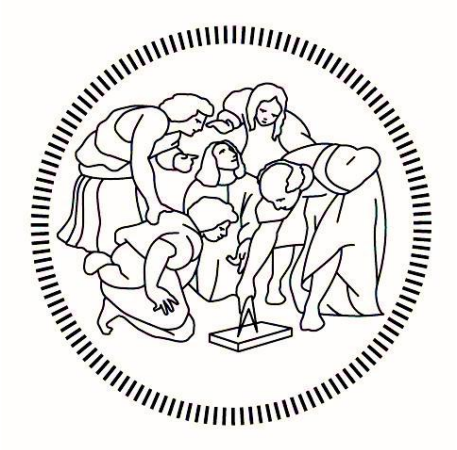
POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

MASTER THESIS IN AUTOMATION AND CONTROL ENGINEERING



Hyperspectral Imaging and Deep Learning for Automatic Food Quality Inspection

Academic Supervisor: Prof. Marco TARABINI

Academic Co-Supervisor: Ing. Davide Maria FABRIS

Master Thesis by:

Filippo Gorlini - 10487839
Robby Izaty Ramadhan - 10655158

Academic Year 2020 - 2021

Thanks to my parents, for always believing in me.
Thanks to my friends, for getting a smile from me when I needed the most.
Thanks to all the Stackoverflow community for saving me a lot of gray hairs (even though I have none).

-Filippo Gorlini

Thanks to my wife, for your unconditional love.
Thanks to my family, for always be there.
Thanks to my friend, for the good and the bad times.
Thank God, for your blessed.

-Robby Izaty Ramadhan

Abstract

This master thesis focuses on the application of Deep Learning algorithms for performing Hyperspectral Image semantic segmentation, with a focus on food quality inspection and sorting. Hyperspectral Imaging is a powerful instrument, still the applicability of this camera is mostly limited to remote sensing applications. The principal objective of this master thesis is the pixel-level classification of different quality of eggplants using Hyperspectral Imaging systems coupled with Deep Learning algorithms. There are several available options for the semantic segmentation algorithm using Deep Learning that can be found in some of scientific databases. In this master thesis, four benchmark Deep Learning models have been adapted and trained on an in-house collected dataset. The HSI data coming from a push-broom type camera. This is the most diffused data acquisition method in industrial applications, as many of the system in industry rely on a conveyor belt to move the products.

The research and development of novel methodologies for carrying out Deep Learning-based HSI segmentation is the final innovative contribution of the present work. Moreover, the work also provides techniques to analyze hyperspectral data from line scanners application. The results of this work show that Hyperspectral Imaging systems, coupled with Deep Learning, lead to very good performance for the semantic segmentation task for automatic food quality inspection. Results were validated through the k-Fold Cross-Validation methodology.

Key words: Deep Learning, Hyperspectral Image Segmentation, Food Sorting

Table of Contents

List of Figures

1. Introduction

This Master Thesis focuses on the application of Deep Learning algorithms for the Hyperspectral Image semantic segmentation, with a focus on food quality inspection and sorting. Hyperspectral Imaging (HSI) systems are commonly used for remote sensing application. As a matter of fact, there is only limited scientific production addressing the application of HSI in the food industry. Moreover, the scarce availability of related data collections in the study is another important challenge to be considered.

The principal objective of the master thesis is the pixel-level classification of different quality of eggplants using HSI systems coupled with Deep Learning Algorithms (DL). In the real industry implementation, this technology will increase the efficiency of the food processing by automating the sorting process. There are several available options for the segmentation algorithm: either relying on conventional Machine Learning (ML) algorithms or on Deep Learning. Both techniques will be briefly described in the following sub-chapters.

The research and development of novel methodologies for carrying out Deep Learning-based HSI segmentation is the final innovative contribution of the present work. Moreover, the work also provides techniques to analyze any HSI data from line scanners application. Several models are trained and tested; results are eventually presented.

1.1. State of the Art

In this section, the state of the art of the three principal technologies on which this master thesis is built on will be briefly described. These subjects are Food Sorting, HSI, and DL.

Food sorting is a crucial activity in the food industry. Therefore, the use of innovative technology is of vital importance for the competitive advantage of any company. Mainly, food sorting is divided into three categories. These are external quality and defect evaluation, internal quality and maturity assessment, and food safety detection [1]. This master thesis is focusing on the external quality and defect evaluation problem, but also on the internal characteristics. There are several technologies already established depending on its enabling technology that will briefly described in the next subchapter.

HSI is a technique that generates a spatial map of spectral variation, making it a useful tool in many applications [2] especially for semantic image segmentation. HSI system produces two-dimensional spatial array of vectors which represents the spectrum at each pixel location [2]. The resulting three-dimensional dataset containing the two spatial dimensions and one spectral dimension is known as the data cube or hypercube [3]. As a comparison, usual RGB camera produce only three spectral data (i.e., corresponding to the red (R), green (G) and blue (B)). In RGB imaging the picture is a matrix of three channels. On the other hand, the data cube might have hundreds of channels. The data richness of data cube is the power of HSI systems, but also at the same time it brings by some challenges.

Deep learning is a supervised process that enables machines to recognize a pattern in the world by studying it beforehand with labelled data. This process is technically implemented using a digital architecture called Neural Network (NN). The design of Neural Network is inspired by how human brain works. As a matter of fact, human brain can process huge amount of information using billions combination of neuron cells. These information are electrical signals that comes from the world senses by the human body's sensors. Later, these signals processes by neuron cells into an understandable information as a class of vision, texture, or sound. The NNs in Deep Learning work the same. NNs is using the combination of,

also so called, neuron to identify complex patterns in the real world. Instead of processing electrical signals, artificial neural networks process numbers that represents the world features into meaningful information. The word deep just comes from the architectures of neural networks that have multiple layers of inter-connected neurons.

1.1.1. Food Sorting Technologies

Food industry is one of the world's biggest market. Food manufacturers are developing their product lines continuously to remain competitive. This is including making sure that their product is always fresh and in a good shape. This is not an easy task when the scale of the company's production capacity is in metric tons. Using human worker to classify food product is not only cumbersome, but also uneconomical. Fortunately, computer vision is now able to do this task with high precision and robustness.

There are numerous methods already developed by other researchers related to food sorting technology using computer vision. Traditional, hyperspectral, and multispectral computer vision systems are the most widely used vision systems in the external quality inspection of food and agricultural products [4]. Traditional computer vision systems are using the usual RGB camera system, widely used for their ease of use. The second technique is multispectral imaging (MSI), which can acquire up to 15 spectral band. The last one is the HSI system, capable of acquiring hundreds of contiguous spectral bands. Each method has its own strengths and weaknesses. Traditional computer vision systems are cheap compared to the other two methods. But they feature limitations on their performance. Multispectral imaging usually work at specific bands to excel at one task while performing poorly in others [4]. HSI systems perform well above the other two. Their ability to acquire hundreds of material spectral band brings huge potential for the application. Nevertheless, the computation requirement for this type of system is very high.

Food sorting is qualitative type of image segmentation. Thus, it belongs to the domain of pattern classification in machine learning algorithm. Several qualitative

models that are used for food sorting have been developed for HSI systems. Among them the following can be cited: Linear Discriminant Analysis (LDA), Correlation Analysis (CA), Principal Components Analysis (PCA), K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Partial Least Square Discriminant Analysis (PLSDA), and Deep Learning [1].

Mehl et al [5] were among the first to apply HSI for surface defect classification in fruits [1]. Later, they also presented Correlation Analysis (CA) method for the same detection problem. Moscetti et al [6] applied multi-class Partial Least Square Discriminant Analysis (PLS-DA) classifier for hazelnuts classification problem. Cheng et al [7] reported combination of PCA-LDA as a hybrid dimension reduction technique with k-NN for classification of physiological disorder during postharvest handling of horticultural commodities of tropic or subtropic origin like cucumbers, apples, and peaches. Finally, the method that increasingly popular in recent years is Deep Learning. Owing to the advancements in computing technologies, especially the utilization of GPUs, Deep Learning able to solve computer vision tasks with superior accuracy [1]. Later, the various Deep Learning state-of-the-art algorithm will be presented and described in subchapter 1.1.3.

1.1.2. Hyperspectral Imaging (HSI)

HSI is a technique that generates a spatial map of spectral variation [2], often times resulting in hundreds of spectral bands. Ordinary camera usually only assigning three spectral bands (red, green, blue) to its image. It is mostly enough for human to identify image using the combination of those colors. The main impetus for developing a HSI system was to integrate spectroscopic and imaging techniques to enable direct identification of different components and samples spatial distribution [2]. Hence, it is superior in its ability to identify material characteristics compared to ordinary monochrome and RGB camera. HSI brings possibilities to analyze spectrum that is invisible for human eyes and to characterize materials.

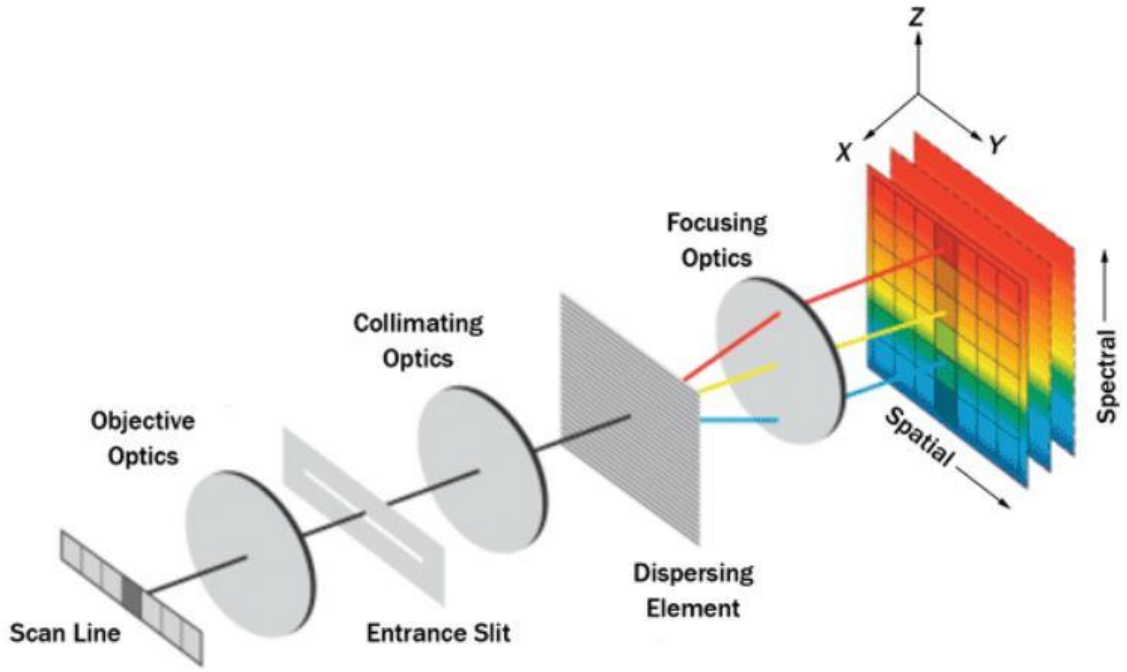


Figure 1 – Image acquisition process in HSI camera

The process starts from the reflection of the light that is coming from the samples going into the entrance slit of the camera as can be seen in **Error! Reference source not found.**[1]. The light is then diffracted into its individual wavelength and captured by the detector array (CCD). The spatial information of the image is maintained as the image preserved the image projection of the world. Furthermore, the spectral information of each band is also preserved as the light intensity image for each spectrum band are stacked onto each other creating a hyperspectral data cube as can be seen in Figure 2[2].

Hyperspectral data cube, known also as hypercube, is three-dimensional data which provide physical and/or chemical information of the samples. It characterized by a very large volume and dimensionality. Data cube can contain information of hundreds of wavebands and hundred thousand of pixels for each waveband. The amount of data is the greatest problem that must be coped with. The first goal of data analysis is therefore to decrease the data size. It is ironic considering the goals

[1] https://www.photonics.com/Articles/Hyperspectral_Imaging_Enables_Industrial/a56804
[2] https://www.cleanpng.com/png-hyperspectral-imaging-data-cube-photon-etc-market-3123261/

of using Hyperspectral camera is to gain samples data as much as possible. Nevertheless, this is something need to be solved as in practice classification of samples must be done in real time.

There are three acquisition modes of HSI[2]. The first is area scanning imaging configuration. This method is performed by gathering the images at one wavelength at a time until all the spectral bands images are taken. The second technique is whiskbroom, or point-scan imaging. This technique scans a single pixel at a time but taking directly all the spectral information of the picture, with the scanning element moving continuously through the image. The third technique is the push-broom, or line-scan imaging. This technique records whole line of an image using two-dimensional dispersing element (grating) and two-dimensional detector array. A narrow line of the specimen is imaged onto a row of pixels on the sensor chip and the spectrograph generate a spectrum for each point on the line, spread across the second dimension of the chip. This technique is commonly used in the food industry because the nature of the product movement in the conveyor belts.

HSI systems cannot stand alone without the help of some software for gaining high performance in acquisition, controlling, and analysis[2]. The first step is the collection of a HSI using hyperspectral camera. Then the spectral data are extracted

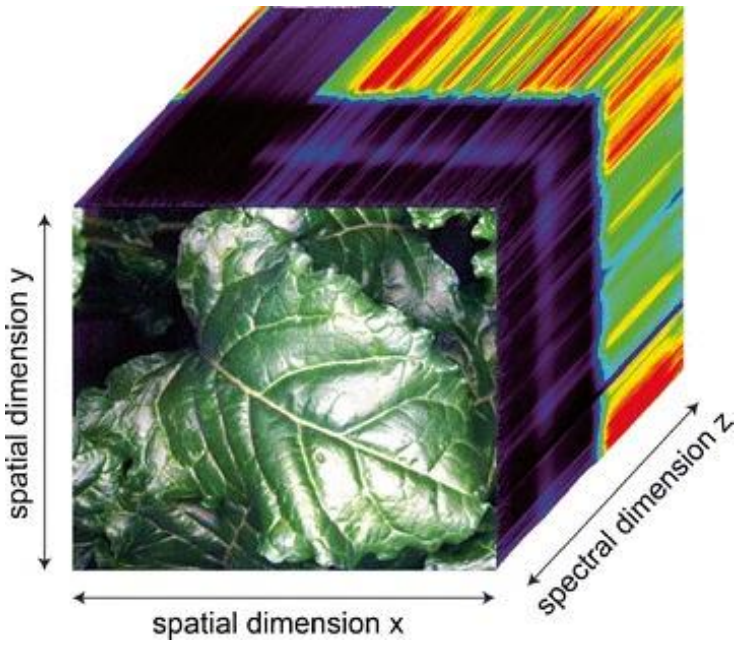


Figure 2 – Visualization of HSI data cube of a leaf

from different regions of interest (ROIs) that present different quality features[2]. Before processing any further, the data must be preprocessed. Many algorithms are available to preprocessed hyperspectral data, depending on whether the interest is to preprocesses the spectral feature or the spatial feature. Earlier research related to HSI analysis focused on Multivariate statistics [8] such as Euclidean distance correlation for correlation technique, and principal components analysis (PCA) for classification. Recent years have set off a wave of deep learning for analysis technique for hyperspectral data [2], mainly by means of Convolutional Neural Networks. This novel technique has excellent capabilities in image processing owing to the advancement of computing technology. This master thesis uses deep learning as its technique to analyze the data cube. Next sub-chapter will briefly introduce the topic.

1.1.3. Deep Learning

Deep learning is a branch of machine learning based on artificial neural networks. The term neural is highly correlated with the human brain. In fact, just like the over 100 billion of neurons in our brain, artificial neural networks aim to extract high-level information from raw data by breaking it down in a collection of low-level simple features.

In the last 20 years, the computational power has increased exponentially along with the amount of available data. These two factors are letting the deep learning to rapidly evolve and out-perform the traditional machine learning algorithm.

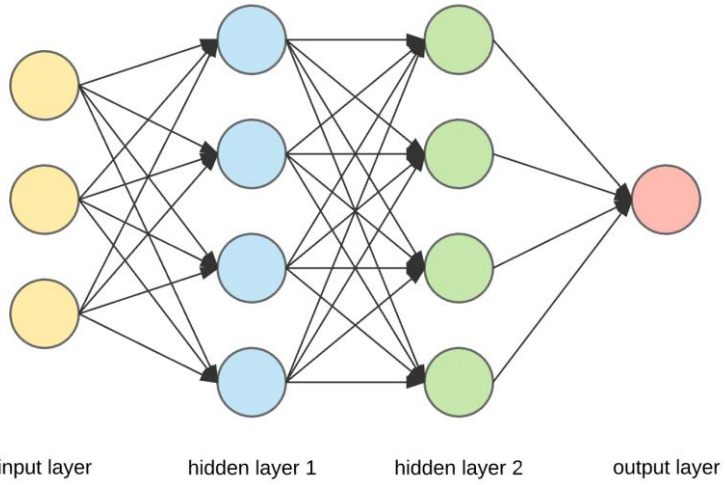


Figure 3 – Visualization of Artificial Neural Network

It is possible to visualize an artificial neural network as a sequence of layers, which represent the depth of the network. A visualization of artificial neural can be seen in the Figure 3[3]. The input is usually the raw data and the output represents the decision such as the classification of a pixel, a yes or no answer, a voice recognition and so on. However, this raw data needs to be prepared before feeding into the network. There are several requirements for the raw data so that the model can work properly, and it is really depending on how the model works. The process to prepare the data is called pre-processing.

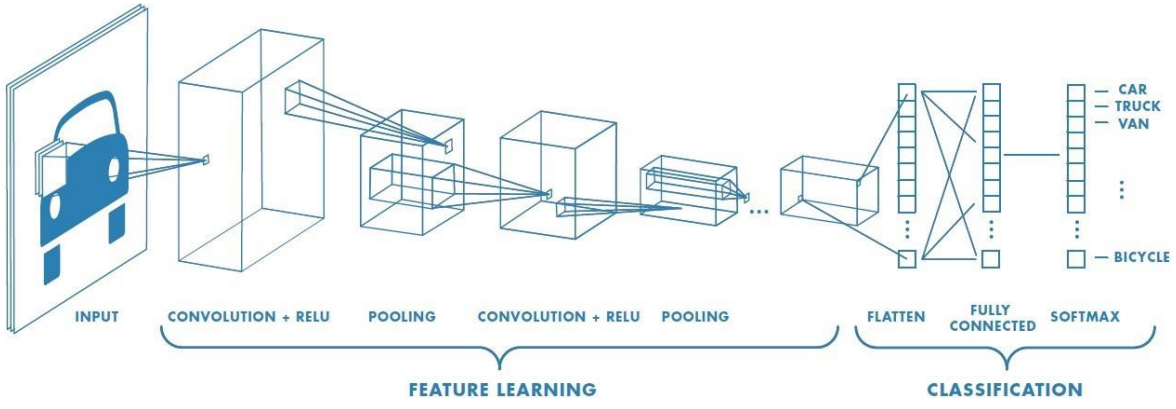


Figure 4 - The visualization of CNN classifies a 2-dimensional image.

[3] https://medium.com/@16611056/machine-learning-2-artificial-neural-network-b57b9b716f78

The process of a training starts from what is called forward propagation. During this process, the raw input is fed to the network and convoluted with the weights of the kernel in each layer. Then the output of each layer is fed to an activation function which results is acting as an input to the next layer. After the last layer, the decision is taken, and a Loss Function gives a numerical value that depicts the distance between the algorithm decision and the true labels. Then an optimization criterion modifies the weights starting from the last layer until the first one with the objective to minimize the result of the Loss Function. This model's weight update operation is called backward propagation. Once all weights are updated, a new input data is fed to the network and the same training process repeated until the last epoch.

Since this master thesis is mostly about image analysis it is important to introduce the most important notion in artificial neural network for image analysis called the Convolutional Neural Network, or CNN. The emphasis of CNN is the use of kernel to convolute the image samples to extract the lower-level features. The example of CNN architecture can be seen in the **Error! Reference source not found.**[4]. CNN is better in exploiting the spectral and spatial correlations of an image. Moreover, convolutional operations along with multi-dimensional kernels allows to reduce the number of variable weights in the development of very deep neural networks.

There are numerous frameworks that enables deep learning with their own strengths and weaknesses. This master thesis developed using Pytorch, an open-source machine learning library based on the Torch library. Pytorch mainly written in Python, but also has a C++ interface. The framework is created with CUDA support by default. It is useful to make the program run faster by harnessing the capabilities of GPU power.

This master thesis uses CNN for HSI segmentation and classification using Pytorch. The goal is to classify food quality in real time scenario. Several state-of-

[4]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

the-art technologies in food sorting application will be introduced and compared with the deep learning technique developed in this master thesis.

1.2. Existing Applications

Most of the research related to HSI System are in remote sensing applications. Traditionally, it is famous method to analyze the composition of different class in the data-cube earth's image. But now some researchers start to apply the camera also for other application. In this section, several deep learning algorithms related to the analysis of the HSI data will be briefly described.

Many algorithms have been applied to analyze the data from HSI camera using deep learning. In image analysis the deep learning algorithm that mostly use is the Convolutional Neural Network (CNN). Here several deep learning algorithms are discussed, both in the remote sensing application and food industry application. Commonly, there are three different CNN architecture used for analyzing HSI image depends on the dimension of the convolution kernel.

Hamidah et al [9] developed 3-D CNN approach to the hyperspectral data sets created by University of Pavia. Hu et al [10] developed 1-D CNN approach to analyze HSI data of Indian pines, Salinas, and also the University of Pavia scenes. Whereas Roy et al [11] developed the hybrid 3-D 2-D CNN approach to analyze the Indian pines, University of Pavia, and Salinas scene. In remote sensing, spatial feature is as important as the spectral feature. Thus, it is useful to use 3D CNN that able to extract both features [11]. Whereas the 2D and 1D CNN only extract respectively the spatial and spectral feature.

Moreover, there are also few developed deep learning algorithms to analyze HSI data from food industry. Wang et al, [12] used ResNet and ResNeXt model to detect internal mechanical damage of blueberries using HSI transmittance data. Zhang et al. [13] developed novel CNN architecture for fine-grained classification of banana's ripening stages. Steinbrener et al. [14] used modified GoogLeNet model to classify various fruit with high accuracy.

As it shows in the previous paragraph, the development of the deep learning algorithm for HSI data analysis is still in the early phase. As Liu et al. [15] suggest in 2017 that deep learning model, especially CNNs should be applied more frequently as it is already shows convincing results in the other area of study.

1.3. Scheme of the thesis

This thesis report is articulated in four main chapters: Setup, Method, Results, and Conclusions.

The Setup chapter is divided into two sub-chapters, Experimental Setup and Software Setup. In Experimental Setup, there are information about the HSI camera that is used to get the data and a brief description of the whole experimental setup in which includes the conveyor belt and specific lighting conditions. Moreover, a gentle introduction to the Software Setup adopted for the program is provided.

Next chapter is related to the applied and developed Method. Here can be found a more in-depth description of the program with all the main logics behind the developed algorithms to accomplish the thesis tasks. This section describes the main metric parameters, preprocessing of the raw dataset, and model architectures adopted to generate the semantic segmentation.

The subsequent chapter is the Results, as the name suggests it refers to all the most relevant results obtained during the large number of experiments. The performances are measured in terms of accuracy and testing time. Independent subsections consider all the comparison between the results obtained with different models. Then a discussion related to the results from the developed algorithms is introduced.

Finally, the Conclusions. This chapter recaps in bullet points the main results of this work, offering at the end our opinions regarding the results obtained and suggestions that hopefully useful for the future research and studies.

2. Setup

The setup for the image acquisition of the eggplants is divided into two main categories. The first one is the experimental setup, that is related to the hardware setup of the HSI imaging system. The second one is the software setup, that is related to the preparation of the environment of the developed program.

2.1. Experimental Setup

The image acquisition of the eggplants is conducted in the ImageS laboratory. The selection and the design of the setup is adapted to the requirements of the master thesis. There are two main system involved, the first one is the Specim FX17 HSI camera, and the second one is the supporting equipment Specim LabScanner 20x20. The camera that is used in this master thesis is line scanning camera, thus it is important to set the conveyor speed of the LabScanner to match the sampling time of the camera. It is also important to calibrate the camera during the experiment.

2.1.1. Specim FX17 Camera

There are not many companies that producing industrial HSI camera for food processing application. One of the leading companies producing HSI camera and imaging systems is Finland's technology firm, Specim. For years, Specim have developed numerous HSI camera for remote sensing applications. It was widely credited for its Thermal Infrared Hyperspectral Cameras, that is the first Hyperspectral Camera that can efficiently be used for outdoor surveillance and UAV applications without an external light source such as the sun or the moon.

Nowadays, Specim broaden their business by producing HSI camera specialized for industrial sorting applications. This camera using line scan technology to inspect the chemical substance of a sample with good performance.

Figure 5 - Specim FX17 made by Specim Spectral Imaging Oy Ltd

FX17 is series of Specim camera operated in near-infrared region (900 nm to 1700 nm), can sample the invisible features of the normal camera or human eyes. Furthermore, the camera can reveal chemical composition of a target, making it superior for sorting technology implementation. It has a spatial resolution of 640 pixels and image speed of 527 FPS for GigE version and 670 FPS for CameraLink version. This specification is sufficient for this master thesis. The camera can be seen in Figure 5.

Camera focus Configuration

To make the camera work properly, it is needed to set manually a few parameters. The first operation is to set the aperture wide open. Once this operation is done, it is necessary to adjust the focus (as the FX17 is a manual focus camera), by rotating the focusing ring. On the PC connected to the camera it is possible to see in a preview of the picture with real time changes in terms of exposure and focus with respect to the setting of the camera parameters we are setting. The visualization of the process can be seen in Figure 6.

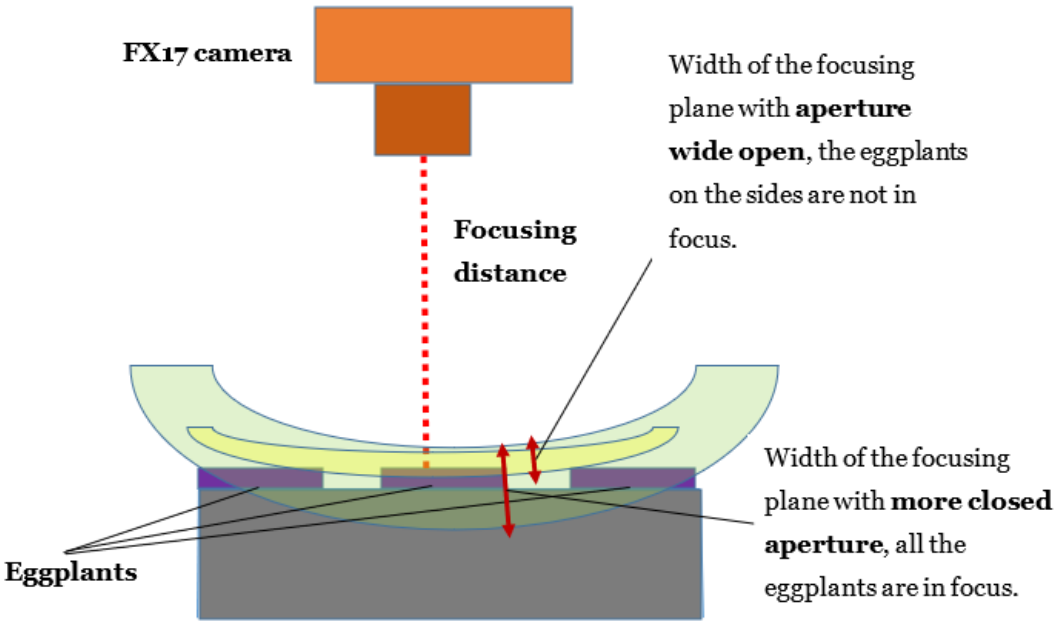


Figure 6 - Picture shows the importance of a correct set-up in terms of aperture and focusing distance.

After following the procedure above, everything on the conveyor belt is focused. Since the aperture is wide open, the focusing plane is the thinnest. The next step is to close the aperture narrow enough to have all the scene perfectly in focus, this operation alone will result also in a darker image, in fact less light is getting inside the lens with the diaphragm more closed. It is necessary to compensate the exposure to have a usable image, so it is needed to increase the exposure time. This is because when the shutter is open for a longer period, than the more light is getting inside the lens. False setting of the focus of the Hyperspectral camera not only will results in blurry spatial features, but spectral features [16] too. Hence reduces the performance of the model that is using the data for training. The result from this procedure is an all-around sharp focused image ready to be used for data acquisition.

Finally, Figure 7 shows the image acquisition setting for the master thesis. The FX17 camera is mounted on the top of the sliced eggplants to acquire the data. Other than that, it is also possible to see the white reference at the edge of the conveyor. This bar is useful for dark current calibration that will be explained in the dark current calibration sub-chapter.

Figure 7 - Dark current calibration and image acquisition setting

2.1.2. Supporting System Tools

The main supporting system of the HSI System [17][17] is Specim's LabScanner 40 x 20, a small scanner system for laboratory use. It has 400 x 200 mm sample tray, a mount of a camera, halogen illumination and optional camera height adjustment. As can be seen from Figure 8, the sample will be moved by the conveyor below the camera that is mounted on the top of the tools. The scanning speed range of the tools is around 0.1 mm/s until 99 mm/s. But this choice of speed is not arbitrary, as it is really depending on the camera parameter such as its exposure time. Failing to set the speed properly will results images having bad spatial data quality as will explained in the next discussion.

Figure 8 - Specim LabScanner 400 x 200 made by Specim Spectral Imaging Oy Ltd

Conveyor Belt Speed Configuration

One of the important aspects for the tool setup of line-scanning camera is the setting of the conveyor belt speed. It is because line scanning camera works by capturing line by line the spatial and spectral features of the samples. Thus, relative motion between the camera and the samples needs to be set. In practice, the camera will be set fix to the ground, and the samples will move with respect to the ground. The setting of the relative speed is performed by putting a circle-shape sample above the conveyor belt that is continuously moving when the data acquisition process starts. If the relative speed is right, then the resulting picture will not change the shape of the samples.

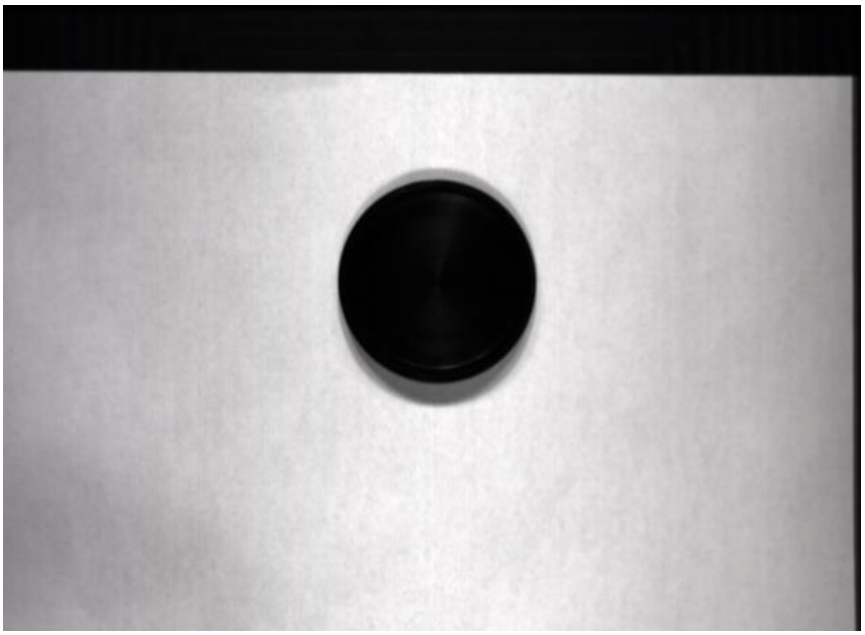


Figure 9 - Elongated circle sample due to the too low conveyor speed

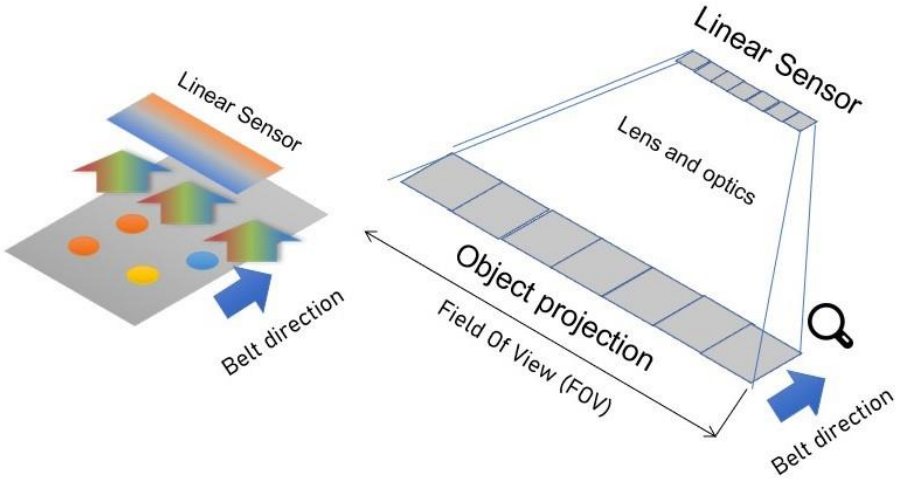


Figure 11 Image acquisition of line-scanning camera

Figure 10 - Elongated circle sample due to the too low conveyor speed

The problem arises when the exposure time of the camera is not match with the speed of the samples. If the speed is too high, then the resulting picture will be wider in the orthogonal direction of the motion. If the speed is too low, the picture of the object will wider in the moving direction. Example of the elongated samples can be seen in the **Error! Reference source not found.**. Using this fact, then the speed of the conveyor belt can be tuned by obtaining a picture of the object with the same shape of the samples.

