities to capture video or screenshots to the top right as shown in Fig. 2.3. In the left side it has several system configurations, in the current window it could go from colored (plasma) images, to gray scaled images with other system settings like number of frames, shutter

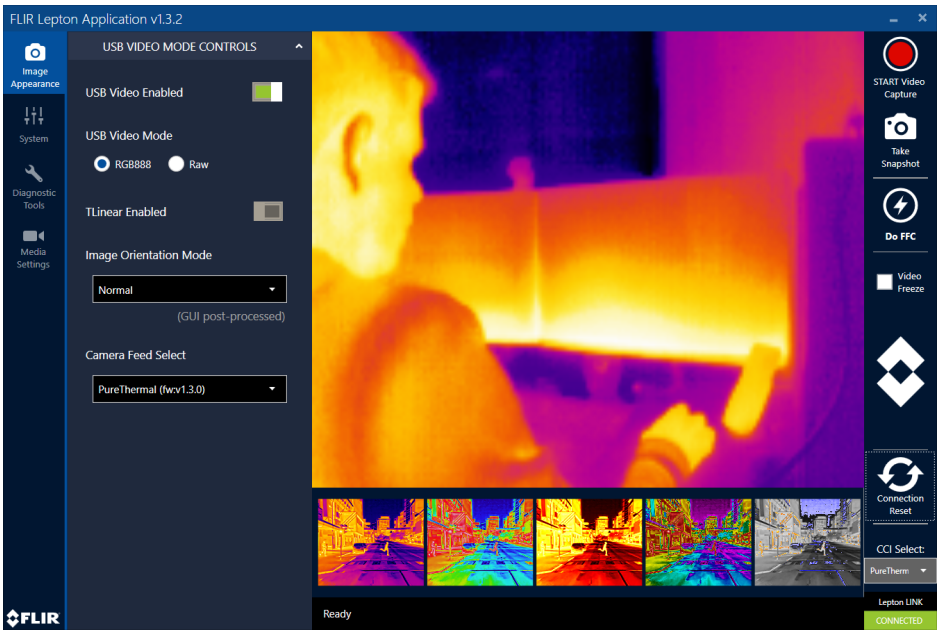


Figure 2.3: Screenshot of Lepton User App

mode, FFC, and delta temperature. Then below it is the media settings, which denotes the video quality, name and save path as well as screenshot name and save path. When all settings are set, it is fully operable and could be used to capture images. Although it is manually done, where every image needs to be taken separately.

2.5 Purethermal 2 breakout board

The Purethermal 2 breakout board is a FLIR Lepton smart I/O module with pre-configured plug-and-play functionality via a USB port. The default settings are compatible with standard webcam applications running on Linux, Windows, Mac, and Android. If the module is configured to be used for other purposes, the firmware is open source [32]. For more information, see Purethermal breakoutboard firmware github [53]. STM32 ST-LINK Utility, a software program for operating STM32 processors, is recommended for flashing the Purethermal 2 breakout board. A FLIR Lepton device is required to operate the Purethermal 2 breakout board [23].

2.6 Convolutional Neural Network

Convolutional neural network (CNN) is popular to use in image classification, as it has high precision and is more robust than other types of neural networks. A computer will see an image in the style of an array of numbers, which means it will transform an image from a matrix shape to an array. It sees the dimensions of the image such as height, width, and depth $(H \times W \times D)$. Height and width being the image resolution and dimension corresponds to the colors of the image (example, grayscale being 1 dimension). The advantage of CNN is to effectively detect the object in a image irrespective of the location of the object in the image [3]. Next, we discuss the different layers of the CNN.

2.6.1 Convolution Layer

In a Convolution Layer, the input image will be seen as a matrix with values corresponding to the pixel colors. These values are then taken through a filter (kernel), which multiply the input matrix with filter matrix and gives the output as another matrix as shown in Fig. 2.4. This filter is normally a 3×3 matrix, but could be altered to fit in the current

model. The new matrix will be filled up with these new values, creating an image with other pixel values. The size of the output matrix also depends on the values of padding, dilation and stride. Padding is to add zeros on edges of image in order to effectively capture the edge values effectively. Padding will also helps in maintaining the output size in similar to input matrix. Dilation will be described in the Dilated Convolution Layer section and stride is how many pixels the filter should move when calculating [3].

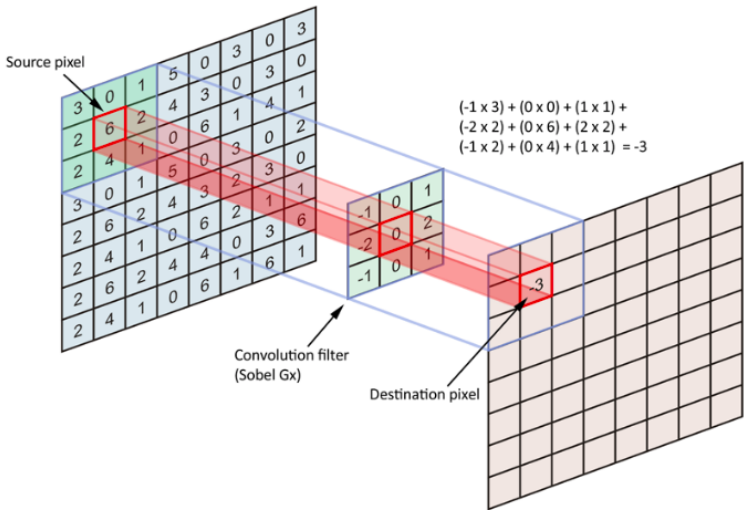


Figure 2.4: Example of convolution filter calcultaion using a filter. Image from [51]

2.6.2 Dilated Convolution Layer

This layer is a type of Convolution Layer whose kernel map's field of view is ≥ 1 [62]. This variable length field of view is parameterised by the variable 'dilation rate'. The dilation rate is a filter with "holes" in it as shown in Fig. 2.5. The "holes" will increase its reception area, while it still maintains the number of parameters. As it is less complex than using other layers to keep the number of parameters down, it is computationally less expensive than pooling layers. While it is easier to compute and also covers more area, it will not reduce the image resolution. This ensures that upsampling will not be necessary for the image to be further processed, as the input and output image have the same resolution. In addition, the output from the dilated convolution will contain details from a larger area of the image [35].

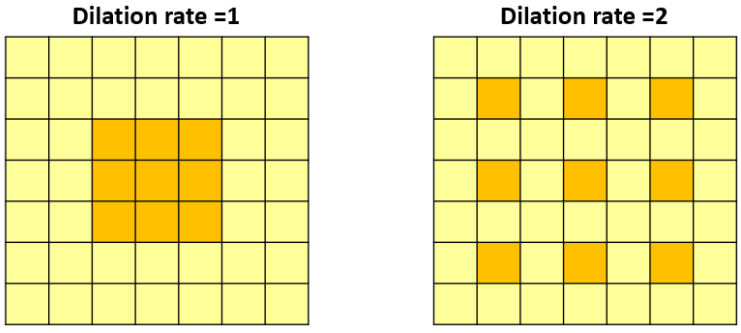


Figure 2.5: Kernel map's field of view for Convolution Layer with dilation rate of 1 and 2.

2.6.3 Max Pool Layer

Pooling layer is used to reduce the dimensions of the feature map while extracting efficient representations. There are different variants available for Pooling layer namely Max Pool

and Average Pool [61]. In pooling layers, the objective is to reduce the size and parameters for the model to calculate and thereby reduce file size so as to increase the speed. Max Pool works based on the size of the input matrix. For example consider an input matrix of size 4×4. It will then take the maximum value of the top left 2×2 matrix from this input matrix and put it as first element of 2×2 output matrix. In the pooling layer, the preferable stride value is greater than 1, but not necessary if there are many filters. It will then perform the same calculation for all 2×2 matrices in the 4×4 matrix, until it has a new 2×2 matrix with all maximum values as shown in Fig. 2.6.

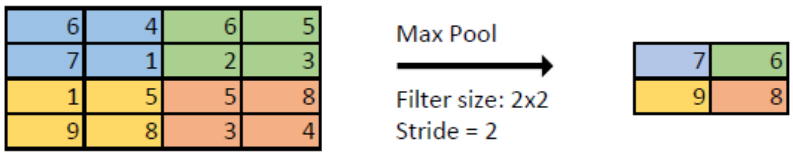


Figure 2.6: Max Pooling Layer visually explained.

Average Pooling layer will take the average of the 2×2 matrix as output, while Sum Pooling will take the sum of the 2×2 matrix as output.

2.6.4 Activation Layer

Activation layers are used to give the input a definition for a neuron in the CNN. The output of an activation layer depends on the type of activation function. Some of the activation functions are explained with examples in [20]. Below, we describe some of the activation functions.
Sigmoid function is a nonlinear activation function which is defined as

$$y = \frac{1}{1 + e^{-x}} \tag{2.1}$$

This activation function results an output between 0 and 1 depending on the level of activation. The disadvantage with Sigmoid activation function is that the output is not zero centered as well as the augment values are zero for saturated values of input [16].
Rectified Linear Unit (ReLU) function is a non linear function whose output is defined as

$$y = \max(0, x) \tag{2.2}$$

where, x denotes the input. Here, $y = 0$ for negative values of x and $y = x$ for positive values of x [20]. The output argument value is zero for negative values of input. To address the problems with ReLU, Leaky ReLU, Parametric ReLU, and Exponential ReLU activation functions are proposed in [17].

2.6.5 Fully Connected Layers

The fully connected layers are placed towards the last part in the CNN models. The Flatten layer transforms the 3D output from a previous layer to a one dimension vector. By adding the output with multiple dimensions, it will create a vector based on the input it gets and remove the other dimensions and feeding it into a fully connected layer. An example of Flatten layer has been described in [19]. The Dense layer is often referred to as a fully connected layer, depending on the form [41]. It is the last layer that converts all inputs to the desired output. It gives the output in one dimension, that is adapted to the desired classes for it to classify. An example of dense layer has been described in [18] where, developers often switches between the activation layers.

2.6.6 Softmax Layer

As explained in [20], the Softmax layer will convert a vector to a probability distribution, to make classification easier to understand. The input belongs to a class that results in highest probability with Softmax Layer [20].

2.6.7 TensorFlow

TensorFlow is a free to use platform for developing and deploying machine learning models. TensorFlow has some large add-on libraries to further develop machine learning possibilities, as it is designed to be usable for both novice and experienced developers. It allows it to use with other application Programming Interfaces (APIs), helping to ease the development of machine learning models. One example of an API would be Keras that helps in building and training machine learning models. It comes in different versions and is possible to use with a full or light version, called TensorFlow Lite (TFLite) [36].

Chapter 3

Methods

In this chapter, we describe thermal image capturing in different environments. We also discuss the proposed CNN model and benchmark models.

3.1 Dataset creation

A big part of this project is to collect data for processing. Then, we created two datasets from captured images. One with plain background and the other in the presence of background noise and different light conditions. The datasets are then fed into a CNN model for image recognition.

3.1.1 Camera stand

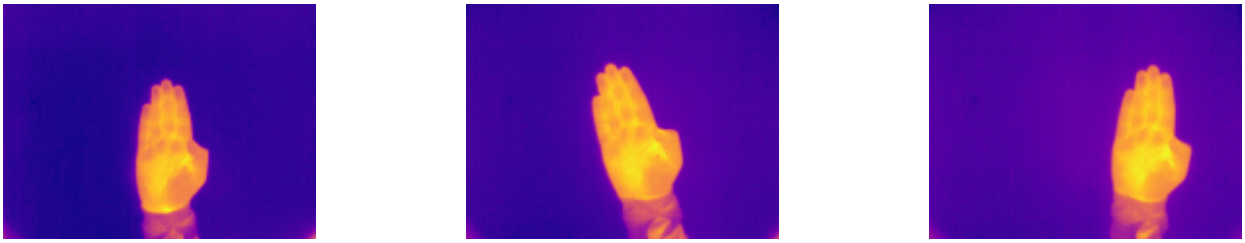
As the camera is only a breakout board and camera lens, it is easier to collect and capture data in a controlled environment using a stand. Fig. 3.1 shows the components required for the stand. The virtual image corresponds to these components will be obtained using Ultimaker Cura that creates a '.ufp' file from the virtual component data which will be processed further using 3D printer namely Ultimaker S5 to print the physical components to be mounted to create the stand for the camera.
Ultimaker Cura is a free software program to prepare drawings to be printed in 3D. The way it works is by importing the desired files, in STL, OBJ, X3D, 3MF, BMP, GIF, JPG, or PNG format. Then select the desired precision, material, and quality before slicing the object. Here, slicing creates a .ufp file that the Ultimaker S5 can read. It contains the drawing and how the machine will print it. When the slicing is done, it will tell the amount of time it takes to print and a preview of the printed objects [9]. Ultimaker Cura works with Windows, MacOS and Linux which makes it easily integrated on most of the computers. To enhance the possibilities in creating objects, it is also possible with plugins such as Solidworks, Autodesk Inventor or other plugin programs [8] that are designed for 2D/3D printing.
The operating screen on Ultimaker S5 is a 4.7" touch screen where, it can be controlled to the desired 3D print file and draw it. The Ultimaker S5 could draw objects in the size of 330 mm x 240 mm x 300 mm, with adjustable layer and XYZ resolution. The nozzle could also be changed to create more details or a larger nozzle to draw faster and thicker objects [10].
The printed parts are found on a website called Thingiverse.com, which have a lot of print ready files to use with Ultimaker Cura. The camera stand is not printed in one solid piece but created by several objects. These objects are created by two or more items that are attached to each other. The printed camera stand consists of 3 pieces including a container for the Lepton camera. The pieces are then attached as follow. To create the stand, the "L" shaped object is attached with bolts and nuts to the ground piece (middle top object in Fig. 3.1). As an extender or for more mobility, "I" shaped object, the left top

Figure 3.1: 3D printed pieces from Ultimaker S5

object in Fig. 3.1, can be mounted either on "L" shaped object or between the "L" shaped object and the object in top middle in Fig. 3.1 to adjust the height of the stand.
The details of the upper row objects in Fig. 3.1 can be found in [22]. While the camera housing is found here [28]. Mounted together they are measured to be 12cm height, around 7.5cm width, and about 6.5cm in depth when mounted as in Fig. 3.8b. A small tripod that is mobile and can easily be mounted on other surfaces or tripods.
The infill within the structures will provide better stability and more robustness when handled. Using infill will also help to carry any weight and have some structural resistance when tightening the bolts that keeps the different parts together. They were also printed with supporting structures, meaning all overhangs in a vertical direction would have infill. The infill is easy to remove afterwards but ensures structural stability when printing. The precision of the PLA is set to 0.15mm and 20% grade of infill in the structures.

3.1.2 Plain background image capturing

Lepton can be accessed with either an application from FLIR [54] or by programming. It can be accessed with Windows PC using SDK and python script can be used to access with Raspberry Pi. We consider Raspberry Pi 4 in this work as it will make the capturing of images easier due to the portability of camera module and Raspberry Pi. The dataset was captured in both fusion colors as well as grayscale to double the size of the dataset. Even though both types capture the same gesture, characteristics are different.



(a) (b) (c)

Figure 3.2: Example of hand movement within frame: (a) Hand position 1; (b) Hand position 2; (c) Hand position 3;

When collecting images, it was very important to have a stable and clear image which can be obtained from a setup as shown in Fig. 3.3. While capturing the images, people were told to move their hands slightly in order to capture the different variations in the

dataset. Some moved their hands willingly, while others moved naturally down as seen in Fig. 3.2. Fig. 3.2a shows the starting position, Fig. 3.2b is captured when the hand is moving naturally downwards, and Fig. 3.2c is captured when the hand is forced towards the body, in an angle that is not as natural.
The python script is modified to capture the images for a certain time or loops. This eases operation and will not require as much handling between each image. The downside is the speed and reliability of the Lepton module and breakout board when operated with software that is not constructed by an official source. The Raspberry Pi will not have the same processing speed and computational power of a full computer which makes it run slower when the dataset is starting to grow, and with 14 400 images the loading speed is slow and the quality control will be more demanding.