Before tuning the speed, it is possible to analyze the required speed of the conveyor buy studying how the line-scanning camera work from Figure 11.

Sensor of the camera works with sampling rate f_{rate}. On the other hand, each pixel in Figure 11 must be exposed by light for $t_{exposition}$ long. Moreover, there is also processing and transmitting data time needed to be considered $t_{processing}$. Thus, the f_{rate} must be fulfilled the following equation to make the picture acquired properly.

$$\frac{1}{f_{rate}} \geq t_{exposition} + t_{processing}$$

The resulting analysis of the speed requirement is not exactly resulting perfect circle image of the reference sample. This is due to the fact that there is small difference between the parameter used to calculate the speed, with the real physical parameter. Thus, tuning of the speed is still important step to do.

2.2. Dataset Samples

The dataset is acquired by taking several classes of eggplants using HSI system already explained above. There are several types of experiments conducted with the data acquisition modules. From those experiments an extracting algorithm performed to gather the pixel samples, with its respective labels, for training.

First, several eggplants (*Solanum melongena*) with identical shape are considered. The first hyperspectral data-cube created from taking the whole image of the healthy eggplant as can be seen from Figure 12. Later the same eggplants will be cut into pieces and the data-cube are created by taking its image using the HSI systems. The second eggplants will be damaged. Then the whole damaged eggplant is scanned using the same HSI system to take its image data-cube. The third eggplants will be cut into four pieces. The first piece was left at it is. The second and the third pieces were baked respectively for 10 minutes and 20 minutes. The last piece was places into a tray, and using a heat lamp, burnt the middle of the pieces. All of four pieces are then placed on the conveyor belt, and one single data-cube is acquired from the HSI system as can be seen in Figure 13.

After all the data-cube are acquired, the next step is to apply the extracting algorithm to create the labelled dataset ready for training. Region of Interest (ROI) function from OpenCV library is used to show the location of the selected area of the data-cube image. Later, the user will need to decide the class of the selected samples. This process is repeated several times depending on how many samples, and how many classes the user wants to create. The results of the dataset extracting algorithm are two mat files. The first one is containing all the selected pixels stacked in a column with the spectrum data lies as its row. The second file is only containing the labels of its respective pixels. It means that the number of the row in both files will be the same and it shows relation between the pixel and its label. Moreover, the

program also saved the information related to the creation of the labelled dataset. A txt file containing information of the created class, the coordinate of the ROI, and the name of image from which the ROI is taken, is created and saved for the documentation purposes.

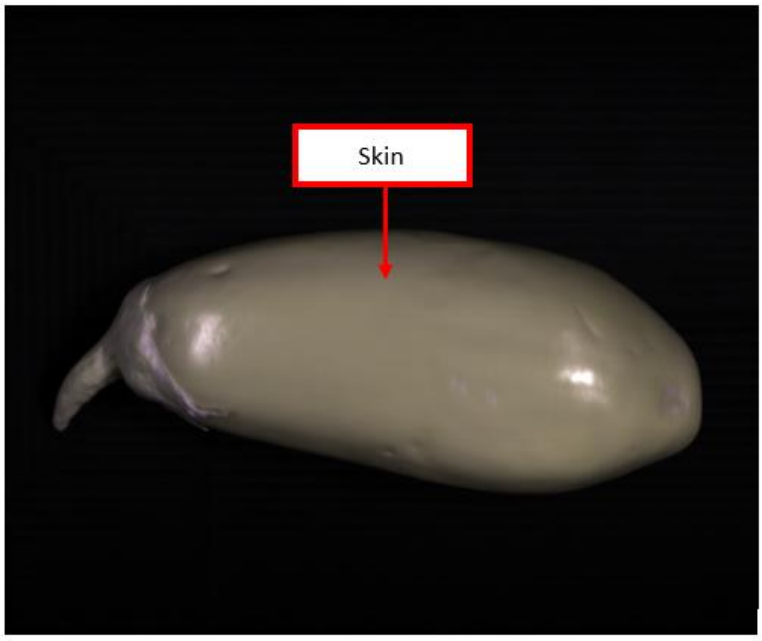


Figure 12 – The false color image of the entire eggplant

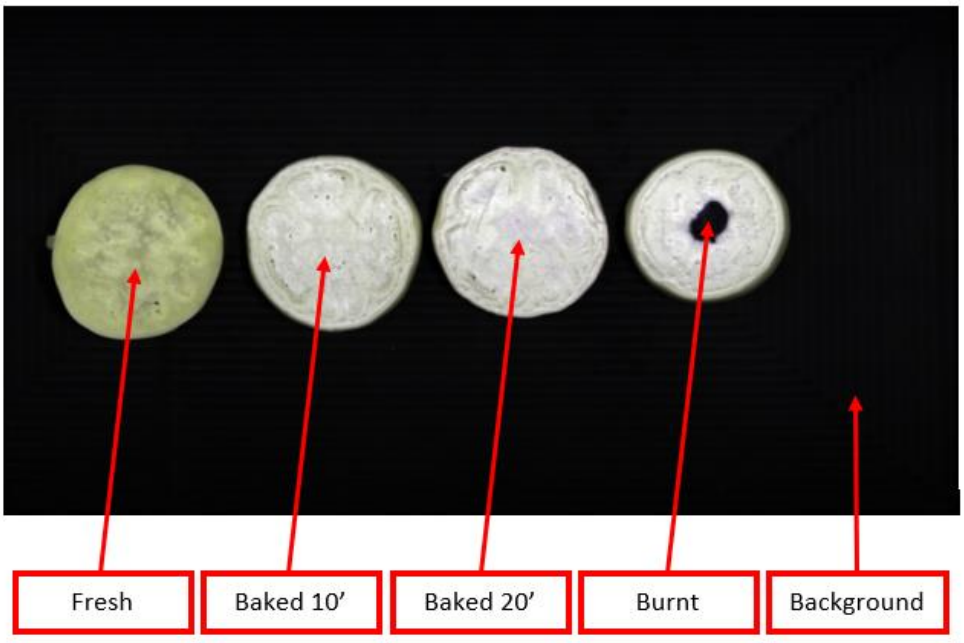


Figure 13 – The false color image of the different class of the cut eggplants

2.3. Software Setup

Since the program creation has a big role in this work, this subchapter aims to provide the reader the knowledge about all the required tools to successfully run and test or replicate the code presented in this work. In the Input Dataset sub-chapter there are an explanation of the dataset used to run the code, it is also explained how to obtain the dataset starting from the most common tools used to acquire hyperspectral data. Later, in Framework and Libraries there is a brief overview of the Python libraries which helped this work the most, introducing their strengths and their specific function to the code. It is important to state that all the programs used in this master thesis are open-source and easy to access from any Python environment.

2.3.1 Input Dataset

By the nature of Convolutional Neural Networks (CNN), there is not a fixed architecture for a specific task. CNN are great for image classification, voice recognition, and in general complex feature selection. However, the same tasks might be also achieved by simpler Fully Connected (FC) sequential layers, if the number of variables is relatively small. For what has just stated, it is important to focus on the heterogeneity of the problems that might be solved with the Neural Network (NN). The hardest challenges in applying NN to real problems is the dataset adaptation. The challenge is to transform real world data into a series of numeric values which can be easily indexed and used by the model.

To train the model described in this work it is necessary to provide these inputs:

Hyperspectral data-cube: it is a 3-dimensional matrix which represents the hyperspectral picture, 2 out of the 3 dimensions are the height and the width of the picture, so the spatial resolution of the camera. The last dimension is the spectral dimension of the camera which can be identified as the depth of the picture, so how many channels are available to be studied for each pixel. In this master thesis, the Hyperspectral data-cube must have the .mat extension format.

Ground Truth: it is a 2-dimensional matrix which has the same shape, in terms of the height and width of the Hyperspectral data-cube. The value inside each element is an integer which represents the class of the pixel positioned in the respective position in the Hyperspectral data-cube. The ground truth also must have the .mat extension format.

The output of the HSI camera is a combination of files, but for the purpose of this master thesis just the raw data is necessary to build the hyperspectral data-cube.

Unlike the hyperspectral data-cube, which is mostly a conversion task, the ground truth creation is a whole different story. It is possible to build it manually, inserting values pixel by pixel on a personal knowledge base, or adopting unsupervised learning techniques or, developing image segmentation algorithms (for this last purpose usually hyperspectral cameras also provide a false color image). Since the scope of this work is not to tackle unsupervised learning technique to create a precise dataset with the smallest possible amount of lost information, therefore a simple program is developed to label part of the picture simultaneously. The program created using Python based on OpenCV and Numpy libraries which allows the user to select regions from the false color images provided by the camera. A selected region should contain only pixels from the same class, the tool then associates these pixels with the corresponding spectra and in parallel create a labelled ground truth. The user then chooses for each selected region which class it corresponds to. This method is wasting a lot of useful data, but it is very fast and easy to apply. It is fundamental to create the most possible precise ground truth, otherwise each pixel that wrongly classified in the ground truth will compromises the training results. Ground truth is in fact the reference that the model uses to state whether its predictions are good or not, and based on that, to update the weights consequently.

2.3.2 Framework and Libraries

The code presented in this work is completely written on Python mainly because of its readability and intuitiveness, it is open source as well as equip with many available libraries and very helpful to tackle in a proper way almost every deep learning task. It is important to say that there are libraries specifically built to develop deep learning models and applications. The list of the most important libraries adopted to make this code possible is presented down below:

NumPy: It is one of the most powerful open-source Python libraries, commonly used in the industry for array computing. It can be utilized to perform many mathematical operations on arrays such as trigonometric, statistical, and algebraic routines. Therefore, the library contains many mathematical, algebraic, and transformation functions. It also allows random methodologies. In the program of the master thesis, NumPy is vastly used to perform preprocessing routines to adapt the dataset to the model.

PyTorch: Significant part of its codebase from the Torch7 project started in 2007 [18], one of the keys to its success is that it allows to write the native looking Python code and get all the benefits of a DL framework like auto-differentiation and built-in optimization. It is an open-source Python library which derives a significant part of its codebase from the Torch7 project started in 2007 [18], one of the keys to its success is that it allows to write native looking Python code and still get all the benefits of a good framework like auto-differentiation and built-in optimization. Moreover, it works with arrays called Tensor, an object which share most of the advantages of NumPy arrays but built to harness the astonishing computational power of the GPU. In the code presented in this work, PyTorch is used to manage and index the dataset and to create the model architectures. In addition, it is also used to tackle all the hidden but essential operations which characterize the training process such as: Forward Propagation, Loss Function, Optimizer, and Back Propagation.

Matplotlib: It is an open-source Python library built to visualize 2-d plots of arrays. Its greatest strength is that it allows visualization of huge amounts of data in an easily readable graph, helped by the fact that it is also built on NumPy arrays. In

this work, Matplotlib has been essential to plot and visualize almost any graph shown such as training and validation loss, accuracy, sample's spectrum, and the semantic segmentation results.

Scikit-Learn: It is another Python open-source library that is a simple and efficient tool for predictive data analysis. It is built on top of NumPy, SciPy, and Matplotlib. Its strength comes by the fact that it is very intuitive and reusable in various contexts. In the code presented in this work, Scikit-Learn is used to provides and visualize an effective confusion matrix.

3. Methods

The Chapter 3 describes all the decisions and techniques adopted to carry on this master theses. The three main sections are contained in this chapter: Metric Parameters, Dataset Preprocessing, and Deep Learning program. Within Metric Parameters are described all the tools and parameters used to understand the results of this work. The Data Preprocessing talks about the management of a raw data and moreover it explains the logic behind the dataset used to train the NNs. Deep Learning program is a section more focused on the NNs Python code and the description of the four different NNs architectures used for this work.

3.1. Metric parameters

To evaluate the performance of the program, several metric parameters are introduced. The first parameter is cross entropy loss that is used to evaluate the loss between the output with the prediction labels. The second one is the Confusion Matrix, which is used to evaluate the prediction performance on a test set. Moreover, considering the limited amount of dataset, cross validation is used to evaluate the generalization performance of the model.

3.1.1 Cross Entropy Loss

Cross Entropy Loss measures the performance of the classification model with the class probability value and the true label as the input. It is the distance between the probability of the output, with the actual label of that output. So, if the distance is small, then it means the model classify the output well. Moreover, the equation of Cross Entropy Loss can be written as the equation below:

$$H(p,q) = -\sum_{\forall x} p(x)\log\left(q(x)\right)$$

The cross-entropy formula takes in two distributions which are $p(x)$, the true distribution, and $q(x)$, the estimated distribution, both define over the discrete variable x. As the equation only receive probabilities, the Softmax function is applied to change the value of the last layer of the NNs into probability.

In this master thesis, Cross Entropy Loss is implemented using PyTorch class with the name CrossEntropyLoss(). The Softmax function is already embedded in this function. Small training loss is a good sign for the model parameter. However, if the loss of the training and the validation start to diverge, then it is a sign that the model works well only in the training data, but not in the validation data. It is called overfitting and may results bad prediction in the testing dataset. Thus, the training should be stop when this condition occurs.

3.1.2 Confusion Matrix

As briefly explained in the previous sub-chapter, Confusion Matrix is used to evaluate the performance of the trained model in the test dataset. It is a compact way to visualize how good the model predicts the labels of the dataset. Basic confusion matrix features can be seen in the picture below.

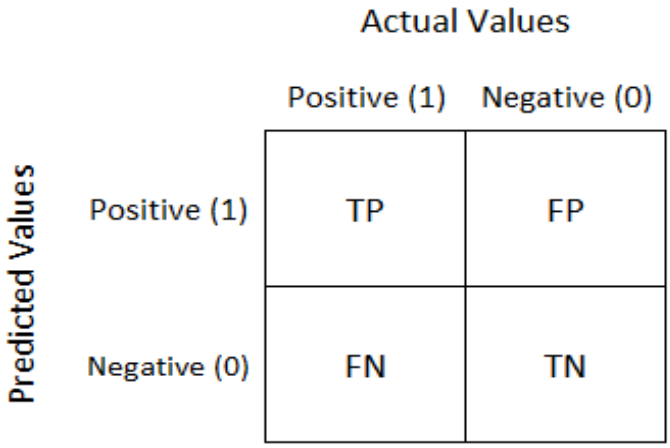


Figure 14 – Example of Confusion Matrix with two class labels

There are four group that represent different combination of predicted and actual labels. The first group is True Positive (TP) it means that the algorithm predicted positive value and it is the same as the actual label. The second one is False Positive (FP), it means that the algorithm predicted positive value, but, it is negative. The third one is False Negative (FN), it means that the algorithm predicted negative value, but, it is positive. The last one is True Negative (TN), it means that the algorithm predicted negative value, and it is the same as the actual label. Each of this group is useful for measuring Recall, Precision, Specificity, and Accuracy.

Recall is the parameter to show how much, out of all the positive cases, the algorithm predicted correctly. It should be as high as possible. Recall can be written with the equation below:

$$Recall = \frac{TP}{TP + FN}$$

On the other hand, Precision defined as, out of all positive case the algorithm has predicted, how many are positive. The same, it should be as high as possible.

$$Precision = \frac{TP}{TP + FP}$$

Because both of parameter shows how good the algorithm predicts, it usually combined to form new parameter called F-score. It is a way to measure Recall and Precision at the same time.

$$F - measure = \frac{2 * Recall * Precision}{Recall + Precision}$$

3.1.3 K-fold Cross-Validation

Generalization is a problem faced by all machine learning model. It is difficult to know a priori how good our model predicts the data outside the training dataset. This problem is worsened by the fact that in this master thesis the amount of the provided data is limited. In this scenario, K-Fold Cross-Validation is a good method

to perform to evaluate the generalization performance of the model with limited amount of data.

First, all the datasets are divided into 6 groups. Thus, this process called 6-Fold Cross-Validation with K value is 6. In the first fold, the training used the nine group of the dataset, then the test conducted to the remaining one group. It is repeated 6 times until the testing process conducted in all the group. It means that one group become training data nine times, and become test data one time, from sixfold processes.

This master thesis conducted the cross-validation process manually by hard code the algorithm using python. It is because the available library that support cross-validation does not work with the PyTorch. Figure 15 is the flow diagram of the cross-validation developed for this master thesis.

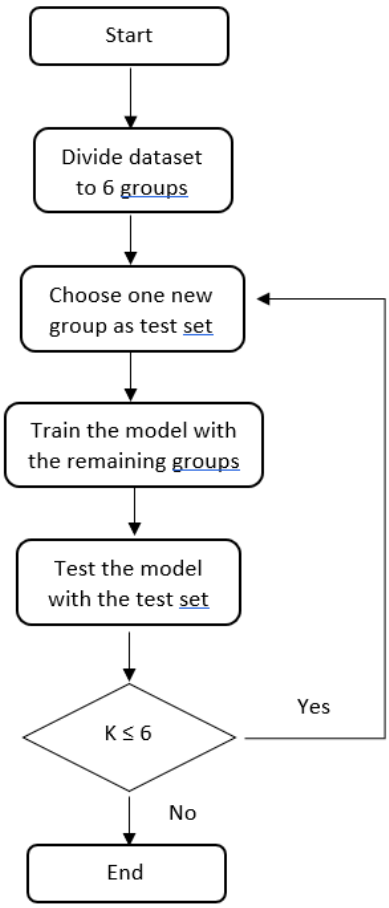


Figure 15 - Flow diagram of 6-Fold Cross-Validation

3.2. Dataset Pre-processing

Before conducting training and testing to the dataset, it is important to make sure that the available datasets is compatible and ready to use. It is because usually the raw dataset from data acquisition is not in the same format with the algorithm requirements, unbalanced, and full of noise. Thus, dataset pre-processing is paramount important for the successful machine learning problem. In the following sub-chapters several techniques are applied to the dataset to make sure that it will results with good model for the prediction.

3.2.1 Dark Current Calibration

Typically, when Hyperspectral Camera is acquired data, there is always electronic current flowing in the detector arrays even without light shining on it [2]. This current is called the dark current, and it is generated from thermally induced electron hole pairs. Thus, it is important to calibrate the output of the camera, so the effect of the dark current is minimized. The calibration is done by normalized the acquired value, with the white reference value and dark reference value.

In this master thesis, the white reference value is resulted from the acquisition of a white reference bar that put-on top of the conveyor belt. Meanwhile, the dark reference value is resulted by the camera when acquires the data with the lens closed (i.e., shutter mode). Figure 7 shows the setting of the data acquisition, with the white bar as the white reference and the cut eggplants is one of the experiments performed. It can be seen also that the LabScanner has six halogen lamp that are put on top of the samples.

The classification conducted in this master thesis is applied for each pixel of the data-cube, usually called semantic segmentation. Thus, one sample in the dataset corresponding to one pixel with all its spectrum. The range of value of the features corresponding to the intensity range of all the spectrum band. To apply the dark current calibration to the dataset, the following equation is applied to each pixel.

$$I = \frac{I - I_{dark}}{I_{white} - I_{dark}}$$

The dark current calibration is easily applied using Numpy. It has *broadcasting* property that makes the manipulation of array very easy. It is also lightweight and efficient for machine learning problem.

3.2.2 Balancing Dataset

Balance dataset is important to give equal priority to each class in the training. This is true especially in a normal condition where all the number of the sample of each class is naturally balance in nature. In this master thesis, there are clearly imbalance number of samples for each class. This can be seen from the **Error! Reference source not found.**.

To minimize the effect of imbalance dataset, a data pre-processing is conducted. It is start by calculating total number of samples in each class. Then, use the class that has the least number of samples, multiply it by 300%, and use it as the maximum number of samples permissible for training for each other classes. With this algorithm, the number of samples in each class is relatively balance, and the model will process each class equally.

Class	Samples per class	Sample per class (balanced)
0 – Background	19251	2697
1 – Fresh	12075	2697
2 – Baked (10 min)	10098	2697
3 – Baked (20 min)	9964	2697
4 – Burnt	899	899
5 – Skin	13716	2697

Table 1 - Total number of samples in each class and total number of samples in balanced dataset

3.2.3 Spectrum band selection

Figure 17 shows the selected dataset spectrum. The complete method for dataset selection will be explained in the next sub chapter. In the beginning and at the end of the spectrum, there are noise due to sensor limitations. Thus, it is advisable to remove the noise to reduce the computational load.

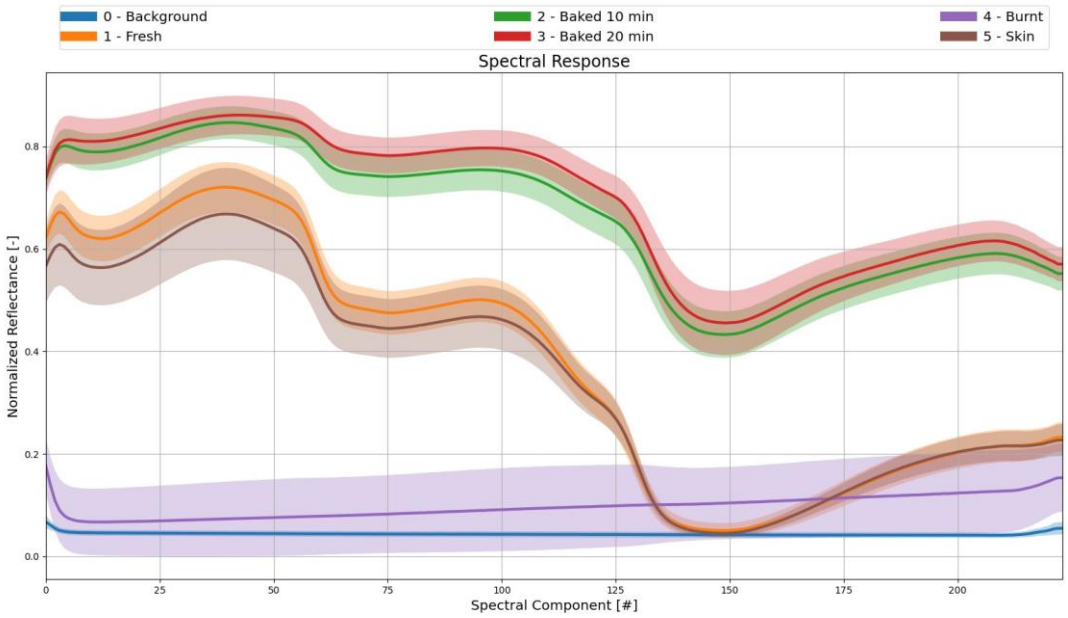


Figure 17 – Spectrum of the selected dataset with the noise

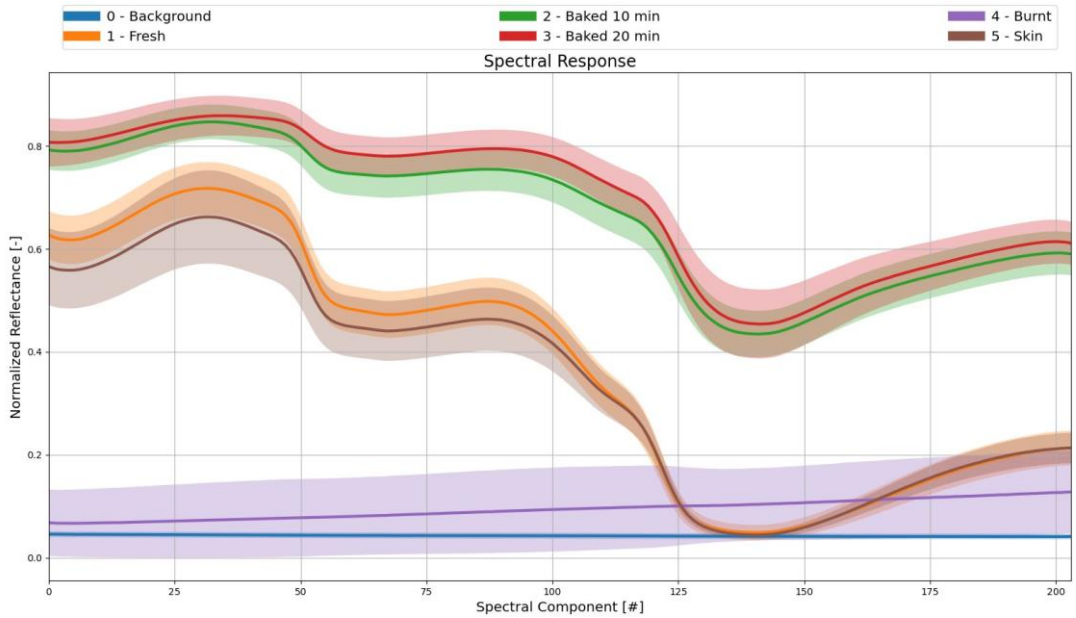


Figure 16– Spectrum of the selected dataset after removing the noise

Figure 16 shows the spectrum without the noise. As with the water absorption spectrum band, it is still used for the training because there is different offset that useful for the classifier.

3.3. Program

Most of the code developed for this master thesis is done using PyTorch and Numpy. Part of Exploration Data Analysis (EDA) is done with Numpy, as well as the data preprocessing. Moreover, part of the training, validation, and testing of the model are done with PyTorch.

To tune the model, and easily troubleshoot some problem arises in the code development, a determinism setting is applied to the program. This is done by setting the seed of the internal python random number generator, so that all the pseudo random process included in the program will be fix anytime the code is running.

3.3.1 Region of Interest Selector

In this master thesis, the dataset is manually acquired using HSI system already explained. Thus, there are no labels assigned on each class, and this task must be done manually.

A program is created to fulfill this function. First, from the spatial image of the sample data-cube, a Region of Interest (ROI) is selected using a square bracket as can be seen from Figure 22. Then the program will investigate the matrix coordinate of the selected region and will take all the spectrum of the pixel inside selected region from the data-cube. After that, a function to assign the class number is invoked, and the user need to fill in it as shown in Figure 19. This process needs to be done several times until all the needed dataset class is created.

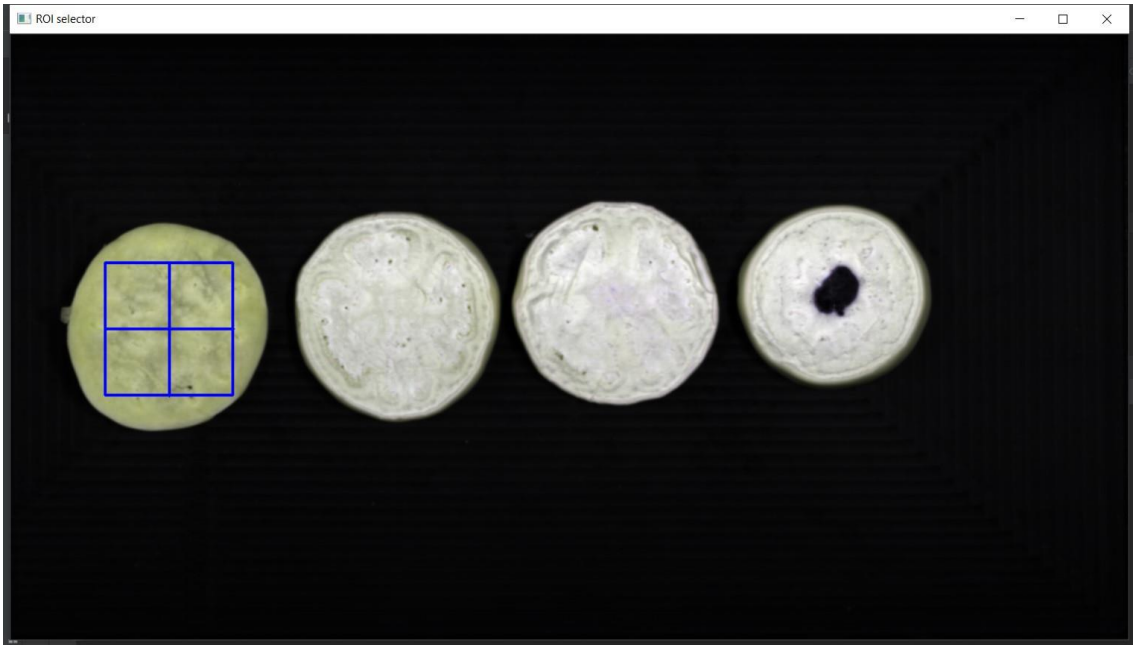


Figure 18 – Choosing Region of Interest using a blue box

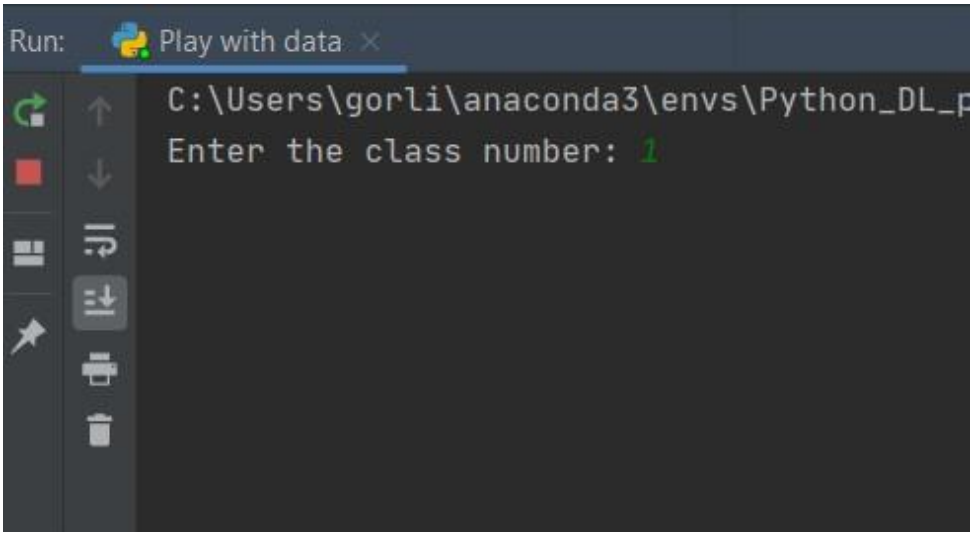


Figure 19 – Assigning the class number to the samples selected using the blue box

3.3.2 Weight initialization

The weight of the model can be start from any value. But the deep learning has difficulties in converging when the weights are initialized using normal distribution

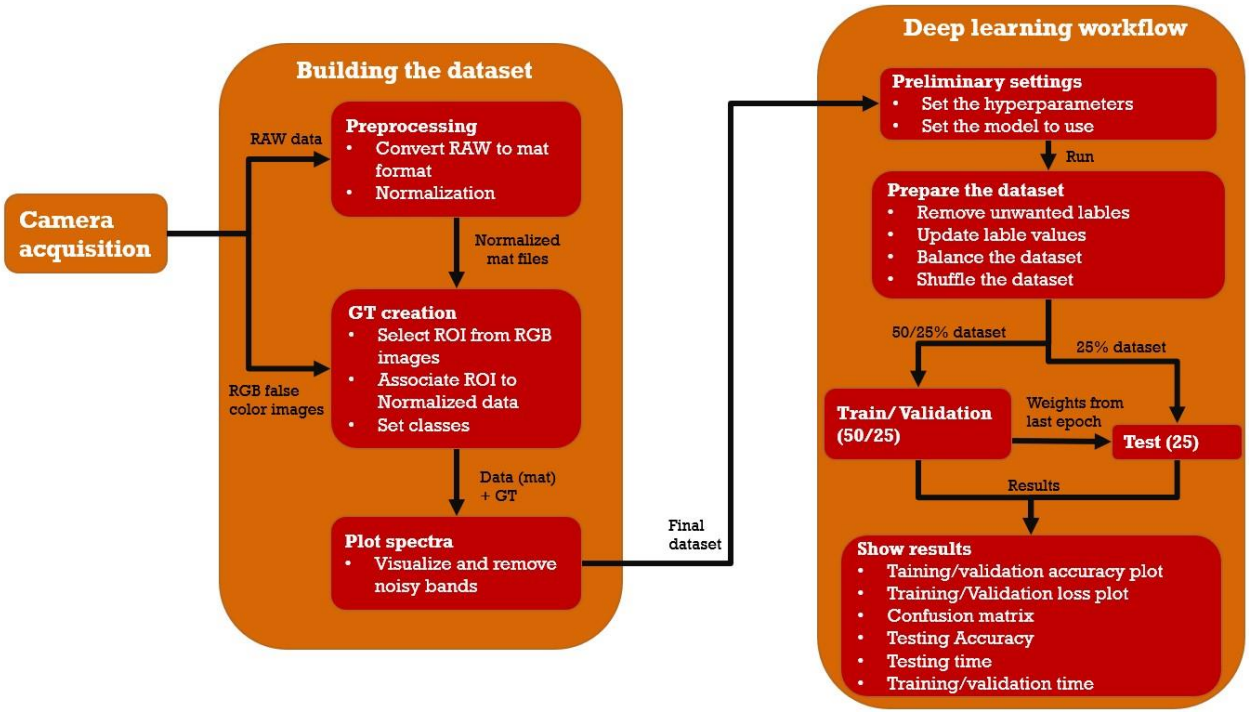


Figure 20 – Data learning workflow

with fixed standard deviation. Fortunately, it is possible to start the weight from value with the help of weight initialization function. This master thesis chooses Kaiming algorithm to initialize the weight.

Kaiming algorithm is used when the ReLU is the activation function. In this master thesis, the activation function used is only the ReLU function. The equation of the Kaiming Initialization can be seen in below:

$$W = \mathrm{N}\left(0, \frac{2}{n^l}\right)$$

3.3.3 Deep Learning Workflow

The whole workflow for the developed pre-processing and deep learning programs can be summarized in Figure 20.