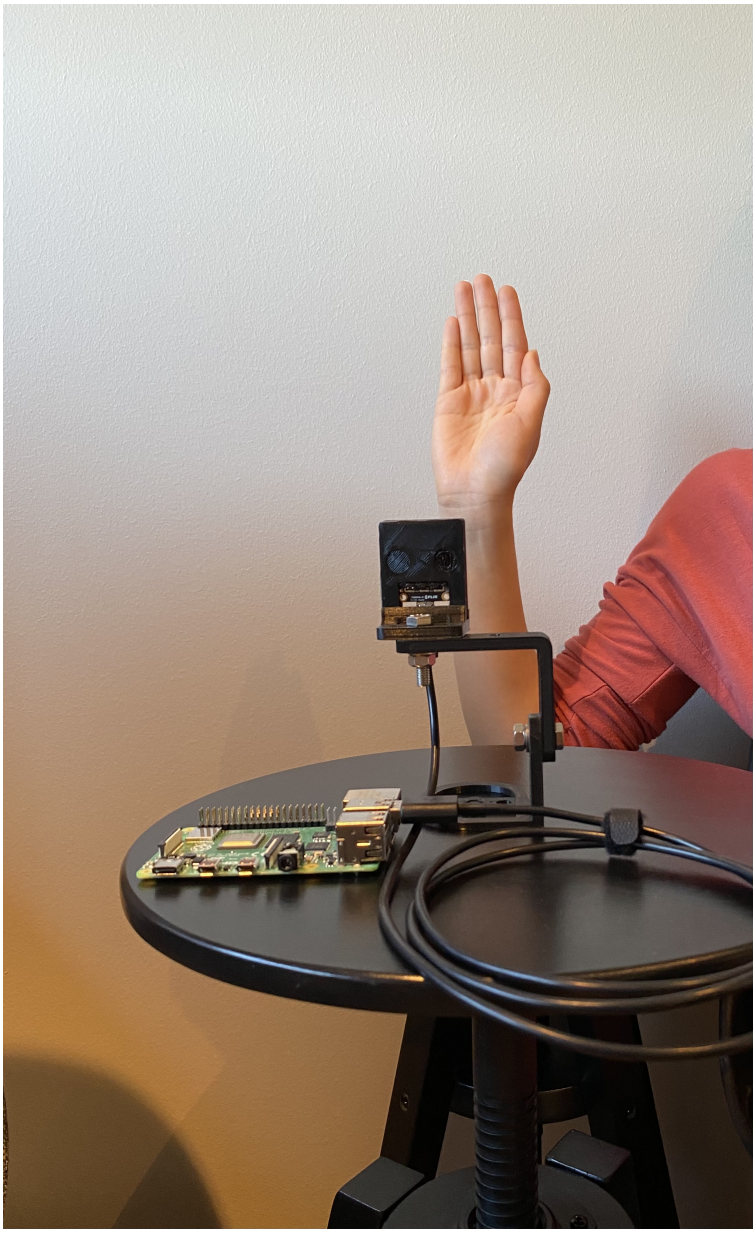


Figure 3.3: Image example of how the images were taken. By placing hand in front of camera with a plain background. The hand is also supported by another barstool to keep the hand as stable as possible.

For the custom Python code, it was used a pre made library. The library used is flirpy, a Python library, to control and interact with FLIR Lepton thermal images [59]. flirpy is a all in one software library even though we use capturing software functions. As the Python software is very helpful to convert raw data into arrays which can be further converted

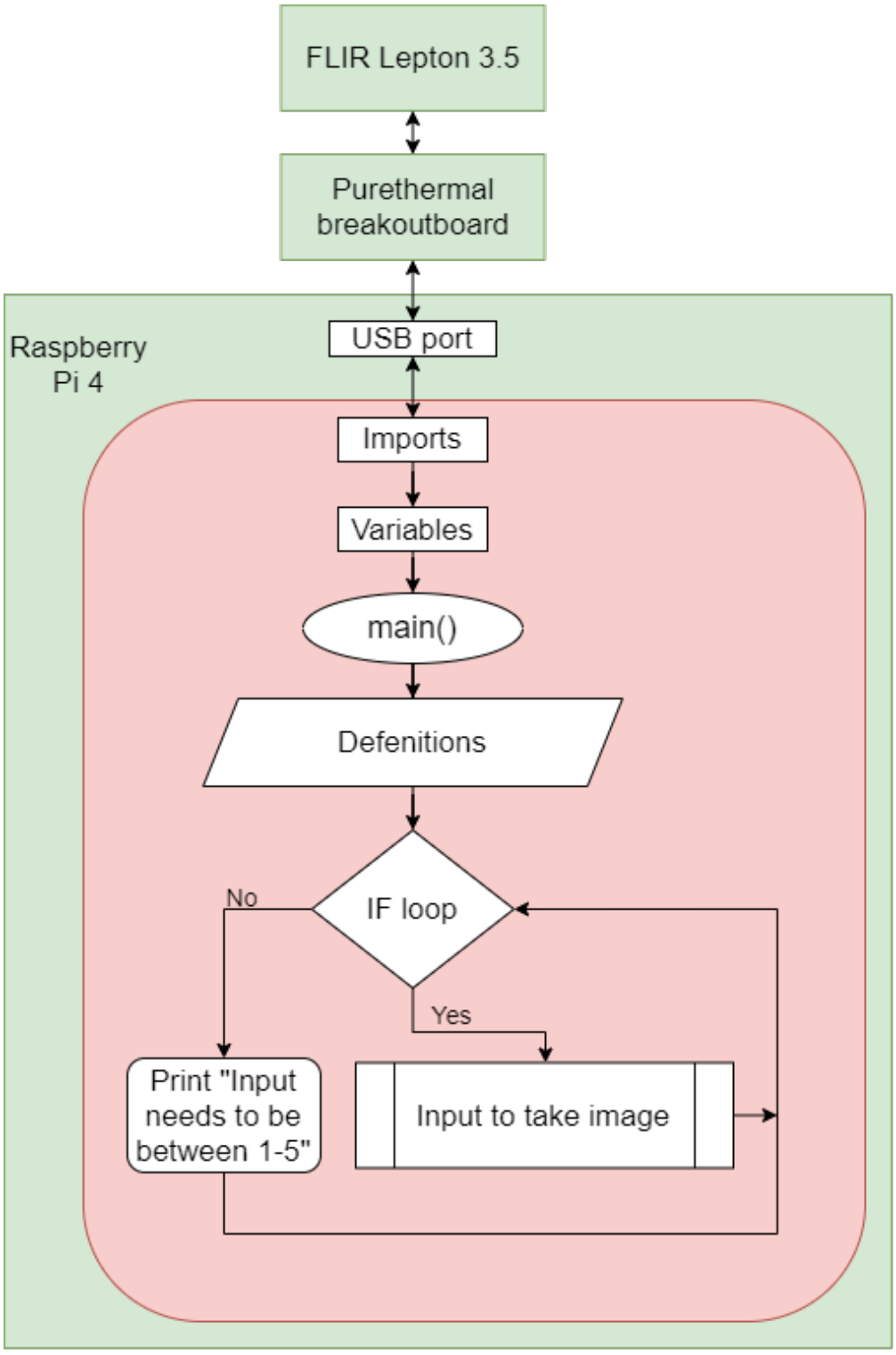


Figure 3.4: Flowchart showing psudo code for capturing the thermal images with Raspberry Pi. With an IF loop to check input, and then choose how many images to take.

into images.

When connected to the Raspberry Pi, the flowchart in Fig. 3.4 describes the python script used to capture images for the dataset. It is a simple program that makes use of the Lepton library for Python programming from flirpy [59]. This library enables Lepton 3.5 to capture images, which are then stored on the Raspberry Pi using the OpenCV (cv2) [42] and matplotlib [29] libraries. The main program is a loop that awaits input in the form of numbers ranging from one to five. When input is equal to two, it takes a new input. This second input is the number of rounds or images it will take before asking for

the first input again.

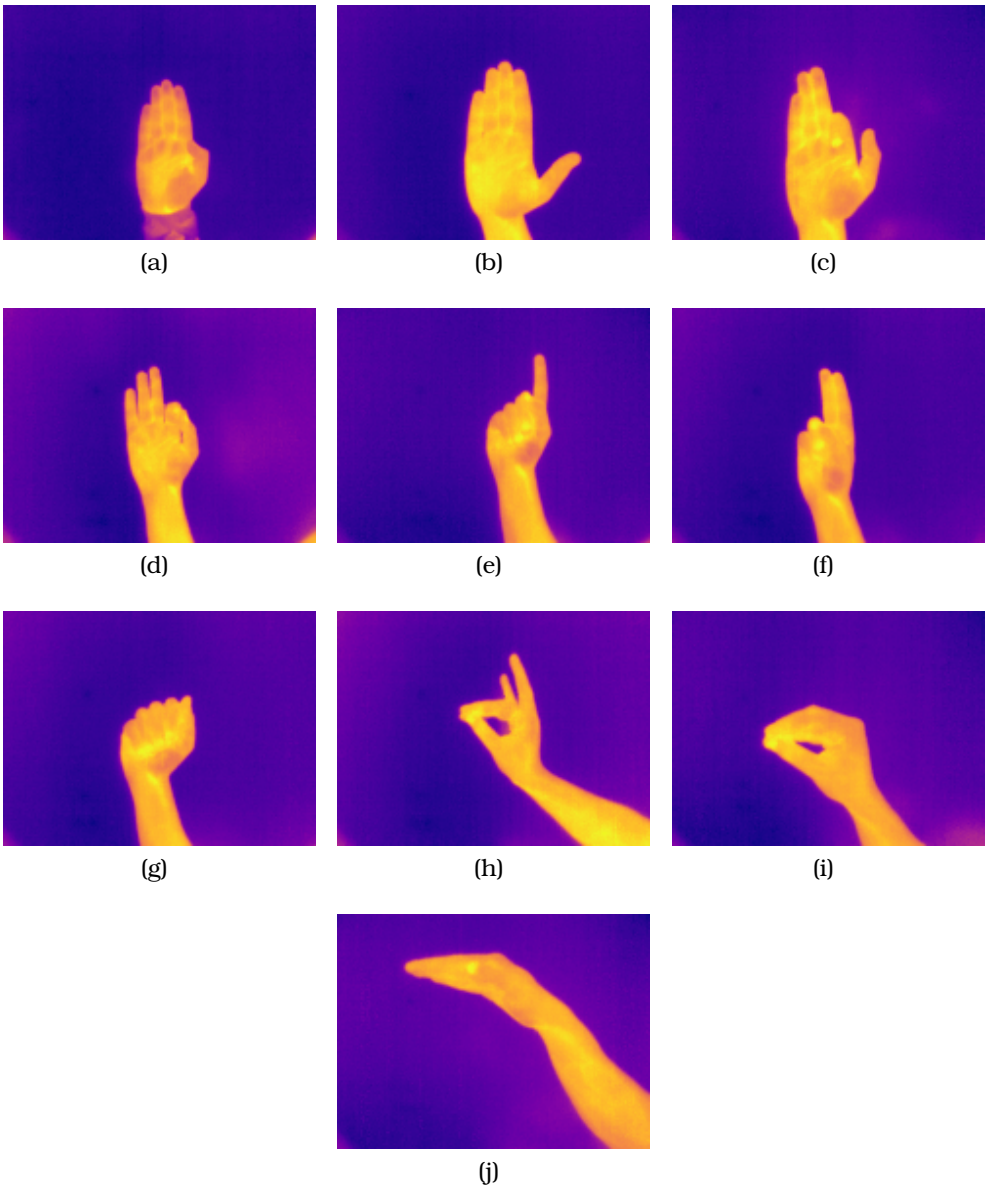


Figure 3.5: A complete set of fusion colored thermal images: (a) Fusion image a; (b) Fusion image b; (c) Fusion image c; (d) Fusion image d; (e) Fusion image e; (f) Fusion image f; (g) Fusion image g; (h) Fusion image h; (i) Fusion image i; and, (j) Fusion image j.

The dataset is created with uniform background. The uniform background will draw attention to the hand and emphasize the unique characteristics of each hand. People are naturally warmer than the inside room where the images were captured, though the quality of the images may vary depending on the circumstances. Fig. 3.7 shows characteristics of hand gestures when captured it as cold. Figures 3.7d, 3.7e, and 3.7f show the problem when the camera is reading the hand as cold. In fact, their hands/fingers are not cold. The blue fingers may come from several things, as the images were taken from January to March and the outside temperatures were lower than in the spring/-summer time. If people would have walked outside prior to taking images, their skin might have been colder than the reference wall behind. Some people might also have a lower body temperatures depending on location in the world. Although this should not have any consequences, a few °C will be shown clearly in thermal images. Personal features may also play its part as wet hands will more likely be colder or closer to the room

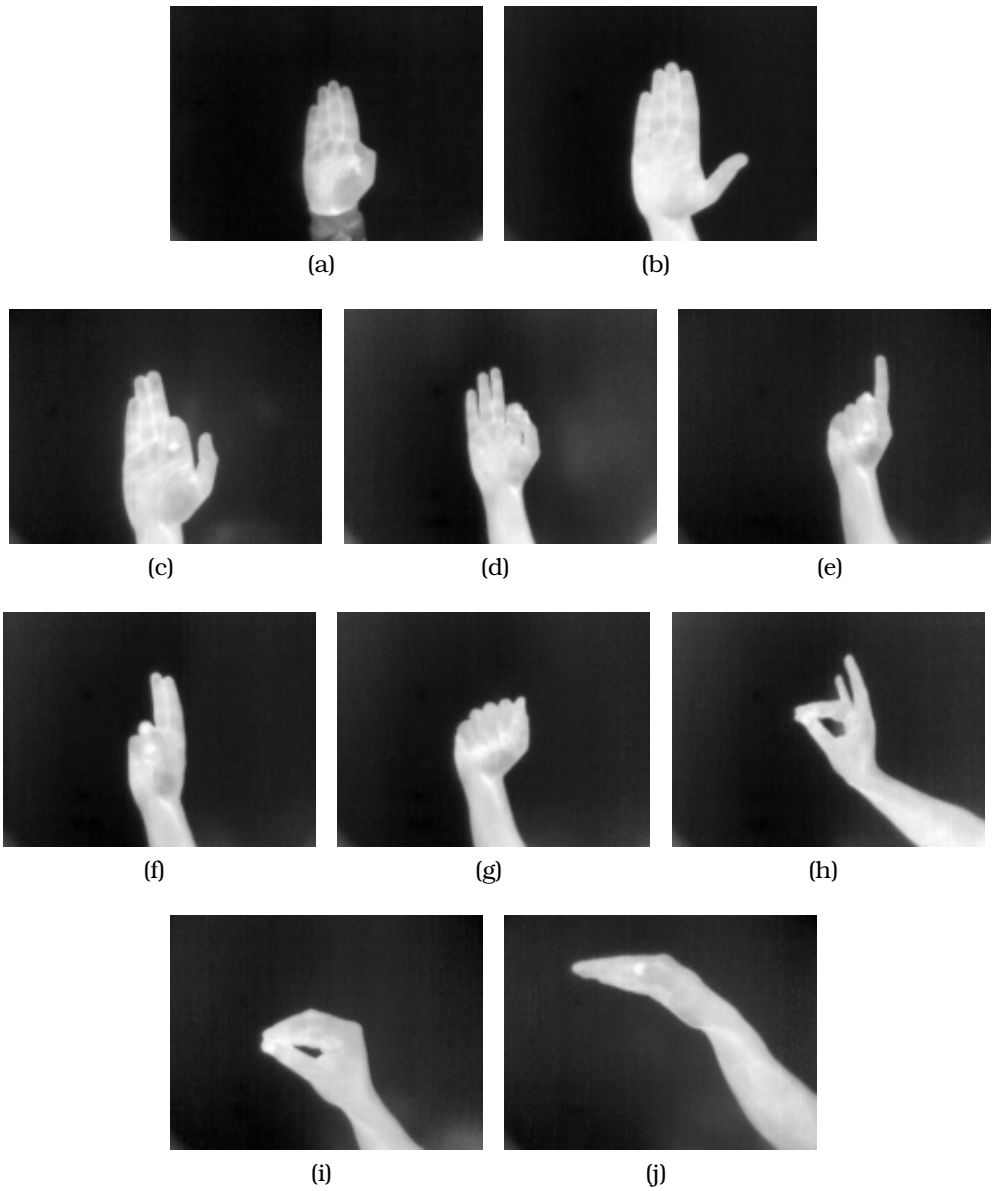
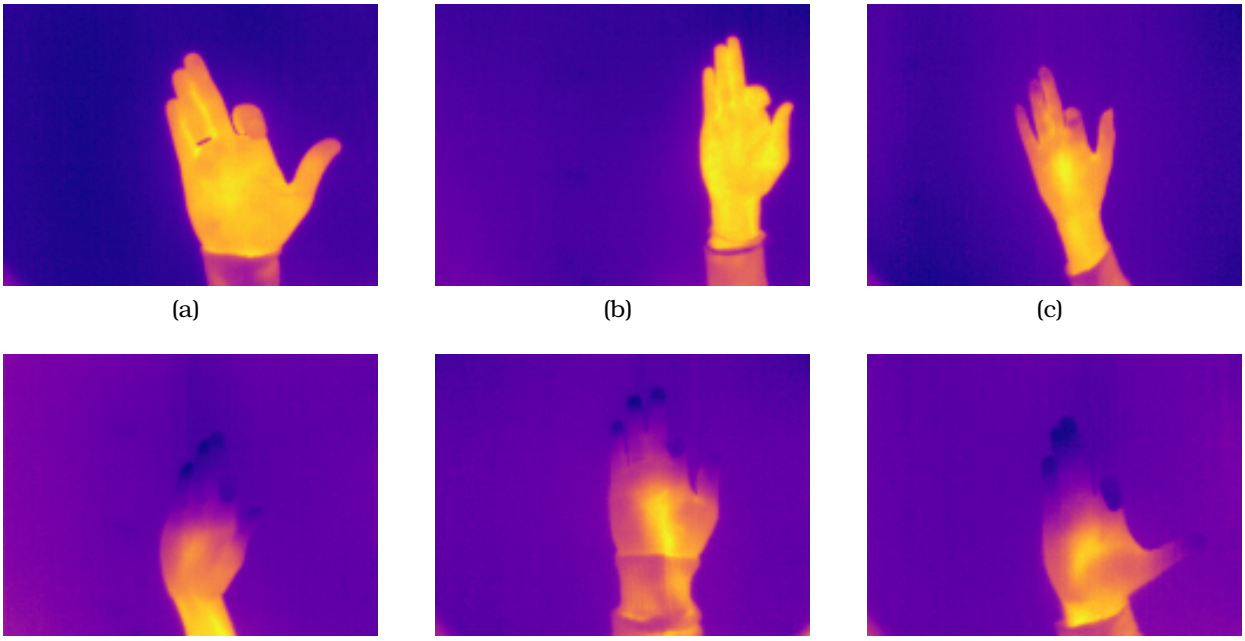


Figure 3.6: A complete set of grayscale thermal images: (a) Grayscale image a; (b) Grayscale image b; (c) Grayscale image c; (d) Grayscale image d; (e) Grayscale image e; (f) Grayscale image f; (g) Grayscale image g; (h) Grayscale image h; (i) Grayscale image i; and, (j) Grayscale image j.

temperature. Other factors may also have an influence but it will all create a variation in the dataset which makes it harder for computers to learn. That also makes it better for testing, to check the quality and performance of the algorithms proposed using such diverse dataset.