As it is already described in the previous chapters, it all begins with data acquisition through Specim FX17 camera. The outputs that need to be saved for the next processes are RAW hyperspectral image, RAW dark reference, RAW white reference, and the RGB false color image. The first three files are converted from

RAW format to the mat format thanks to a MATLAB program which is not described in this work. After that, the same three files are taken as input by the Python program normalization_tool which applied a normalization and resulted a single normalized mat file. This output together with the false color image are the new inputs to the Python program called play_with_data. This program allows the user to create the Ground Truth (GT) by selecting the Regions of Interest (ROI) from the false color image. Then the ROI coordinates are used to extract the respective hyperspectral normalized dataset from the mat file. Finally, the user writes the corresponding class for each ROI. The dataset is now saved in two distinct mat files, one contains the normalized data as a list of spectra, the other contains the GT as a list of integers, which represents the classes of the spectra positioned on the same index coordinate. Since Specim FX 17 sensor has a lot of noise in the first and last bands of its spectrum, it is important to plot the results to visualize which bands are better to be removed to obtain better performances during the training, this part is possible due to the Python program called spectra_plotter.

Now the dataset is ready to be used to train the weights of the DL model. Two Python programs are developed for that purpose, tvt_tool and kfold_cv_tool, the last one uses cross validation while the first one follows the train-validation-test pattern. The following paragraphs describes more in depth of the tvt_tool, and the last paragraph will concentrate on the differences with kfold_cv_tool.

The user must set the classical hyperparameters such as the number of epochs, learning rate, and batch size and others which are not so common such as the maximum number of samples per class that is created specifically for this master thesis. Moreover, if during the visualization part with spectra_plotter file, it is necessary to remove bands, then there is section of the program inside the file to remove them before training the model. If there are classes that are not consider in the training, the user must type them in the related section of the program. Then, the user must choose whether to save the weights at the end of the training, activate the early stop algorithm, and apply a dynamically decreasing learning rate during training. In addition to that the user also must choose a model between these four: Lucas NNN, Hu et al.[10], 2D CNN and FC NN.

After the program is run, the tvt_tool program starts adjusting the dataset, in fact imagine that the total number of labels is n and 2 unwanted labels have been

selected to be removed, and since the classes must be in range from 0 to n with no gaps in between 0 and n, the algorithm should update/shift the value of the labels to always have increasing values from 0 to n – 2. The program then does not simply take every sample inside the dataset to train it, there is high probability to take an unbalanced sample by doing that, so it looks at all the classes and sees the class which has the least number of samples, then it multiplies this number by three and that is the maximum number of samples per class adopted for all the classes in the dataset. It is also important to note that all the samples from each class are chosen randomly. In addition to that, the program shuffles all the samples before using them for the training.

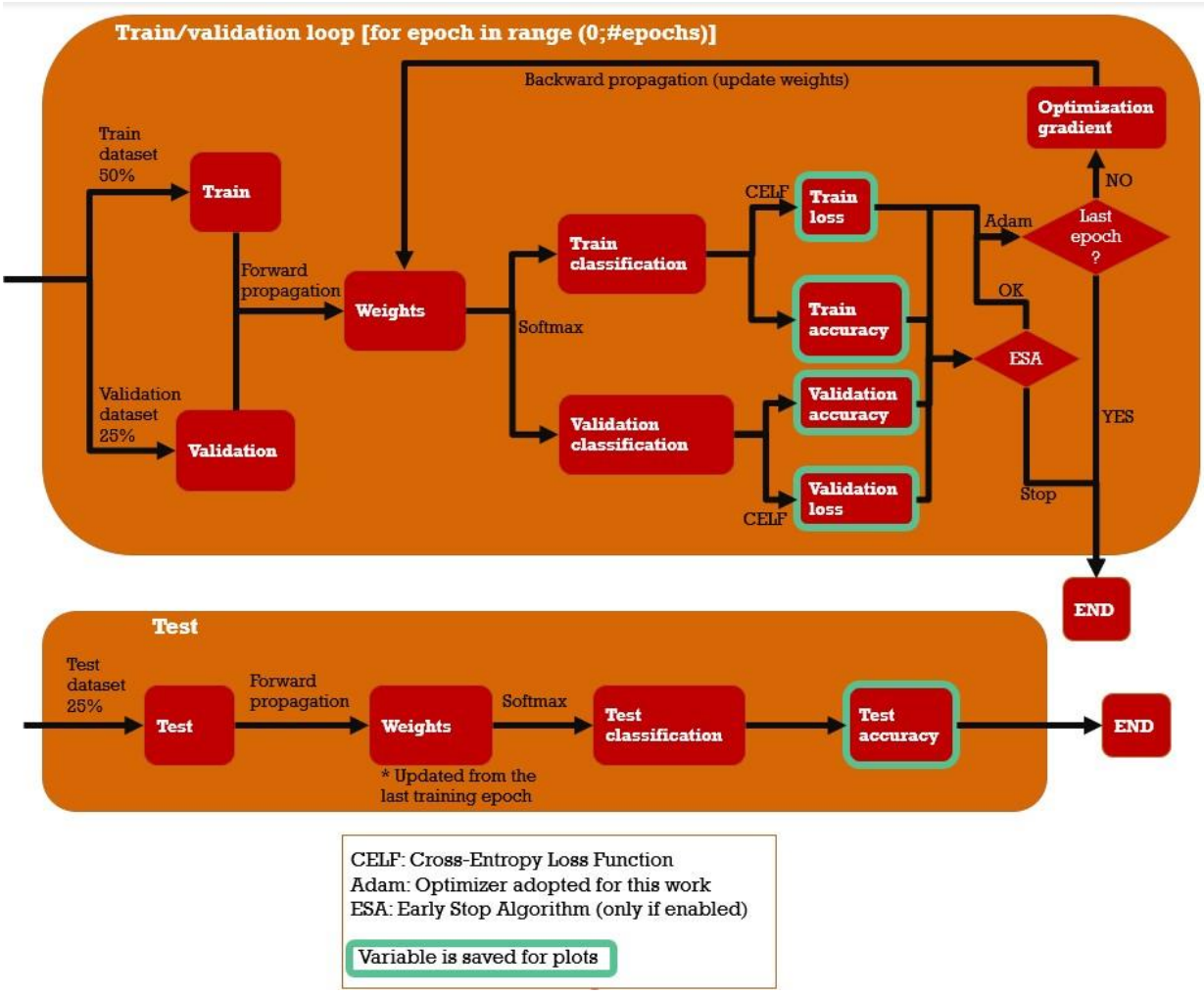


Figure 21 - Train/validation/test workflow

To properly index each sample with the corresponding label, PyTorch offers a specific object called Dataset to implement it. After the dataset is indexed inside the Dataset object, it is necessary to input it to the Dataloader object that splits the whole dataset in train, validation, and test set in accordance with the proportion set by the user at the beginning of the program. Two loops are designed then, one is for training and validation, and the other is for testing, as can be seen from Figure 21. Through the forward propagation, the input data pass through the network weights, then a softmax function evaluates the most probable solution from the outcomes. The outputs for both training and validation branch are accuracy and loss values. The function to calculate the loss is the Cross-entropy loss function. Next step is to calculate the outputs of the train branch that through the SGD optimizer and

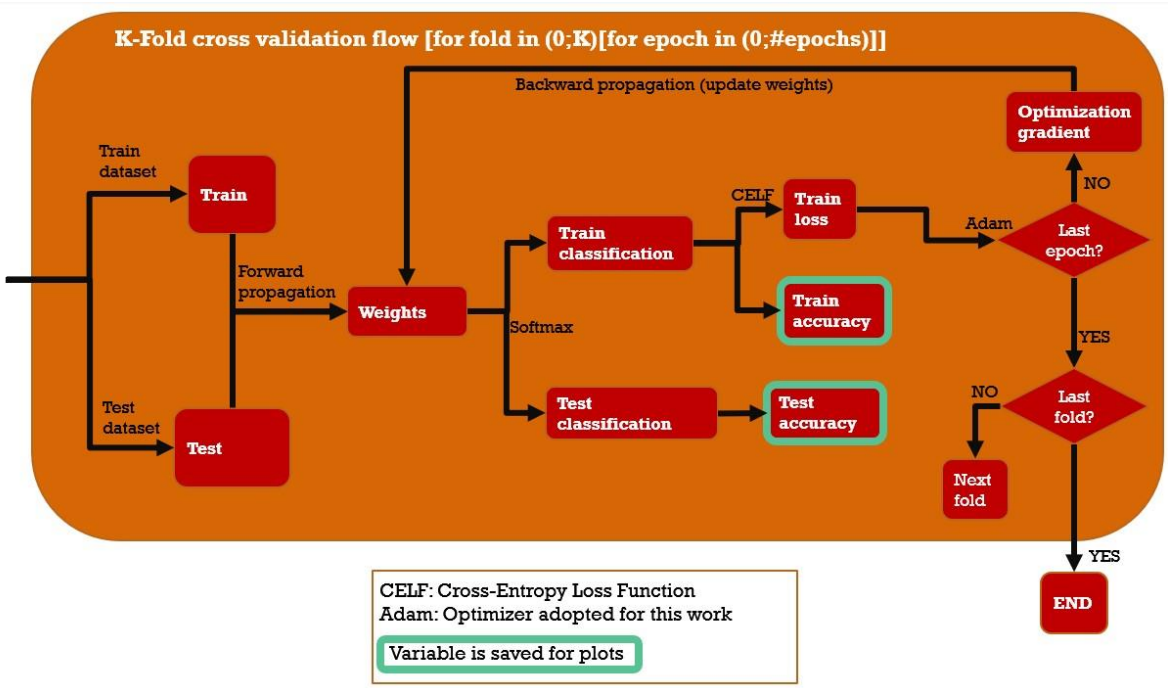


Figure 22 - K-Fold Cross Validation workflow

accordingly calculate the gradient which allows the classification to be closer to the solution. In this process, the weights which are used for the next training and validation loop, are updated (only if the epoch is not the last one or the early stop algorithm stops the process). The weights resulting from the last epoch in the train or validation process is used for the test part. Finally, the test dataset is simply fed to these weights and, with the usual softmax function, all the classification are evaluated.

On the other hand, the Python program kfold_cv_tool uses K-fold Cross Validation technique to evaluate the reliability and robustness of the model. It shares a lot of common processes with the train and validation flow, but with the test branch instead of the validation branch as can be seen from Figure 22. Now the dataset is no more split in train-validation-test but in K number of folds that the user sets. The train-test loop is done for all the epochs necessary to finish the training, but this process is repeated K times. Note that for each fold, one group is used for the test, and the remaining are grouped for the training.

3.3.4 Models

In this section are described the model used for this semantic segmentation work: Lucas NNN, Hu et al 1D CNN, FC NN, and 2D CNN. The first two model are already applied to classify hyperspectral data coming from remote sensing images taken with drones, and the remaining two are developed for this master thesis. Since this work aims to study hyperspectral data pixel by pixel, all the models do not consider spatial feature correlations. In the following subsections, there are more detailed descriptions.

Lucas NNN

Lucas NNN (in the original paper [19] referred as LucasCNN) is a neural network developed in 2019 by Riese and Kellers to study and classify the freely available *Land Use/Cover Area Frame Statistical Survey* (LUCAS) Soil dataset. It includes hyperspectral and soil texture data from measurements all over Europe.

The Lucas NNN consists of four convolutional layers, each followed by a ReLU activation function and max-pooling layer. After flattening the output of the fourth convolutional layer, two FC layers are implemented, again followed by a ReLU activation function, and one FC layer with a softmax activation. Finally, six outputs are placed at the end of the network.

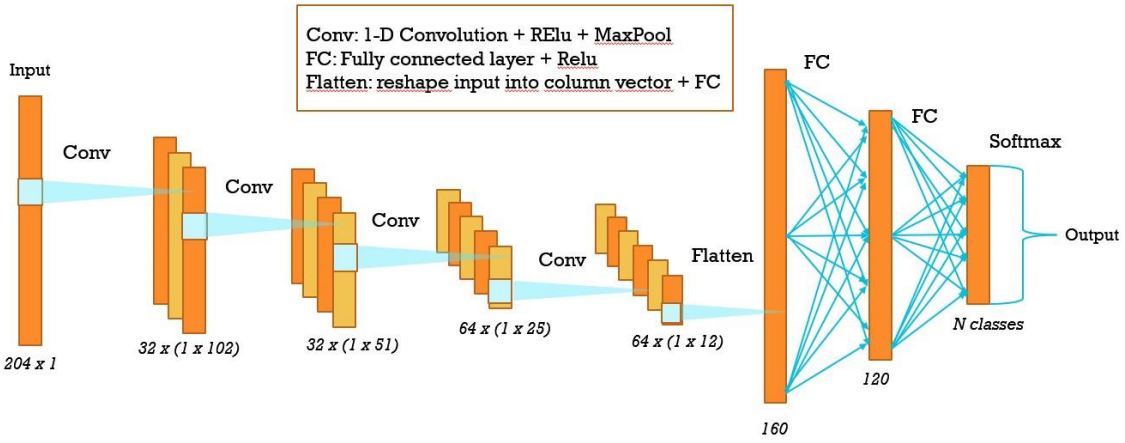


Figure 23 - LucasNNN architecture

OPERATION	INPUT	OUTPUT	KERNEL	STRIDE	PADDING
Conv (first)	1 x 204	32 x 1 x 102	3	1	1
Conv (second)	32 x 1 x 102	32 x 1 x 51	3	1	1
Conv (third)	32 x 1 x 51	64 x 1 x 25	3	1	1
Conv (fourth)	64 x 1 x 25	64 x 1 x 12	3	1	1
MaxPool	-	-	2	-	-
Flatten	768	160	-	-	-
FC (first)	160	120	-	-	-
FC (second)	120	N classes (6)	-	-	-

Table 2 - Table with the parameters related to LucasNNN architecture

Hu et al. 1D CNN

Hu et al. [10] developed a neural network architecture that is employed to classify HSI image directly in spectral domain. The algorithms originally implemented on several freely available HSI data sets including Indian Pines, Salinas, and University of Pavia.

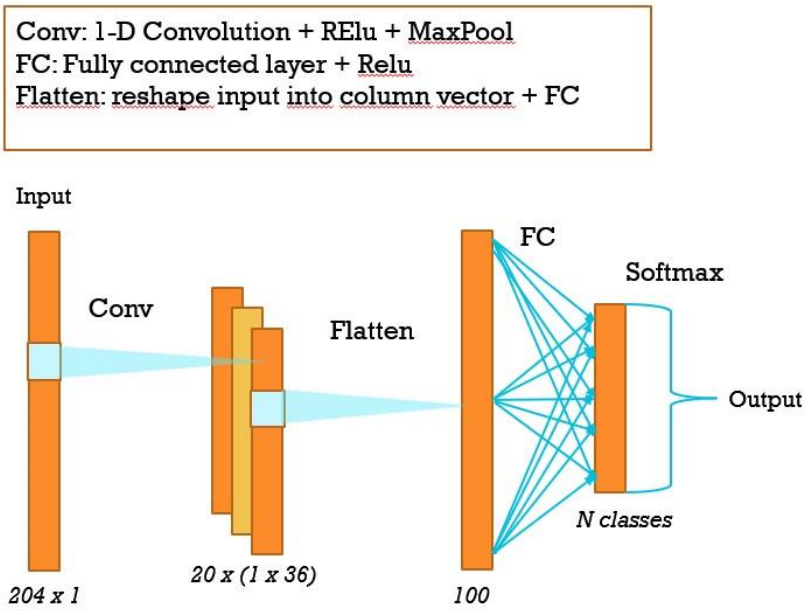


Figure 24 - Hu et al. architecture

The architecture consists of one convolutional layer, followed by a ReLU activation function and max-pooling layer. After flattening the output of the max-pooling layer, two FC layers are implemented, again followed by a ReLU activation function, and one FC layer with a softmax activation is placed at the end of the network.

OPERATION	INPUT	OUTPUT	KERNEL	STRIDE	PADDING
Conv	1 x 204	20 x 1 x 36	25	1	-
MaxPool	-	-	5	-	-
Flatten	720	100	-	-	-
FC	100	N classes (6)	-	-	-

Table 3 - Table with the parameters related to Hu et al. architecture

FC NN

This architecture is developed just to see if a simpler structure such as a sequence of three fully connected layers might compete in terms of performances and accuracy against the two more structured architectures described before. Figure 25 represents the architecture of the network, and Table 4 shows the evolution of the input and output for each layer.

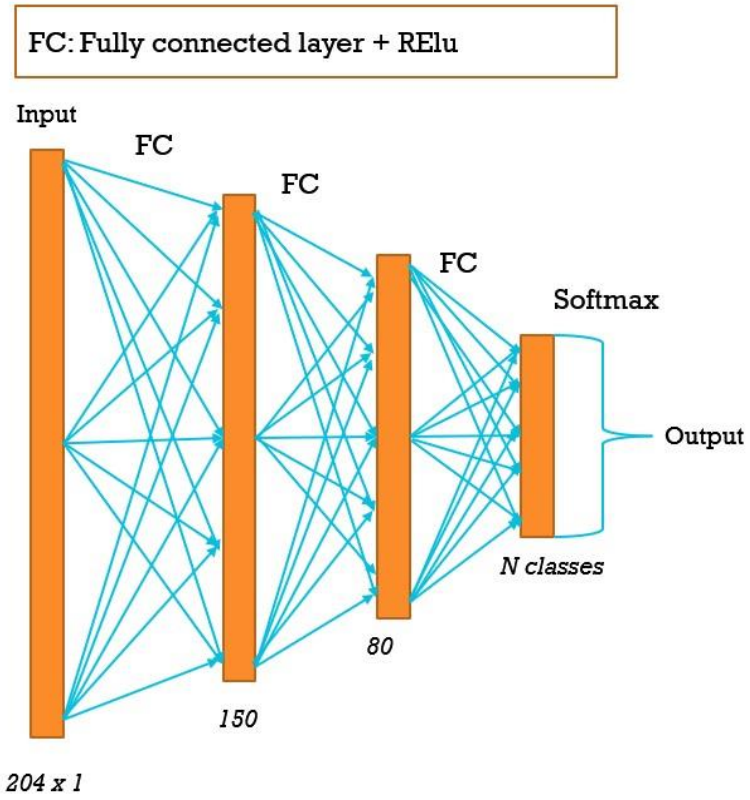


Figure 25 - FC NN architecture

OPERATION	INPUT	OUTPUT	KERNEL	STRIDE	PADDING
FC (first)	204	150	-	-	-
FC (second)	150	80	-	-	-
FC (third)	80	N classes (6)	-	-	-

Table 4 - Table with the parameters related to FC NN architecture

2D CNN

This network is very similar to classical 2D architectures used to analyze spectral-spatial correlations in 2D RGB images. It is decided to also study this architecture to see if even a different approach might be useful to classify an input dataset made by a 1D vector and not a 2D image. To build the requested 2D input, the input of 1D vector as a column is multiplied by itself as a row. Figure 26 described the architecture of the network, and Table 5 shows the evolution of the input and output for each layer in the network.

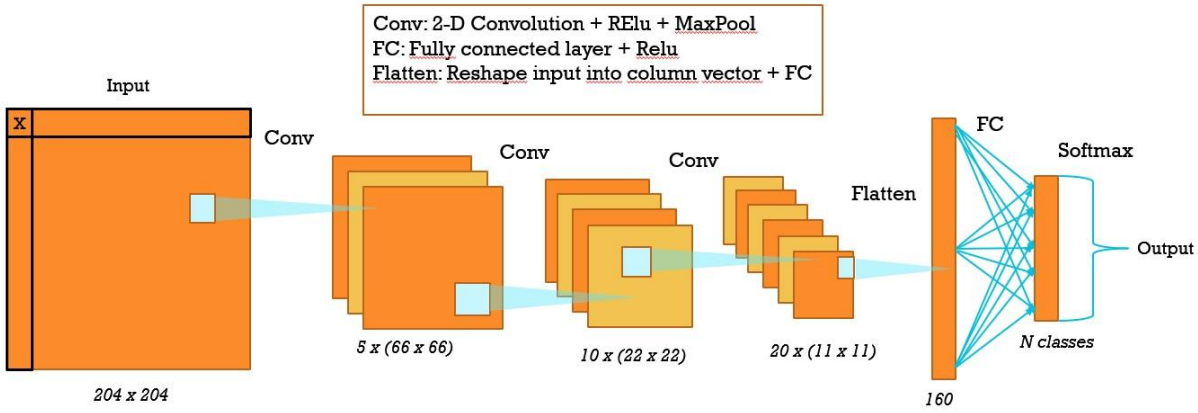


Figure 26 - 2D CNN architecture

OPERATION	INPUT	OUTPUT	KERNEL	STRIDE	PADDING
Conv (first)	1 x 204 x 204	5 x 66 x 66	7	1	-
Conv (second)	5 x 66 x 66	10 x 22 x 22	3	1	1
Conv (third)	10 x 22 x 22	20 x 11 x 11	3	1	1
MaxPool (first/second)	-	-	3	-	-
MaxPool (third)	-	-	2	-	-
Flatten	2420	160	-	-	-
FC	160	N classes (6)	-	-	-

Table 5 - Table with the parameters related to 2D CNN architecture

3.3.5 Early Stop Algorithm

The main problem in training is deciding when to stop it. This problem is classical machine learning problem that still become the main research topic around the world. Too fast to stop the training, then the model will not fully develop and will underperformed even in the training data. Too long to stop the training, then the model will overfit the training data, causing the model underperformed for the unseen testing data.

In this master thesis, simple early stop algorithm is developed to minimize the effect of overfitting. The code is manually developed using some criteria that is the assumption of well-developed model.

When training a model, it is important to have enough computational power to complete many epochs in a limited time span, on the other end it would be an error to think that the more epochs you are doing the more the model is getting better, in fact the risk of overfitting is always behind the corner. Overfitting happens when the model is updating the weights in a way that it is not generalizing the classification anymore, but it is just trying to fit all the data contained in the training dataset with the respective values. The result is a model which classify perfectly what is inside the training dataset, but completely fail the recognition of any sample outside from the training dataset.

Spotting overfitting is not easy task, but without going to much in depth, it is possible to say that whenever an increasing divergency happens between training and validation curves with respect to both accuracy and loss, overfitting is taking over. The algorithm described below aims to recognize overfitting by monitoring validation and training curves during training, whenever overfitting is detected the algorithm stops the training loop.

To tackle this problem the algorithm creates 2 lists with a defined number of elements, the elements cab be either 0 or 1, one list is related to the divergency between training and validation in the accuracy of the model, the other to the divergency between training and validation in the loss of the model. The algorithm flow is described in the Figure 27.

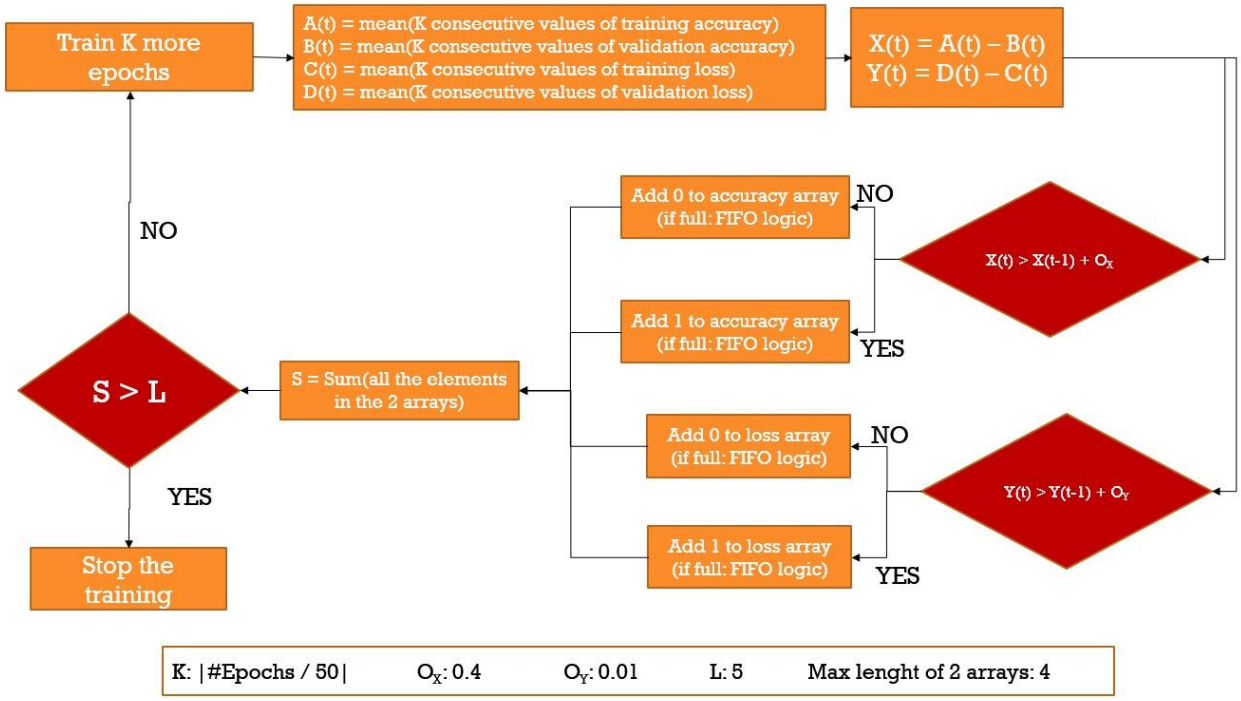


Figure 27 - Early stop algorithm flowchart

4. Results

The Chapter 4 is going to exhibit the results of the master thesis. The results of each model will be presented sequentially as Lucas NNN, 2D CNN, Hu et al, FC NN. As it was anticipated in the previous chapters the results from the models will be analyzed in two ways, first one is the classic learning pattern composed by training, validation, and test, the second one is made with k-fold cross validation. For what concerns the first pattern, the first experiments focus on the number of epochs (400, 200) and the ability of early stop algorithm to stop the training whenever overfitting occurs, the next experiments will focus on the different learning rates (0.001, 0.005, 0.0002) and how they affect the training. As a support to these experiments, there will be three different plots such as training and validation accuracy curves throughout all the epochs, training and validation loss curves throughout all the epochs, confusion matrix related to the results from the test, and other parameters like training and validation time, testing time and testing accuracy.

On the other hand, with the k-fold cross validation (k equals to 6) pattern the focus of the related experiments will be mostly on the different learning rates (0.001, 0.005, 0.0002). As a support to these experiments there will be a plot which describes the different training and testing accuracy along the different folds and other parameters like cross validation time, mean and standard deviation of testing accuracy along the different folds.

Besides these different experiments there is a common ground between all of them which is presented in Table 6.

Batch Size	Loss Function	Optimizer	Weights initialization	Processor
64	Cross Entropy	Adam Algorithm	Kaiming normal	Nvidia GeForce GTX 1080 Ti

Table 6 - Common function and parameters

4.1. LucasNNN

In this introductory section there will be described results from experiments regarding Lucas NNN architecture. A general overview about the combination of parameters used for each test outcoming from train, validation and test pattern is visible in **Error! Reference source not found.**, while in Table 8 are presented the parameters used for cross validation pattern. In the next two sections there is a more in-depth description of the results from the tables presented down below, also with references to the related plots.

Model	Epochs (E)	Learning Rate (LR)	Training, Validation time (s)	Test time (s)	Test Accuracy (%)
LucasNNN	**400**	**0.001**	**382.9**	**0.1436**	**99.11**
	200	0.001	192.5	0.1376	96
	200	0.005	192.3	0.1388	96.75
	200	0.0002	192.1	0.1466	94.22

Table 7 - Training, Validation, and Testing's parameter and results of LucasNNN model

Model	Epochs (E)	Learning Rate (LR)	Cross Validation time (s)	Average Test Accuracy	Standard deviation test Accuracy
LucasNNN	200	0.001	1810.3	98.89	0.2573
	200	**0.005**	**1810.4**	**99.09**	**0.4**
	200	0.0002	1814.4	98.21	0.4

Table 8 - Parameters adopted for cross validation pattern in LucasNNN architecture

4.1.1 Training, Validation, and Testing Results

In this section there are described only the experiments resulting from the train, validation and test pattern described in Table 7.

LucasNNN: 400 epochs, 0.001 learning rate

From the plots in Figure 28 and Figure 30 in it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting, but this was inevitable since the parameters are the same with the previous experiment, except for the number of epochs which is lower.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 29 is showing good results in terms of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

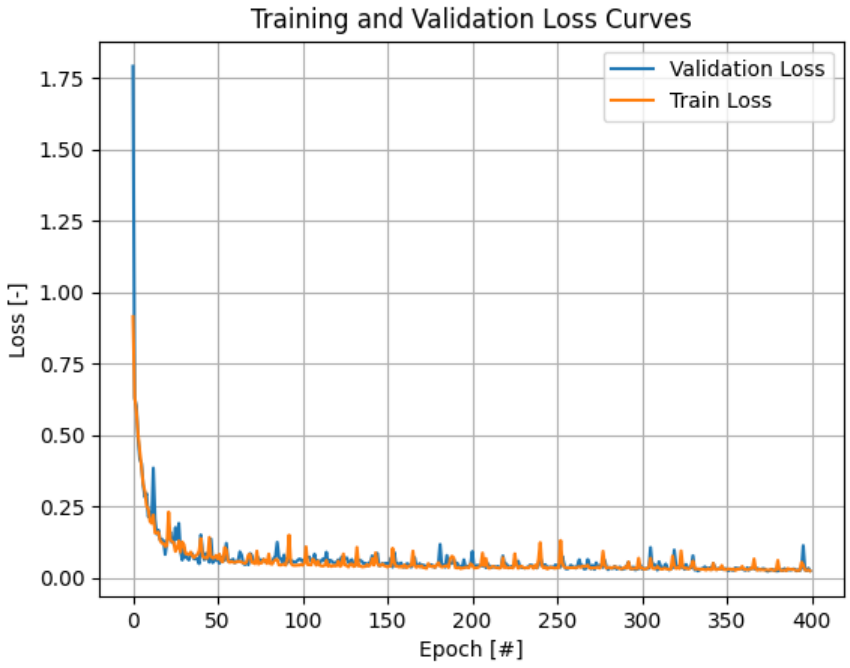


Figure 28 - The training validation loss of LucasNNN with E 400 and LR 0.001

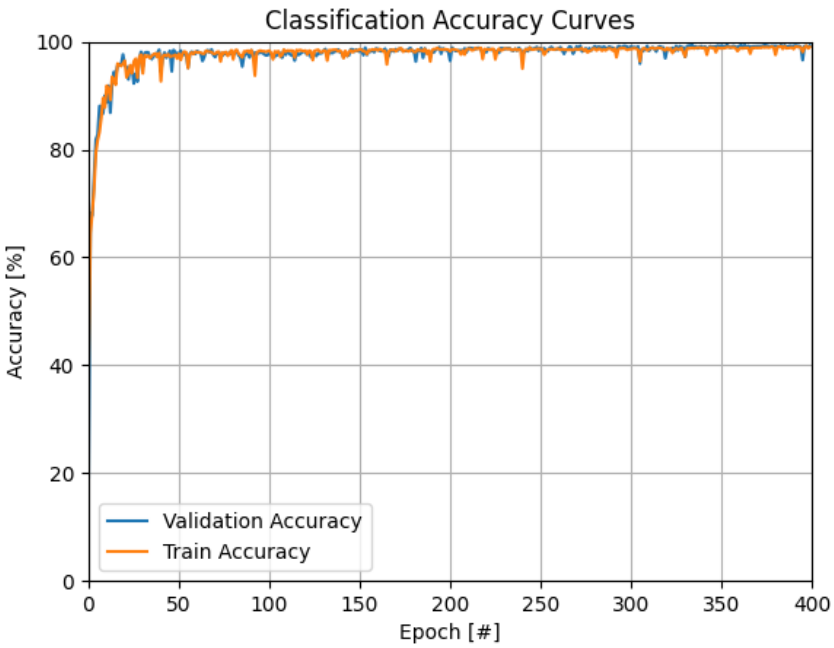


Figure 30 - The training validation accuracy of LucasNNN with E 400 and LR 0.001

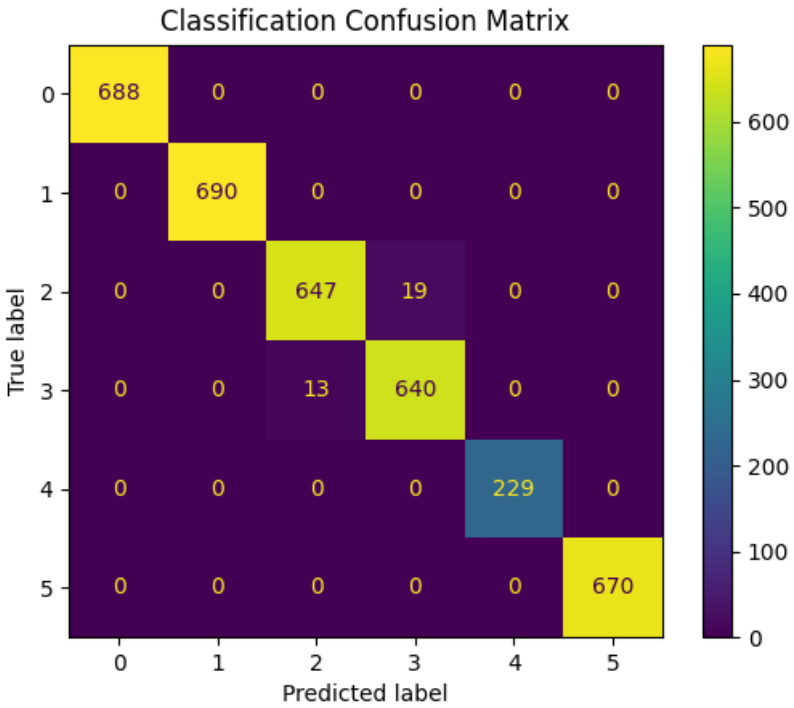


Figure 29 - Confusion Matrix of a LucasNNN with E 400 LR 0.001

LucasNNN: 200 epochs, 0.001 learning rate

From the plots in Figure 31 and Figure 33 it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting, but this was inevitable since the parameters are the same with the previous experiment, except for the number of epochs which is lower.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 32 is showing good results in terms of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