While taking the thermal images, people had to hold their arms to their sides directly in front of the camera in order to capture hands properly. Ideally, the people photographed had their arms to the side with a 90° angle upwards. The position in which people have been holding their arms is depicted in Fig. 3.3. Because the camera tripod is not very tall, it must be placed at a different height than the elbows. In some cases, the tripod has been placed on top of other objects or it has been raised by placing it on a variable-height object. Keeping their arms in such static positions may be too difficult for the person. As a result, a table or stool was placed for the people to support their elbows. It was sometimes necessary to change angles completely and turn their bodies the same direction as their hand was pointing with gestures h, i, and j. The Lepton 3.5's

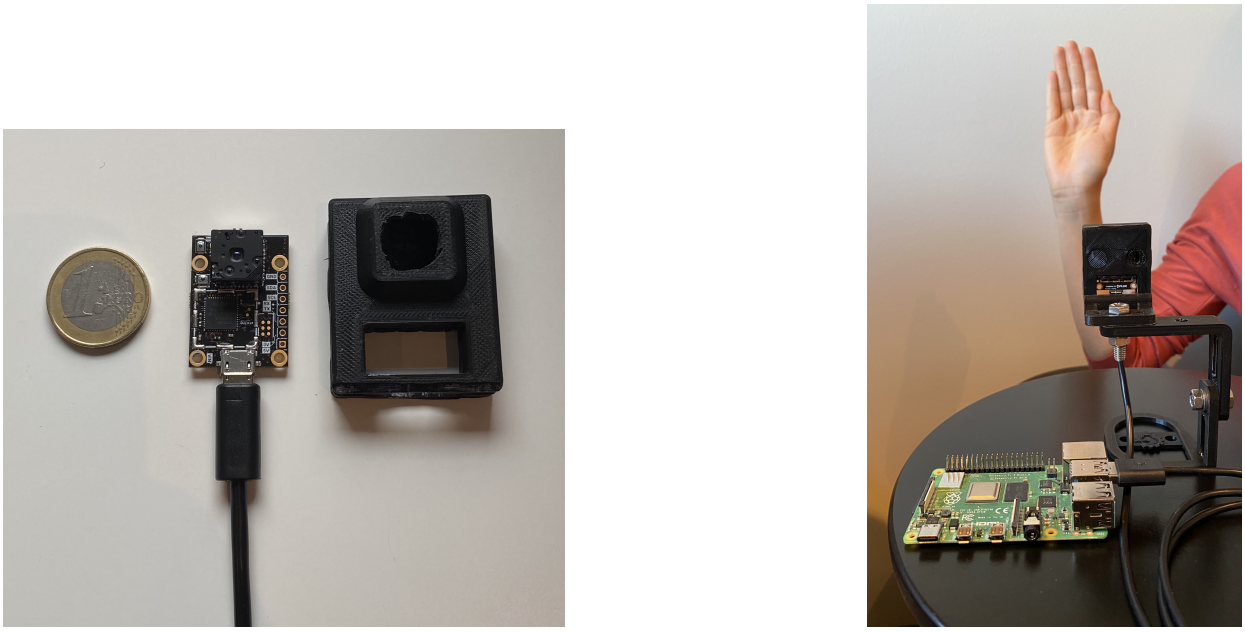


(a) (b) (c)

(d) (e) (f)

Figure 3.7: Example of difference between images, with position, gesture and temperature differences: (a) Good temperature difference, good hand position; (b) Good temperature difference, good hand gesture; (c) Good temperature difference, medium hand gesture; (d) Poor temperature difference, good hand gesture; (e) Poor temperature difference, medium hand gesture; (f) Poor temperature difference, poor hand gesture;

Horizontal FOV (HFOV) is 57° indicating that it was designed to capture more details in images rather than objects.
All images are taken from the same distance and angle from the camera. The distance maintained between the camera and the hand is 40 to 60 cm. It can capture images with a high level of detail at that distance. However, it is technically possible to capture images at greater distances as well. Fig. 3.7a shows an example of a high-detail image in which different skin temperatures are clearly visible and all fingers are visible. Images will begin to lose details as they are moved further away as shown in Fig. 3.7c. The camera tripod can be adjusted up and down as well as sideways to point the camera straight at the hands as can be seen in Fig. 3.8b. Because the tripod is so small, adjusting to a different setting is simple because the setup is mobile and portable. Making it easier to meet people close to them and not being confined to a single location.
The dataset is made up of 14 400 images that are divided into two color tones, which are then divided into 10 gestures of 24 images per person. Figures 3.5 and 3.6 show a set of ten gestures from a person. All images should look similar to these gestures with only the natural/personal differences for each person. When sorted by gesture, the dataset contains 720 images corresponds to each gesture in both fusion and grayscale. The size of the images vary from fusion to grayscale. Fusion colors will be of larger size with each image taking up 25kB with a total of around 18MB. Grayscale images are smaller, weighing in at 17kB per image and totaling 12.2MB isze for a single gesture. With a total of 30 people, the total size of the dataset is around 300MB of data resulting in some detailed thermal images suitable for image recognition. The algorithm will be more accurate with a larger dataset and more data but, it may slow down in terms of how long it takes to train. The thermal images of less size is advantageous because it will make integration into any machine learning algorithm and computing platform easier.
Because the images do not represent any numbers or letters, the gestures are thought to be similar. Here, the gestures use some of the same fingers but in different positions and placement as shown in the first four images of Figures 3.5 and 3.6. The palm of the hand



(a) (b)

Figure 3.8: Camera stand and camera stand on a tripod: (a) FLIR Lepton in the 3D printed stand; (b) Thermal camera stand on tripod to gain height when taking images;

is in the center for all these images but, the fingers are in various positions. Depending on the image's quality and temperature, a small change in position can be difficult to detect. As a result, having a large number of images reduces the likelihood of selecting the incorrect label for the image. The dataset is doubled when images are taken in two different colors and could also be increased further by changing the rotation of images if needed.

3.1.3 Complex background image capturing

In the complex background images are different from normal images in terms of background and hardware used.
As the Raspberry Pi 4 Model B was starting to run slower with many images due to higher folder size. It was easier to install and run a virtual machine such as VMware workstation. Thus, we consider a virtual machine installed with Ubuntu 18.04 LTS [11] which is the same operating system as in Raspberry Pi 4 Model B. Using a virtual machine on a full PC will also be more practical and it is computational powerful. Further, switching between PC and virtual machine is smooth and fast. When dealing with a PC it can also be more portable than a Raspberry Pi that requires a monitor. The stability of a PC is also helping, as the loading of images and interruptions are less frequent. A folder with many files will be heavier to load for any computer, which will make quality control of images much easier. For this dataset, the Lepton user app was also considered, but not chosen as it would be less configurable and need more time to do the same work.
Capturing the actual images was done using the same libraries, but with some modifications to the python code. After moving from Raspberry Pi to a PC, the path to folders need to be updated to make the capturing smoother. Sorting of images was made more automatic by choosing a gesture based folder in addition to the main folder which contains all images. These folders were named from a to j and they corresponds to the gesture order shown in Figures 3.5 and 3.6. Also, the way to check the number of images in a folder was coded. This in turn reduces the amount of time needed to open folders and manually count images in sets of 24. The Python code was written to return a modulus of 24, which tells how many images from modulo 24 that were taken.

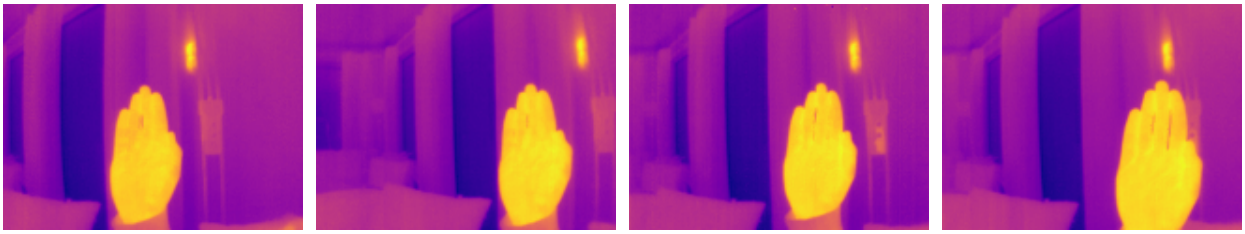
Figure 3.9: Image of the tripod that was used to capture complex background images.

The tripod to mount the camera on is also, a camera tripod shown in Fig. 3.9. This makes capturing images with different backgrounds easier, as the tripod is light and compact to move around. The possible height adjustments will also increase the possibility of making more differences in images. These differences are randomly captured in images, as the goal is to have many backgrounds with a varied style.

Moving the tripod is critical to capture backgrounds with different lighting and reflections. As this movement and variation will prevent overstimulating the machine learning model, where it will only be able to correctly read the training images. Here, the plain background needed a uniform background, to take good images, the complex background will have differences in both hand placement and background as seen in Fig. 3.10.

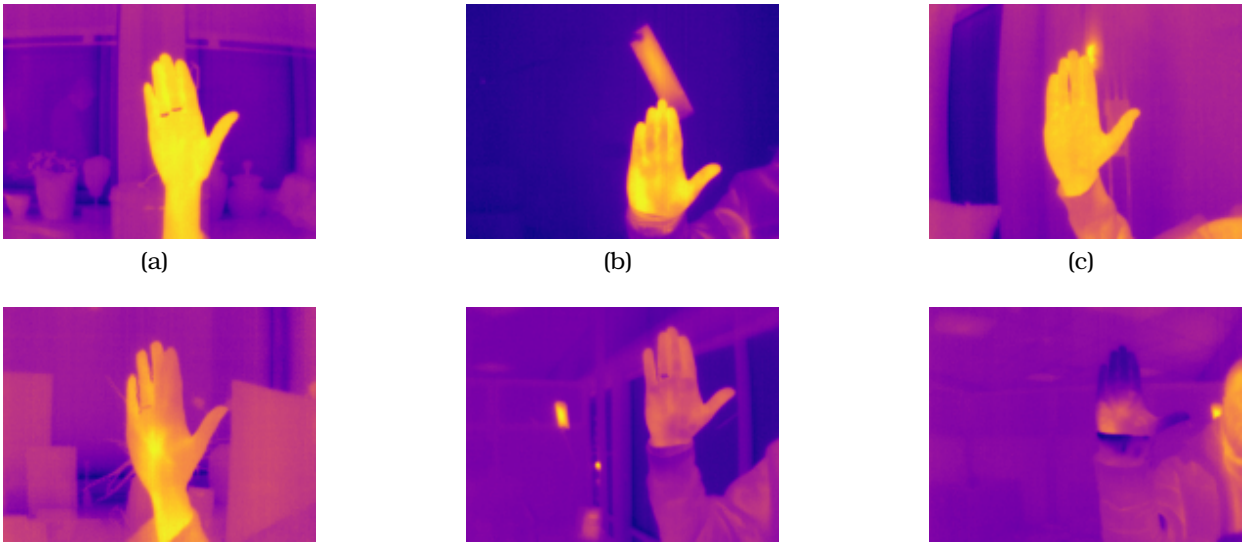
The movement of camera and tripod is vital when trying to capture various and complex backgrounds in a thermal image as the lens will be reacting to background temperature in regards of the hand in front. Thus, there will be some images where the background is more plain and some with dots/areas of color. The variation of these are also contributing to increase the difficulty to learn that requires a large database in order to capture all the different variations, and to learn about the important parts in the image. Placing light sources and heated things in front of the images will increases the difficulty as the machine learning model will not only have shapes of one color. The hands may be colder than the background, which will be reflected in the images with the opposite colors of the normal hands. Fig. 3.11 shows an example figure for the same. Especially the two on the bottom center in Fig. 3.11e and left in Fig. 3.11f will be hard to find the correct gesture.

This dataset contains much more details in the images where, it might be needed to perform object detection before doing image recognition. This will require more samples to make a working model. With more images, it might be possible to use image recognition only depending on the content of the second dataset. While some images like Figures 3.11b and 3.11f, it will be harder to detect. As the example dataset in Figures 3.12 and 3.13, the background is not as dominating and has no bright spots. This is however not consistent through the dataset, meaning it will be beneficial to perform object detection before doing image recognition to get a valid result.



(a) (b) (c) (d)

Figure 3.10: Example of hand movement within frame: (a) Hand position 1, Starting point; (b) Hand position 2, Camera directed to the left; (c) Hand position 3, Starting point; (d) Hand position 4, Hand is moved closer;



(a) (b) (c)

(d) (e) (f)

Figure 3.11: Example of difference between images, with position, gesture and temperature differences. Poor positioning is when other parts are hotter than the hand: (a) Good temperature difference, good hand position; (b) Good temperature difference, good hand position; (c) Good temperature difference, medium hand position; (d) Poor temperature difference, good hand position; (e) Poor temperature difference, medium hand position; (f) Poor temperature difference, poor hand position;

3.1.4 Thermal imaging in low lighting conditions

Images are captured in various lighting conditions to demonstrate the robustness of the FLIR Lepton 3.5 thermal camera. This section shows images of how the camera works in various lighting conditions, ranging from dim to completely dark. The Fig. 3.14 depicts three different light shades in the room. The images were taken in the same session, and the lighting was reduced from normal, as shown in Figures 3.3. The hand was kept in the same position to see any differences as shown in the Lepton image at the bottom right of Figures 3.14. During this test, the camera was setup as a web camera from a virtual machine and was live streaming the images.
It can be seen in Figures 3.14a that this scenario has less thermal leakage than others, which can occur when the camera is live streaming and constantly calculating the temperature within the frame. This could be due to a number of factors, such as a new temperature sensor calibration or a bias calculation of the background temperature when the light went off. The last image taken in Fig. 3.14c has much higher temperature difference to the right of the hand. This could be explained by the body moving more towards the hand, heating the area around or the hand heating up the area around. Another rea-

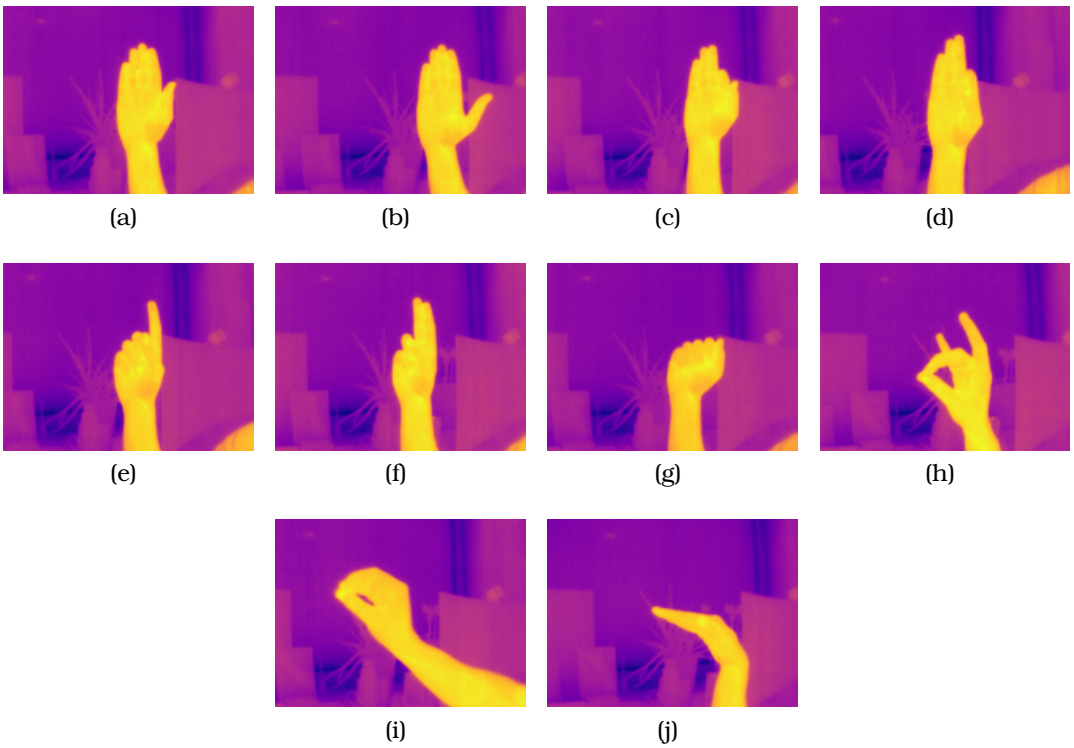


Figure 3.12: A complete set of fusion colored thermal images with complex background: (a) Fusion image a; (b) Fusion image b; (c) Fusion image c; (d) Fusion image d; (e) Fusion image e; (f) Fusion image f; (g) Fusion image g; (h) Fusion image h; (i) Fusion image i; and, (j) Fusion image j.