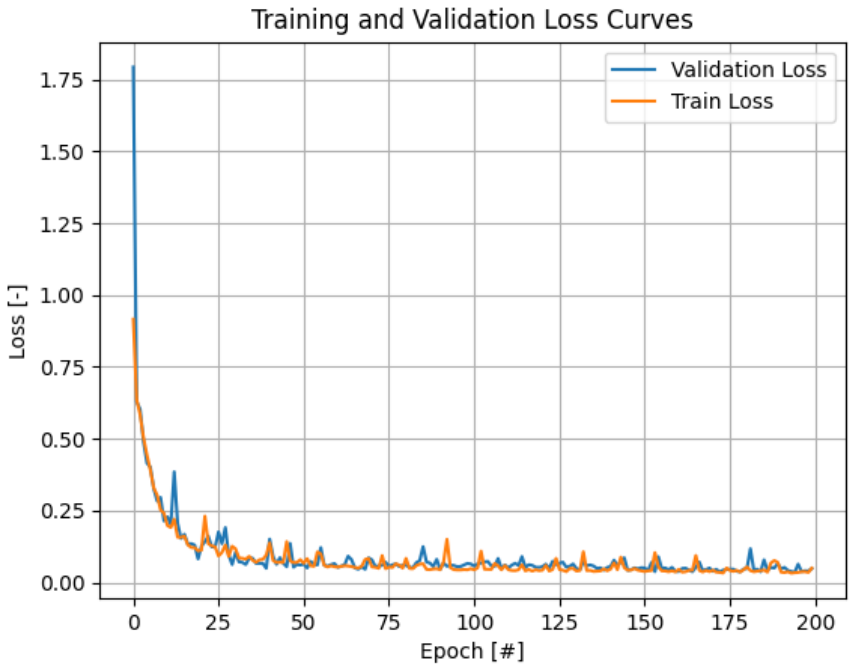


Figure 31 - The training validation loss of LucasNNN with E 200 LR 0.001

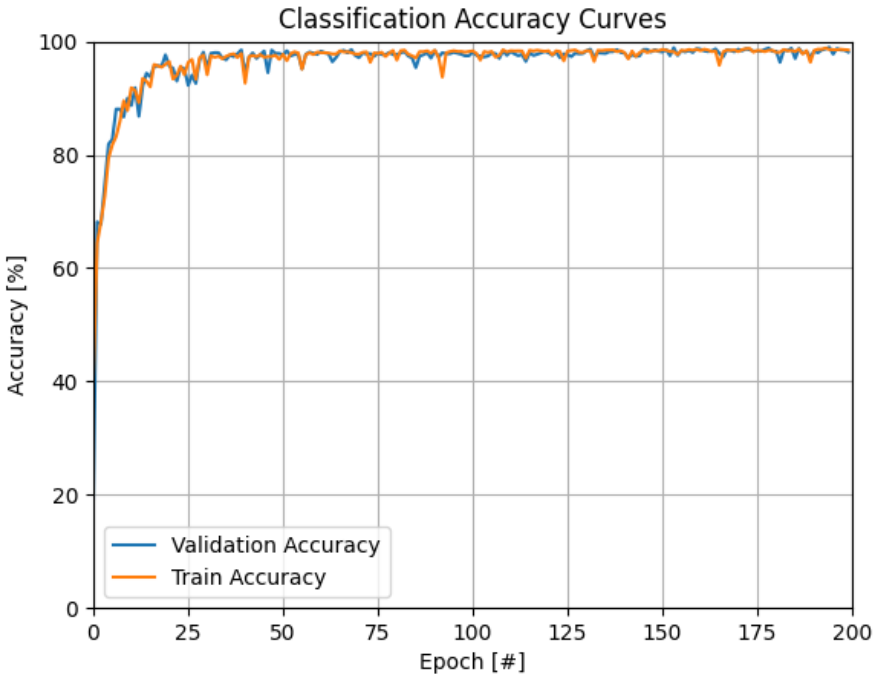


Figure 33 – The training validation accuracy of LucasNNN with E 200 LR 0.001

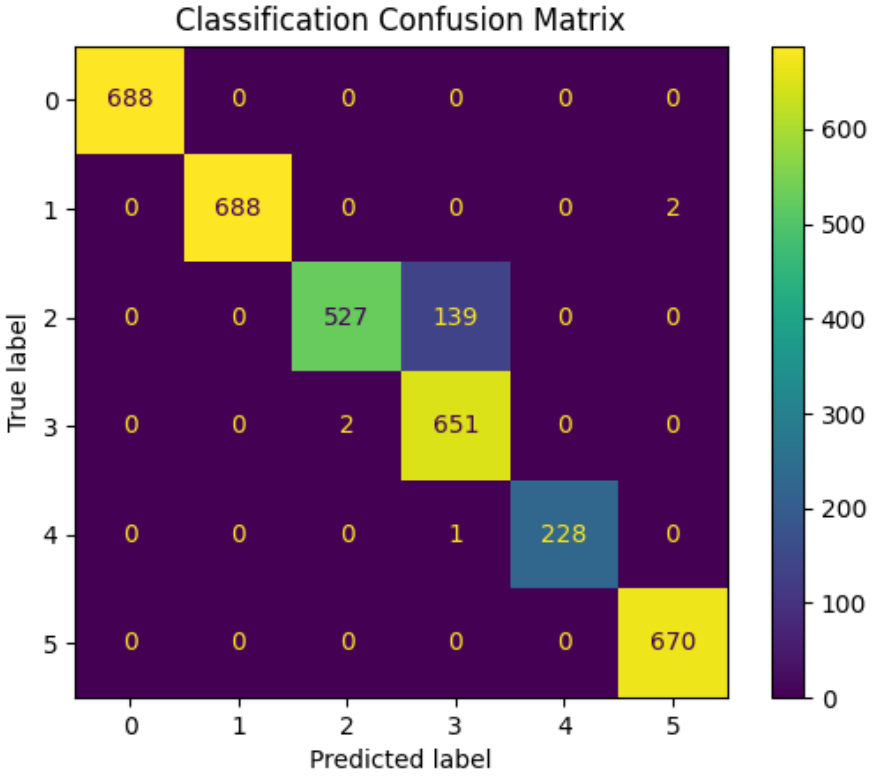


Figure 32 - Confusion Matrix of a LucasNNN with E 200 LR 0.001

LucasNNN: 200 epochs, 0.005 learning rate

From the plots in Figure 34 and Figure 36 it is possible to see that in the first 20 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, more than the 0.001 case, this is because the learning rate is higher, but still the training was evolving well. Confusion matrix in **Error! Reference source not found.** is showing amazing results in terms of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

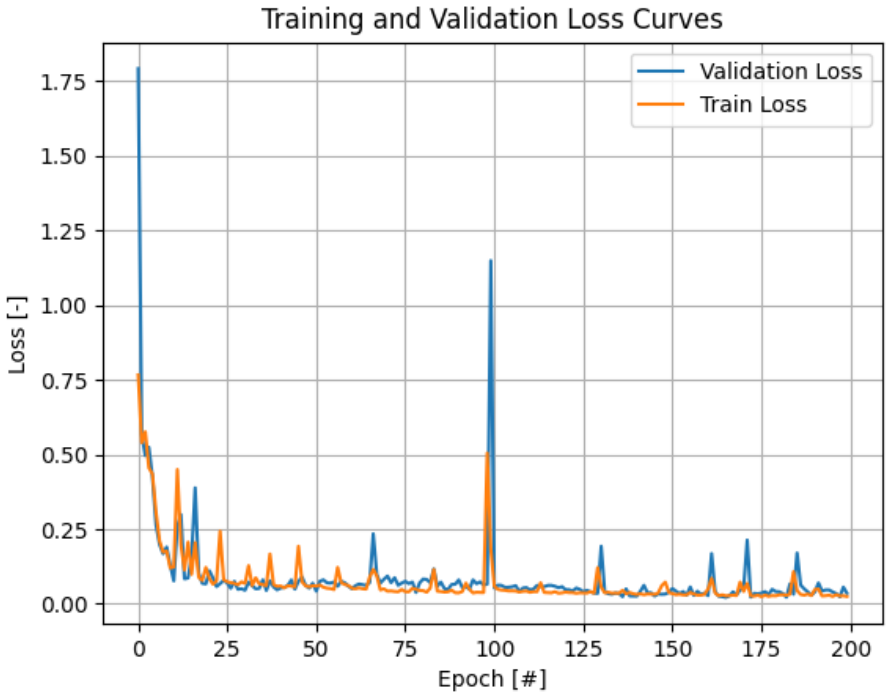


Figure 34 - The training validation loss of LucasNNN with E 200 LR 0.005

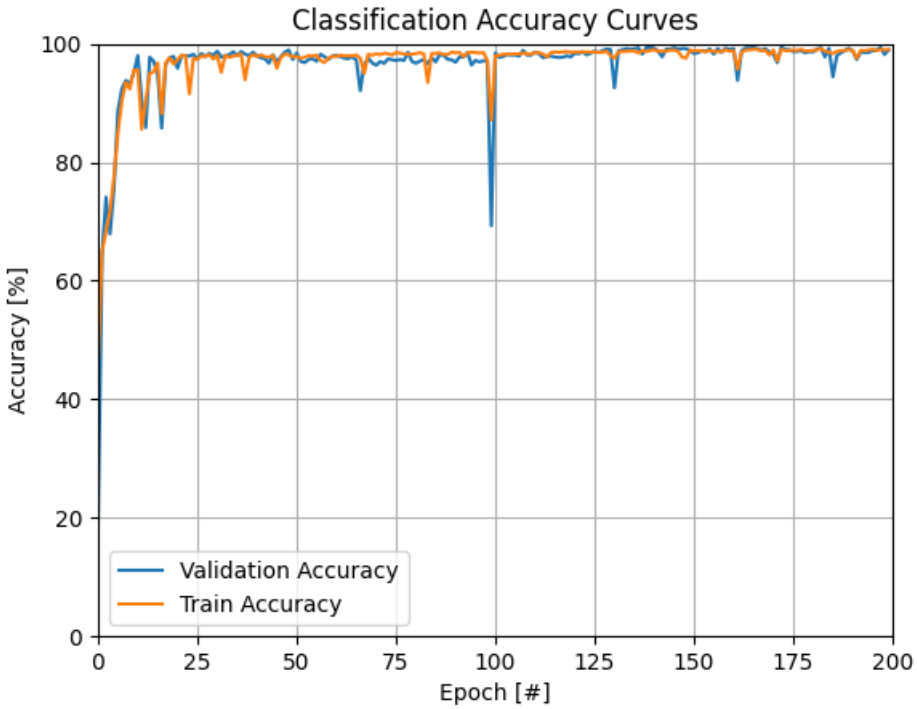


Figure 36 – The training validation accuracy of LucasNNN with E 200 LR 0.005

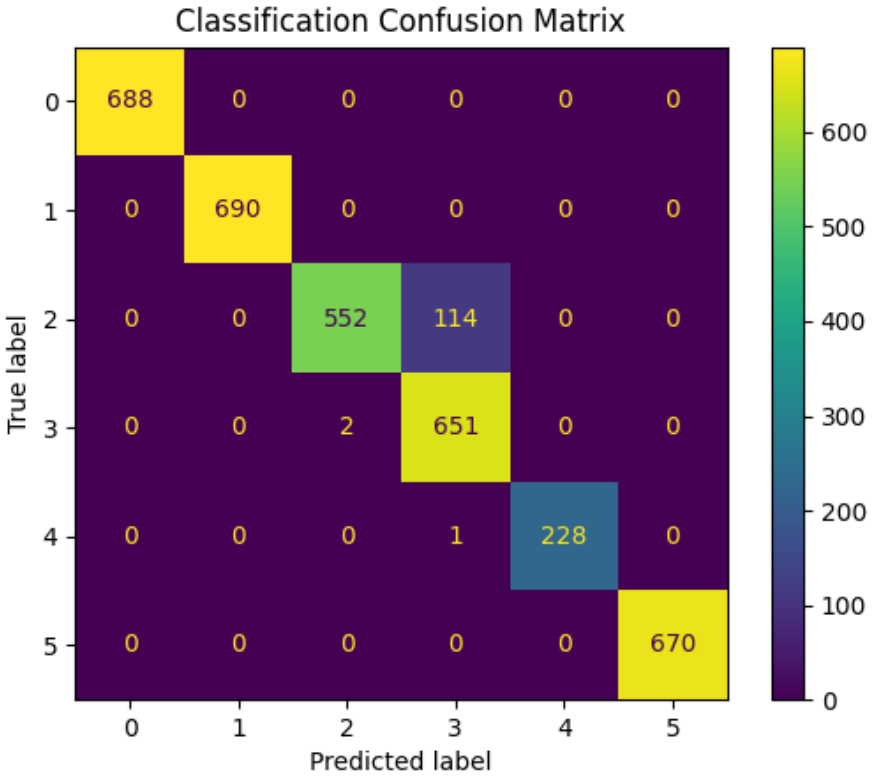


Figure 35 - Confusion Matrix of LucasNNN with E 200 LR 0.005

LucasNNN: 200 epochs, 0.0002 learning rate

From the plots in Figure 37 and Figure 39 it is possible to see that in the first 50 epochs the curves have almost reached their steady state values, this number is higher as expected from the previous experiments, in fact a lower learning rate means smaller steps for the training, so lower learning speed.

In accuracy plot there is a slight sign of divergency, but nothing to be worried about, the model is still generalizing and training well avoiding any overfitting.

In general, during the evolution of the curves in the two plots the smooth behavior is associated with the lower learning rate, in fact moving in training with smaller steps means also to always move from one relative minimum gradually to another, without rough jumps. Confusion matrix in Figure 38 is showing good results in terms of test accuracy, bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min) and regarding classes 1 and 5 (fresh and skin).

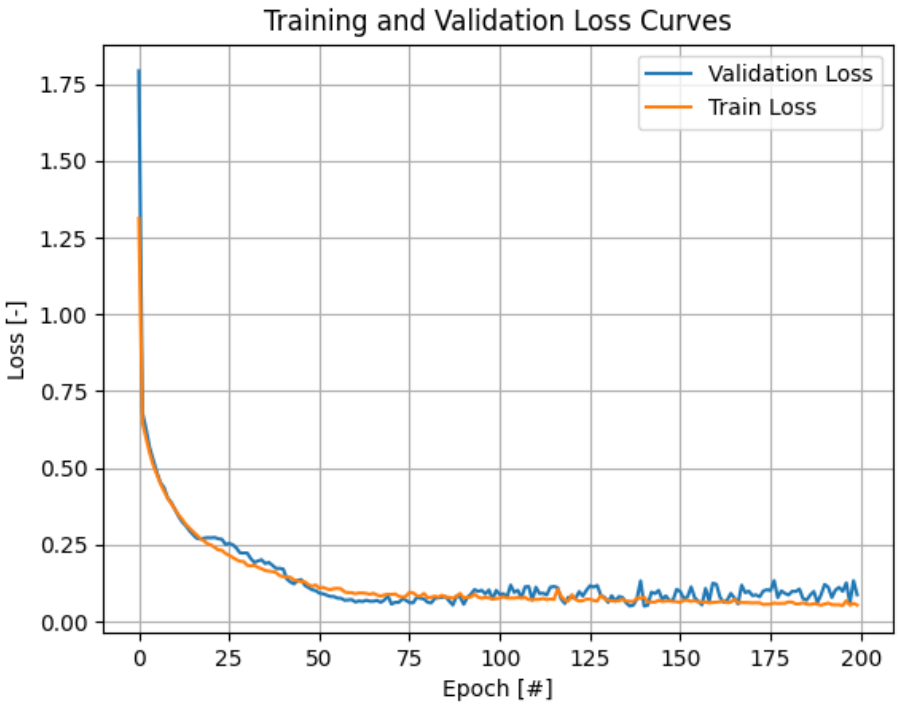


Figure 37 - The training validation loss of LucasNNN with E 200 LR 0.0002

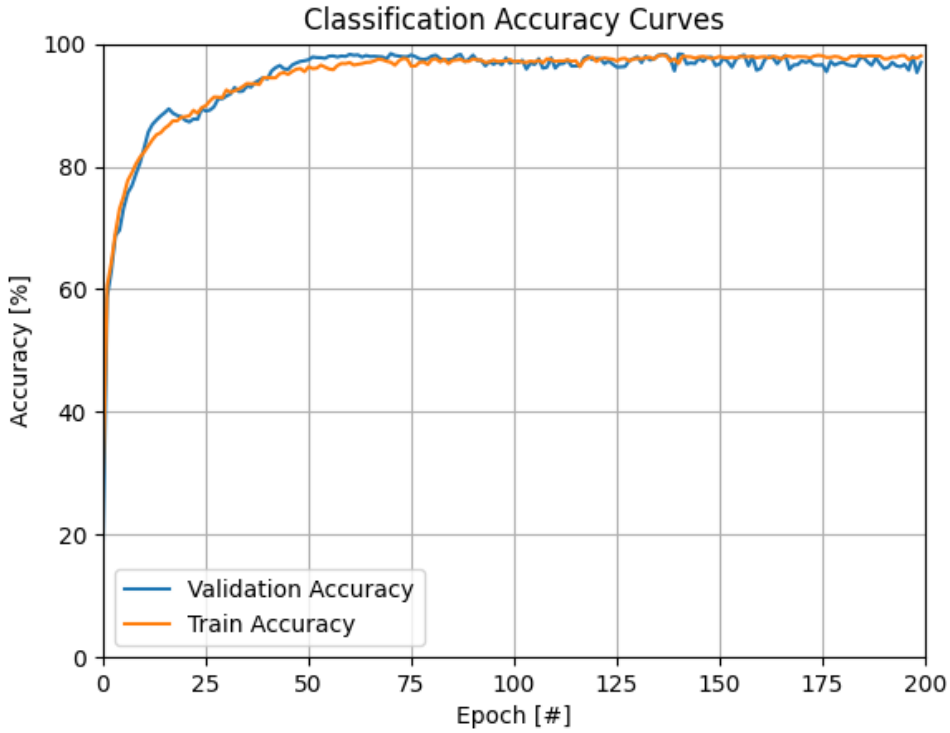


Figure 39 – The training validation accuracy of LucasNNN with E 200 LR 0.0002

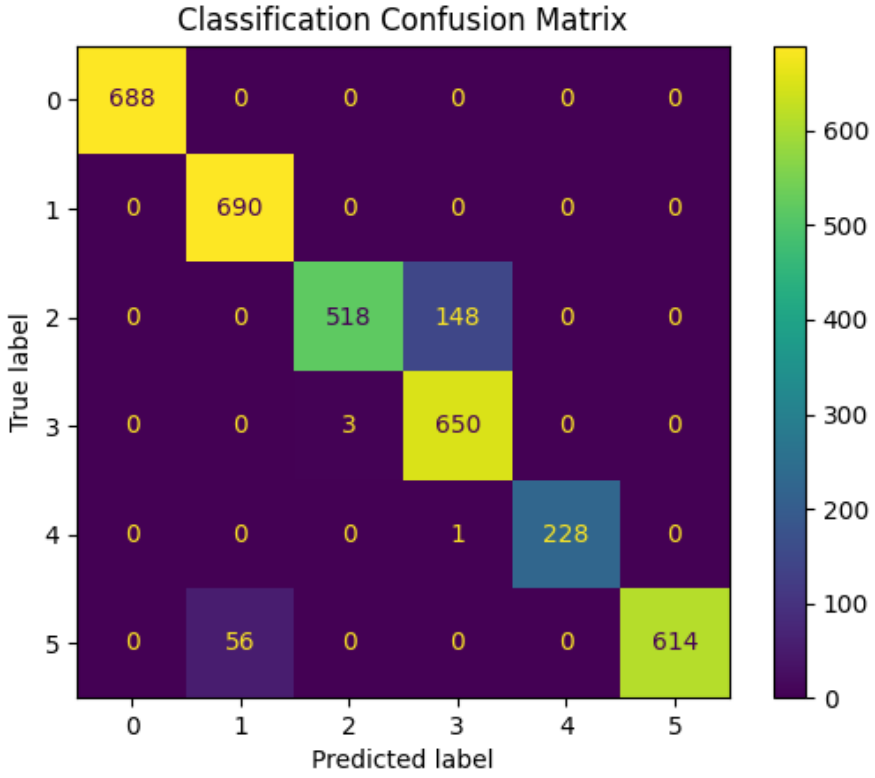


Figure 38 - Confusion Matrix of a LucasNNN with E 200 LR 0.0002

Early Stop

Since the divergencies were non existing, especially in the case used to test the early stop algorithm with a learning rate of 0.001, the training did not stop before the assigned number of epochs of 200.

4.1.2 6-folds Cross-Validation

In order, Figure 40,Figure 41, and Figure 42, are referred to the following learning rate: 0.001, 0.005 and 0.0002. The results are very consistent with a small standard deviation and a high mean. That leads to the conclusion that LucasNNN model is in general very robust and with a good repeatability.

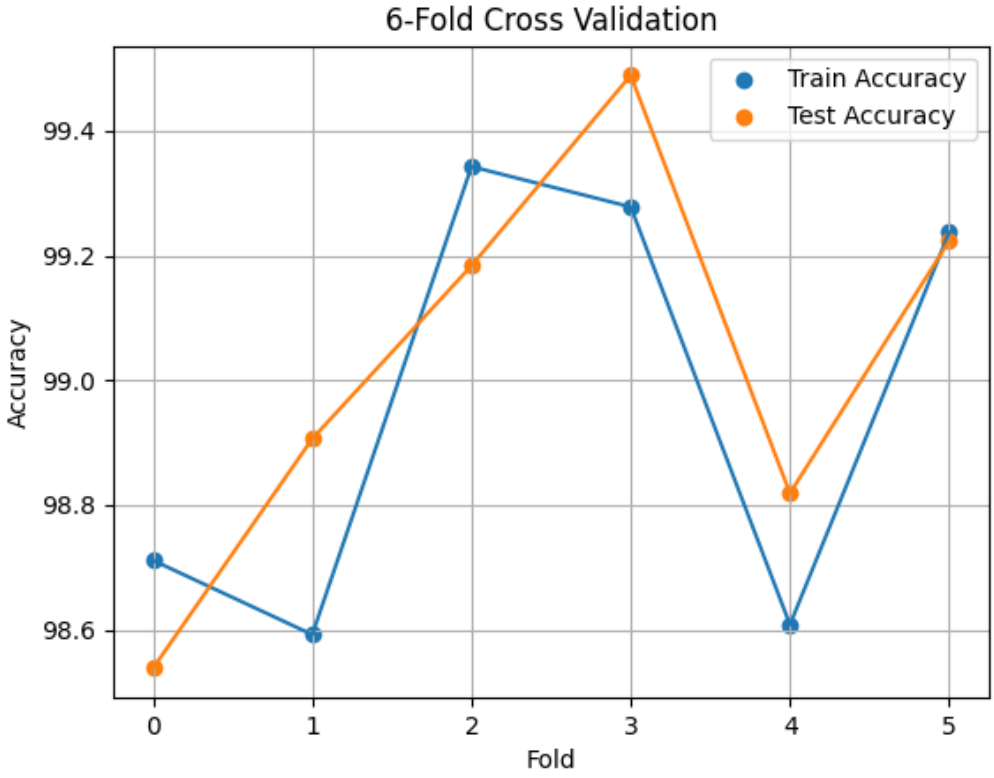


Figure 40 – The cross validation of LucasNNN with E 200 LR 0.001

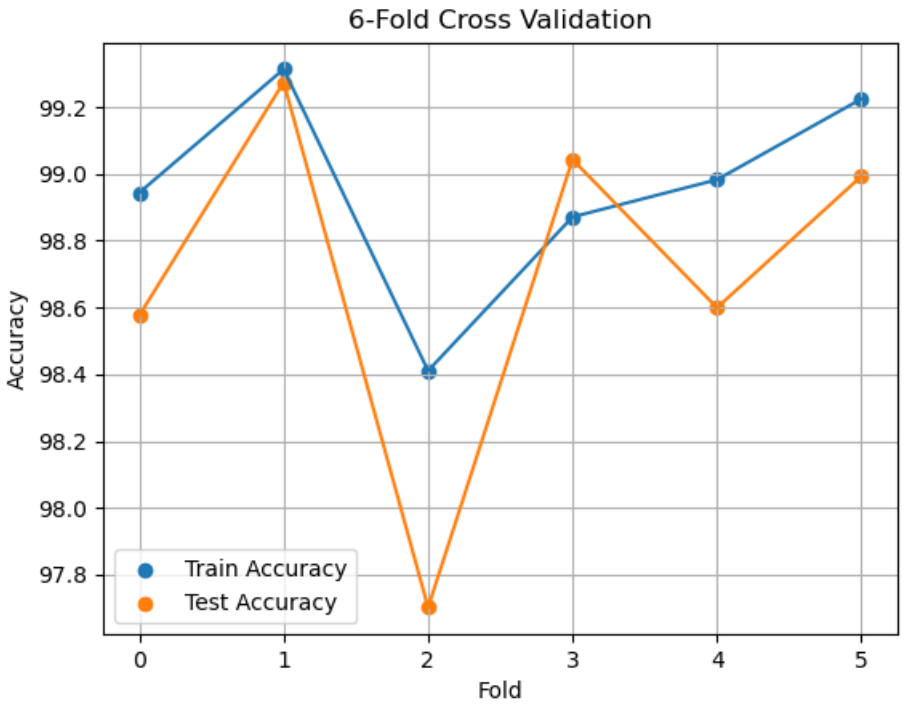


Figure 41 – The cross validation of LucasNNN with E 200 LR 0.005

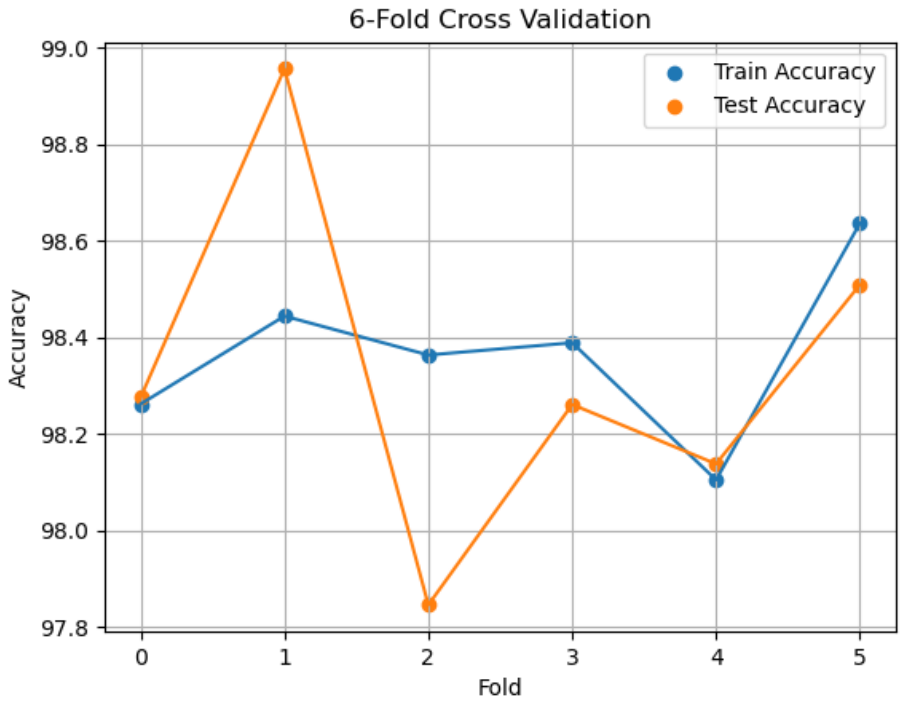


Figure 42 – The cross validation of LucasNNN with E 200 LR 0.0002

4.1.3 Discussion

From Cross Validation experiments is possible to say that the model in general is very robust and reliable to train. Looking at the train, validation and test part is clear that the number of epochs could have been also higher than 400 to output even better results. The learning rate which seems the best in terms of trade-off between spiky curves and speed of training is 0.001. That learning rate also represents the smaller standard deviation in the cross-validation experiments.

4.2. TwoDCNN

In this introductory section there will be described the results from experiments regarding TwoDCNN architecture. A general overview about the combination of parameters used for each of the test outcoming from the train, validation and test pattern are visible in **Error! Reference source not found.**, while in

Model	Epoch (E)	Learning Rate (LR)	Training, Validation time (s)	Test time (s)	Test Accuracy (%)

Model	Epoch (E)	Learning Rate (LR)	Training, Validation time (s)	Test time (s)	Test Accuracy (%)
TwoDCNN	400	0.001	1292.1	0.7749	98.78
	200	0.001	641.2	0.7679	98.92
	200	**0.005**	**642.1**	**0.7699**	**99.17**
	200	0.0002	642.1	0.7669	98.58

Table 10 - Training, Validation, Testing's parameters from TwoDCNN architecture

Model	Epochs (E)	Learning Rate (LR)	Cross Validation time (s)	Average Test Accuracy	Standard deviation test Accuracy
TwoDCNN	200	0.001	5549.5	99.01	0.28
	200	**0.005**	**5334.8**	**99.03**	**0.37**
	200	0.0002	5456.6	98.85	0.37

Table 9 - Parameters adopted for cross validation pattern in TwoDCNN architecture

TwoDCNN	400	0.001	1292.1	0.7749	98.78
	200	0.001	641.2	0.7679	98.92
	200	**0.005**	**642.1**	**0.7699**	**99.17**
	200	0.0002	642.1	0.7669	98.58

Table 10 presented the parameters used for cross validation pattern. In the next two sections there are more in-depth description of the results from the tables presented down below, also with references to the related plots.

4.2.1 Training, Validation, and Testing Results

In this section there are described only the experiments resulting from the train, validation and test pattern described in Table 10 - Training, Validation, Testing's parameters from TwoDCNN architecture**Error! Reference source not found.**.

TwoDCNN:400 epochs, 0.001 learning rate

From the plots in Figure 43 and Figure 45 it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 44 is showing amazing results in terms

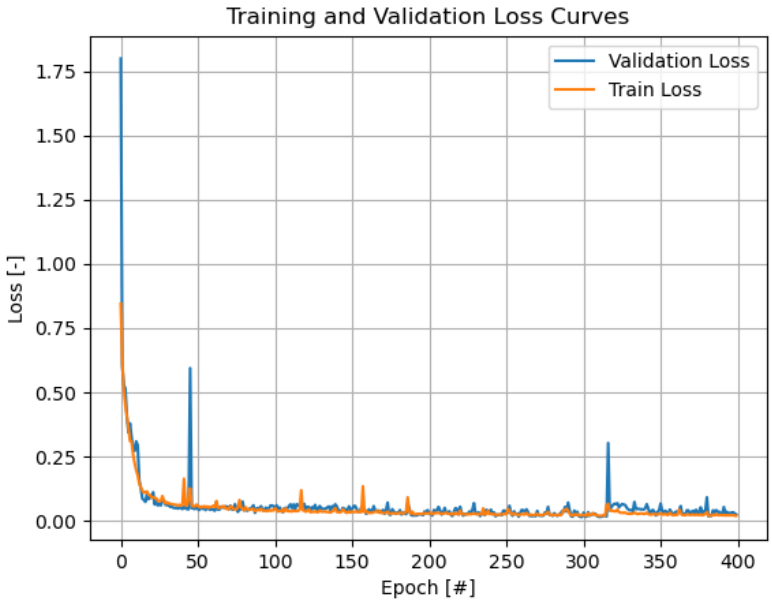


Figure 43 - The training validation loss of TwoDCNN with E 400 LR 0.001

of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

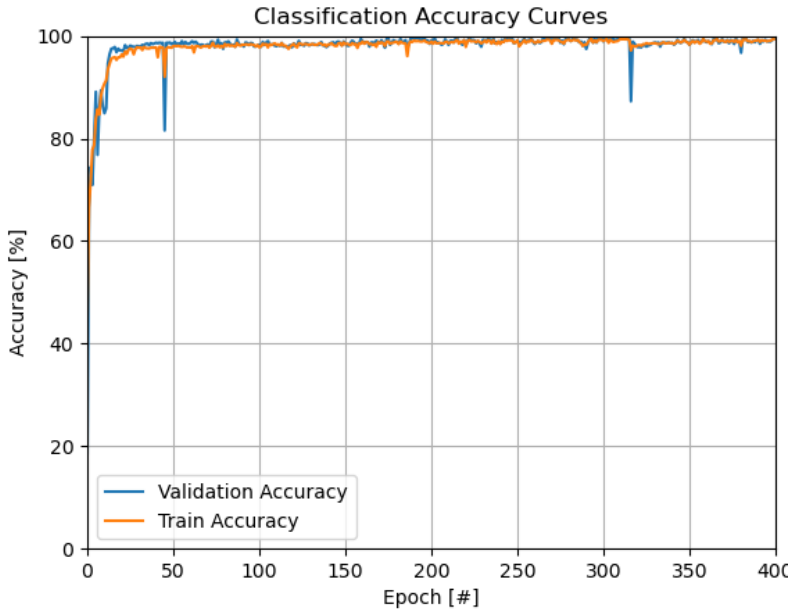


Figure 45 – The training validation accuracy of TwoDCNN with E 400 LR 0.001

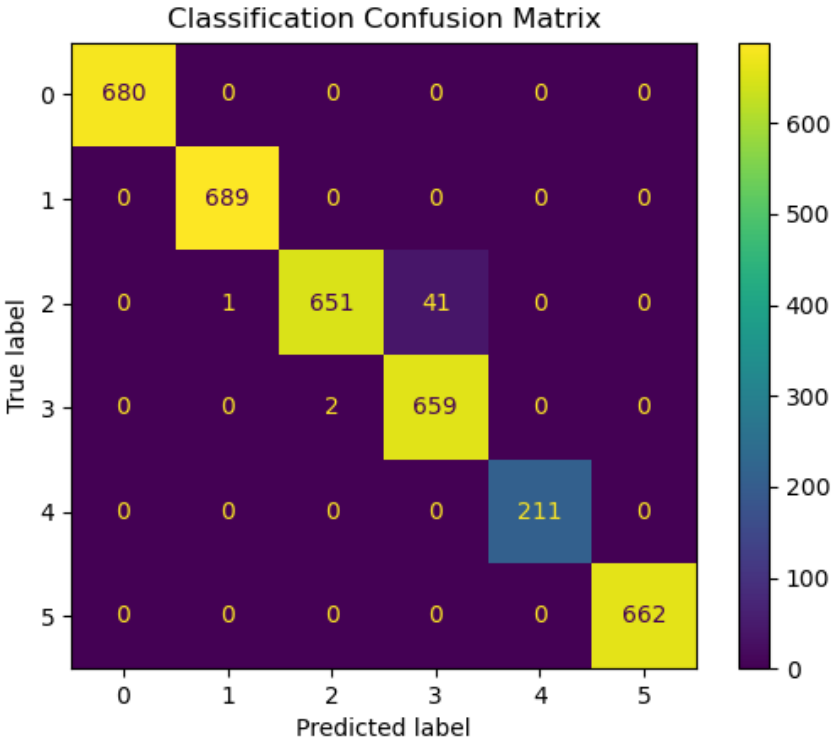


Figure 44 - Confusion Matrix of a TwoDCNN with E 400 and LR 0.001

TwoDCNN: 200 epochs, 0.001 learning rate

From the plots in Figure 46 and Figure 47 it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting. This was inevitable since the parameters are the same with the previous experiment, except for the number of epochs which is lower.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 48 is showing amazing results in terms of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

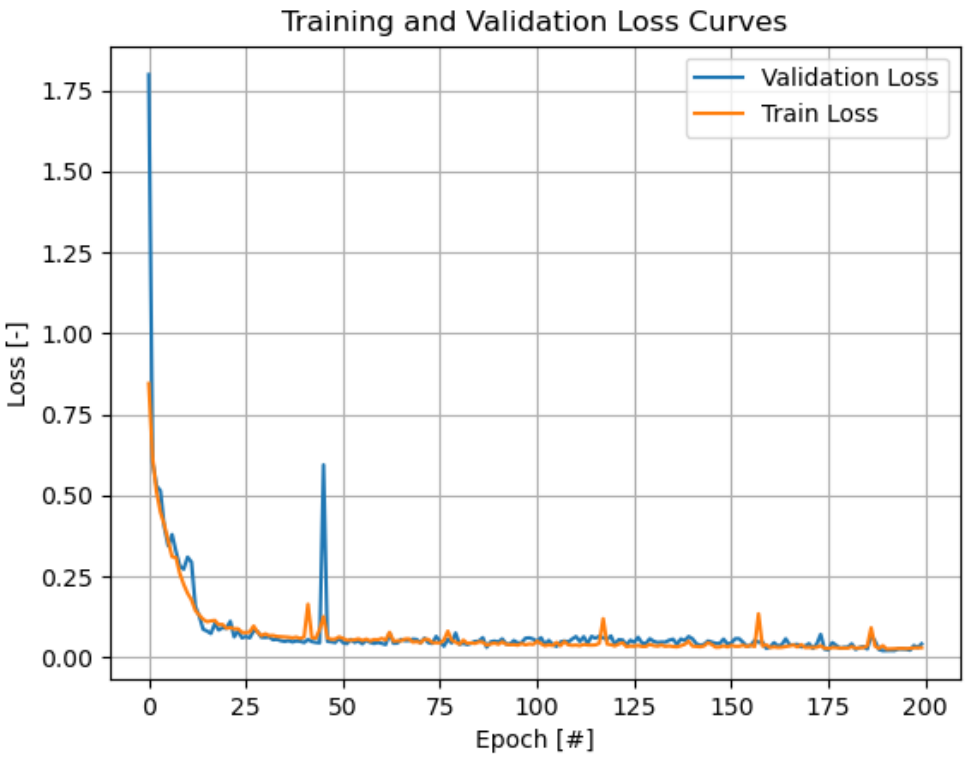


Figure 46 - The training validation loss of TwoDCNN with E 200 LR 0.001

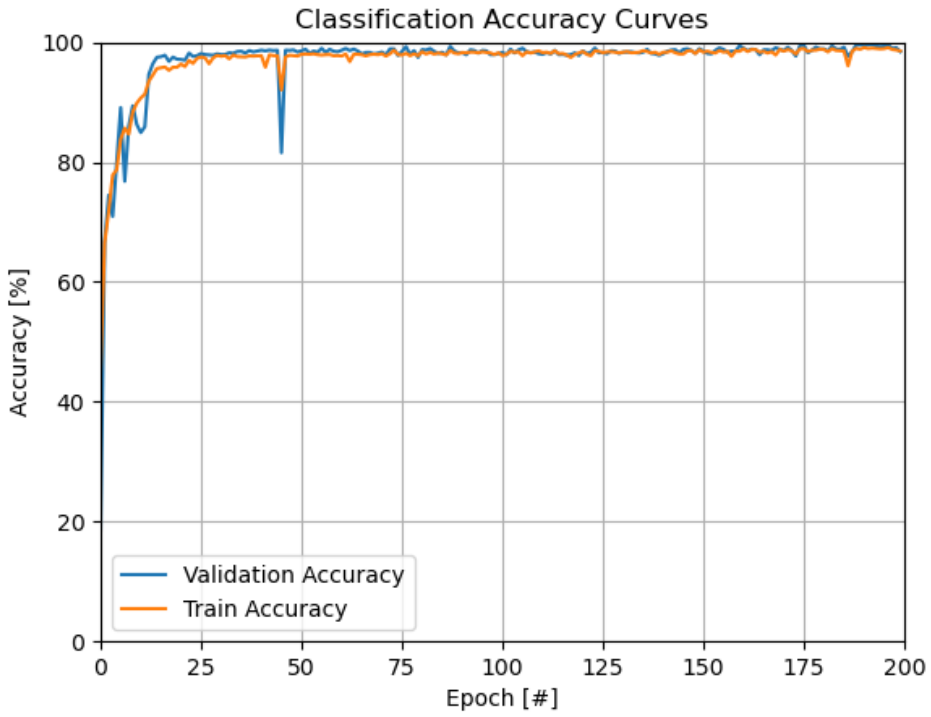


Figure 47 - The training validation accuracy of TwoDCNN with E 200 LR 0.001

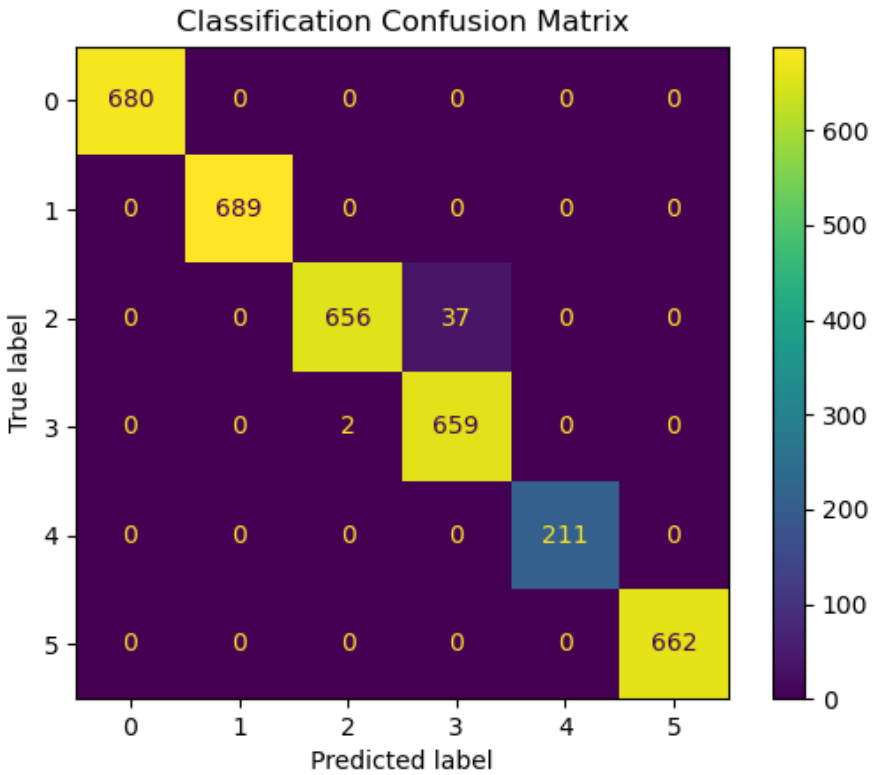


Figure 48 –Confusion Matrix of a TwoDCNN with E 200 and LR 0.001

TwoDCNN: 200 epochs, 0.005 learning rate

From the plots in Figure 49 and Figure 51 it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 50 is showing amazing results in terms of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

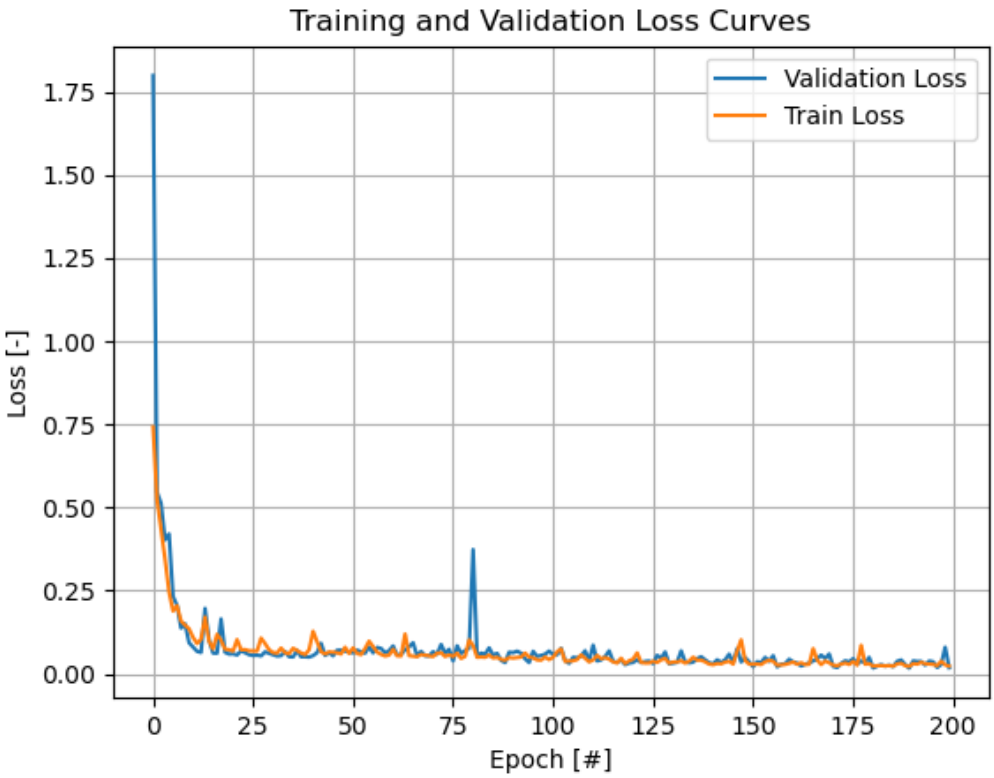


Figure 49 - The training and validation loss of TwoDCNN with E 200 LR 0.005

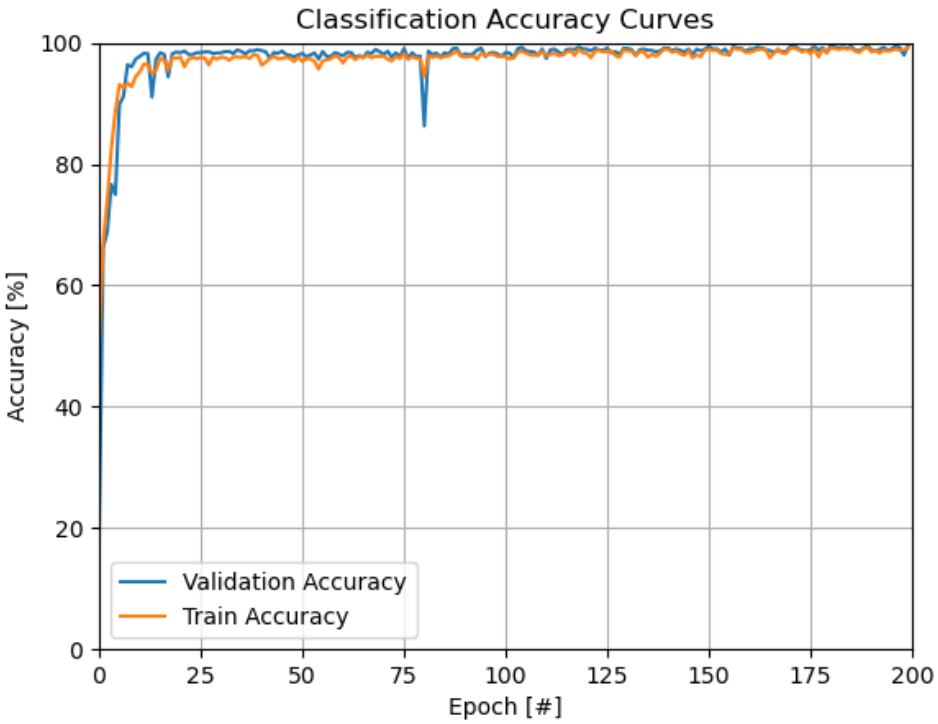


Figure 51 - The training and validation accuracy of TwoDCNN with E 200 LR 0.005

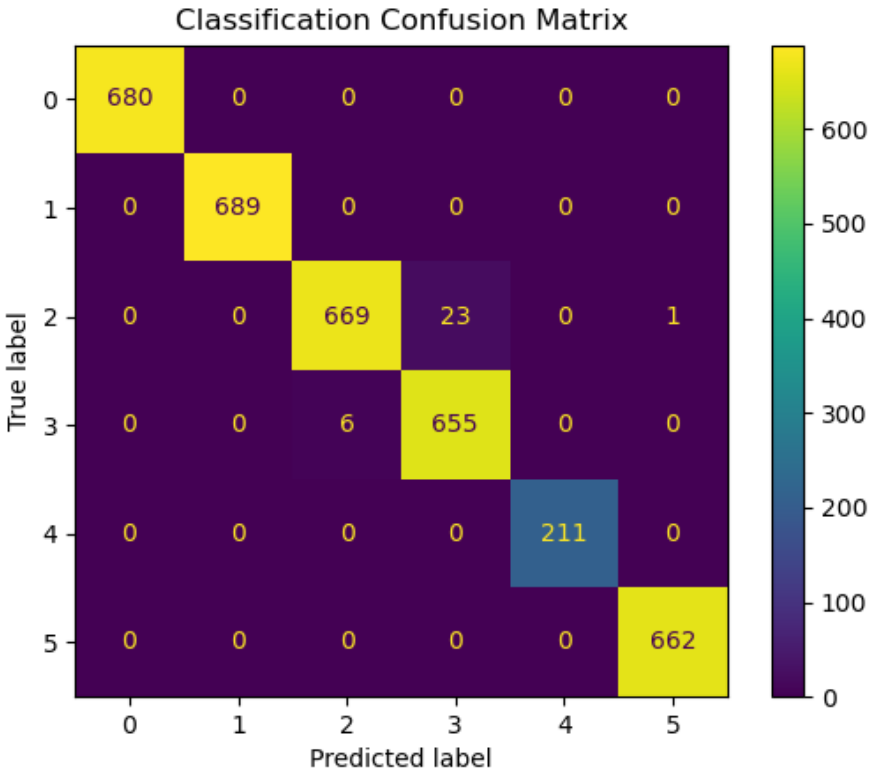


Figure 50 –Confusion Matrix of a TwoDCNN with E 200 and LR 0.005

TwoDCNN: 200 epochs, 0.0002 learning rate

From the plots in Figure 52 and Figure 53 it is possible to see that in the first 50 epochs the curves have almost reached their steady state values, this number is higher as expected from the previous experiments, in fact a lower learning rate means smaller steps for the training, so lower learning speed.

In accuracy plot there is a slight sign of divergency, but nothing to be worried about, the model is still generalizing and training well avoiding any overfitting.

In general, during the evolution of the curves in the two plots the smooth behavior is associated with the lower learning rate, in fact moving in training with smaller steps means also to always move from one relative minimum gradually to another, without rough jumps. Confusion matrix in Figure 54 is showing good results in terms of test accuracy, bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min) and regarding classes 1 and 5 (fresh and skin).

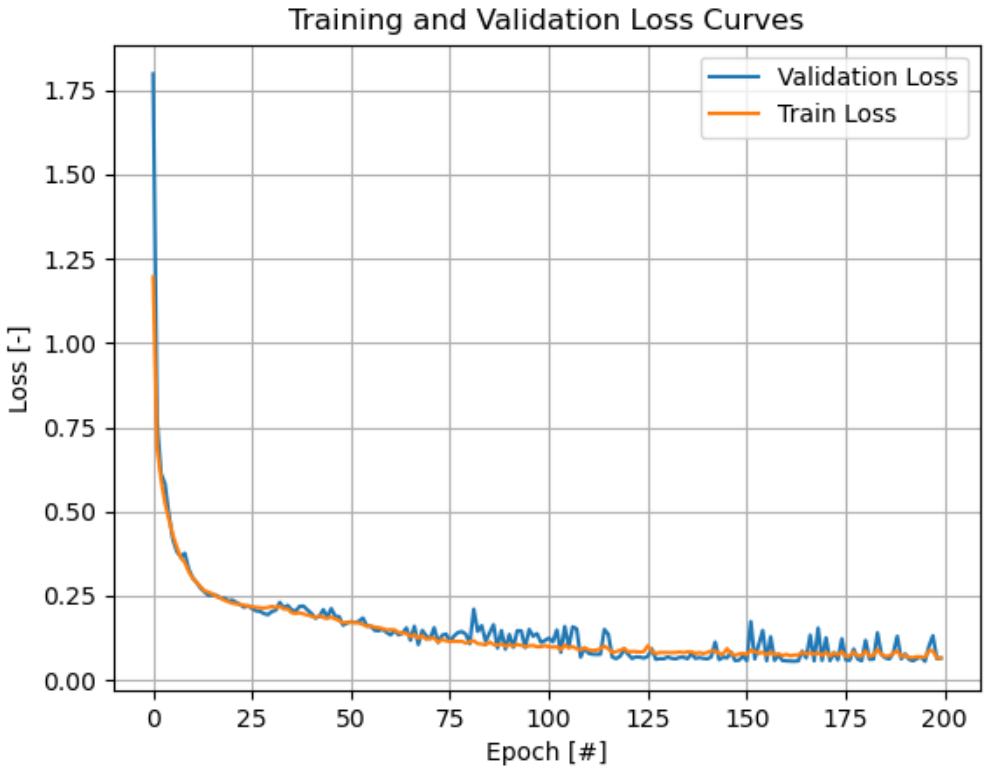


Figure 52 - The training and validation loss of TwoDCNN with E 200 LR 0.0002

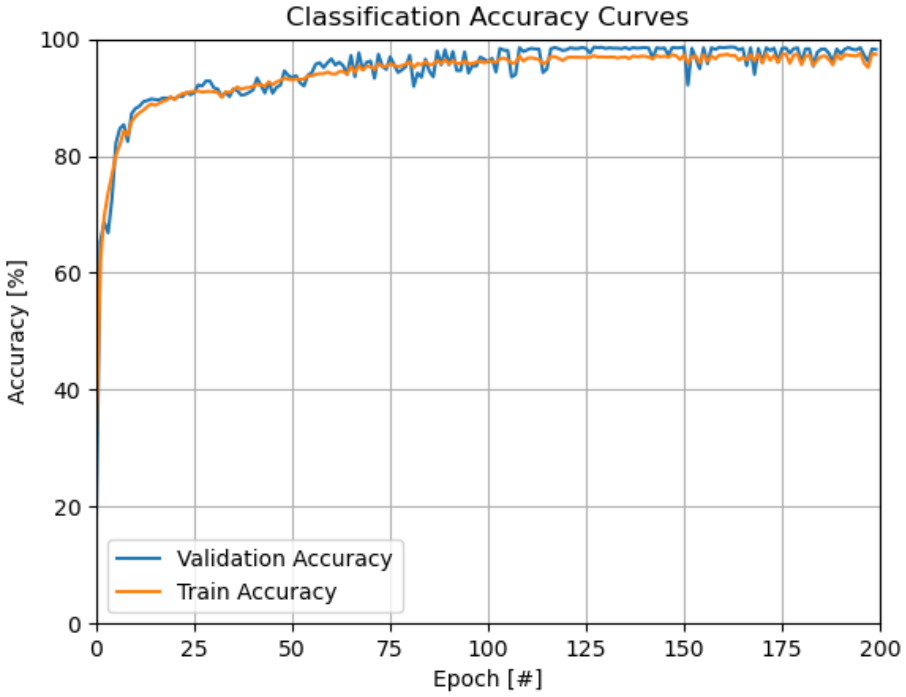


Figure 53 – The training validation accuracy of TwoDCNN with E 200 LR 0.0002

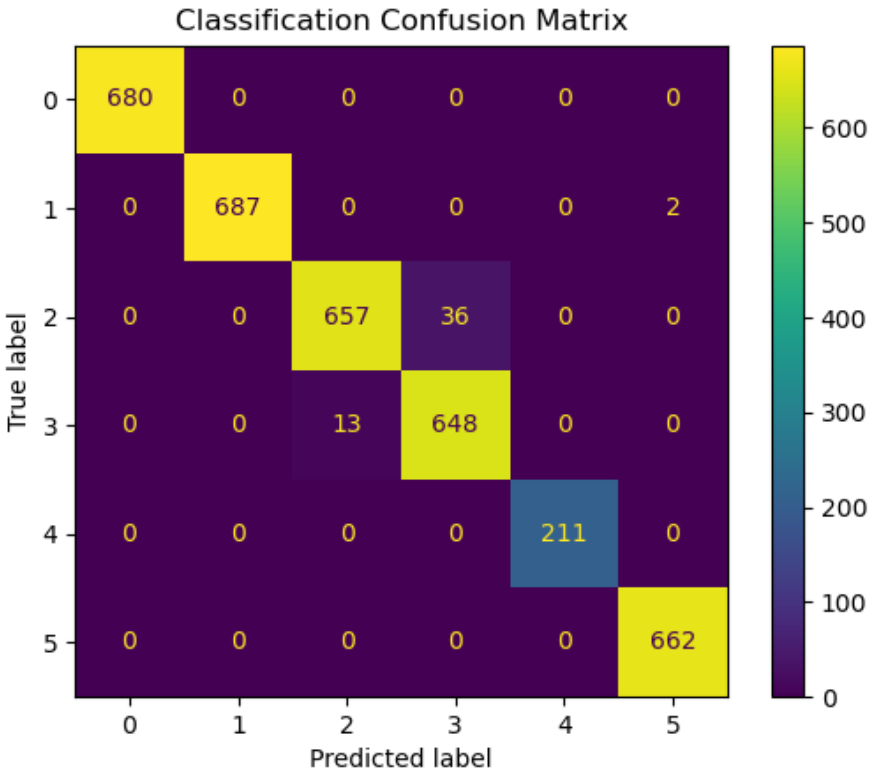


Figure 54 - Confusion Matrix of a TwoDCNN E 200 and LR 0.0002

Early Stop

There is no sign of divergency between training and validation plot for 200 epochs. Thus, the early stop algorithm condition is not met.

4.2.2 6-folds Cross-Validation

Figure 55, Figure 56, and Figure 57 are referred to the following learning rate: 0.001, 0.005 and 0.0002. The results are very consistent with a small standard deviation and a high mean. That leads to the conclusion that LucasNNN model is in general very robust and with a good repeatability.

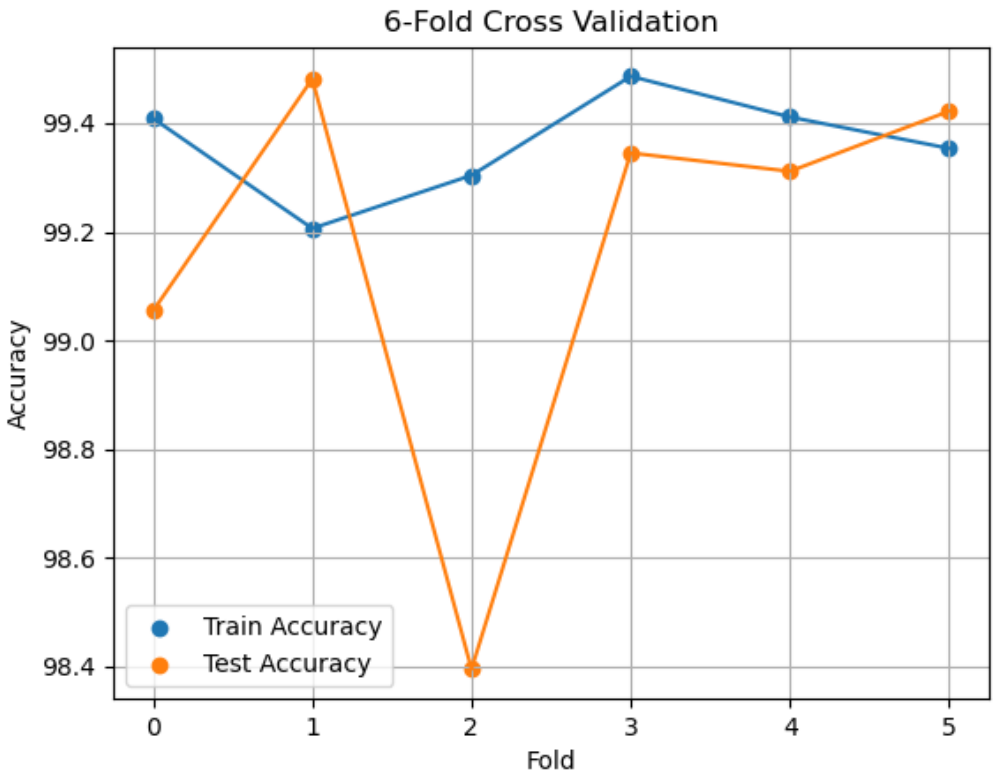


Figure 55 – Cross Validation of TwoDCNN with E 200 LR 0.001

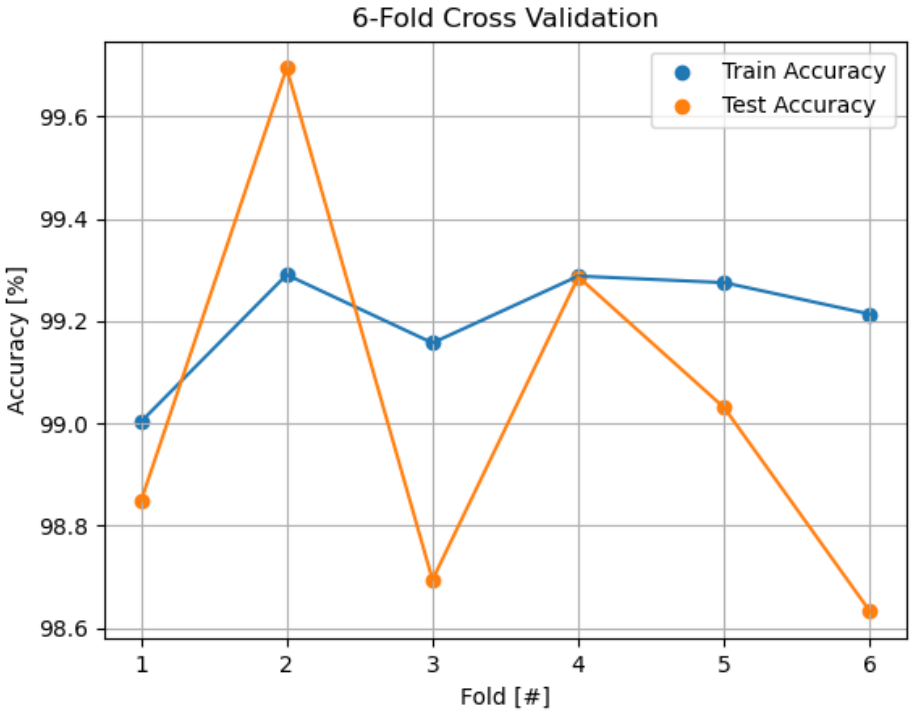


Figure 56 – Cross Validation TwoDCNN with E 200 LR 0.005

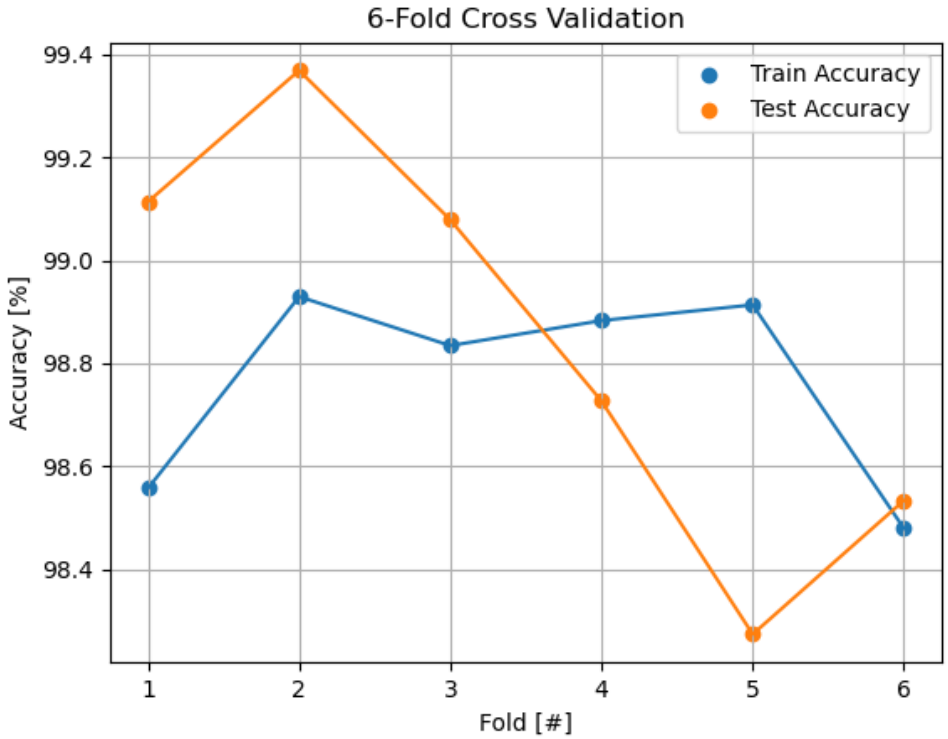


Figure 57 – Cross Validation of TwoDCNN with E 200 LR 0.0002

4.2.3 Discussion

From Cross Validation experiments is possible to say that the model in general is very robust and reliable to train. Looking at the train, validation and test part is clear that the number of epochs could have been also higher than 400 to output even better results. The learning rate which seems the best in terms of tradeoff between spiky curves and speed of training is 0.001. That learning rate also represents the smaller standard deviation in the cross-validation experiments.

4.3.Hu et al.

In this introductory section there will be described results from experiments regarding Hu et al. [10] architecture. A general overview about the combination of parameters used for each test outcoming from train, validation, and test pattern is visible in **Error! Reference source not found.**, while in **Error! Reference source not found.** are presented the parameters used for cross validation pattern. In the next two sections there is a more in-depth description of the results from the tables presented down below, also with references to the related plots.

Model	Epoch (E)	Learning Rate (LR)	Training, Validation time (s)	Test time (s)	Test Accuracy (%)
Hu et al. [10]	400	0.001	204.8	0.0728	99.36
	200	**0.001**	**102.2**	**0.0738**	**99.53**
	200	0.005	102.9	0.0738	99.42
	200	0.0002	102.8	0.0738	98.69

Table 12 - Training, Validation, Testing's of Hu et al architecture

Model	Epochs (E)	Learning Rate (LR)	Cross Validation time (s)	Average Test Accuracy	Standard deviation test Accuracy
Hu et al. [10]	**200**	**0.001**	**954.1**	**99.21**	**0.11**
	200	0.005	960.1	99.12	0.34
	200	0.0002	5456.6	98.85	0.37

Table 11 - Parameters adopted for cross validation pattern in Hu et al architecture

4.3.1 Training, Validation, and Testing Results

In this section there are described only the experiments resulting from the train, validation and test pattern described in Table 7.