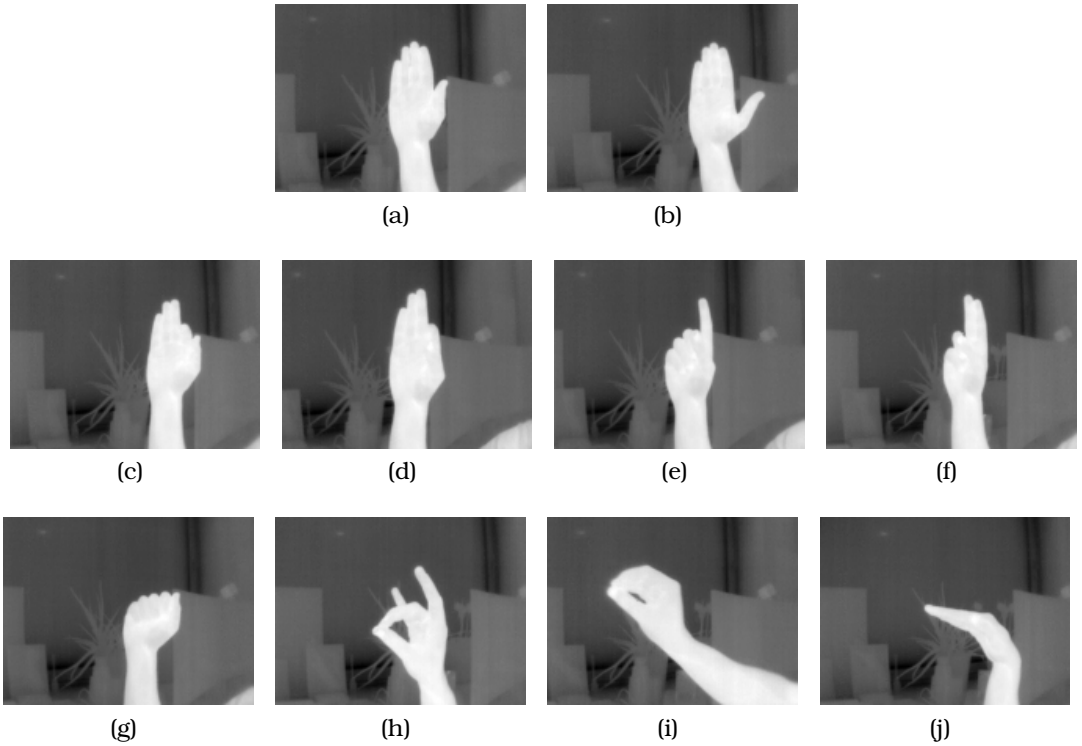
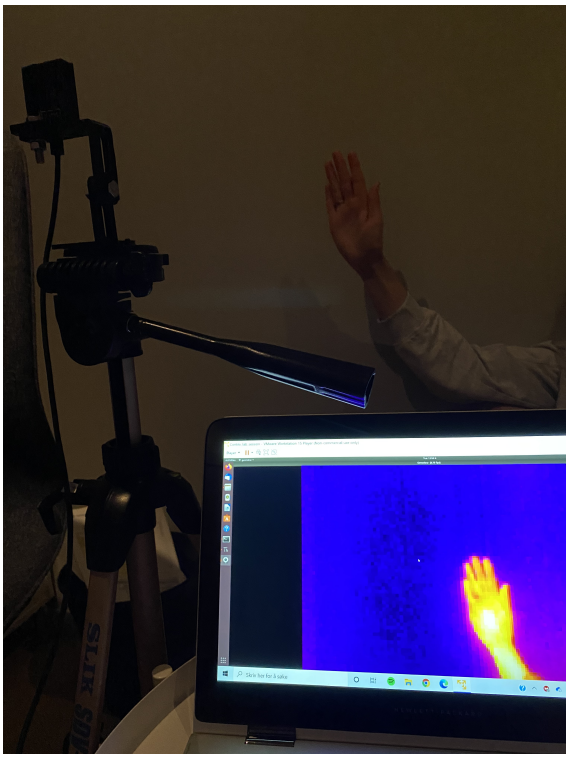
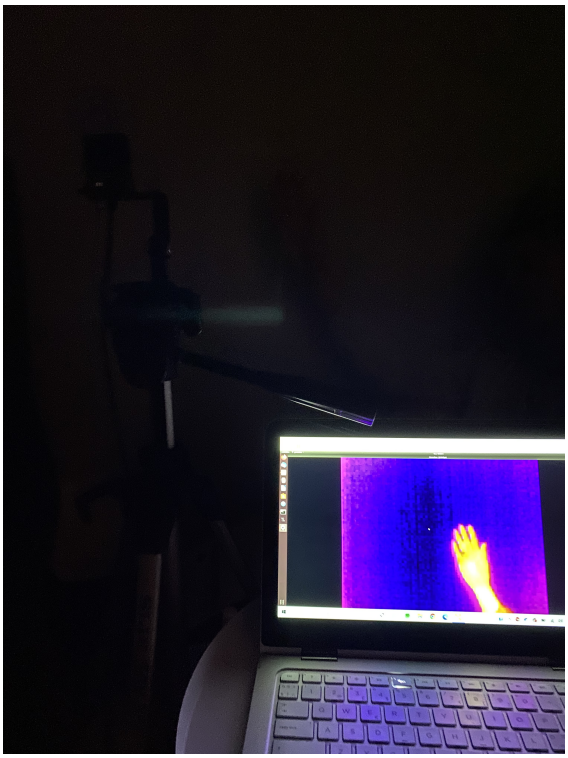


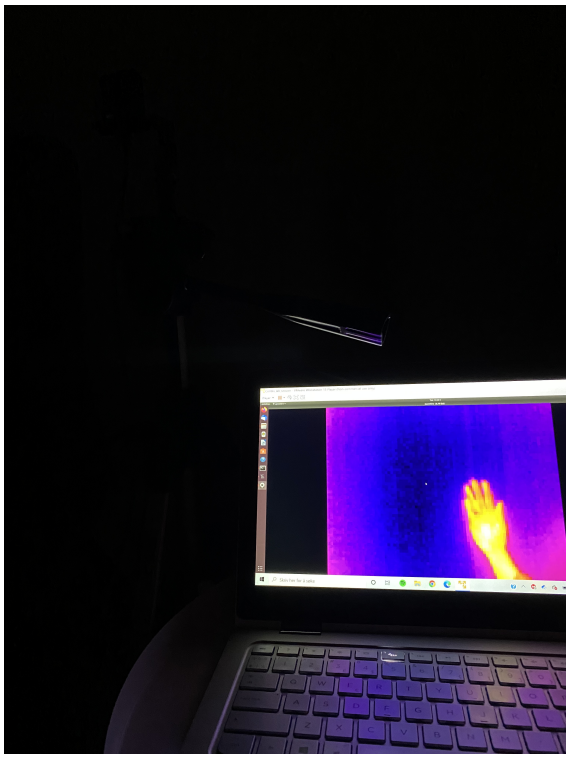
Figure 3.13: A complete set of grayscale thermal images with complex background: (a) Grayscale image a; (b) Grayscale image b; (c) Grayscale image c; (d) Grayscale image d; (e) Grayscale image e; (f) Grayscale image f; (g) Grayscale image g; (h) Grayscale image h; (i) Grayscale image i; and, (j) Grayscale image j.



(a)



(b)



(c)

Figure 3.14: Image of how the reflection is in a low light scenario: (a) Lower light than normal; (b) Almost completely dark; (c) Without light;

son for this temperature variation would be the presence of the people around the area. The darkest blue part is calculated to be the coldest and the temperature difference is more as the hand is placed at larger distances within the same area.

The fact that all images are captured using the 3D printed casing as shown in Fig. 3.15. It also demonstrates that the camera is unaffected by external or surrounding light.

Figure 3.15: FLIR Lepton 3.5 in casing.

Because all other sensors are covered by the casing in this image, only the lens will be able to detect any surrounding light. The casing will provide more stable conditions that will not be affected by small external and rapid variations in temperature, like wind gusts.

3.2 Convolutional Neural Network

We divide the available dataset into two parts such as Train dataset and Test dataset. The test dataset consists of 20% of the entire dataset. We further divide the training dataset into two parts namely Train dataset and Validation dataset. The division of the dataset is done in a manner such that equal number of samples are taken from each class so as to avoid an imbalance distribution. Table 3.1 shows the details of the three datasets.

Table 3.1: Number of samples per class in Train, Validation and Test Datasets.

Class	Train Dataset	Validation Dataset	Test Dataset
0	519	57	144
1	519	57	144
2	519	57	144
3	519	57	144
4	519	57	144
5	519	57	144
6	519	57	144
7	519	57	144
8	519	57	144
9	519	57	144

3.2.1 Convolution Layer

In this layer the kernel maps ('k') performs the convolution operation on the input feature map. These layers consist of some of the main parameters such as dimensions of the

kernel map ($H \times W \times D$). Here, 'H' is the height of kernel map, 'W' is the width of the kernel map, 'D' is the depth of the kernel map, and 'N' number of kernel maps. In this work $H \neq W$ i.e., a rectangular kernel map. We also use a stride of 1 for each convolution layer. The convolution operation is done as follows [61]

$$y_{i^{l+1},j^{l+1},n} = \sum_{i=0}^{H} \sum_{j=0}^{W} \sum_{d=0}^{D} k_{i,j,d,n} \times x^{l}{}_{i^{l+1}+i,j^{l+1}+j,d} \tag{3.1}$$

In (3.1), x^l is the output of the previous layer ($l-1$) which becomes the input feature map to the current layer (l). '$y_{i^{l+1},j^{l+1},n}$' is the intermediate output after performing the convolution operation. We repeat this for for all 'N' maps to obtain 'Y'. We then input this 'Y' to an non linear activation function, Residual Linear Unit (ReLU). The expression for ReLU function is obtained as [61].

$$f(x) = \max(0, x) \tag{3.2}$$

The ReLu function is an activation function that gives the gradient outputs for each inputs. The ReLu function is a variant much used in CNN, as it uses the equation (3.2). It gives the maximum value of the input or 0 depending on how the input is activated. Thus, the final output from the Convolution Layer is obtained as [61].

$$X^{l+1} = f(Y) \tag{3.3}$$

3.2.2 Dilated Convolution Layer

In this work, we use 4 convolution layers, 2 with dilation rate of 1 and 2 with dilation rate of 2. Each of the convolution layer is followed by a batch normalization operation to improve training speed and decrease overfitting [27]. We also use 1 Max Pooling layer. The entire model architecture is shown in Table 3.2 and Fig. 3.16. This Dilated Convolution will not decrease the input resolution as much as Pooling layers. The reason for some decrease are the fact that Zero Padding is not used, and the filter will not reach the outer edges of the image.

Table 3.2: Architecture details of the proposed CNN model. Conv1 and Conv2 are convolutioon layers with dilation rate 1 and 2 respectively.

Layer	Output Shape
Input	(None,120,160,1)
Conv1	(None, 118,159,16)
Conv1	(None, 116,158,32)
Max Pool	(None, 38,79,32)
Conv2	(None, 30, 73, 64)
Conv2	(None, 22, 67,128)
Max Pool	(None, 7, 33, 128)
Flatten	(None, 29568)
Dense	(None,10)

3.2.3 Max Pooling

In this work we use Max Pool Layer, whose operation is shown in the equation (3.4) [61]. The Max Pool Layer is decreasing the parameters between the first and second part of convolution layers. The big cut in parameters is a big step in cutting size, as the filter size is a 3×2 matrix. It will then reduce the amount of parameters by 3 in height and 2 times in width. This gives us the output from Max Pool layer one in table 3.2. This is

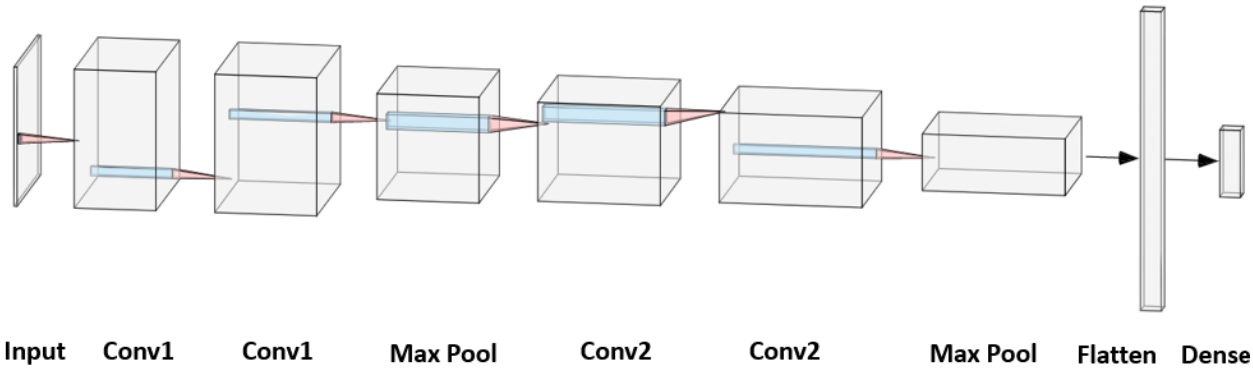


Figure 3.16: Proposed CNN model architecture. Conv1 and Conv2 corresponds to convolution layers with dilation rate of 1 and 2 respectively.

then repeated for a second time before input into the flatten layer. This Max Pooling will make the model look at larger areas of input as the resolution is decreased. This saves computation power and helps avoid overfitting in the model.

$$y_{i^{l+1},j^{l+1},n} = \max_{0 \leq i \leq H, 0 \leq j \leq W} x^{l}{}_{i^{l+1} \times H + i, j^{l+1} \times W + j, d} \tag{3.4}$$

3.2.4 Training

We train the proposed CNN model using 'Adam Optimizer' with a learning rate of 0.001 [34] and a batch size of 32. All the weights were initialized using 'Kaiming initializer' [25]. The model was trained using 10 fold cross validation i.e the entire train dataset was divided into 10 parts and for each iteration 9 parts were used for training and one part was used as validation dataset. This procedure was repeated for 10 times hence the name 10 fold cross validation. The training of each individual fold took about 10 minutes, which is affected by the size of each fold and the computational power available. After training the folds, they were all tested on their corresponding part of test dataset. Before they were combined and the folds were tested on the whole test dataset, as this minimizes the chances of the result to be a lucky strike and that the model has actually learned.

3.2.5 Benchmark Model

We compare the proposed CNN model with MobileNetV3 model as benchmark. MobileNetV3 is the '3^{rd}' version among the MobileNet family of architectures. MobileNet models are designed for optimised performance on mobile and edge computing devices. These models are specifically trained to have low latency while maintaining the accuracy of the model. There are two variants of MobileNetV3 that was proposed in [26], MobileNetV3 Small and MobileNetV3 Large. The difference between the two is the total number of parameters used to train the model. In this work we use both the variants of the model pre trained on ImageNet dataset. Before training the model we first have to slightly modify the benchmark models to adapt for the given task. We remove the classification layer (output layer) of the benchmark model and add a new classification layer with 10 classes for the given task. We also use a global average operator layer to flatten the output of the benchmark model. This layer is then to the new classification layer. We train the benchmark model using transfer learning technique called as 'Fine Tuning' method.

3.2.6 Fine Tuning

In this method few layers of the benchmark model along with the new classification layer are trained on the given task. This methodology of training can be very useful as compared to training from scratch. This is because the benchmark model's pre-trained weights act as a good parameter initializer and can optimize better on the given task.
We train both the benchmark models via 10 fold cross validation method. We also use an RMS Prop optimiser [58] with a batch size of 32 for training. All the models including the proposed model are trained on Google Colab i.e on Nvidia's T4 GPU with 12 GB GPU RAM, using Keras Deep Learning Library [15].

Chapter 4

Numerical Results

The project is mainly divided into two parts, one is to gather images for a dataset, while the second is to create a CNN model for hand recognition. Collecting a dataset of thermal images, was done with each participant taking 24 images of all 10 gesture and in total 240 for each person in both color and grayscale. For the model to be able to learn correctly, it was collected 30 people. Enough to make the model learn without overfitting.
The 10 fold cross validation results of all the models in Fig. 4.1. The average 10 fold validation accuracy for the proposed model, MobileNetV3 Large and MobileNetV3 Small is 98.42%, 99.42%, 99.86% respectively. The learning rate of the proposed CNN model is shown in the accuracy plot, where it shows the convergence after 50 epoch in Fig. 4.2.

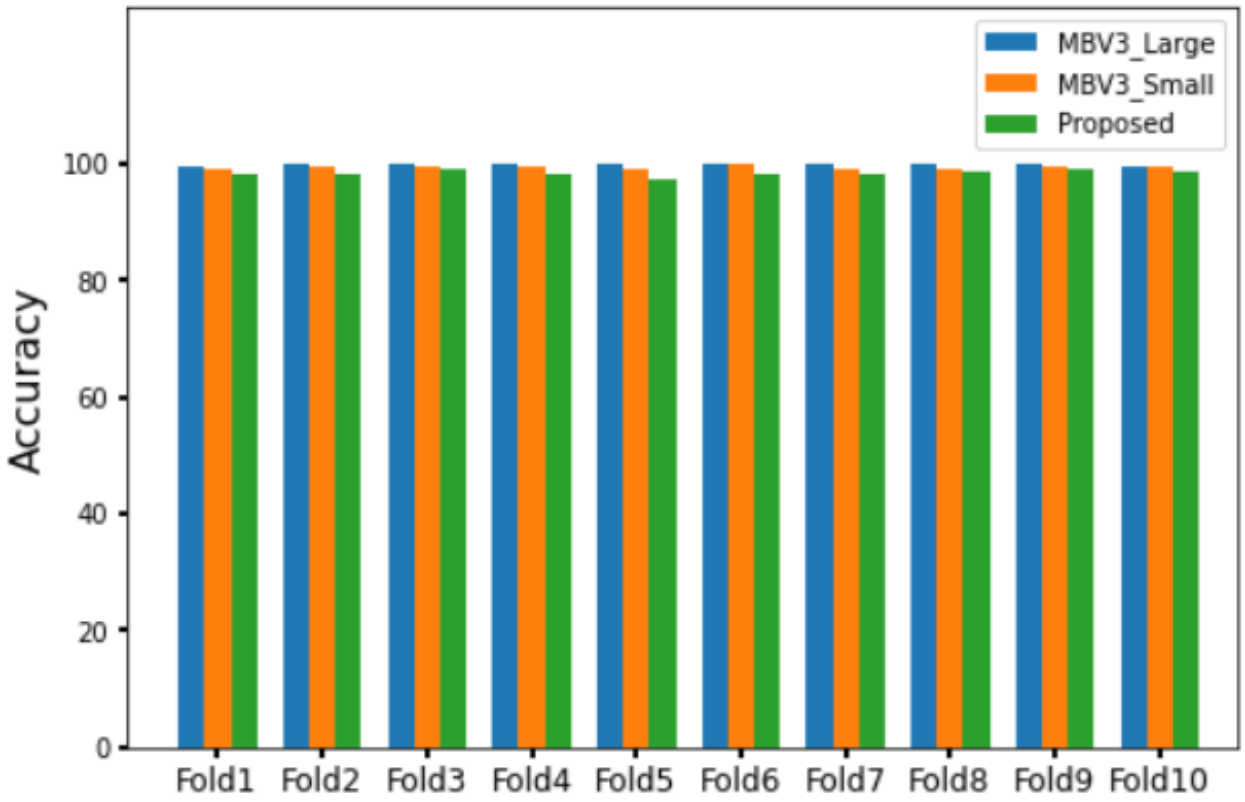


Figure 4.1: 10 fold cross validation accuracy of the proposed model and the benchmark models.

Next the models were compared based on their test accuracy. After 10 fold cross validation it is 10 models available for each model, as each fold will be its own model. We can get the test accuracy by either combining the results from all the 10 models [50] [7] or obtain a single model by training the model on the entire train dataset with optimal hyperparamter values obtained during 10 fold cross validation [44]. It was choosen to

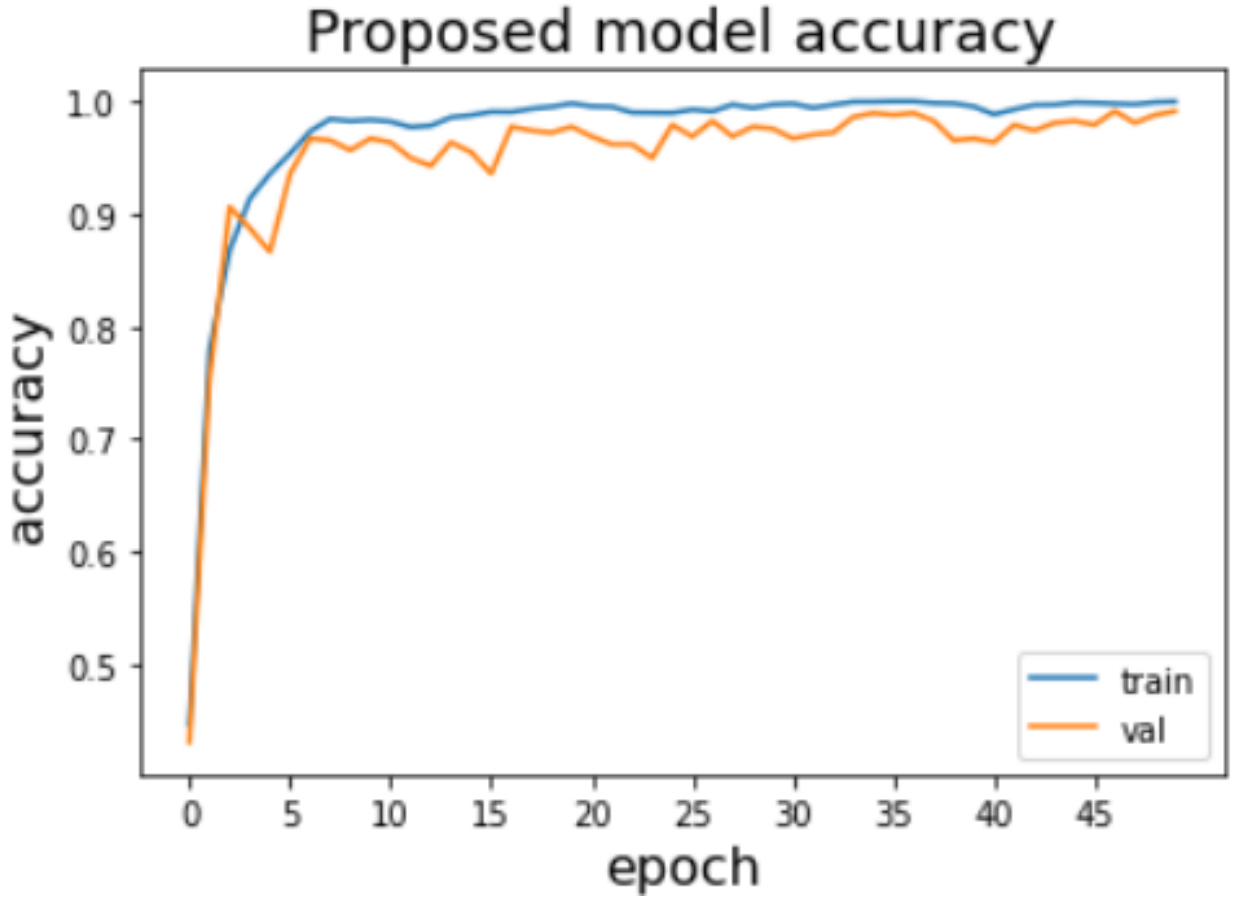


Figure 4.2: Training and Validation accuracy of the proposed CNN model for fold 10.

use the latter method for simplicity sake, by obtaining 3 models corresponding to the Proposed CNN model, MobileNetV3 Small and MobileNetV3 Large respectively. We then evaluate these models on the test dataset to get the test accuracy values. These values are summarised in Table 4.1.

Table 4.1: Test accuracy values of the Proposed CNN model, MobileNetV3 Small and MobileNetV3 Large models.

Model	Test Accuracy
MobileNetV3 Small	99.72%
MobileNetV3 Large	99.98%
Proposed CNN model	98.81%

After this we compare the proposed CNN model with the benchmark models in terms of model's size both TensorFlow and TensorFlow Lite (TFLite) versions and also compare them in terms of the number of parameters. These results are summarized in Table 4.2 and 4.3. As seen from Table 4.3, the Proposed CNN model's TFLite version is 3 times smaller than the MobileNetV3 Small model and 8 times smaller than the MobileNetV3 Large model.
We next see a plot for the various performance metrics such as 'Precision', 'Recall', 'F1 score' for the proposed CNN model [21]. The confusion matrix of the proposed CNN model is shown in the Fig. 4.3. The performance metric values are shown in the Table 4.4.
We lastly show the inference time of the TFLite version models deployed on Raspberry Pi 4 Model B and Nvidia Jetson AGX Xavier. These values are summarized in Table 4.5 and 4.6. Showing the average time after 100 iterations, of recognizing a gesture on the two platforms.
The timing tables 4.5 and 4.6, shows that the hardware is a big factor when looking at

Table 4.2: Test accuracy values of the Proposed CNN model, MobileNetV3 Small and MobileNetV3 Large models.

Model	Total Parameters	Trainable Parameters
MobileNetV3 Small	1,540,218	1,430,058
MobileNetV3 Large	4,239,242	3,725,818
Proposed CNN model	504,858	504,378

Table 4.3: Model's Size of Tensorflow (TF) and Tensorflow Lite version (TFLite) of the Proposed CNN model, MobileNetV3 Small and MobileNetV3 Large models. Model size is measured in MB.

Model	TF Model Size (MB)	TFLite Model Size (MB)
MobileNetV3 Small	12	6
MobileNetV3 Large	31	16
Proposed CNN model	6	2

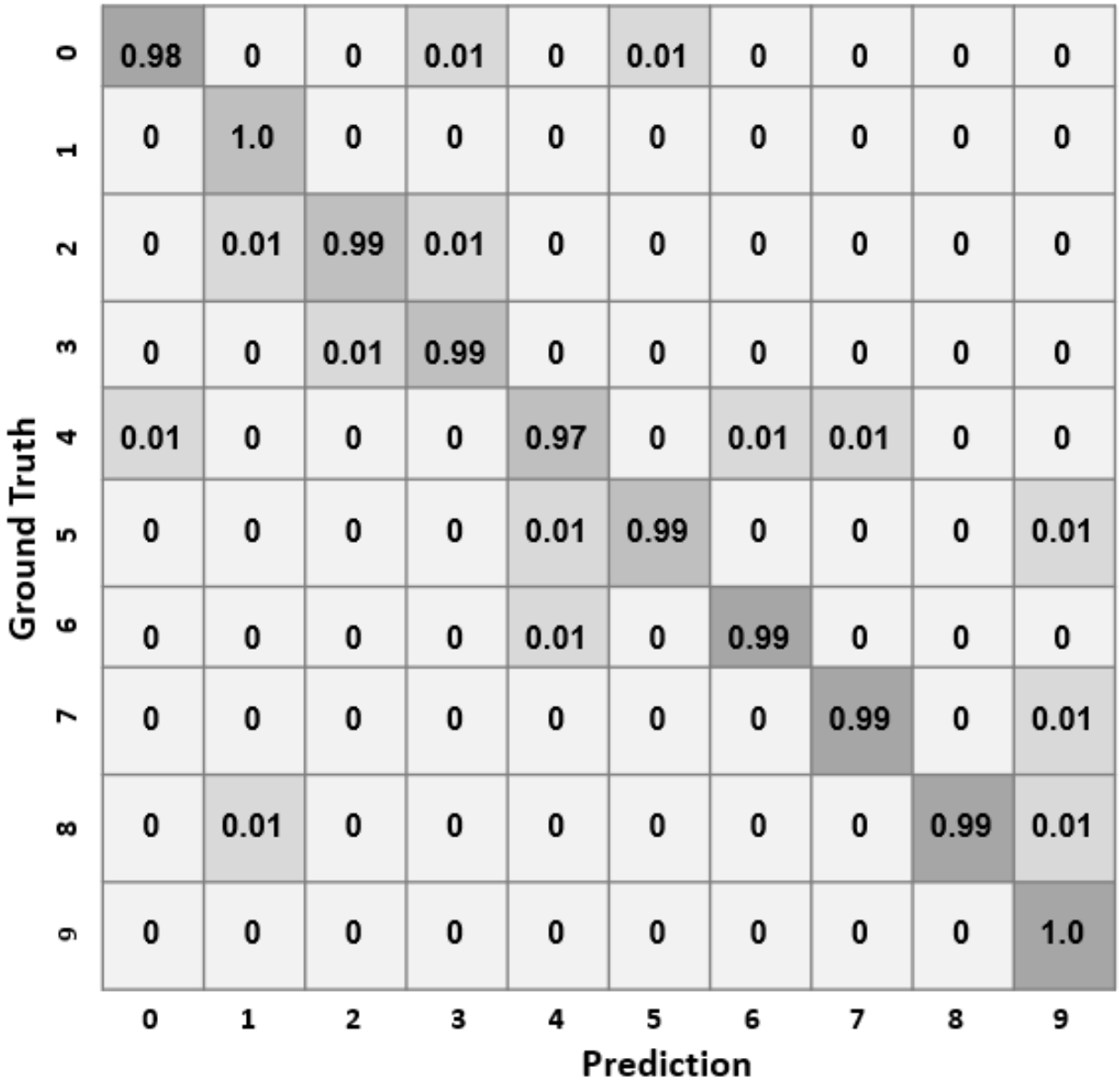


Figure 4.3: Confusion matrix of the proposed model.

Table 4.4: Precision, Recall and F1 Score values of the Proposed CNN model for each class.

Class	Precision	Recall	F1 Score
0	0.99	0.98	0.99
1	0.99	1.0	0.99
2	0.99	0.99	0.99
3	0.98	0.99	0.98
4	0.99	0.97	0.98
5	0.99	0.99	0.99
6	0.99	0.99	0.99
7	0.99	0.99	0.99
8	1.0	0.99	0.99
9	0.98	1.0	0.99

Table 4.5: Inference time of all the models on Raspberry Pi 4B. TFLite is the Tensorflow Lite version of the models.

TFLite model	Inference Timing
MobileNetV3 Small	0.033844s
MobileNetV3 Large	0.079605s
Proposed CNN Model	0.140968s

Table 4.6: Inference time of all the models on Jetson AGX Xavier. TFLite is the Tensorflow Lite version of the models.

TFLite model	Inference time
MobileNetV3 Small	0.013664s
MobileNetV3 Large	0.035300s
Proposed CNN model	0.075138s

the inference time. As the more general Raspberry Pi 4B will have less power, compared to the deep learning machine Jetson AGX Xavier platform.

Chapter 5

Discussions

5.1 Project plan

As part of the master thesis and project, I created an article based on the same work that I have presented in this thesis. At first, the plan was to create two articles which were based on the two datasets. After careful consideration of the time it takes to develop and gather data the second article was put on ice. Therefore will the article, using the thermal images with a plain background be completed. As the dataset with a complex or noisy background is not completed, the article and models could be completed at a later stage.

From the start of planning the project for a thesis, it was proposed some different tasks. Among the tasks it was to create an image recognition and a collision avoidance system, using thermal cameras and mmWave radar. The applications were both aimed at image recognition using mmWave radar and thermal cameras. The first application was aimed towards recognition of different walls in order to recognize different materials of walls. This helps in identifying the wall type such as brick, wood, cement, or other types. This will be applicable for both outdoor and indoor walls. Then, it should be combined with the mmWave radar that collects range data. Combining the data from both, we can identify the wall type and its distance. For the second application, the goal was to create a classification system. This system was supposed to have two parts, both including mmWave radar and thermal camera. First part is using images in an outdoor environment then, moving it forward to a moving vehicle while still performing the same classification. Which creates a new challenge of using live video and not still images, while also do classification at live speed.