Hu et al. [10]: 400 epochs, 0.001 learning rate

From the plots in Figure 58 and Figure 60 it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 59 is showing amazing results in terms of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

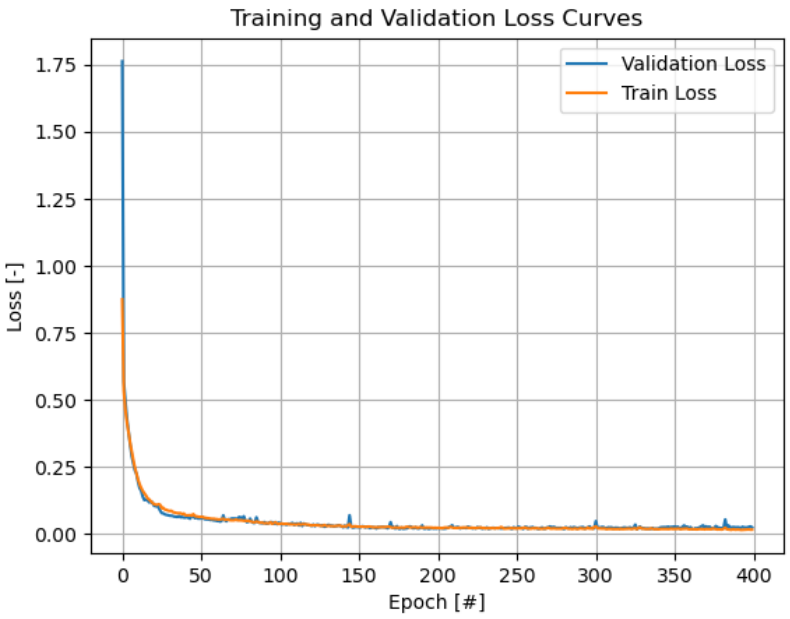


Figure 58 - The training validation loss of Hu et al. with E 400 LR 0.001

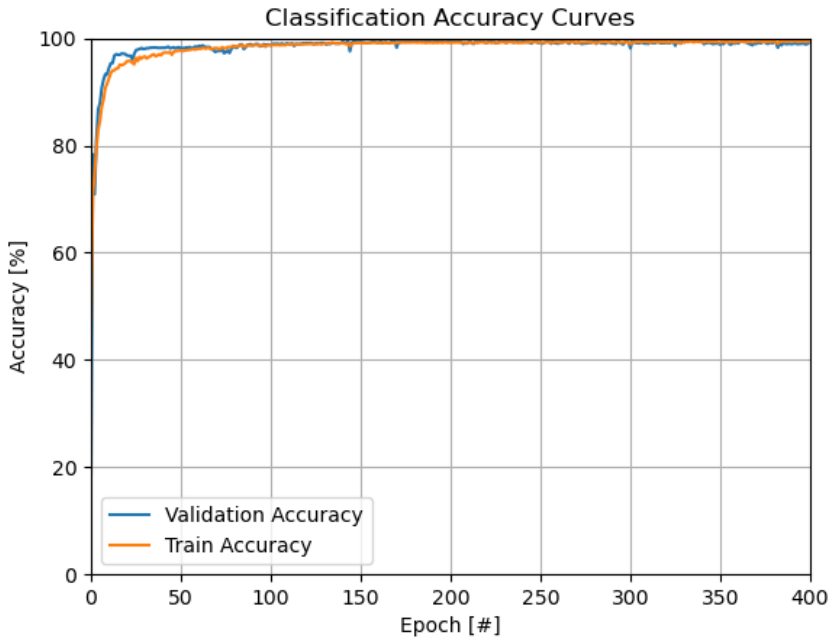


Figure 60 – The training validation accuracy of Hu et al. with E 400 LR 0.001

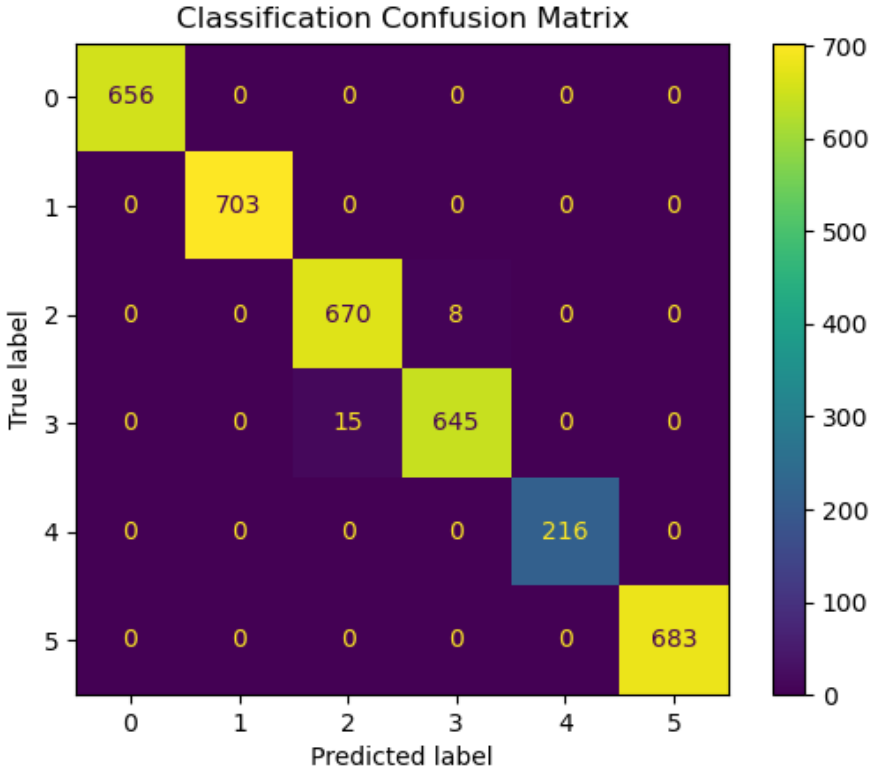


Figure 59 - Confusion Matrix of a Hu et al. with E 400 LR 0.001

Hu et al. [10]: 200 epochs, 0.001 learning rate

From the plots in Figure 61 and Figure 63 it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting. This was inevitable since the parameters are the same with the previous experiment, except for the number of epochs which is lower.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 62 is showing amazing results in terms of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

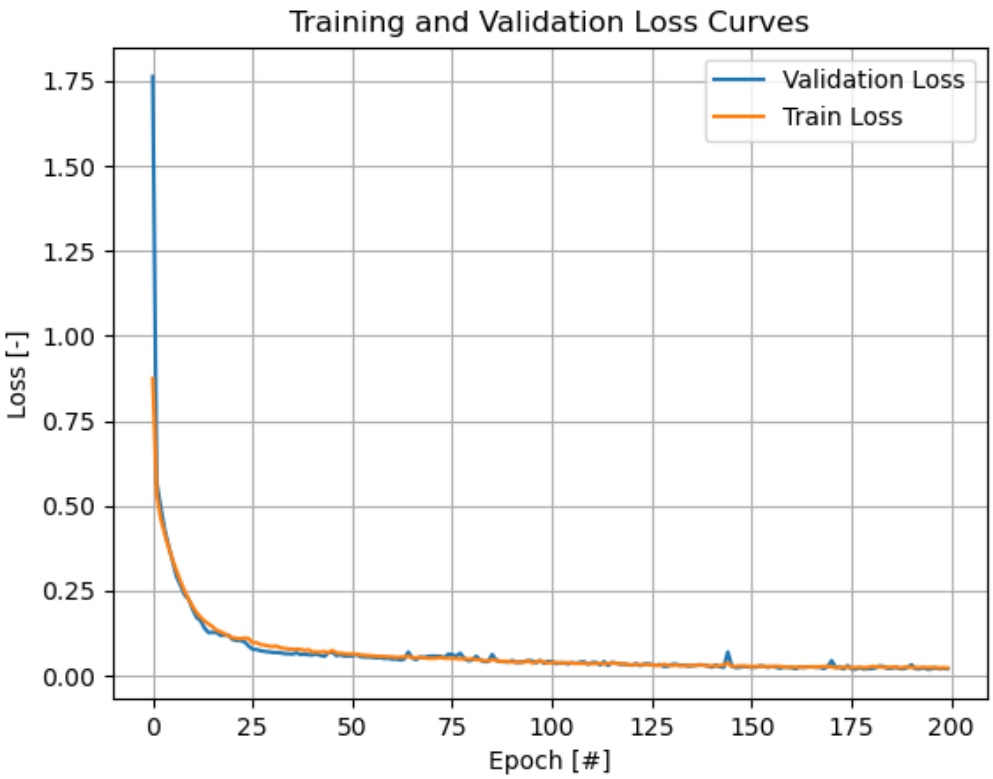


Figure 61 - The training validation accuracy of Hu et al. with E 200 LR 0.001

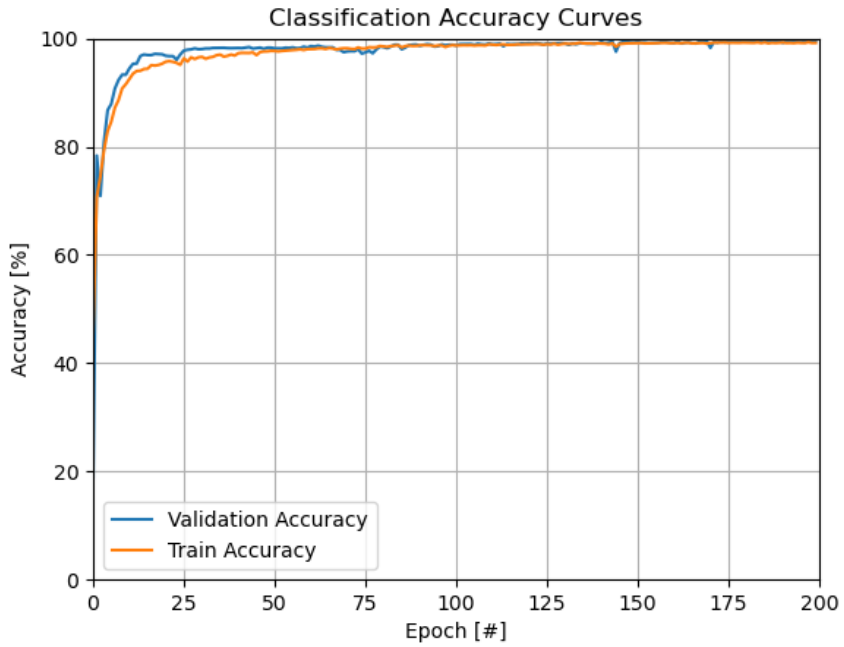


Figure 63 - The training and validation loss of Hu et al. with E 200 LR 0.001

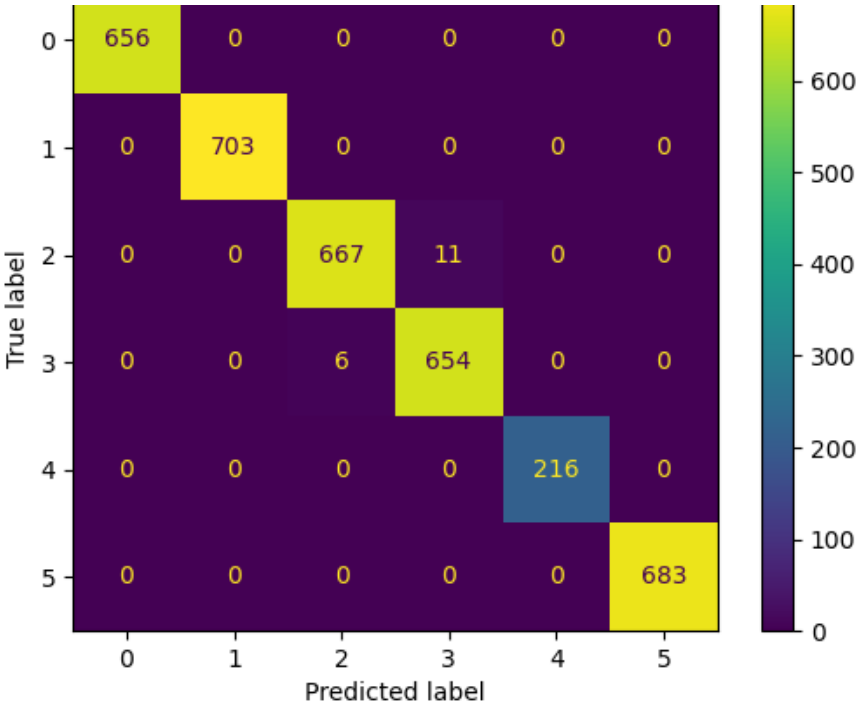


Figure 62 - Confusion Matrix of a Hu et al. with E 200 LR 0.001

Hu et al. [10]: 200 epochs, 0.005 learning rate

From the plots in Figure 64 and in Figure 66 it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 65 is showing amazing results in terms of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

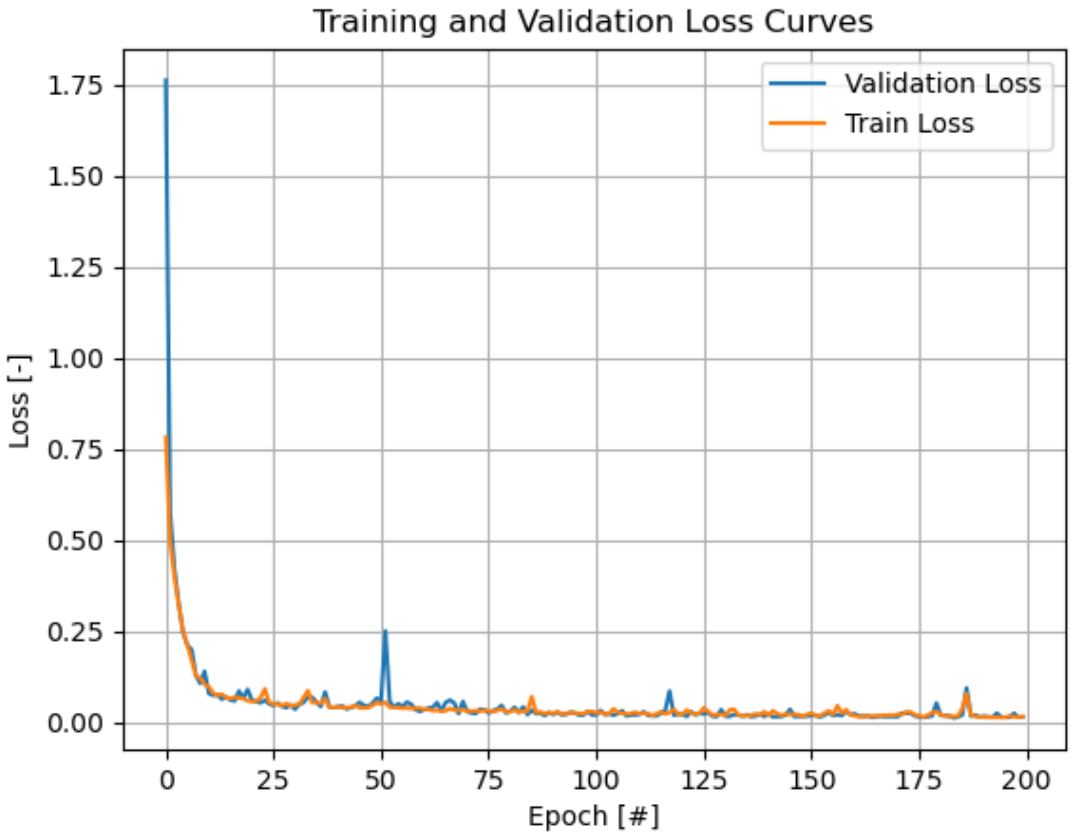


Figure 64 - The training and validation loss of Hu et al. with E 200 LR 0.005

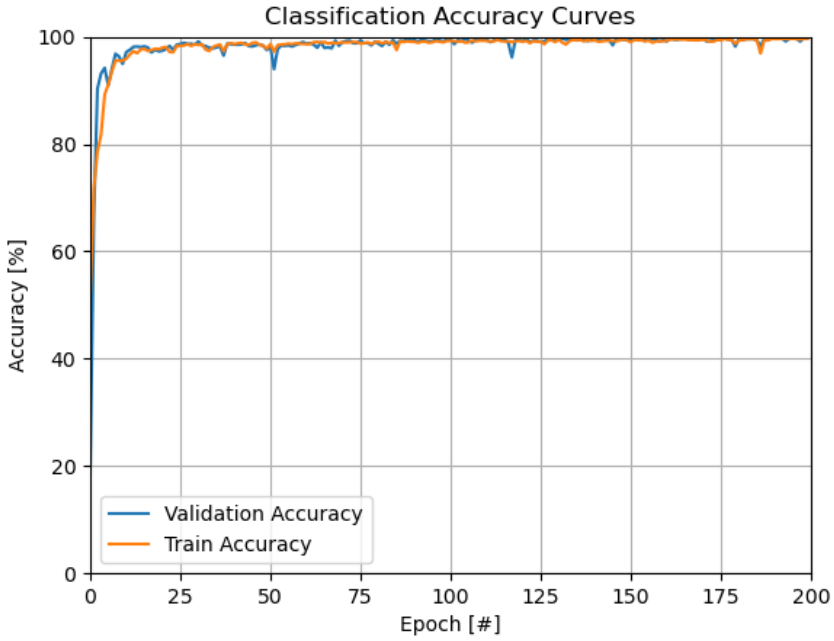


Figure 66 - The training and validation accuracy of Hu et al. E 200 LR 0.005

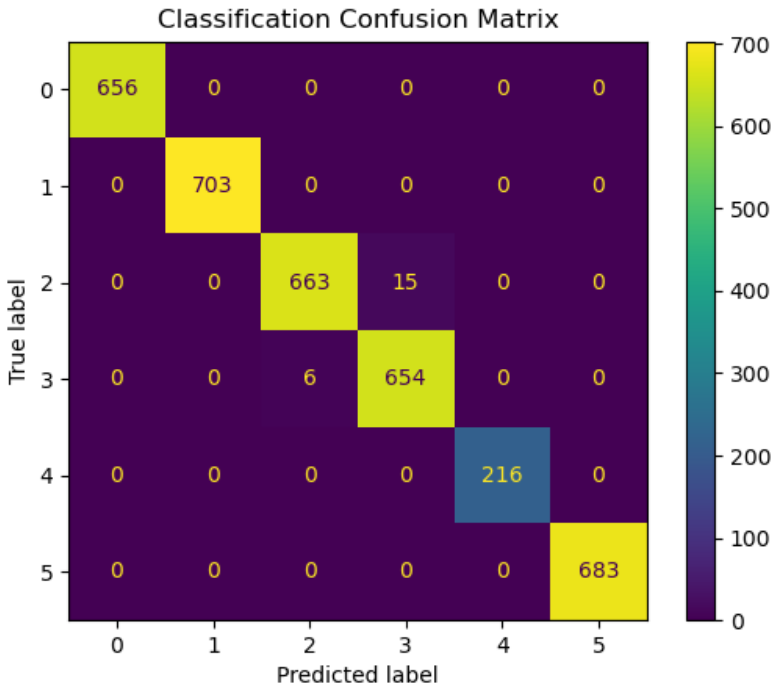


Figure 65 –epochs Confusion Matrix of a Hu et al. with E 200 LR 0.005

Hu et al. [10]: 200 epochs, 0.0002 learning rate

From the plots in Figure 67, and Figure 69 it is possible to see that in the first 50 epochs the curves have almost reached their steady state values, this number is higher as expected from the previous experiments, in fact a lower learning rate means smaller steps for the training, so lower learning speed.

In both loss and accuracy plots there is no sign of divergency, that means that the model is still generalizing and training well avoiding any overfitting.

In general, during the evolution of the curves in the two plots the smooth behavior is associated with the lower learning rate, in fact moving in training with smaller steps means also to always move from one relative minimum gradually to another, without rough jumps. Confusion matrix in Figure 68 is showing amazing results in terms of test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

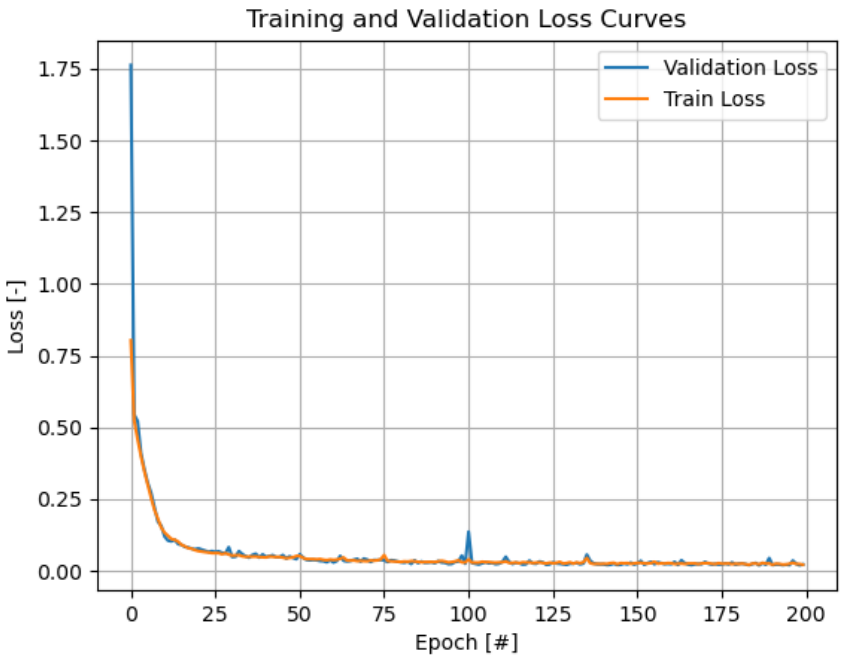


Figure 67 - The training and validation loss of Hu et al. with E 200 LR 0.0002

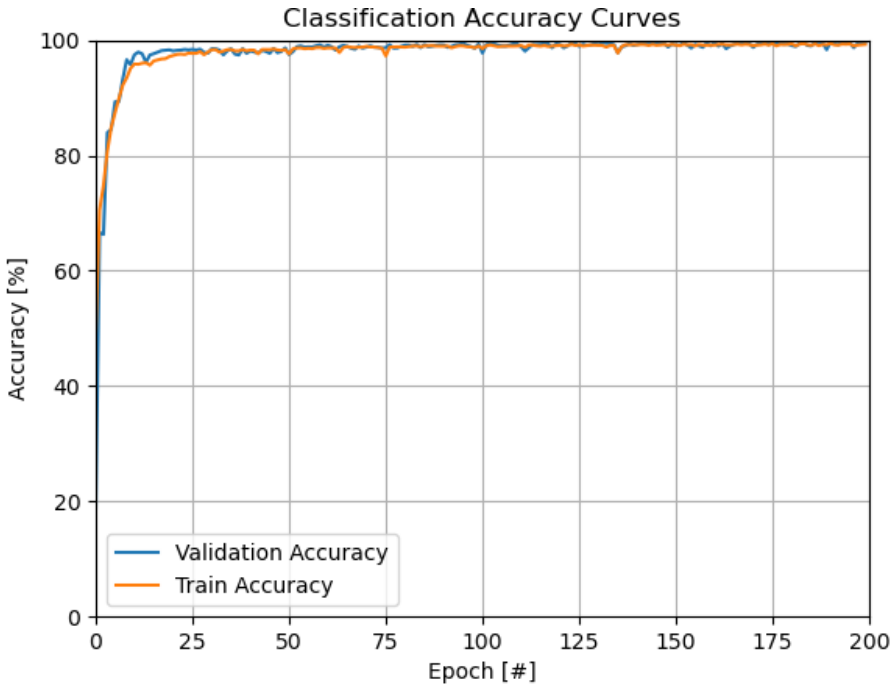


Figure 69 – The training and validation accuracy of Hu et al. with E 200 LR 0.0002

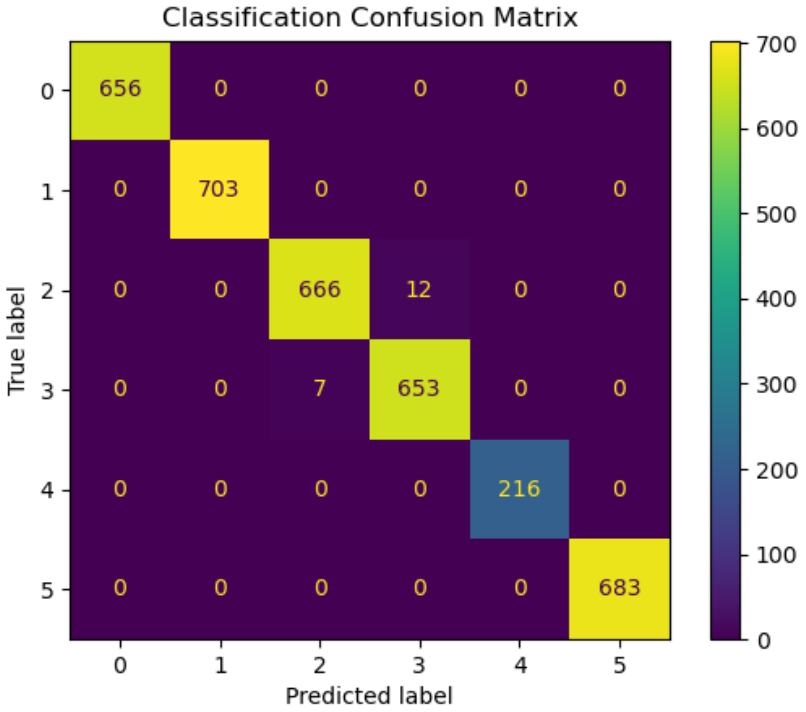


Figure 68 - Confusion Matrix of a Hu et al. with E 200 LR 0.0002

4.3.2 6-folds Cross-Validation

Figure 70, Figure 71, and Figure 72 are referred to the following learning rate: 0.001, 0.005 and 0.0002. The results are very consistent with a small standard deviation and a high mean. That leads to the conclusion that Hu et al. [10] model is in general very robust and with a good repeatability.

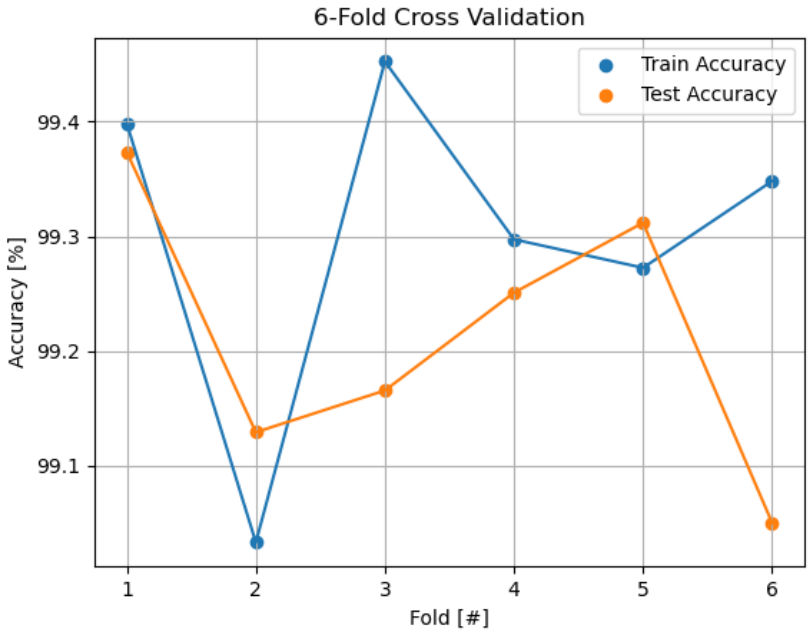


Figure 70 – Cross Validation of Hu et al. with E 200 LR 0.001

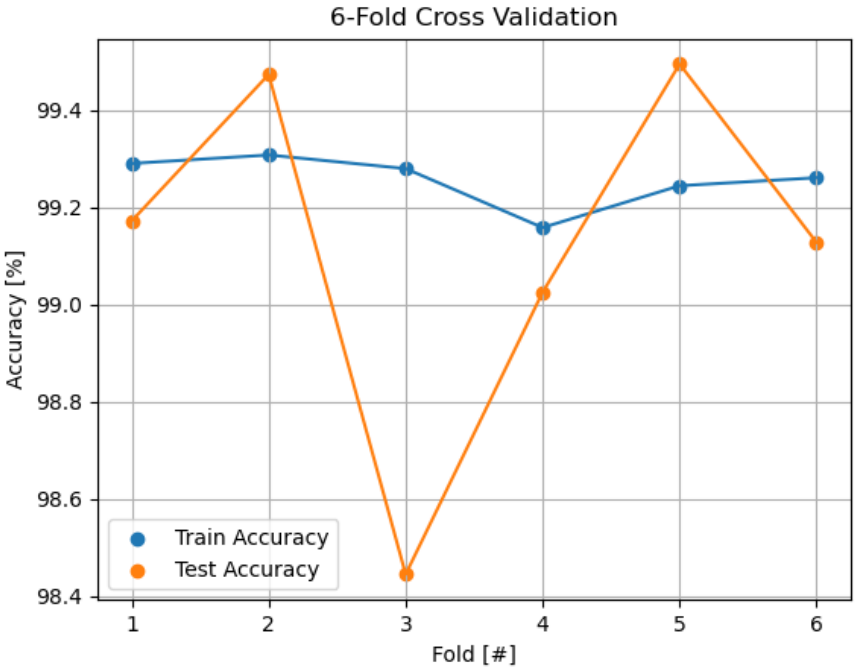


Figure 71 - Cross Validation of Hu et al. with E 200 LR 0.005

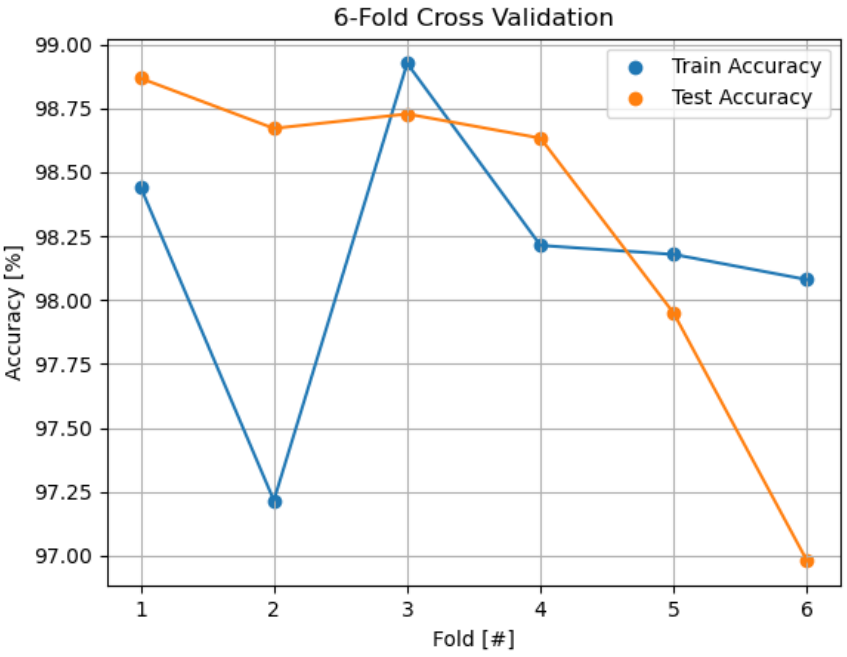


Figure 72 – Cross Validation of Hu et al. with E 200 LR 0.0002

4.3.3 Discussion

From Cross Validation experiments is possible to say that the model in general is very robust and reliable to train. Looking at the train, validation and test part is clear that the number of epochs could have been also higher than 400 to output even better results. The learning rate which seems the best in terms of tradeoff between spiky curves and speed of training is 0.001. That learning rate also represents the smaller standard deviation in the cross-validation experiments.

4.4.FCNet

In this introductory section there will be described results from experiments regarding FCNet architecture. A general overview about the combination of parameters used for each test from train, validation and test pattern is visible in Table 13, while in **Error! Reference source not found.** are presented the parameters used for cross validation pattern. In the next two sections there is a more

Model	Epoch (E)	Learning Rate (LR)	Training, Validation time (s)	Test time (s)	Test Accuracy (%)
FCNet	**400**	**0.001**	**205.5**	**0.1097**	**97.08**
	200	0.001	101.5	0.072	94.83
	200	0.005	104.5	0.0652	87.29
	200	0.0002	167.7	0.1287	97.05
	120 (ESA)	0.001	63.9	0.0668	96.85

Table 13 - Training, Validation, Testing's of FCNet models

Model	Epoch (E)	Learning Rate (LR)	Cross Validation time (s)	Average test accuracy	Standard deviation test accuracy
FCNet	**200**	**0.001**	**848,1**	**97,74**	**1,27**
	200	0.005	852,7	97,46	0,7
	200	0.0002	848,6	97,6	0,75

Table 14 - Cross Validation results from FCNet models

in-depth description of the results from the tables presented down below, also with references to the related plots.

4.4.1 Training, Validation, and Testing Results

In this section there are described only the experiments resulting from the train, validation and test pattern described in Table 13**Error! Reference source not found.**.