If the mmWave radar were added as part of the thesis, it would require a lot more work. As that would be an even higher level of knowledge and expertise to solve. The mmWave radar is a completely new concept for myself and would claim a substantial amount of time to understand and function. The thermal images were always in our focus and the mmWave radar was dropped from the project after realising the difficulty of implementing it. As further work or a higher degree proposal it would be more fitting.

All different challenges and time consuming tasks, made it obvious that completing one article was possible and not two. For the second article, the dataset is not fully completed and it needs a machine learning algorithm for classification. The self created machine learning and dataset creation will however be sufficient work for a complete thesis, which also includes the creation of the second dataset with complex backgrounds. For this second dataset with complex background, the model would need to be created with the possibility to detect objects. If not, the images might have some background noise the model believes to be a hand gesture making classifying fail.

5.2 Process

The working process has been altered a bit as the changes in plan has been moved towards the finished end product. The initial focus was to develop thermal camera application for gathering images for the datset. It was cruical that it worked and was able to grab images from FLIR's Lepton thermal camera. Programming this was not the hardest part, with the option to use an SDK which works out of the box [54]. The Python code in the SDK is similar but was not used directly in the actual image capturing.
As the task of developing the range profile was dropped, the focus on the mmWave Radar was dropped. Further development of range profile will be a big step up in complexity. Such that this thesis will not be covering this. The fact that the mmWave radars were not used, shifted more focus to the image capturing and machine learning parts.
In the image capturing phase, there was another problem as we needed lots of people. The global pandemic was preventing large groups of people meeting to collect images. In the public room, as on campus Grimstad, it was advised with online lessons to prevent people gathering in big groups. The other was at home visitors, as it was advised to reduce the numbers of social contacts. All these rules and guidelines introduced in the beginning of the year, made the gathering of images a lot more time consuming.
When the final formulation of the project was made, it was time to start developing the machine learning algorithm. As the machine learning model design is dependent on what parameters it will be provided. With everything in place, the shaping of layers and parameters were decided. There was no real issues until the deployment phase, where the installation of TensorFlow for certain hardware was proving difficult. When this problem was solved, the product was ready to be tested and finalized. As for time, it was now too late starting on another model with the complex background. Therefore the dataset collection of images were also stopped some time before. Making this the logical step to further develop this project.

5.3 End product

The end product is a combination of the proposed model and the dataset capturing camera. The camera module was supported by a plastic tripod, which was 3D printed and then again supported by another camera tripod. As the first tripod is of fixed height, it was only used to keep the camera module still. The second tripod was adjustable, as most cameras require some movement. It could be adjusted from its legs or by lengthen its neck, as seen in Fig. 3.9. The camera module was then attached to a computer, for both power supply and data transfer.
The proposed model is deviating some from the original idea, as it does not include the mmWave radar and is classifying hand gestures instead of different walls. While the thermal camera and images were kept. With the current solution, the end product is a CNN model with hand gesture recognition using thermal images. It is a very small size model, which funciotn very good for edge computing on computers with low memory capacity. this makes it more fitted to be used with the likes of Raspberry Pi and other low powered computers. The system is a robust and accurate system with an accuracy of 98.81%, which is a good result looking at the parameters and model size. The development of a small size model allows for some more flexibility, as further development could be to add features or speed, depending on use area.

5.4 Implementation

First task was to create a dataset to process using a machine learning algorithm. The datasets were collected at various locations, both for plain background and complex back-

ground. Initially, we created first dataset with plain background to understand the possibility of working of the model.
The dataset and proposed CNN model were developed independently on separate computers. Then, we merged them together after both parts were complete, when it was ready to train on the given dataset. The dataset was divided into the separate gestures and labeled each from “a” to “j” to the numbers 0 to 9. As the CNN model is trained using supervised training, it requires labeled data in order to validate and correct the weights. As mentioned earlier in Chapter 4, the dataset was divided into 10 folds containing the colored images from gestures“a” to “j” as in Fig. 3.5. Here, the colored image data is enough to get the desired performance. Adding more grayscale images would not increase accuracy further, as the colors are not treated differently.
The weights used to train the model are randomly initialized using the Kaiming He which is a probability distribution function. As you want most weights to be within a certain range. It was trained through 50 epochs, as this is a level of epochs in which the model should have improved if possible. If the accuracy of the validation set still is not increasing after 50 epochs, the model will need some adjustments.
When the proposed model was complete, it was tested against other CNN models such as MobilNet in both small and large models. As these models are already trained, they were only optimized with our dataset for the models to correctly recognize thermal hand gestures. The MobileNet models were slightly modified, by removing the dense layer with the original output by another dense layer trained on our dataset. This fine-tuning of an already existing model makes the accuracy even better, with weights being adapted to a similar dataset.
To create the model, it was used Keras Deep learning library and TensorFlow 2, which has functions to do the hard calculations automatic. This makes the creation of the model more streamlined, by using functions, not reinventing the same layers. The functionality of Keras and TensorFlow is very wide and is used for doing most of the mathematics. From calculating layers to number of parameters, making our work easier. This makes creation of a deep CNN model easier, faster, and more performance focused. Keras utilizes Python, which has many features and possible add-ons for even further development, which is ideal for developers to create custom networks.
For any practical use, in a small memory computer, it is very helpful for the model to run on TFLite. Which is a way for TensorFlow to decrease the size of the model, while it still keeps the accuracy. This way the proposed model has its size decreased from 6MB to 2MB, but still with the same accuracy. This was also done to the MobileNet models. This enables the usage of low power edge computers such as Raspberry Pi to run advanced CNN models with limited memory.

5.5 Challenges

The main challenge that was encountered with the process, hardware, software, or other issues during the project is mentioned here.
As a lasting problem with the FLIR Lepton camera module, it had some timing issues. These issues made it disconnect for a brief moment, stopped capturing at random intervals. This error could be in the library, breakout board or within the module communication, as the problem only appeared when being used with a custom Python program. We realized that the problem is not with the camera module after checking it with Leptop user application in Windows. Thus, we updated the python module and fixed the problem.
Originally the model size was 9MB, which was too large for a low memory edge computing device. The challenge was then to reduce the size while maintaining the accuracy. For this it would need to be tested a lot and made some sacrifices to one side of the performance. As the smaller sized model would decrease the accuracy, but with a good

starting accuracy it was possible to sacrifice 1% to reduce the model size. With it, it is very accurate with a reduced size.
In the deployment phase there were some troubles with TensorFlow versions as the TensorFlow version should be same for both training and testing. Thus, the edge computing devices should also have TensorFlow 2.4 which was used for training. The Raspberry Pi module needs to run a 64-bit operating system to run on the TensorFlow v2.4, 32-bit operating system works till TensorFlow 2.2. In a small computer as the Raspberry Pi, installation of these big libraries increases the usage of the processor which heats it up and could slow it down due to overheating. Other times the process of installing would just be really slow which even takes hours for installation of big libraries.
To test the proposed CNN model in more than one computer, it was desired to test it on a Sitara AM572X from Texas Instruments (TI). The problem with this machine was to install another operating system (OS). TI supports a range of OS for the Sitara machine, which are Linux based, but they do not provide any solution on how to install custom software. Thus, it was very difficult to install desired software or use in a test of the proposed model and benchmark models. It was not possible to install a general Linux machine like Ubuntu 18.04 LTS, as the machine would not boot this OS.

Chapter 6

Conclusion

A complete end-to-end system with a robust hand recognition model has been presented in this thesis. The system is designed to be highly portable, and a thermal dataset is created. The dataset includes 30 people and 14,400 thermal images of hand gestures, with 7200 in fusion color and 7200 in grayscale. The images were then classified into ten different categories. It was tested with three different machine learning models in this work. The size of the proposed CNN model is 6MB with just above 500000 parameters, and a TFLite verions of 2MB in size. Compared to the MobileNet small model with a size of 12MB and TFLite version of 6MB, the MobileNet large model is 31MB and 16MB in the TFLite version. The model also achieves an accuracy of 98.81%, compared to benchmark models with accuracy of 99.72% and 99.98% for small and large models, respectively. The inference time of the proposed model is 0.075138s on Nvidias Jetson AGX, compared to the benchmark models which results in 0.013664s for small and 0.0353s for large model. When tested on Raspberry Pi 4B, the inference time of the proposed model was 0.140968s compared to small and large benchmark models that result in 0.033844s and 0.079605s, respectively. Because of reliable thermal imaging, the proposed hand gesture recognition is robust and unaffected by external light sources. In addition, a dataset has also been created with complex background.

6.1 Further work

The inference time for the proposed model can be reduced on any edge computing device by improving the model. It can be observed from the MobileNetV3 models that it is possible to decrease the inference time, without being significantly increased in model size.
Adding more features is also a possible way to develop further without any complexity in the system. It requires a dataset with good quality and enough samples, depending on the features and image resolution. Similar features as presented and with same image resolution, would require about the same amount of data.
Dataset 2 is collected in the presence of background noise or other objects. Having more number of samples in this dataset may increase the accuracy of any classification model. This can be achieved by collecting the data from 10 to 20 more people in a similar fashion. We will collect the data from more people and propose new CNN model that fits with this dataset as part of future work. Another possible work lies in the direction of processing the live stream data. This can be achieved by designing a CNN model with less inference time. Moving it towards the faster MobileNet V3 models, which are bigger but also faster. This model may also require some sort of object detection, depending on the amount of background noise and objects present, in order to detect and recognize the correct part of the image.

Bibliography

[1] Zhwan Ahmed and Jamal Hussein. “An Interactive and Predictive Pre-diagnostic Model for Healthcare based on Data Provenance.” In: *UHD Journal of Science and Technology* 3 (Oct. 2019), p. 59. DOI: 10.21928/uhdjst.v3n2y2019.pp59-73.

[2] Pramod Kumar et al. “Hand posture and face recognition using a fuzzy-rough approach.” In: *International Journal of Humanoid Robotics* 07.03 (2010), pp. 331–356. DOI: https://doi.org/10.1142/S0219843610002180.

[3] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. “Understanding of a convolutional neural network.” In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[4] Bilel Ben Atitallah et al. “Simultaneous Pressure Sensors Monitoring System for Hand Gestures Recognition.” In: *2020 IEEE SENSORS*. 2020, pp. 1–4. DOI: 10.1109/SENSORS47125.2020.9278884.

[5] Bergsala AB. *Wii: spillkonsollen som endret verdens syn pÃ¥ tv-spill*. URL: https://www.nintendo.no/support/146-om-wii. (accessed: 18.05.2021).

[6] Daniel S. Breland et al. “Deep Learning-Based Sign Language Digits Recognition From Thermal Images With Edge Computing System.” In: *IEEE Sensors Journal* 21.9 (2021), pp. 10445–10453. DOI: 10.1109/JSEN.2021.3061608.

[7] Lauran Brewster et al. “Development and application of a machine learning algorithm for classifcation of elasmobranch behaviour from accelerometry data.” In: *Marine Biology* 165 (Mar. 2018). DOI: 10.1007/s00227-018-3318-y.

[8] Ultimaker BV. *Plugins*. URL: https://marketplace.ultimaker.com/app/cura/plugins. (accessed: 25.01.2021).

[9] Ultimaker BV. *Ultimaker Cura*. URL: https://ultimaker.com/software/ultimaker-cura. (accessed: 26.01.2021).

[10] Ultimaker BV. *Ultimaker S5 Powerful, reliable, versatile 3D printing*. URL: https://ultimaker.com/3d-printers/ultimaker-s5. (accessed: 25.01.2021).

[11] Canonical Ltd. *Alternative downloads*. URL: https://ubuntu.com/download/alternative-downloads. (accessed: 12.05.2021).

[12] Canonical Ltd. *Download Ubuntu Desktop*. URL: https://ubuntu.com/download/desktop. (accessed: 12.05.2021).

[13] Canonical Ltd. *List of releases*. URL: https://wiki.ubuntu.com/Releases. (accessed: 12.05.2021).

[14] Canonical Ltd. *Ubuntu Desktop for Developers*. URL: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. (accessed: 12.05.2021).

[15] François Chollet et al. “Keras: The python deep learning library.” In: *ascl* (2018), ascl–1806.

[16] Somayeh Ezadi, Tofigh Allahviranloo, and Salar Mohammadi. "Two new methods for ranking of Z-numbers based on sigmoid function and sign method." In: *International Journal of Intelligent Systems* 33.7 (2018), pp. 1476–1487. DOI: https://doi.org/10.1002/int.21987. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/int.21987. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/int.21987.

[17] Francois Chollet, et al. *Activation layers*. URL: https://keras.io/api/layers/activation_layers/. (accessed: 02.06.2021).

[18] Francois Chollet, et al. *Dense layer*. URL: https://keras.io/api/layers/core_layers/dense/. (accessed: 02.06.2021).

[19] Francois Chollet, et al. *Flatten layer*. URL: https://keras.io/api/layers/reshaping_layers/flatten/. (accessed: 02.06.2021).

[20] Francois Chollet, et al. *Layer activation functions*. URL: https://keras.io/api/layers/activations/. (accessed: 02.06.2021).

[21] K. M. Ghori et al. "Performance Analysis of Different Types of Machine Learning Classifiers for Non-Technical Loss Detection." In: *IEEE Access* 8 (2020), pp. 16033–16048. DOI: 10.1109/ACCESS.2019.2962510.

[22] GroupGets. *Adjustable Thermal Camera Stand*. URL: https://www.thingiverse.com/thing:1277474. (accessed: 18.01.2021).

[23] GroupGets. *PureThermal 2 - FLIR Lepton Smart I/O Module*. URL: https://groupgets.com/manufacturers/getlab/products/purethermal-2-flir-lepton-smart-i-o-module. (accessed: 03.03.2021).

[24] Hari Prabhat Gupta et al. "A Continuous Hand Gestures Recognition Technique for Human-Machine Interaction Using Accelerometer and Gyroscope Sensors." In: *IEEE Sensors Journal* 16.16 (2016), pp. 6425–6432. DOI: 10.1109/JSEN.2016.2581023.

[25] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[26] Andrew Howard et al. "Searching for MobileNetV3." In: *CoRR* abs/1905.02244 (2019). arXiv: 1905.02244. URL: http://arxiv.org/abs/1905.02244.

[27] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *arXiv preprint arXiv:1502.03167* (2015).

[28] IonEwe. *Lepton USB Housing*. URL: https://www.thingiverse.com/thing:3275190. (accessed: 18.01.2021).

[29] John Hunter, et al. *Matplotlib: Visualization with Python*. URL: https://matplotlib.org/stable/index.html. (accessed: 03.06.2021).

[30] Juan Pablo Wachs et al. "Vision-based hand-gesture applications." In: *International Journal of Humanoid Robotics* 54.02 (2011), pp. 60–71. DOI: http://hdl.handle.net/10945/52012.

[31] Hyun Kang, Chang Lee, and Keechul Jung. "Recognition-based gesture spotting in video games." In: *Pattern Recognition Letters* 25 (Nov. 2004), pp. 1701–1714. DOI: 10.1016/j.patrec.2004.06.016.

[32] Kurt Kiefer. *PureThermal 1/2/Mini Reference Firmware*. URL: https://github.com/groupgets/purethermal1-firmware. (accessed: 03.03.2021).

[33] Seo Yul Kim et al. "A Hand Gesture Recognition Sensor Using Reflected Impulses." In: *IEEE Sensors Journal* 17.10 (2017), pp. 2975–2976. DOI: 10.1109/JSEN.2017.2679220.

[34] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *International Conference on Learning Representations* (Dec. 2014).

[35] Yuhong Li, Xiaofan Zhang, and Deming Chen. “CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes.” In: (Feb. 2018).

[36] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[37] NVIDIA Corporation. *JETSON AGX XAVIER Deploy AI-Powered Autonomous Machines at Scale*. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/. (accessed: 19.05.2021).

[38] NVIDIA Corporation. *Jetson AGX Xavier Developer Kit*. URL: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit. (accessed: 12.05.2021).

[39] NVIDIA Corporation. *Jetson Download Center*. URL: https://developer.nvidia.com/embedded/downloads#?search=Jetpack. (accessed: 12.05.2021).

[40] NVIDIA Corporation. *JETSON FAQ*. URL: https://developer.nvidia.com/embedded/faq#xavier-faq. (accessed: 12.05.2021).

[41] Timothy Oâ€™Shea and Jakob Hoydis. “An Introduction to Deep Learning for the Physical Layer.” In: *IEEE Transactions on Cognitive Communications and Networking* 3.4 (2017), pp. 563–575. DOI: 10.1109/TCCN.2017.2758370.

[42] OpenCV team. *Releases*. URL: https://opencv.org/releases/. (accessed: 03.06.2021).