FCNet: 400 epochs, 0.001 learning rate

From the plots in Figure 73 and Figure 74 it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there are clear signs of divergency, especially around epoch 120, that means that the model is not generalizing well but is trying just to imitate the behavior of the training dataset, resulting in overfitting.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 75 is showing good results in terms of

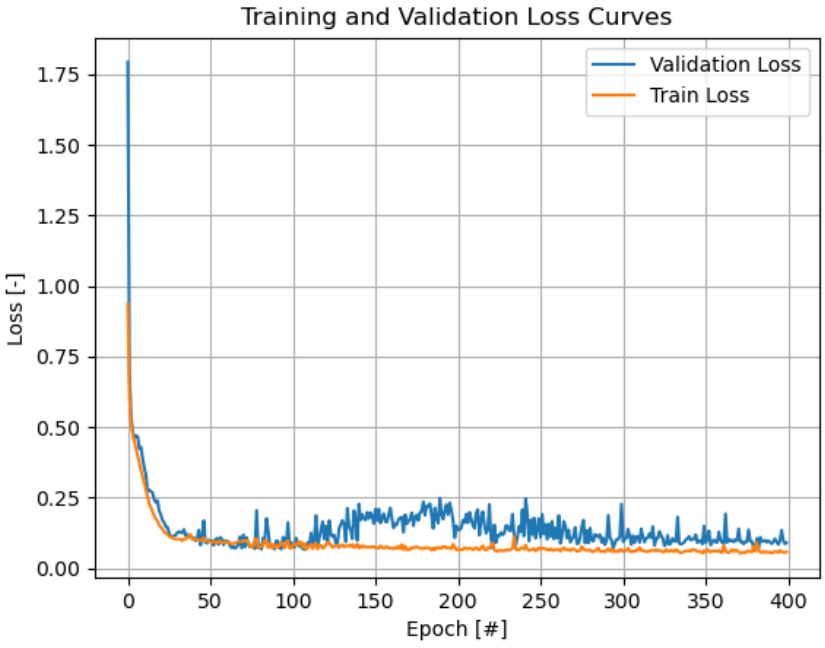


Figure 73 - The training and validation loss of FCNet with E 400 LR 0.001

test accuracy, just a bit of confusion regarding classes 2 and 3 (baked 10 min and

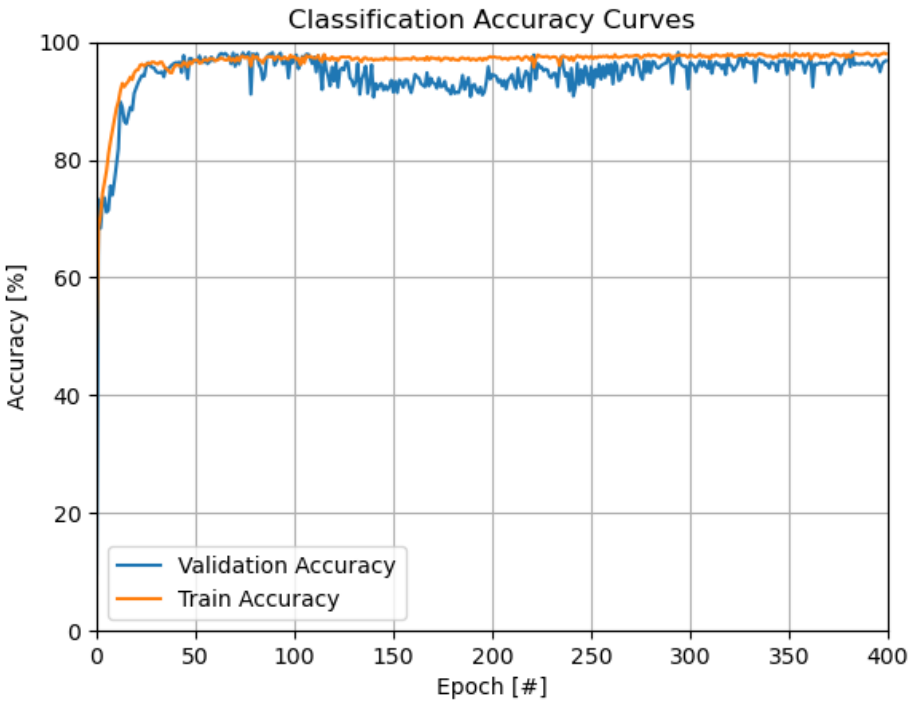


Figure 74 – The training and validation accuracy of FCNet with E 400 LR 0.001

baked 20 min).

FCNet: 200 epochs, 0.001 learning rate

From the plots in Figure 76 and Figure 77 it is possible to see that in the first 30 epochs the curves have almost reached their steady state values.

In both loss and accuracy plots there are clear signs of divergency, especially around epoch 120, that means that the model is not generalizing well but is trying just to imitate the behavior of the training dataset, resulting in overfitting.

In general, during the evolution of the curves in the two plots is possible to see a spiky behavior, this is because the learning rate is a bit too big, but still the training was evolving well. Confusion matrix in Figure 78 is showing good results in terms of test accuracy, a bit of confusion regarding classes 2 and 3 (baked 10 min and baked 20 min), more than the 400 epochs case.

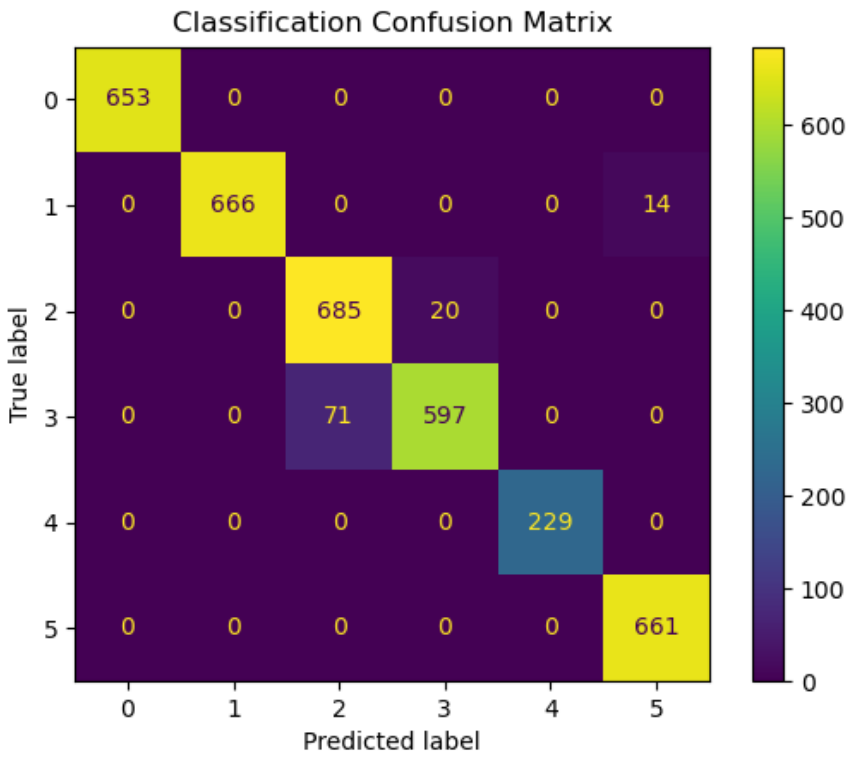


Figure 75 - Confusion Matrix of a FCNet with E 400 LR 0.001

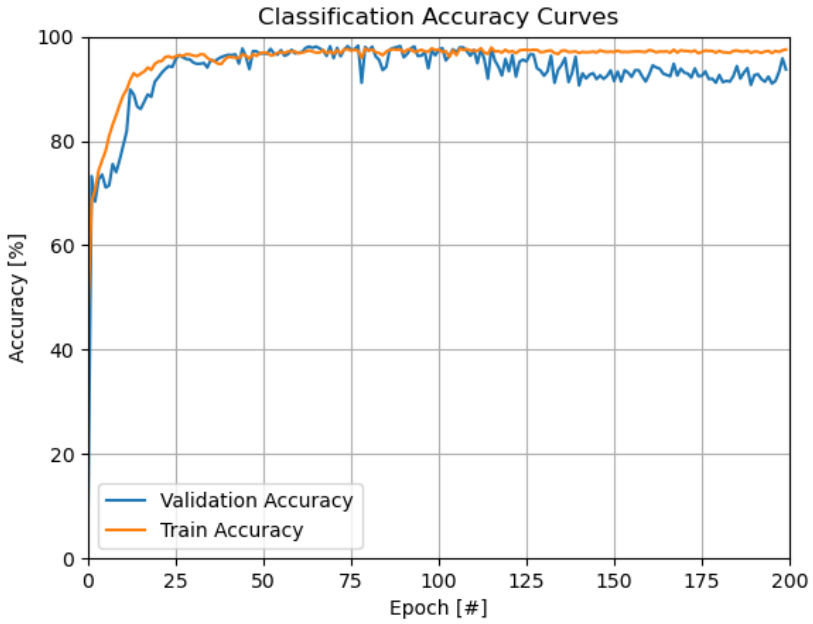


Figure 77 – The training and validation accuracy of FCNet with E 200 LR 0.001

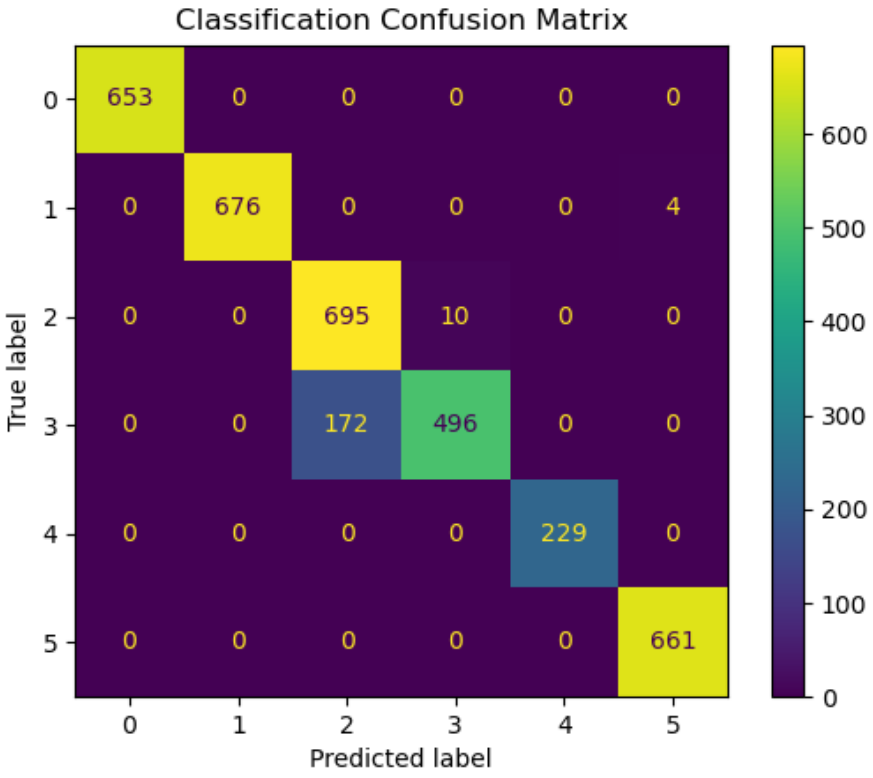


Figure 78 - Confusion Matrix of a FCNet with E 200 and LR 0.001

FCNet: 200 epochs, 0.005 learning rate

From the plots in Figure 79 and Figure 80 it is possible to see that in the first 20 epochs the curves have almost reached their steady state values, so faster than the 0.001 learning rate case.

In both loss and accuracy plots there are signs of divergency, especially around epoch 160, that means that the model is not generalizing well but is trying just to imitate the behavior of the training dataset, resulting in overfitting.

In general, during the evolution of the curves in the two plots is possible to see a very spiky behavior, this is because the learning rate is too big, but still the training was evolving well. Confusion matrix in Figure 81 is showing decent results in terms of test accuracy, confusion regarding classes 2 and 3 (baked 10 min and baked 20 min) and classes 1 and 5 (fresh and skin).

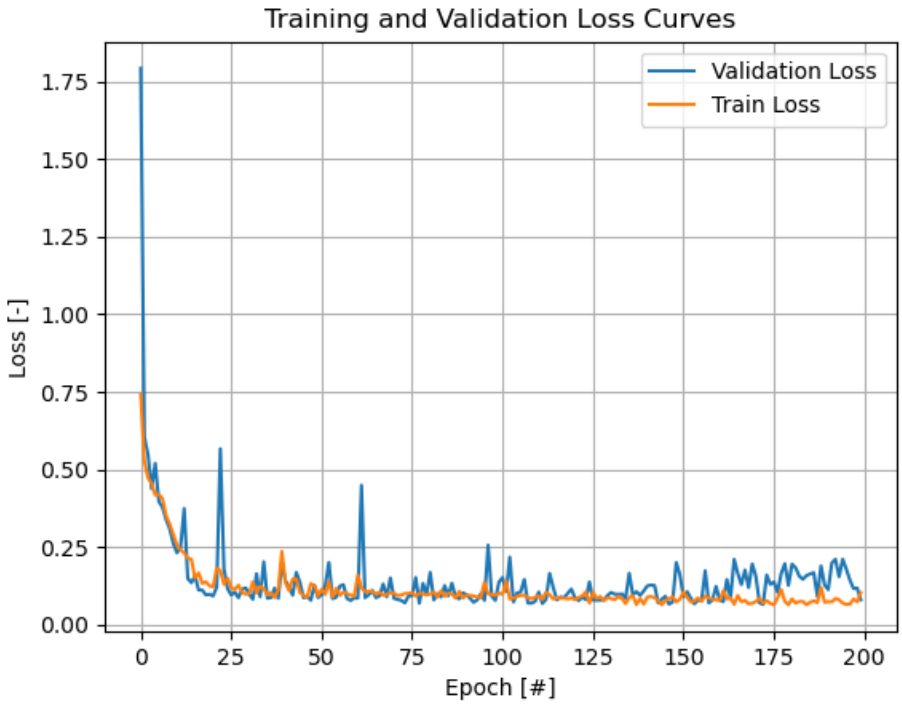


Figure 79 - The training and validation loss of FCNet with E 200 and LR 0.005

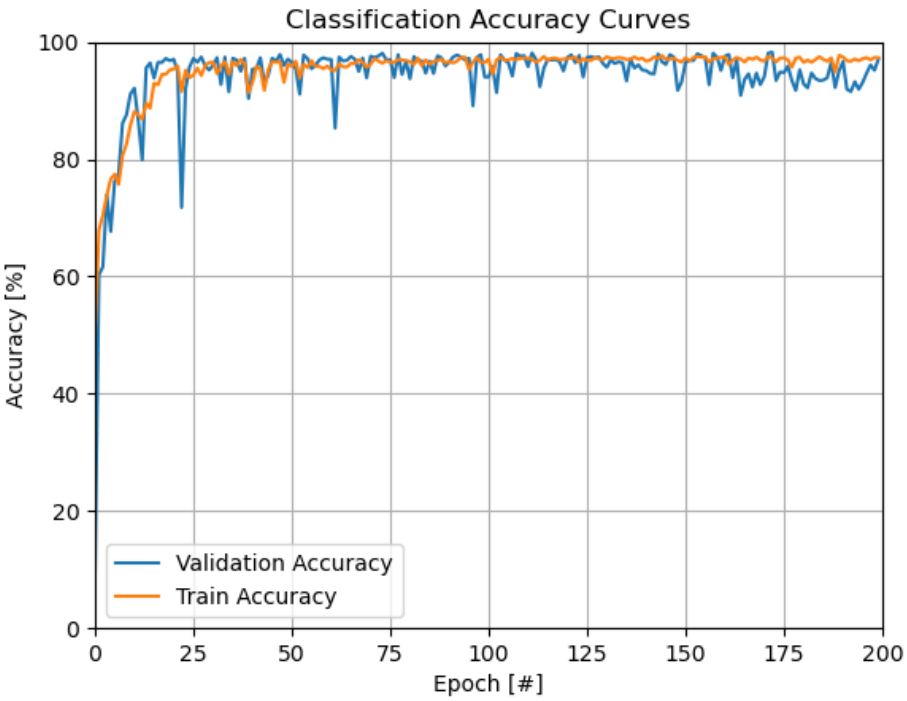


Figure 80 – The training and validation accuracy of FCNet with E 200 and LR 0.005

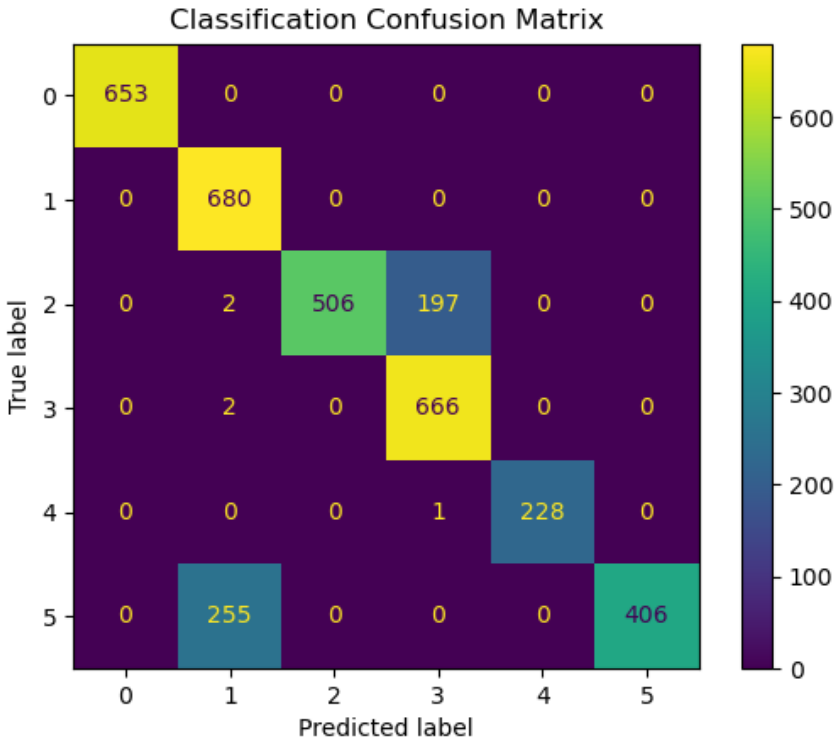


Figure 81 - Confusion Matrix of a FCNet with E 200 and LR 0.005

FCNet: 200 epochs, 0.0002 learning rate

From the plots in Figure 82 and Figure 84 it is possible to see that in the first 100 and 50 epochs the curves have almost reached their steady state values, so way slower with respect to the previous cases, this is due to the small learning rate value.

In both loss and accuracy plots there is no sign of divergency, that means that the model is generalizing well, no overfitting is occurring.

In general, during the evolution of the curves in the two plots is possible to see a very smooth behavior, this is because the learning rate is very small, the training was evolving well better than the other cases. Confusion matrix in Figure 83 is showing the best results in terms of test accuracy, just a small confusion regarding classes 2 and 3 (baked 10 min and baked 20 min).

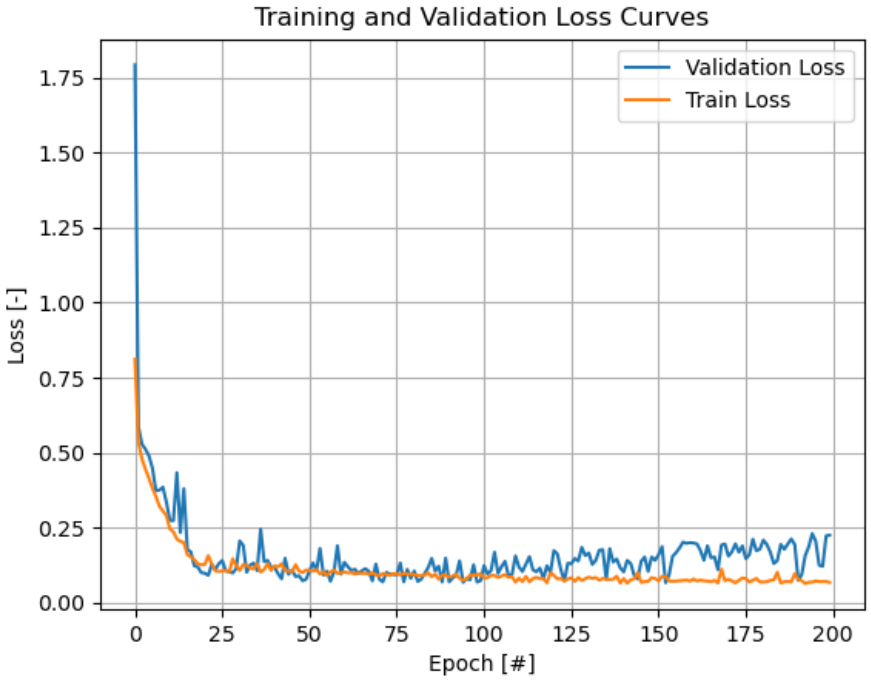


Figure 82 - The training and validation loss of FCNet with E 200 and LR 0.0002

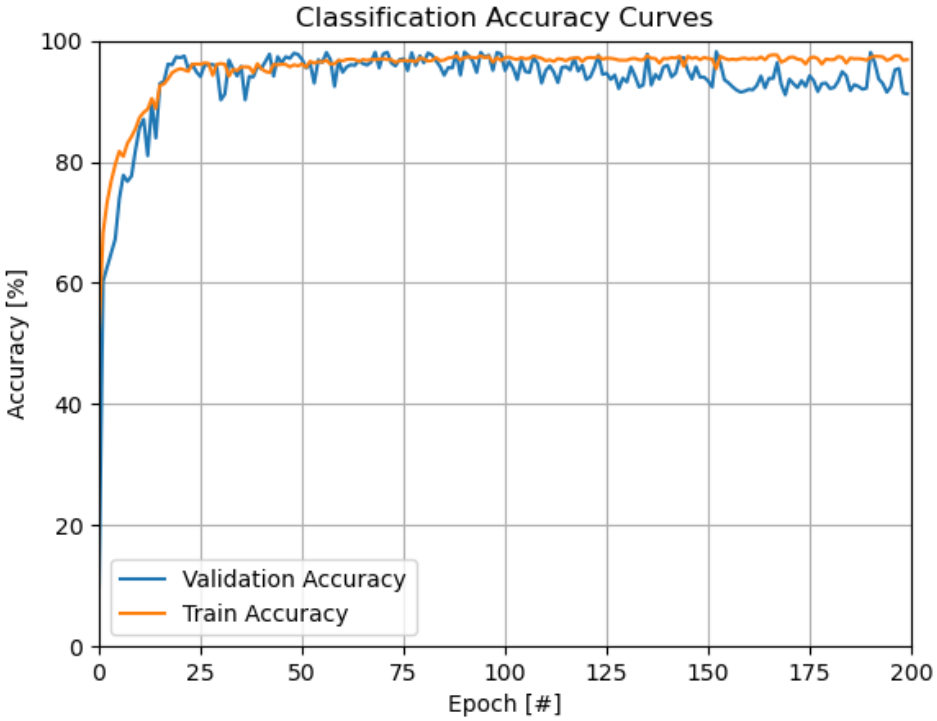


Figure 84 – The training and validation accuracy of FCNet with E 200 and LR 0.0002

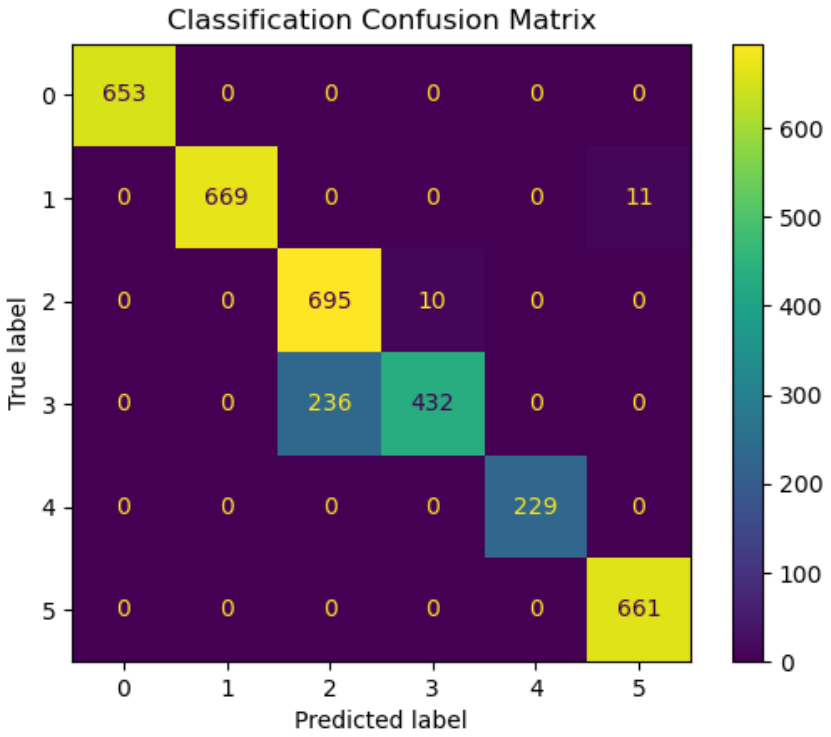


Figure 83 - Confusion Matrix of a FCNet with E 200 and LR 0.0002

4.4.2 6-folds Cross-Validation

In order, Figure 85, Figure 86 and Figure 87 are referred to the following learning rate: 0.001, 0.005 and 0.0002. The results are very consistent with a low standard deviation, and a high mean. The standard deviation in general is higher than the other models, that shows that this model is slightly less robust than the others. That leads to the conclusion that LucasNNN model is in general very robust and with a good repeatability.

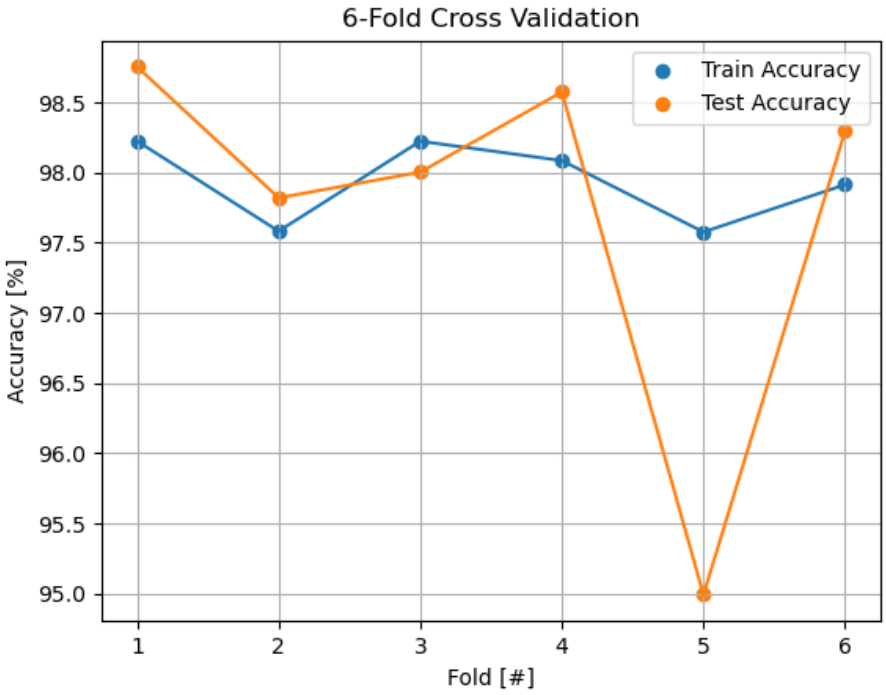


Figure 85 – Cross Validation of FCNet with E 200 and LR 0.001

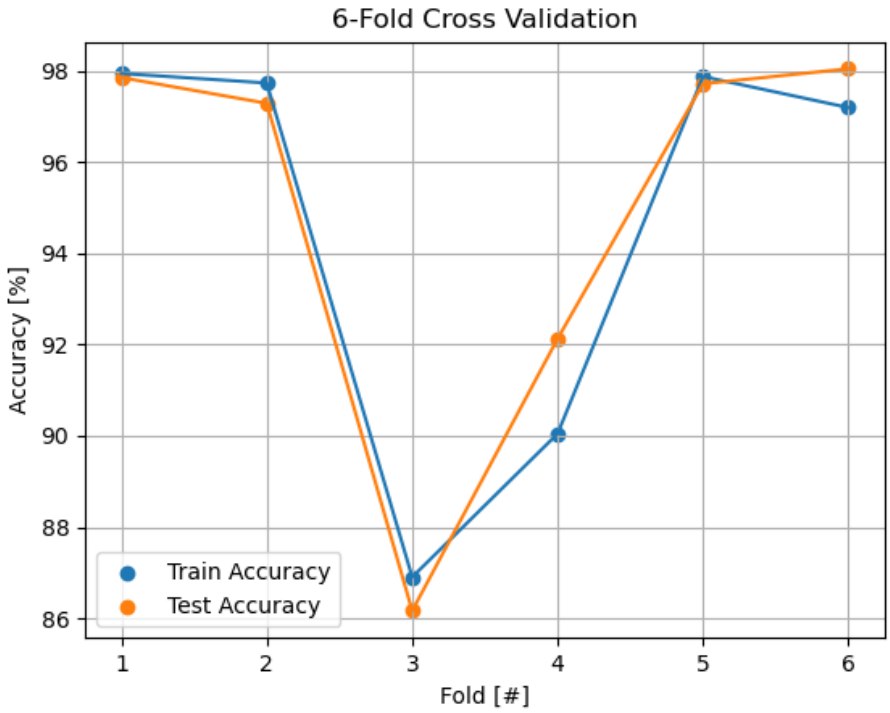


Figure 86 – Cross Validation of FCNet with E 200 and LR 0.005

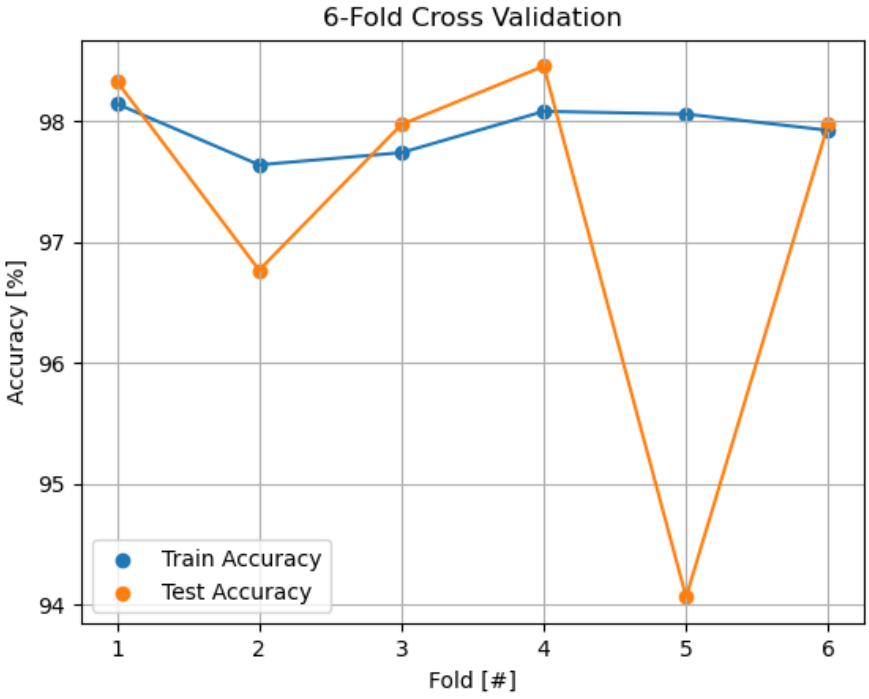


Figure 87 – Cross Validation of FCNet with E 200 and LR 0.0002

Early Stop

In this model, early stop algorithm stops earlier the training preventing from the divergencies visible in Figure 88 and Figure 90. The training is stopped at epoch 120, the experiment is done with an initial number of epochs set to 200 and a learning rate equals to 0.001. The results available in Figure 89 show better accuracy than the experiment with the same parameters but with the early stop algorithm disabled.