[43] Pramod Kumar et al. *NUS dataset*. URL: https://www.ece.nus.edu.sg/stfpage/elepv/NUS-HandSet/. (accessed: 25.02.2021).

[44] Sebastian Raschka. *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*. 2020. arXiv: 1811.12808 [cs.LG].

[45] RASPBERRY PI FOUNDATION. *Operating system images*. URL: https://www.raspberrypi.org/software/operating-systems/#raspberry-pi-os-32-bit. (accessed: 12.05.2021).

[46] RASPBERRY PI FOUNDATION. *Raspberry Pi 4*. URL: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. (accessed: 11.05.2021).

[47] RASPBERRY PI FOUNDATION. *Raspberry Pi 4 Tech Specs*. URL: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/. (accessed: 12.05.2021).

[48] Linga Reddy Cenkeramaddi et al. “A Survey on Sensors for Autonomous Systems.” In: *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. 2020, pp. 1182–1187. DOI: 10.1109/ICIEA48937.2020.9248282.

[49] Rikke Gade, Thomas B. Moeslund. “Thermal Cameras and Applications.” In: *Machine Vision & Applications* 25.01 (2014), pp. 246–262. DOI: https://10.1007/s00138-013-0570-5.

[50] Dymitr Ruta and Bogdan Gabrys. “Classifier selection for majority voting.” In: *Information Fusion* 6.1 (2005). Diversity in Multiple Classifier Systems, pp. 63–81. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2004.04.008. URL: https://www.sciencedirect.com/science/article/pii/S1566253504000417.

[51] Shuchen Du. *Understanding Deep Self-attention Mechanism in Convolution Neural Networks*. URL: https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-networks-e8f9c01cb251. (accessed: 27.05.2021).

[52] Sruthy Skaria et al. “Deep-Learning for Hand-Gesture Recognition with Simultaneous Thermal and Radar Sensors.” In: *2020 IEEE SENSORS*. 2020, pp. 1–4. DOI: 10.1109/SENSORS47125.2020.9278683.

[53] STMicorelectronics. *STM32 ST-LINK utility*. URL: https://www.st.com/en/development-tools/stsw-link004.html#overview. (accessed: 03.03.2021).

[54] FLIR System. *Software & SDK*. URL: https://lepton.flir.com/software-sdk/. (accessed: 04.02.2021).

[55] FLIR Systems. *FLIR LEPTON 3 & 3.5*. URL: https://lepton.flir.com/wp-content/uploads/2015/06/Lepton3_3.5-Data-Sheet.pdf. (accessed: 04.02.2021).

[56] FLIR Systems. *FLIR LEPTON Engineering Datasheet*. URL: https://www.flir.com/globalassets/imported-assets/document/flir-lepton-engineering-datasheet.pdf. (accessed: 05.02.2021).

[57] FLIR Systems. *LEPTON*. URL: https://www.flir.com/products/lepton/?model=3.5%20Lepton. (accessed: 05.02.2021).

[58] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.

[59] Josh Veitch-Michaelis. *LJMUAstroecology/flirpy: v.0.1.0 Zenodo*. Version v.0.1.0. May 2020. DOI: 10.5281/zenodo.3866331. URL: https://doi.org/10.5281/zenodo.3866331.

[60] Dinesh Vishwakarma and Rajiv Kapoor. "An Efficient Interpretation of Hand Gestures to Control Smart Interactive Television." In: *International Journal of Computational Vision and Robotics* 7.4 (2017), pp. 454–471. DOI: 10.1504/IJCVR.2017.10005393.

[61] J. Wu. "Introduction to Convolutional Neural Networks." In: 2017.

[62] Fisher Yu and Vladlen Koltun. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2016. arXiv: 1511.07122 [cs.CV].

[63] Xiaoliang Zhang et al. "Cooperative Sensing and Wearable Computing for Sequential Hand Gesture Recognition." In: *IEEE Sensors Journal* 19.14 (2019), pp. 5775–5783. DOI: 10.1109/JSEN.2019.2904595.

Appendix A

Python Code: Plain Background Image Capture

```
#!/bin/env python
#--------Imports--------#
import cv2
import sys
import time
import os.path
import numpy as np
import matplotlib.pyplot as plt
from flirpy.camera.lepton import Lepton
from PIL import Image


#--------Arrays--------#
image = []


#--------Definitions--------#

def print_image_info(): # Flirpy

    print(image)                # Prints the pixel values in an array
    print(camera.frame_count)
    print(camera.ffc_temp_k)
    print (camera.fpa_temp_k)

#-----------------------------------------------------------------#

def capture_image_grayscale(image_number): # Example from: https://github.com/groupgets/pylepton

    camera= Lepton()
    image = camera.grab()

    #cv2.normalize(image, image, 0, 65535, cv2.NORM_MINMAX) # Extend/increase contrast
    #np.right_shift(image, 8, image) # Fit image into 8 bits
    #cv2.imwrite("grayscale_output.jpg", np.uint8(image)) # Save image
    arr = image
                                                            # Make an array for image pixels
    plt.imsave(fname = '/home/ubuntu/FLIR_Lepton_docs/Lepton_imaging_git/plain_background_images/NUS/Grayscale/gray_image_{}.png'.format(image_number), arr = image ,cmap = 'gray') # Save image using pyplot

    camera.close()

#-----------------------------------------------------------------#
```

```
42
43 def capture_image_colorscale(image_number): # Matplotlib
44
45     camera = Lepton()
46     image = camera.grab()
47
48     arr = image
                                                    # Make an array for
    image pixels
49     plt.imsave(fname = '/home/ubuntu/FLIR_Lepton_docs/Lepton_imaging_git/
    plain_background_images/NUS/Colorscale/plasma_image_{}.png'.format(image_number
    ), arr = image ,cmap = 'plasma') # Save image using pyplot
50
51     camera.close()
52
53 #-----------------------------------------------------------------#
54
55 def capture_images(image_number_grey, image_number_color):
56
57     camera= Lepton()
58     image = camera.grab()
59     arr = image # Make an array for image pixels
60
61     plt.imsave(fname = '/home/ubuntu/FLIR_Lepton_docs/Lepton_imaging_git/
    plain_background_images/NUS/Grayscale/gray_image_{}.png'.format(
    image_number_grey), arr = image ,cmap = 'gray')       # Save image grayscale
62     plt.imsave(fname = '/home/ubuntu/FLIR_Lepton_docs/Lepton_imaging_git/
    plain_background_images/NUS/Colorscale/plasma_image_{}.png'.format(
    image_number_color), arr = image ,cmap = 'plasma') # Save image colorscale
63     del arr
64     camera.close()
65
66 #-----------------------------------------------------------------#
67
68 def check_folder_grayscale():
69
70     directory_path = os.path.dirname('/Grayscale/')
                # Fnds the directory for the path
71     path = os.getenv('HOME') + '/FLIR_Lepton_docs/Lepton_imaging_git/
    plain_background_images/NUS/Grayscale/'              # Finds the path to place
    files
72     number_files = len([f for f in os.listdir(path)if os.path.isfile(os.path.join(
    path, f))]) # Calculates the number of files in folder
73     print("Number of images in folder {}: {}".format(directory_path, number_files)
    )
74     print(" ")
75     return number_files + 1
76
77 #-----------------------------------------------------------------#
78
79 def check_folder_colorscale():
80
81     directory_path = os.path.dirname('/Colorscale/')
                # Fnds the directory for the path
82     path = os.getenv('HOME') + '/FLIR_Lepton_docs/Lepton_imaging_git/
    plain_background_images/NUS/Colorscale/'              # Finds the path to place
     files
83     number_files = len([f for f in os.listdir(path)if os.path.isfile(os.path.join(
    path, f))])  # Calculates the number of files in folder
84     print("Number of images in folder {}: {}".format(directory_path, number_files)
    )
85     print(" ")
86     return number_files + 1
87
88 #-----------------------------------------------------------------#
89
90 #--------main--------#
```

```
91 def main():
92
93 #--------Variables--------#
94     image_number = 0
95     #grayscale_folder = 1
96     #colorscale_folder = 1
97     number_files = 1
98
99     #for x in range (1):              # For loop to read through the folders at
       start
100    colorscale_folder = check_folder_colorscale()
101    grayscale_folder = check_folder_grayscale()
102
103    while 1:
104
105        print("Press 1 for 1 image")
106        print("Press 2 for x images")
107        print("Press 3 for colorscale image, only for example")
108        print("Press 4 for grayscale image, only for example")
109        print("Press 5 to see files in directories")
110        print("Press 9 to exit")
111
112        num_input = int(input()) # Takes the input into a var.
113
114        if num_input >= 1 and num_input <= 9:
115            if num_input == 1:
116                capture_images(grayscale_folder, colorscale_folder)    # Captures
       both type of images
117                grayscale_folder = check_folder_grayscale()   # Checks how many
       images of grayscale images the folder contains
118                colorscale_folder = check_folder_colorscale() # Checks how many
       images of colorscale images the folder contains
119            elif num_input == 2:
120                print("Enter the amount of images to take")
121                rounds = int(input())
122                for x in range (rounds):
123                    capture_images(grayscale_folder, colorscale_folder)
124                    grayscale_folder = check_folder_grayscale()
125                    colorscale_folder = check_folder_colorscale()
126                    #time.sleep (0.2)
127            elif num_input == 3:
128                print("Enter the amount of images to take")
129                rounds = int(input())
130                for x in range (rounds):
131                    capture_image_colorscale(colorscale_folder)
132                    colorscale_folder = check_folder_colorscale()
133            elif num_input == 4:
134                print("Enter the amount of images to take")
135                rounds = int(input())
136                for x in range (rounds):
137                    capture_image_grayscale(grayscale_folder)
138                    grayscale_folder = check_folder_grayscale()
139            elif num_input == 5:
140                print("Folder content: ")
141                check_folder_grayscale()
142                check_folder_colorscale()
143            else:
144                #camera.close()
145                break;
146        else:
147            print("Input needs to be selected as stated")
148
149
150
151
152 #--------Start main--------#
```

```
153 if __name__ == "__main__": main()
```

Listing A.1: Plain background image capture code

Appendix B

Python Code: Complex Background Image Capture

```
#!/bin/env python
#--------Imports--------#
import cv2
import sys
import time
import os.path
import numpy as np
import matplotlib.pyplot as plt
from flirpy.camera.lepton import Lepton
from PIL import Image


#--------Arrays--------#
image = []


#--------Definitions--------#

def print_image_info(): # Flirpy

    print(image)                # Prints the pixel values in an array
    print(camera.frame_count)
    print(camera.ffc_temp_k)
    print (camera.fpa_temp_k)

#-----------------------------------------------------------------#

def capture_image_grayscale(gesture_folder, image_number_gray): # Example from: https://github.com/groupgets/pylepton

    camera= Lepton()
    image = camera.grab()

    #cv2.normalize(image, image, 0, 65535, cv2.NORM_MINMAX) # Extend/increase contrast
    #np.right_shift(image, 8, image) # Fit image into 8 bits
    #cv2.imwrite("/home/pi/Documents/Lepton_imaging_git/complex_background_images/NUS/grayscale/grayscale_{}png".format(image_number_gray), np.uint8(image)) # Save image
    #cv2.imwrite("/home/pi/Documents/Lepton_imaging_git/complex_background_images/{}/grayscale/grayscale_{}png".format(gesture_folder, image_number_gray), np.uint8(image)) # Save image

    arr = image
                                                    # Make an array for image pixels
    plt.imsave(fname = '/home/pi/Documents/Lepton_imaging_git/
```

```
    complex_background_images/NUS/grayscale/gray_image_{}.png'.format(
    image_number_color), arr = image ,cmap = 'gray')       # Save image grayscale
40      plt.imsave(fname = '/home/pi/Documents/Lepton_imaging_git/
    complex_background_images/{}/grayscale/gray_image_{}.png'.format(gesture_folder
    , image_number_color), arr = image ,cmap = 'gray') # Save image grayscale
41      del arr
42      camera.close()
43
44  #-----------------------------------------------------------------#
45
46  def capture_image_colorscale(gesture_folder, image_number_color): # Matplotlib
47
48      camera = Lepton()
49      image = camera.grab()
50
51      arr = image
                                                          # Make an array for
    image pixels
52      plt.imsave(fname = '/home/pi/Documents/Lepton_imaging_git/
    complex_background_images/NUS/colorscale/plasma_image_{}.png'.format(
    image_number_color), arr = image ,cmap = 'plasma')       # Save image grayscale
53      plt.imsave(fname = '/home/pi/Documents/Lepton_imaging_git/
    complex_background_images/{}/colorscale/plasma_image_{}.png'.format(
    gesture_folder, image_number_color), arr = image ,cmap = 'plasma') # Save image
     colorscale
54      del arr
55      camera.close()
56
57  #-----------------------------------------------------------------#
58
59  def capture_images(gesture_folder ,image_number_gray, image_number_color):
60
61      camera= Lepton()
62      image = camera.grab()
63      arr = image # Make an array for image pixels
64
65      plt.imsave(fname = '/home/pi/Documents/Lepton_imaging_git/
    complex_background_images/NUS/grayscale/gray_image_{}.png'.format(
    image_number_gray), arr = image ,cmap = 'gray')       # Save image grayscale
66      plt.imsave(fname = '/home/pi/Documents/Lepton_imaging_git/
    complex_background_images/{}/grayscale/gray_image_{}.png'.format(gesture_folder
    , image_number_color), arr = image ,cmap = 'gray') # Save image grayscale
67
68      plt.imsave(fname = '/home/pi/Documents/Lepton_imaging_git/
    complex_background_images/NUS/colorscale/plasma_image_{}.png'.format(
    image_number_color), arr = image ,cmap = 'plasma') # Save image colorscale
69      plt.imsave(fname = '/home/pi/Documents/Lepton_imaging_git/
    complex_background_images/{}/colorscale/plasma_image_{}.png'.format(
    gesture_folder, image_number_color), arr = image ,cmap = 'plasma') # Save image
     colorscale
70      del arr
71      camera.close()
72
73  #-----------------------------------------------------------------#
74
75  def check_folder_grayscale():
76
77      directory_path = os.path.dirname('/grayscale/')
                  # Fnds the directory for the path
78      path = os.getenv('HOME') + '/Documents/Lepton_imaging_git/
    complex_background_images/NUS/grayscale/'              # Finds the path to
    place files
79      number_files = len([f for f in os.listdir(path)if os.path.isfile(os.path.join(
    path, f))]) # Calculates the number of files in folder
80      print("Number of images in folder {}: {}".format(directory_path, number_files)
    )
81      return number_files + 1
```

```
82
83 #-----------------------------------------------------------------#
84
85 def check_folder_colorscale():
86
87     directory_path = os.path.dirname('/colorscale/')
                  # Fnds the directory for the path
88     path = os.getenv('HOME') + '/Documents/Lepton_imaging_git/
       complex_background_images/NUS/colorscale/'            # Finds the path to place
        files
89     number_files = len([f for f in os.listdir(path)if os.path.isfile(os.path.join(
       path, f))])  # Calculates the number of files in folder
90     print("Number of images in folder {}: {}".format(directory_path, number_files)
       )
91     return number_files + 1
92
93 #-----------------------------------------------------------------#
94
95 #--------main--------#
96 def main():
97
98 #--------Variables--------#
99     image_number = 0
100    #grayscale_folder = 1
101    #colorscale_folder = 1
102    number_files = 1
103
104    #for x in range (1):              # For loop to read through the folders at
       start
105    colorscale_folder = check_folder_colorscale()
106    grayscale_folder = check_folder_grayscale()
107
108    print("\n#####################################")
109    print("Select gesture folder to add images")
110    gesture_folder = str(input())
111
112
113    while 1:
114
115        print("\n#####################################")
116        print("Press 1 for 1 image of both colors")
117        print("Press 2 for x images of both colors")
118        print("Press 3 for colorscale image, only for example")
119        print("Press 4 for grayscale image, only for example\n")
120        print("Press 5 to see files in directories\n")
121        print("Press 6 to change gesture_folder")
122        print("Press 7 to see if images are modulus of 24\n")
123        print("Press 9 to exit")
124        print("#####################################\n")
125
126        num_input = int(input()) # Takes the input into a var.
127
128        if num_input >= 1 and num_input <= 9:
129            if num_input == 1:
130                print("\n#####################################")
131                capture_images(gesture_folder, grayscale_folder, colorscale_folder
       )    # Captures both type of images
132                grayscale_folder = check_folder_grayscale()   # Checks how many
       images of grayscale images the folder contains
133                colorscale_folder = check_folder_colorscale() # Checks how many
       images of colorscale images the folder contains
134                print("#####################################\n")
135            elif num_input == 2:
136                print("\n#####################################")
137                print("Enter the amount of images to take")
138                rounds = int(input())
139                for x in range (rounds):
```

```
                    capture_images(gesture_folder, grayscale_folder,
    colorscale_folder)
                    grayscale_folder = check_folder_grayscale()
                    colorscale_folder = check_folder_colorscale()
                    #time.sleep (0.2)
                print("#####################################\n")
            elif num_input == 3:
                print("\n#####################################")
                print("Enter the amount of images to take")
                rounds = int(input())
                for x in range (rounds):
                    capture_image_colorscale(gesture_folder, colorscale_folder)
                    colorscale_folder = check_folder_colorscale()
                print("#####################################\n")
            elif num_input == 4:
                print("\n#####################################")
                print("Enter the amount of images to take")
                rounds = int(input())
                for x in range (rounds):
                    capture_image_grayscale(gesture_folder, grayscale_folder)
                    grayscale_folder = check_folder_grayscale()
                print("#####################################\n")
            elif num_input == 5:
                print("\n################################################")
                print("Folder content: ")
                check_folder_grayscale()
                check_folder_colorscale()
                print("################################################\n")
            elif num_input == 6:
                print("\n#####################################")
                print("Current gesture folder: %s" % gesture_folder)
                print("Type new gesture folder")
                gesture_folder = str(input())
                print("#####################################\n")
            elif num_input == 7:
                if int(colorscale_folder - 1) % 24 == 0:
                    print("\n#####################################")
                    print ("Images are modulus of 24 (even) \nTotal number of
    images: {} + {} = {}".format(int(colorscale_folder -1), int(grayscale_folder -
    1), int(colorscale_folder + grayscale_folder - 2)))
                    print("#####################################\n")
                else:
                    print("\n#####################################")
                    print("\nNot a modulus of 24 (odd)")
                    print("%i images left to take" %((int(colorscale_folder - 1) %
     24) - 24))
                    print("#####################################\n")
            else:
                #camera.close()
                break;
        else:
            print("\nInput needs to be selected as stated")



#--------Start main--------#
if __name__ == "__main__": main()
```

Listing B.1: Complex background image capture code

Appendix C

Google Colab: Preprocessing Data

```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

Listing C.1: Import Google Drive

```
import glob
letter=['a','b','c','d','e','f','g','h','i','j']
dict={'a':[],'b':[],'c':[],'d':[],'e':[],'f':[],'g':[],'h':[],'i':[],'j':[]}
for i in letter:
  dict[i]=(glob.glob("/gdrive/MyDrive/thermal_sign_data/"+i+"/Colorscale/*.png"))
print(len(dict['a']))
```

Listing C.2: Label data

```
for i in letter:
  print(len(dict[i]))
```

Listing C.3: Print files in array

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

X=np.empty((7200,120,160,1))
Y=np.empty((7200,1))
k=0
for j in letter:
  for i in dict[j]:
    X[k,:,:,0]=cv2.imread(i,cv2.IMREAD_GRAYSCALE)
    k=k+1
'''
for i in range(0,10):
  for j in range(1,321):
    X[k,:,:,0]=cv2.imread("/gdrive/My Drive/thermal_images/"+str(i)+"/image_"+str(
    j)+".png",cv2.IMREAD_GRAYSCALE)
    Y[k]=i
    k=k+1
'''
print(k)
```

Listing C.4: Print total entries of "k"

```
print(X.shape)
```

Listing C.5: Print shape

```
import numpy as np
np.save("/gdrive/My Drive/thermal_images/X.npy",X)
```

Listing C.6: Save to Google Drive

```
k=0
label_dict={'a':0,'b':1,'c':2,'d':3,'e':4,'f':5,'g':6,'h':7,'i':8,'j':9}
for j in letter:
  for i in dict[j]:
    Y[k]=label_dict[j]
    k=k+1
print(k)
```

Listing C.7: Print total labeled entries of "k"

```
np.save("/gdrive/My Drive/thermal_images/Y.npy",Y)
```

Listing C.8: Save to Google Drive

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
test=np.empty((32,32,1))
im=cv2.imread("/gdrive/MyDrive/thermal_sign_data/a/Colorscale/plasma_image_4.png",
    cv2.IMREAD_GRAYSCALE)
#test[:,:,0]=im
print(im.shape)
plt.imshow(im)
plt.show()
plt.imshow(test[:,:,0])
plt.show()
```

Listing C.9: Show thermal and test image

Appendix D

Google Colab: Load Data as Arrays

```
import numpy as np
X=np.load("/gdrive/My Drive/thermal_images/X.npy")
Y=np.load("/gdrive/My Drive/thermal_images/Y.npy")
```

Listing D.1: Import images

```
print(X.shape)
print(Y.shape)
```

Listing D.2: Print shapes

```
X_=np.stack((X[:,:,:,0],)*3, axis=-1)
```

Listing D.3: Stack array

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
plt.imshow(X[1438,:,:,0])
print(Y[1438])
```

Listing D.4: Show image

```
print(Y.shape)
a=[0,0,0,0,0,0,0,0,0,0]
for i in range(Y.shape[0]):
  a[int(Y[i])]=a[int(Y[i])]+1
print(a)
```

Listing D.5: Print Y shape of array

Appendix E

Google Colab: Create Train and Test Dataset

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X_, Y, test_size=0.2,
    random_state=42,stratify=Y)

print(Y_test.shape)
a=[0,0,0,0,0,0,0,0,0,0]
for i in range(Y_test.shape[0]):
  a[int(Y_test[i])]=a[int(Y_test[i])]+1
print(a)
```

Listing E.1: Split dataset

```
print(X_train.shape)
```

Listing E.2: Print train data shape

```
X=None
#X_=None
```

Listing E.3: Set X = None

```
import numpy as np

X_train=np.empty((5760,120,160,1))
Y_train=np.ones((5760,1))
```

Listing E.4: Set train dataset

Appendix F

Google Colab: Create Model

```
import tensorflow as tf
from tensorflow import Tensor
from tensorflow import keras
from tensorflow.python.keras.layers import Input, Conv2D, ReLU, BatchNormalization
    ,\
                                    Add, AveragePooling2D, Flatten, Dense
from tensorflow.python.keras.models import Model
from tensorflow.keras.layers import MaxPool2D, AveragePooling2D, SeparableConv2D
initializer = tf.keras.initializers.he_normal()

def scheduler(epoch, lr):
  if (epoch%8)==0:
    return lr * tf.math.exp(-0.1)
  else:
    return lr

def relu_bn(inputs: Tensor) -> Tensor:
    relu = ReLU()(inputs)
    bn = BatchNormalization()(relu)
    return bn

def relu_u(inputs: Tensor) -> Tensor:
    relu = ReLU()(inputs)
    #bn = BatchNormalization()(relu)
    return relu

def bottleneck(x: Tensor, kernels:int, dilation: int) -> Tensor:


    y = Conv2D(kernel_size=1,
               strides= 1,
               filters=int(kernels/4),
               padding="same")(x)
    y = relu_bn(y)

    y = Conv2D(kernel_size=3,
               strides=1,
               filters=int(kernels/4),
               padding="same")(y)
    y = relu_bn(y)

    y = Conv2D(kernel_size=1,
               strides= 1,
               filters=kernels,
               padding="same")(y)
    y = relu_bn(y)

    out = Add()([x, y])

    y1 = Conv2D(kernel_size=1,
```

```
                strides= 1,
                filters=int(kernels/4),
                padding="same")(out)
    y1 = relu_bn(y1)

    y1 = Conv2D(kernel_size=3,
                strides=1,
                filters=int(kernels/4),
                dilation_rate=dilation,
                padding="same")(y1)
    y1 = relu_bn(y1)

    y1 = Conv2D(kernel_size=1,
                strides= 1,
                filters=kernels,
                padding="same")(y1)
    y1 = relu_bn(y1)

    out1 = Add()([out, y1])

    return out1

def resblock(x: Tensor, kernels:int, dilation: int) -> Tensor:
    y1 = Conv2D(kernel_size=1,
                strides= 1,
                filters=int(kernels/4),
                padding="same")(x)
    y1 = relu_bn(y1)

    y1 = Conv2D(kernel_size=3,
                strides=1,
                filters=int(kernels/4),
                dilation_rate=dilation,
                padding="same")(y1)
    y1 = relu_bn(y1)

    y1 = Conv2D(kernel_size=1,
                strides= 1,
                filters=kernels,
                padding="same")(y1)
    y1 = relu_bn(y1)

    out1 = Add()([x, y1])

    return out1

def create_net():

    inputs = Input(shape=(120,160,1))
    '''
    t = Conv2D(kernel_size=3,
                strides=2,
                filters=64,
                padding="valid")(inputs)
    t = relu_bn(t)

    t = Conv2D(kernel_size=3,
                strides=2,
                filters=128,
                padding="valid")(t)
    t = relu_bn(t)

    t = bottleneck(t,kernels=128,dilation=2)

    t = Conv2D(kernel_size=3,
                strides=2,
                filters=256,
```

```
                padding="valid")(t)
    t = relu_bn(t)

    t = bottleneck(t,kernels=256,dilation=4)


    t = Conv2D(kernel_size=3,
                strides=2,
                filters=128,
                padding="same")(t)
    t = relu_bn(t)

    t = bottleneck(t,kernels=128,dilation=8)

    t = Conv2D(kernel_size=3,
                strides=2,
                filters=64,
                padding="valid",kernel_initializer=initializer)(t)
    t = relu_bn(t)

    t=bottleneck(t,kernels=64,dilation=4)

    t = Conv2D(kernel_size=3,
                strides=2,
                filters=16,
                padding="valid",kernel_initializer=initializer)(t)
    t = relu_bn(t)



    t=bottleneck(t,kernels=32,dilation=2)

    t = Conv2D(kernel_size=3,
                strides=2,
                filters=16,
                padding="valid")(t)
    t = relu_bn(t)
'''
    t = Conv2D(kernel_size=(3,2),
                filters=16,
                padding="valid",data_format="channels_last")(inputs)
    t = relu_bn(t)

    #t= MaxPool2D((2,2))(t)
    #t=Dropout(0.3)(t)

    #t = Conv2D(kernel_size=(3,4),
                #filters=4,
                #padding="valid",kernel_initializer=initializer,data_format="
    channels_last")(t)
    #t = relu_bn(t)

    #t= MaxPool2D((2,2))(t)
    #t=Dropout(0.3)(t)
    #t= resblock(t,kernels=16,dilation=2)
    t = Conv2D(kernel_size=(3,2),               filters=32,                padding="
    valid",data_format="channels_last")(t)
    t = relu_bn(t)
    #t = Conv2D(kernel_size=3,filters=32,                padding="valid",
    data_format="channels_last")(t)
    #t = relu_bn(t)

    t= MaxPool2D((3,2))(t)
    #t=Dropout(0.3)(t)

    t = Conv2D(kernel_size=(5,4),filters=64, dilation_rate=2, padding="valid",
    data_format="channels_last")(t)
```

```
    t = relu_bn(t)

    #t= MaxPool2D((2,2))(t)
    #t=Dropout(0.35)(t)
    #t= resblock(t,kernels=32,dilation=4)
    #t=MaxPool2D((2,2))(t)
    t = Conv2D(kernel_size=(5,4) ,filters=128, dilation_rate=2,padding="valid",
    data_format="channels_last")(t)
    t = relu_bn(t)
    t=MaxPool2D((3,2))(t)

    #t=Dropout(0.35)(t)
    #t = Conv2D(kernel_size=5,filters=256, padding="valid",kernel_initializer=
    initializer,data_format="channels_last")(t)
    #t = relu_bn(t)


    t = Flatten()(t)
    outputs = Dense(10, activation='softmax')(t)

    model = Model(inputs, outputs)

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=0.001),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    return model
```

Listing F.1: Create model architecture

```
net = create_net()
print(net.summary())
```

Listing F.2: Print summary

Appendix G

Google Colab: Train model

```
net = create_net()
callback = tf.keras.callbacks.LearningRateScheduler(scheduler)
a=net.fit(X_train,Y_train,batch_size=32,epochs=50,validation_split=0.1,shuffle=
    True,callbacks=[callback])
```

Listing G.1: Training parameters

Appendix H

Google Colab: 10 Fold Validation

```
from sklearn.model_selection import StratifiedKFold
from numba import cuda
from keras import backend as K
import gc
kfold = StratifiedKFold(n_splits=10, shuffle=True)
acc_per_fold = []
loss_per_fold = []
fold_no=1
a=[0,0,0,0,0,0,0,0,0,0]
#test_X=np.empty((5,1440,120,160,1))
#test_Y=np.empty((5,1440,1))
for train, test in kfold.split(X_train,Y_train):
  '''
  print(X[train].shape)
  print(X[test].shape)

  for i in range(Y[test].shape[0]):
    a[int(Y[test][i])]=a[int(Y[test][i])]+1
  print(a)
  break
  '''
  net = create_net()
  callback = tf.keras.callbacks.LearningRateScheduler(scheduler)
  history = net.fit(X_train[train], Y_train[train],
              batch_size=32,
              epochs=50,
              verbose=1,callbacks=[callback],validation_data=(X_train[test],
    Y_train[test]))

  # Generate generalization metrics
  scores = net.evaluate(X_train[test], Y_train[test], verbose=0)
  print(f'Score for fold {fold_no}: {net.metrics_names[0]} of {scores[0]}; {net.
    metrics_names[1]} of {scores[1]*100}%')
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])

  # Increase fold number
  fold_no = fold_no + 1
  if (fold_no == 11):
    h=history
    model=net
    #test_X=X[test]
    #test_Y=Y[test]
  #test_X[fold_no-2,:,:,:,:]=X[test]
  #print(Y_test.shape)
  #test_Y[fold_no-2,:,:]=Y[test]
  del net
  K.clear_session()
  gc.collect()

```

```
print('------------------------------------------------------------------------')
print('Score per fold')
for i in range(0, len(acc_per_fold)):
  print('------------------------------------------------------------------------'
    )
  print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy: {acc_per_fold[i]}%')
print('------------------------------------------------------------------------')
print('Average scores for all folds:')
print(f'> Accuracy: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)}')
print('------------------------------------------------------------------------')
```

Listing H.1: 10 fold validation

```
from keras import backend as K
import gc
del net
K.clear_session()
gc.collect()
```

Listing H.2: Garbage handling

Appendix I

Google Colab: Final Test Accuracy

```
print("Evaluate on test data")
results = net.evaluate(X_test, Y_test, batch_size=1)
print("test loss, test acc:", results)
```

Listing I.1: Evaluate model

Appendix J

Google Colab: Save the Model

```
net.save('/gdrive/My Drive/thermal_images/model_final_separable_.h5')
```

Listing J.1: Save model

```
net.save('/gdrive/My Drive/thermal_images/model_final_new_99.88_.h5')
```

Listing J.2: Save model

```
np.save('/gdrive/My Drive/thermal_images/k_test_X.npy',test_X)
np.save('/gdrive/My Drive/thermal_images/k_test_Y.npy',test_Y)
```

Listing J.3: Save images

```
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

import seaborn as sns
from tensorflow import keras
moe=keras.models.load_model('/gdrive/My Drive/thermal_images/model_98.1_k_.h5')
```

Listing J.4: Load model

```
moe.summary()
```

Listing J.5: Print summary

Appendix K

Google Colab: Convert TensorFlow to TensorFlow Lite

```
import tensorflow as tf
from tensorflow import lite
from tensorflow import keras

#mode = keras.models.load_model('/gdrive/My Drive/thermal_images/
    model_final_new_k_.h5')
#mode.save('/gdrive/My Drive/final_model_.h5')
converter = lite.TFLiteConverter.from_keras_model(net)
tflite_model = converter.convert()
'''
#net = keras.models.load_model('/gdrive/My Drive/thermal_images/
    model_final_new_tf_2.2.0_k_.h5')
batch_size = 2
input_shape = net.inputs[0].shape.as_list()
input_shape[0] = batch_size
func = tf.function(net).get_concrete_function(
    tf.TensorSpec(input_shape, net.inputs[0].dtype))
converter = tf.lite.TFLiteConverter.from_concrete_functions([func])

tflite_model = converter.convert()
'''
open('/gdrive/My Drive/thermal_images/model_final_separable_.tflite', 'wb').write(
    tflite_model)
```

Listing K.1: Convert TF to TFLite

Appendix L

Google Colab: Timing code

```
import numpy as np
import tensorflow as tf
import time

interpreter = tf.lite.Interpreter(model_path='/gdrive/My Drive/thermal_images/
    model_final_separable_.tflite') # change path
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

interpreter.allocate_tensors()


t=[]
for i in range(20):
  t1=time.time()
# Test the model on random input data.
  input_shape = input_details[0]['shape']

  #x=np.empty((1,120,160,3),dtype=np.float32)# for benchmark models
  x=np.empty((1,120,160,1),dtype=np.float32) # for our model
  interpreter.set_tensor(input_details[0]['index'], x)

  interpreter.invoke()

  output_data = interpreter.get_tensor(output_details[0]['index'])
  t2=time.time()
  t.append(t2-t1)
print(np.average(np.array(t)))
```

Listing L.1: Timing code