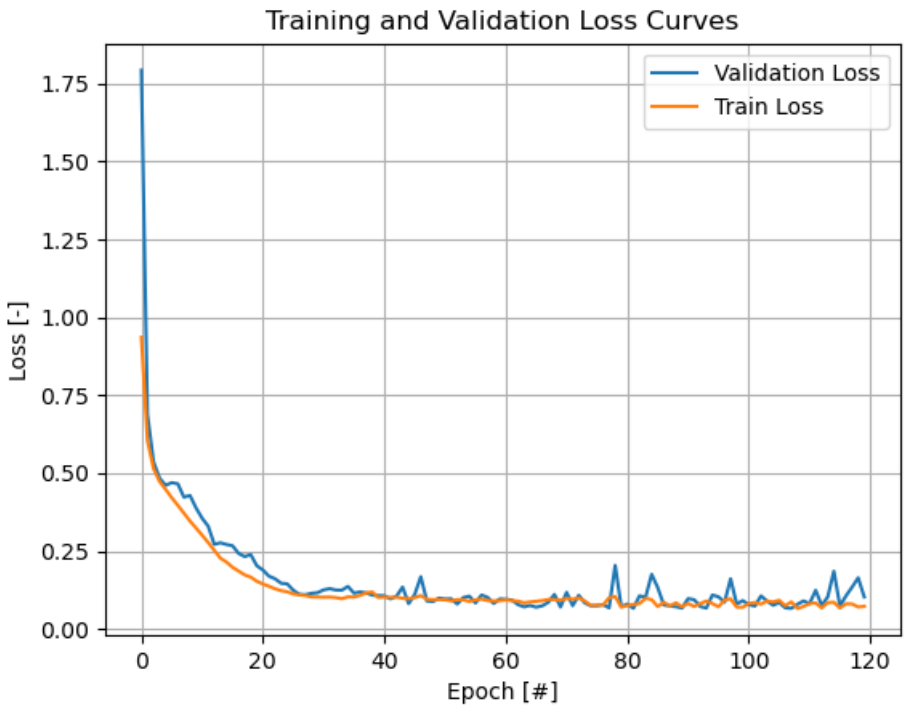


Figure 88 – The training stops at E 120 for FCNet due to early stop algorithm

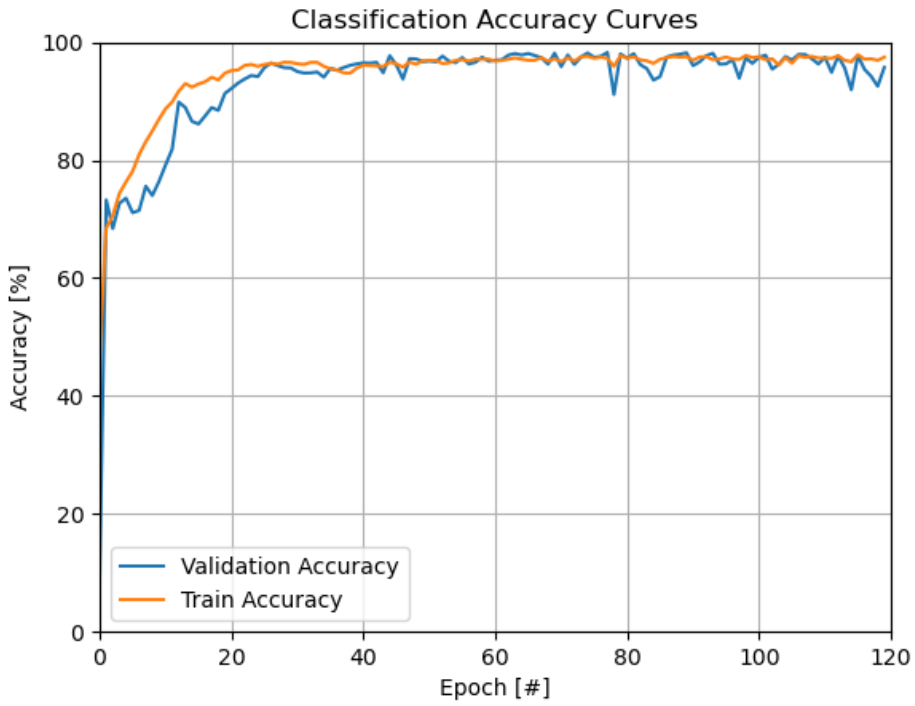


Figure 90 - The training stops at E 120 for FCNet due to early stop algorithm

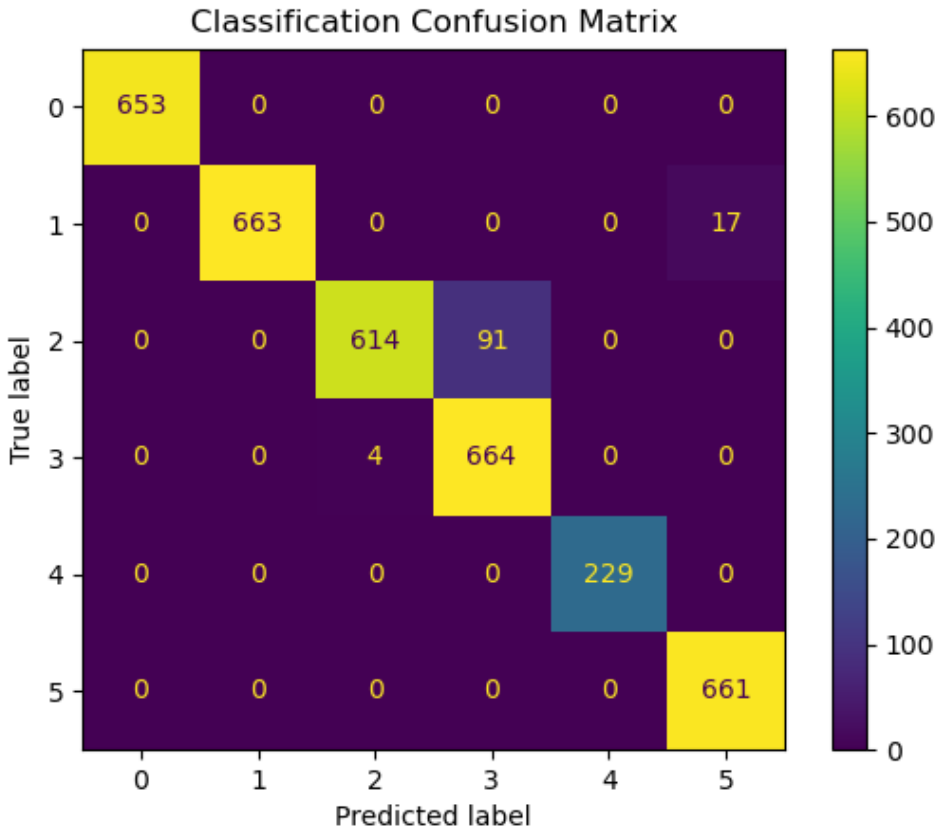


Figure 89 –Confusion Matrix of a FCNet that stop at E 120 due to early stop

4.4.3 Discussion

From Cross Validation experiments is possible to say that the model in general is very robust and reliable to train. Looking at the train, validation and test part is clear that the number of epochs could have been also higher than 400 to output even better results. The learning rate which seems the best in terms of tradeoff between spiky curves and speed of training is 0.001. That learning rate also represents the smaller standard deviation in the cross-validation experiments.

4.5. Comparison between models

From the experiments conducted in the previous sub-chapters, some comparison can be discussed. First, all the train-validation time's experiments for each model with 200 epochs is averaged. It is shown that FCNet results the fastest train-validation time, followed by Hu et al, LucasNNN, and TwoDCNN. However, from the related average of the test accuracy, it is interesting to note that FCNet performed the worst. Model that is relatively fast to train, but has the highest accuracy on the test data is Hu et al. On the other hand, even though TwoDCNN has the second highest accuracy on the test data, it is taking significantly long to train compared to other models. While for LucasNNN is moderate both in training time and test accuracy.

As the generalization performance, it can be deducted from the cross-validation results. First, it is important to note that all LucasNNN, TwoDCNN, and Hu et al models almost have similar average accuracy for the cross validation, while FCNet has the lowest accuracy. Moreover, FCNet also performed worst in average cross validation standard deviation, while for the other three are almost the same. However, FCNet is the fastest to train compare the other model. While for TwoDCNN, is again also the longest time to train.

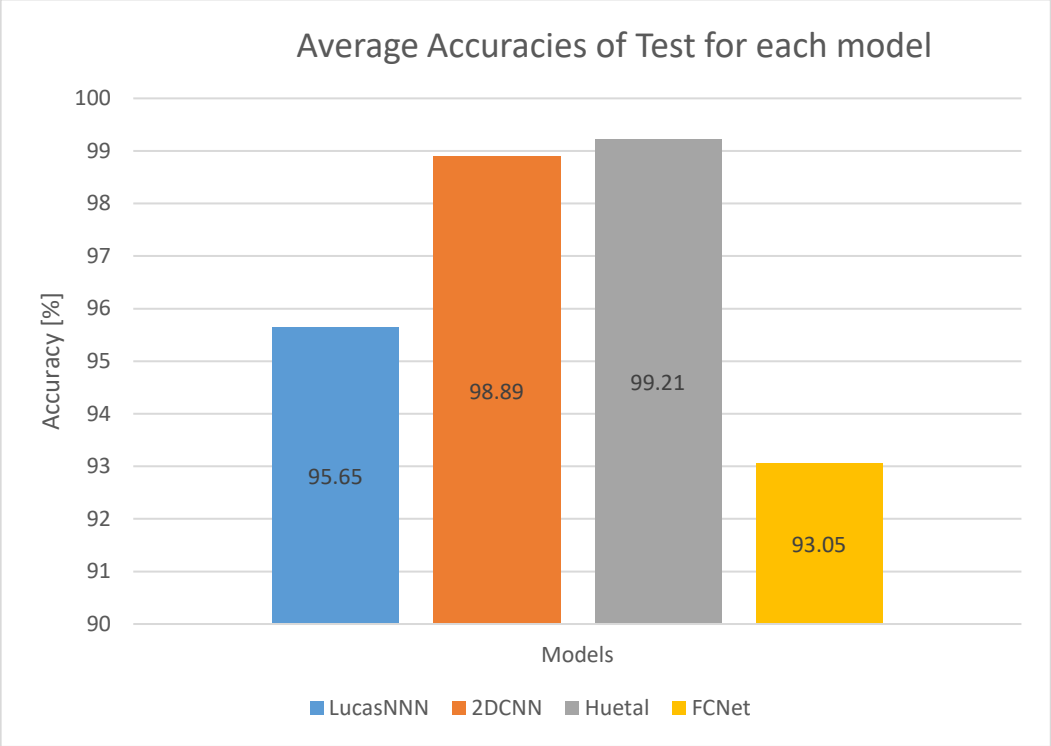


Figure 92 –Average Accuracy of the test for each model with E 200

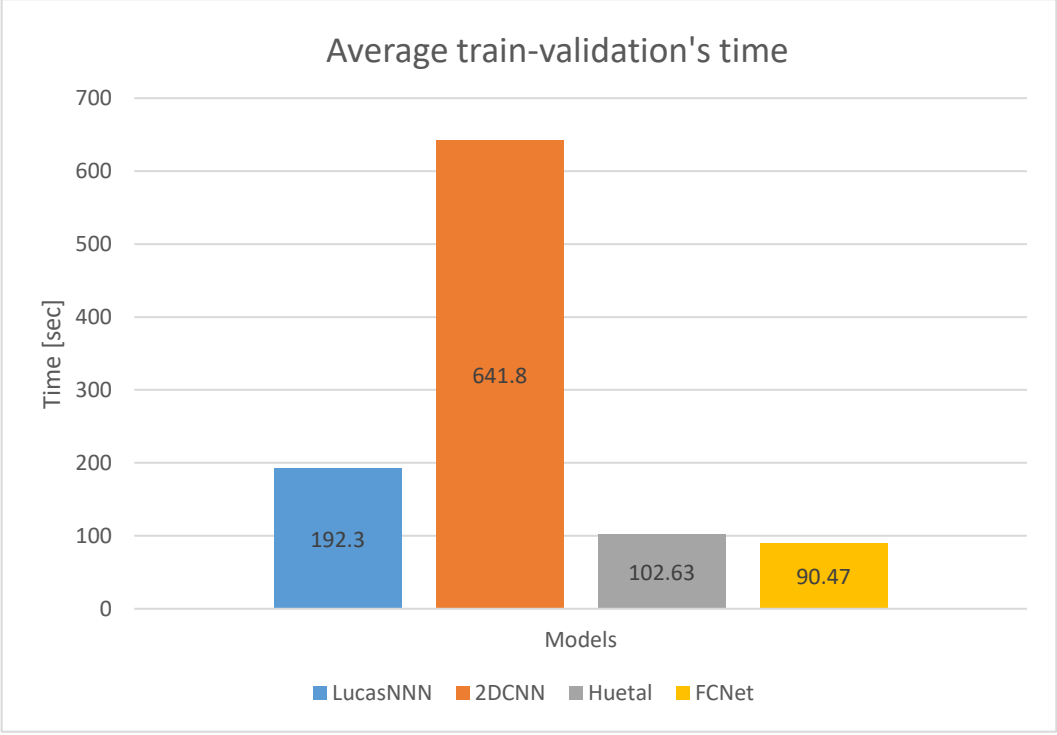


Figure 91 –Average train-validation processing time for each model with E 200

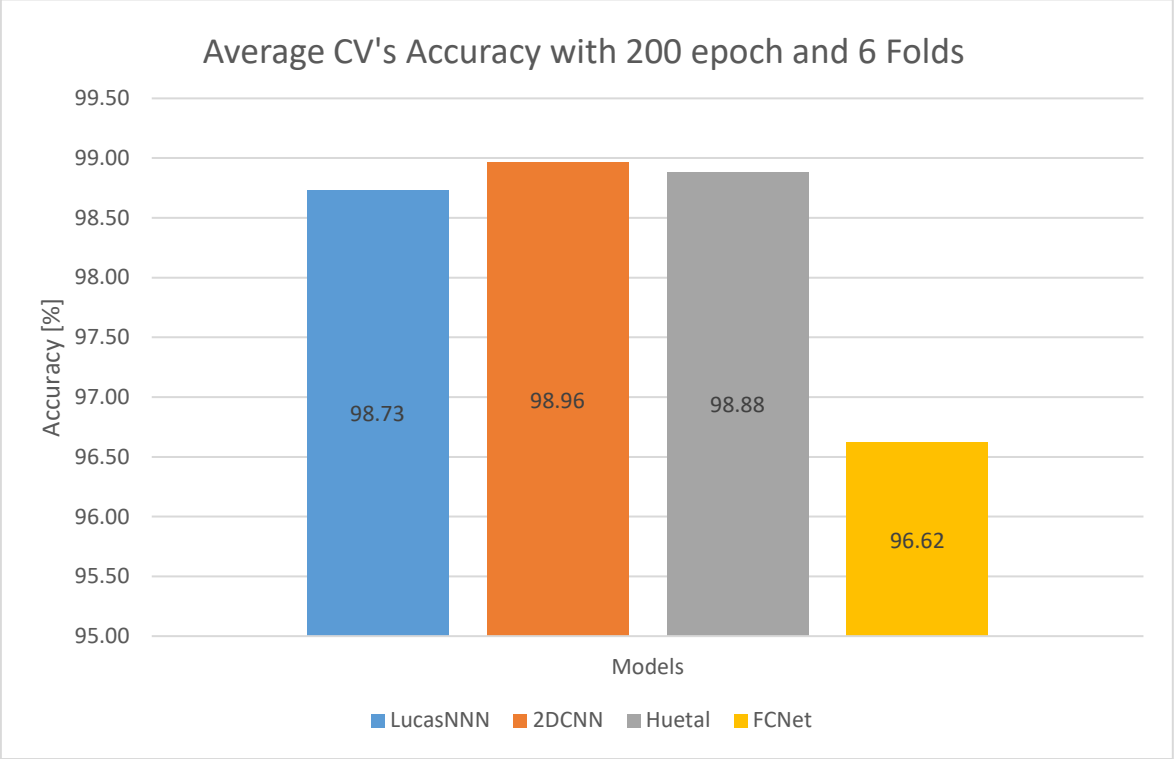


Figure 94 –Average CV accuracy for each model with E 200

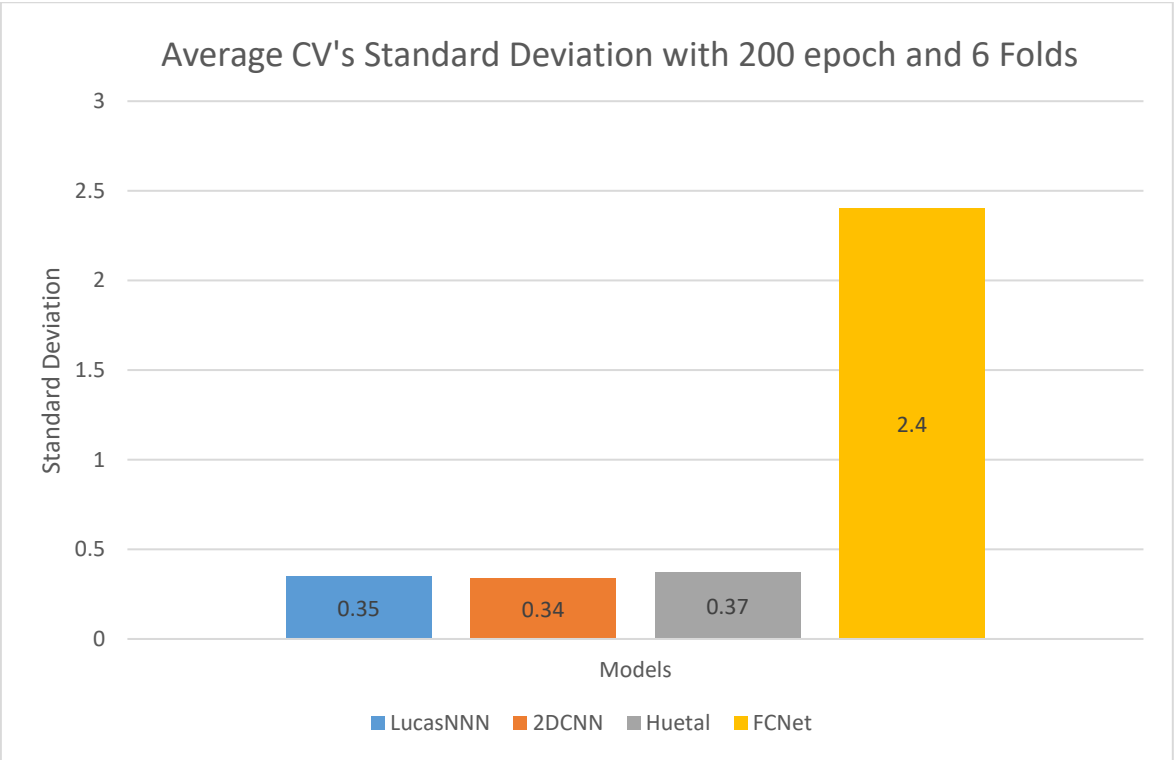


Figure 93 –Average Standard Deviation for each model with E 200

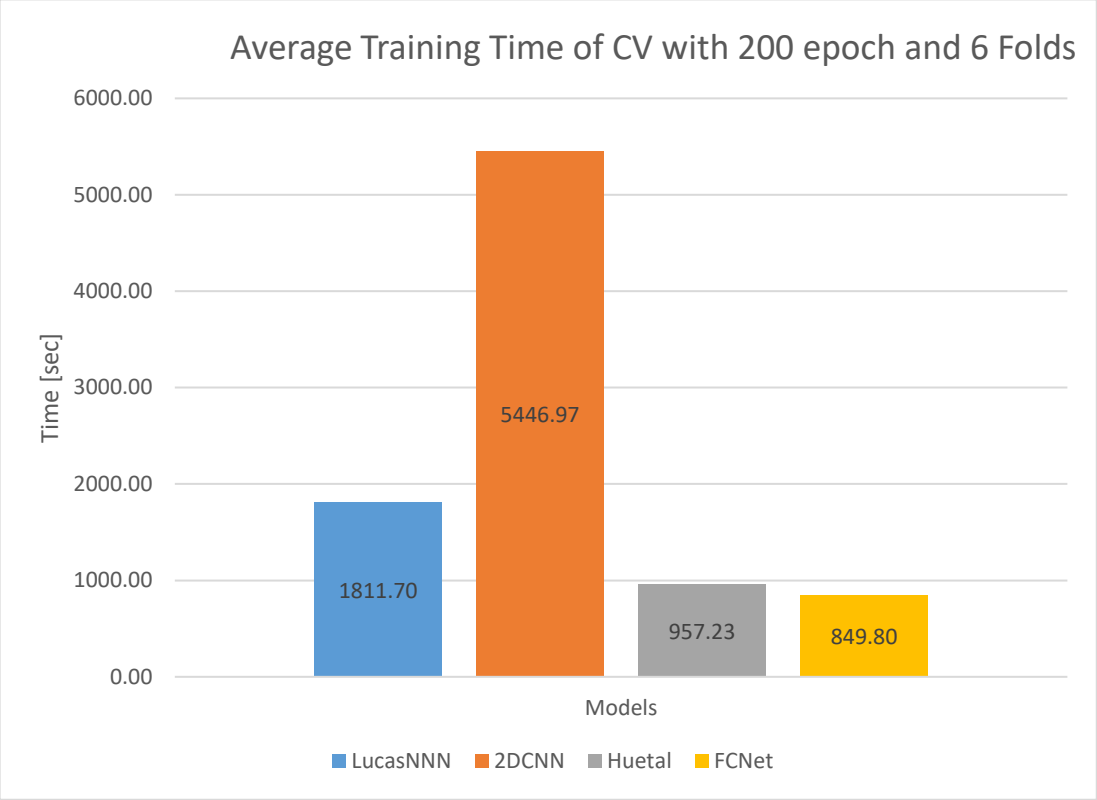


Figure 95 –Average training time for each model with E 200

Finally, all the trained weights from the best combination were tested to create an intuitive false color image to show the classification results as can be seen from Figure 96. Here one big point raised, in fact it seems that apart from FCNet, all the other models confuse some part of the background with burnt class. The images are taken from the outputs of weights coming from a training with a balanced dataset. Balanced dataset for this work means that if the class with less samples has N samples, than all the other classes must have at maximum 300% of N samples inside the dataset used for training, for simplicity referred as "300" spc (sample per class) value. Thus, a new experiment is done using more spc and the results can be seen from Figure 97. By considerably increasing the spc, way better results appear from all the models.

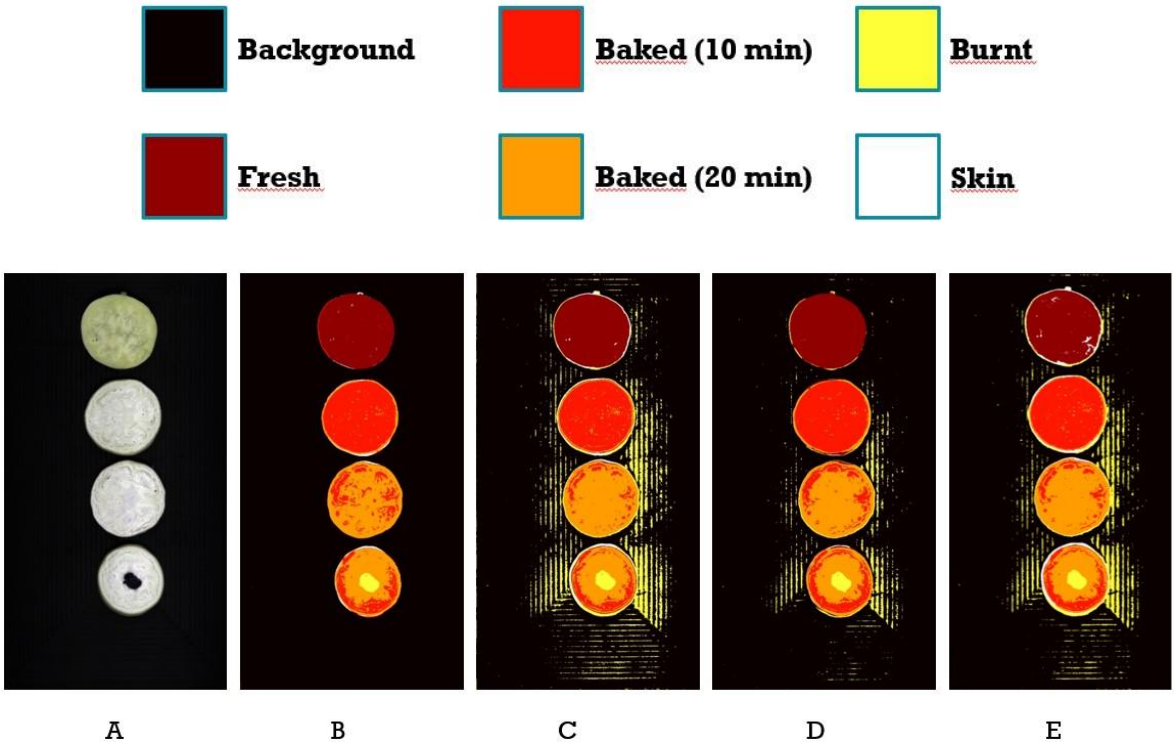


Figure 96 - Classification results using spc = 300: A) False color image from FX17; B) FCnet; C) LucasNNN; D) Huetal; E) 2DCNN

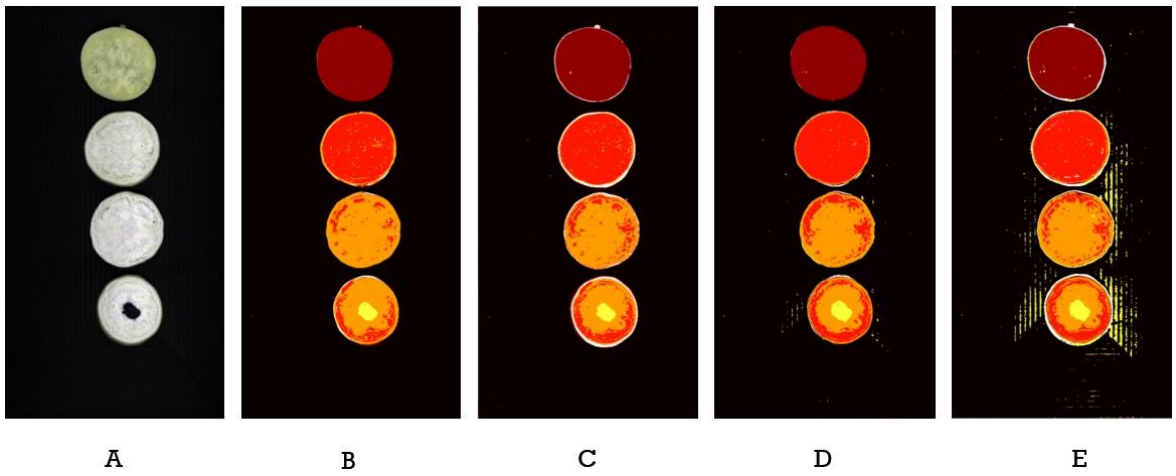


Figure 97 - Classification results using spc = 700: A) False color image from FX17; B) FCnet; C) LucasNNN; D) Huetal; E) 2DCNN

5. Conclusions

This Master Thesis project achieved these important milestones in the application of Hyperspectral Imaging and Deep Learning for performing Food Quality investigations.

1. All the tested networks show very good performance for the semantic segmentation task for automatic food quality inspection with accuracy all above 90%. Furthermore, the result from 6-Fold Cross-Validation confirms also that all the neural networks are robust to different random dataset.
2. In more detail, Hu et al. [10] performed best with average accuracy for the test at 99.25%. It means that statistically speaking, not a single pixel from 100 pixels in semantic segmentation task is misclassified. Furthermore, the FCNet, which is the worst model, still performed good for industrial application, with average test accuracy at 94%. Only six misclassified pixels out of 100 possible pixels. This is a good result for food quality inspection, as almost impossible for the algorithm to miss an eggplant that has the bad region, because it must be taken at least dozens of pixels.
3. Moreover, the generalization performance of the models is also good. The 6-Fold Cross Validation results shows that the model is robust for randomly selected eggplants pixel. Moreover, it is hard to come across significant differences between LucasNNN, TwoDCNN, and FCNet models. All are having average CV accuracy around 98% with standard deviation around 0.3. Not more than two pixels on about 100 pixels in semantic segmentation task are misclassified. However, once again FCNet model underperformed compared to the other models with accuracy at 96.62% and standard deviation at 2.4%.

There is some suggestion for the next research for this thesis topic:

1. Upgrade the GT creation program such that it is possible to select multiple areas related to the same class from a single picture, or even better multiple areas from different classes from the same picture.

2. Upgrade the program into real time GT classification.
3. Increase the semantic segmentation with other possible useful classes (such as rotten eggplants).
4. Converts the programs from Pytorch to Tensorflow because it is used more for industrial application and has better deep learning packages (i.e., the cross validation is less hard coded).

Finally, it is reasonable to say that Hyperspectral Image Systems with the combination of Deep Learning algorithm result in really good performance for food industry sorting application. Moreover, it is also proud to say that this master thesis has provided very good tools to analysis the HSI data, since the creation of the ground truth until the testing of the model. Hence, the next researcher can focus more with the data analysis rather than building the program from scratch.

Bibliography

[1] Y. Lu, W. Saeys, M. Kim, Y. Peng, and R. Lu, "Hyperspectral imaging technology for quality and safety evaluation of horticultural products: A review and celebration of the past 20-year progress," *Postharvest Biol. Technol.*, vol. 170, no. August, p. 111318, 2020, doi: 10.1016/j.postharvbio.2020.111318.

[2] D. Sun, G. Elmasry, and D. Sun, *Hyperspectral Imaging for Food Quality Analysis and Control Principles of Hyperspectral Imaging Technology*. Dublin: Academic Press, 2010.

[3] Y. Chen, K. Chao, and M. S. Kim, "Machine v ision technology for agricultural applications," vol. 36, pp. 173–191, 2002.

[4] B. Zhang *et al.*, "Principles, developments and applications of computer vision for external quality inspection of fruits and vegetables: A review," *Food Res. Int.*, vol. 62, pp. 326–343, 2014, doi: 10.1016/j.foodres.2014.03.012.

[5] P. M. Mehl, K. Chao, M. Kim, and Y. R. Chen, "Detection of defects on selected apple cultivars using hyperspectral and multispectral image analysis," *J. Agric. Saf. Health*, vol. 18, no. 2, pp. 219–226, 2012, doi: 10.13031/2013.7790.

[6] R. Moscetti *et al.*, "Hazelnut Quality Sorting Using High Dynamic Range Short-Wave Infrared Hyperspectral Imaging," *Food Bioprocess Technol.*, vol. 8, no. 7, pp. 1593–1604, 2015, doi: 10.1007/s11947-015-1503-2.

[7] X. Cheng, Y. R. Chen, Y. Tao, C. Y. Wang, M. S. Kim, and A. M. Lefcourt, "A novel integrated PCA and FLD method on hyperspectral image feature extraction for cucumber chilling damage inspection," *Trans. Am. Soc. Agric. Eng.*, vol. 47, no. 4, pp. 1313–1320, 2004, doi: 10.13031/2013.16565.

[8] W. Lv and X. Wang, "Overview of Hyperspectral Image Classification," *J. Sensors*, vol. 2020, 2020, doi: 10.1155/2020/4817234.

[9] A. Ben Hamida, A. Benoit, P. Lambert, and C. Ben Amar, "3-D deep learning approach for remote sensing image classification," *IEEE Trans. Geosci.*

Remote Sens., vol. 56, no. 8, pp. 4420–4434, 2018, doi: 10.1109/TGRS.2018.2818945.

[10] W. Hu, Y. Huang, L. Wei, F. Zhang, and H. Li, "Deep convolutional neural networks for hyperspectral image classification," *J. Sensors*, vol. 2015, 2015, doi: 10.1155/2015/258619.

[11] S. K. Roy, G. Krishna, S. R. Dubey, and B. B. Chaudhuri, "HybridSN: Exploring 3D-2D CNN Feature Hierarchy for Hyperspectral Image Classification," *arXiv*, vol. 17, no. 2, pp. 277–281, 2019.

[12] Z. Wang, M. Hu, and G. Zhai, "Application of deep learning architectures for accurate and rapid detection of internal mechanical damage of blueberry using hyperspectral transmittance data," *Sensors (Switzerland)*, vol. 18, no. 4, pp. 1–14, 2018, doi: 10.3390/s18041126.

[13] Y. Zhang, J. Lian, M. Fan, and Y. Zheng, "Deep indicator for fine-grained classification of banana's ripening stages," *Eurasip J. Image Video Process.*, vol. 2018, no. 1, 2018, doi: 10.1186/s13640-018-0284-8.

[14] J. Steinbrener, K. Posch, and R. Leitner, "Hyperspectral fruit and vegetable classi fi cation using convolutional neural networks," *Comput. Electron. Agric.*, vol. 162, no. October 2018, pp. 364–372, 2019, doi: 10.1016/j.compag.2019.04.019.

[15] F. Liu, L. Snetkov, and D. Lima, "Summary on fruit identification methods : A literature review," *Adv. Soc. Sci. Educ. Humanit. Res.*, vol. 119, no. ESSAEME, pp. 1629–1633, 2017.

[16] Y. E. Esin, O. Ozdil, S. Ozturk, and B. Demirel, "Practical Focus Adjustment Method for Hyperspectral Cameras," 2019.

[17] J. Naranjo-Torres, M. Mora, R. Hernández-García, R. J. Barrientos, C. Fredes, and A. Valenzuela, "A review of convolutional neural network applied to fruit image processing," *Appl. Sci.*, vol. 10, no. 10, 2020, doi: 10.3390/app10103443.

[18] E. Stevens and L. Antiga, *Deep Learning with PyTorch Essential Excerpts*. 2019.

[19] F. M. Riese and S. Keller, "SOIL TEXTURE CLASSIFICATION with 1D CONVOLUTIONAL NEURAL NETWORKS BASED on HYPERSPECTRAL DATA," *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. 4, no.

2/W5, pp. 615–621, 2019, doi: 10.5194/isprs-annals-IV-2-W5-615-2019.

Appendix

All the tools described in this section are developed in Python, the list of programs strictly related to this work are: tvt_tool, kfold_cv_tool, play_with_data, spectra_plotter, false_color and normalization_tool.

Tvt_tool

This code aims to train the model chosen by the user following the classic machine learning training workflow: training, validation, testing.

Inputs

- Input Dataset (Data and GT in .mat)
- Model to train
- Hyperparameters (number of epochs, batch size, learning rate, weight decay, number of samples per class)
- Bands to take into account
- Percentages to split the dataset (train, validation, test percentages)
- Classes to do not take into account
- Save the weights (True/False)
- Use an Early Stop Algorithm (True/False)
- Use a scheduler to dynamically change the learning rate (True/False, set the weight decay)

Outputs

- Plot with the Average Training and Validation Accuracy

- Plot with the Average Training and Validation Loss
- Confusion Matrix (referred to the results from testing)
- Time occurred to train/validate the NN
- Time Occurred to test the NN
- Testing Accuracy
- NN model with the weights trained up to last epoch

Kfold_cv_tool

This code aims to validate both the dataset and the training model, it tests the reliability of the results through a k-fold Cross Validation algorithm. It only trains and tests the NN.

Inputs

- Input Dataset (Data and GT in .mat)
- Model to train
- Hyperparameters (number of epochs, batch size, learning rate, weight decay, number of samples per class)
- Bands to take into account
- Classes to do not take into account
- Select K, so how many folds to use to apply the K-fold Cross Validation technique
- Use a scheduler to dynamically change the learning rate (True/False, set the weight decay)

Outputs

- Plot with the final Training and Testing Accuracy values from each of the K training processes
- Average Testing Accuracy between the accuracies resulting from the k training processes

- Standard Deviation Testing Accuracy between the accuracies resulting from the k training processes
- Time occurred for the whole k-Fold cross validation process

Play_with_data

This code aims to create the ground truth necessary to train model, the user needs to create two folders first. In one folder are saved the png images, in the other one are saved the corresponding mat files.

Inputs

- Directory to the folder containing png images
- Directory to the folder containing mat files

Outputs

- Mat file containing a list of spectra
- Mat file containing a list of integers, each integer represents the class of the spectrum in the corresponding position in the previous mat file

Spectra_plotter

This code aims to plot the spectra of different classes in a 2-dim graph, x-axis represents the frequencies while y-axis represents the intensities. Each line is not a single spectrum but the mean of all the spectra considered for that class, in transparency is also represented the standard deviation.

Inputs

- Data and ground truth
- Classes whose spectra should be plotted

- Frequencies to do not consider when plotting the spectra

Outputs

- Graph with the spectra in terms of mean and standard deviation

Normalization_tool

This code aims to preprocess the data normalizing them with the following formula (I - Ib)/(Iw - Ib).

Inputs

- Mat file with inside: white ref, black ref, hyperspectral data

Outputs

- Mat file containing just the normalized hyperspectral data