I hereby declare that this work has not been submitted for any other degree/course at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original and entirely the result of my own work at the University of Strathclyde under the supervision of Paul Murray."

Hand Identification system for RGB and hyperspectral images that can recognise individuals from hand images

Konstantinos Gkogkalatze

Reg. Number: 201922539

Supervisor: Dr Paul Murray

Date: 31/03/2023

Abstract

This project aimed to create an algorithm for person identification based on RGB and hyperspectral dorsal hand images. The algorithm uses a less complex approach that delivers accurate results in various datasets with different backgrounds and light conditions. The RGB images were pre-processed by removing the background and using a guided filter with a high pass filter, while PCA was applied to the first three principal components of the hyperspectral images. This project used the pre-trained VGG16 CNN to identify hand features and train the model. The model was then evaluated using a variety of metrics and preprocessing methods. The outcomes demonstrated that the best accuracy and loss were generated by the high-detail BGR preprocessing technique. For the 11k hands dataset, the 11k hands algorithm did better than the suggested solution, but it was less accurate when it applied to the combined dataset. However, the proposed algorithm showed better performance in the hyperspectral dataset in comparison to RGB datasets. It showed some bias towards hand posture and position, which could be addressed using a Support Vector Machine (SVM) model in future work. This project offered useful insights into different algorithms and showed the potential of pre-processing and pre-trained models for hand human recognition. The outcomes could be helpful for other jobs like gesture recognition, estimating hand poses, and computer vision training and research.

Acknowledgements

I would like to thank Dr Paul Murray from the Electronic and Electrical Engineering department for the opportunity to take this project and for the help and support given during duration of the project.

I would also like to thank Dr Andrew John Cambell for the help and the suggestions given about the project.

Lastly, I would like to thank Dr Mahmoud Afifi, The Hong Kong Polytechnic University, and the University of Las Palmas de Gran Canaria for the hand datasets that they provided.

Table of Contents

Table of Figures

Index of Tables

1. Introduction

Nowadays, phones, computers or other devices can get unlocked by identifying the face of the user. Those devices use image processing and machine/deep learning techniques that detect and recognise human beings by their face biometric features. However, sharing a person's face with a device is something that many people feel uncomfortable with due to privacy concerns that has been highly mentioned in the public domain [1]. So, what if those devices can get unlocked by just viewing a person’s hand, a much more comfortable and convenient way?

Deep learning algorithms are used more frequently in this day and age. They showed promising results in image processing tasks like object and face recognition. Convolutional Neural Networks (CNNs) are also types of deep learning that learn directly from the data. They have a huge advantage compared to older neural network architectures, as they detect the features needed automatically and with no need for human supervision [2].

This project focuses on the design, implementation, and testing of image processing techniques for hand recognition. The research started with finding hand biometrics features that are essential in identidying human beings. Those biometrics can be the shape of the hand, ridge patterns, skin texture, creases and wrinkles, and knuckle patterns [3], [4]. The above biometrics can be extracted from both RGB and hyperspectral images, with the appropriate preprocessing techniques for each of them. However, there is one biometric that can only be studied and analysed for hyperspectral images, and that is the vein patterns. Vein recognition is emerging as an effective method for personal identification, as the vein pattern for every person is unique and unrepeatable [5].

After the preprocessing process, a compatible CNN model was deployed for feature extraction and then was trained on those features. The model was then evaluated under different hyperparameters to obtain its best version and used different manual datasets to test its functionality in unseen data with different backgrounds.

1.1. Background

Image processing is commonly used in a variety of applications, including image reconstruction, traffic sensing technology, and medical image reconstruction. [6]. One of the most important applications of image processing that is used today is face recognition, which has been the focus of research for many years [7]. Recently, there is an increasing interest in being able to recognise humans in other ways, including unique features on their hands [8]. It anticipated that techniques like those used for facial recognition could be extended to hand recognition.

Some algorithms that have been produced for hand identification include hand geometry measurements and many deep learning approaches. The most popular algorithms are the Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs). However, these algorithms are extremely complex, and require huge computational costs, resulting sometimes in inefficient resource consumption. Also, they are modified and trained for specific hand image databases, making them inaccurate for new and unseen data.

Most hand identification image processing systems are using RGB images, as those images are used more frequently in the everyday life. The widespread use of RGB images in hand recognition systems is justified by the ease of acquisition However, RGB images have their limitations in terms of identifying a human being. In more details, those images have restricted spectral information as they are composed of three spectral channels, red (around 700 nm), green (around 550 nm), and blue (around 450 nm) [9]. This can be a disadvantage when we try to identify hands in different environmnets, and it is the reason why we introduced hyperspectral image recognition in the late stages of this project. Since hyperspectral images can capture spectral information at hundreds or thousands of wavelengths, providing more spectral data that can be used to identify hands more accurately. Therefore, using hyperspectral images can lead to higher evaluations in different environmental conditions.

1.2. Motivation

The motivation of this project is to develop a more accurate and robust hand identification system that can work both on RGB and hyperspectral images.

There is limited resource online about algorithms that can identify people by hand images. Face recognition is much more popular in this, as it has been used more frequently in our everyday life. However, this carries many risks as data privacy, and sharing our faces online can sometimes be uncomfortable. Creating a system that recognises people from hand images can eliminate those factors, as human beings are not able to identify people from their hand with their bare eyes.

Although RGB images are used in machine vision systems, they have limited ability to identify differences in skin colour and texture. However, hyperspectral images can capture skin features that are not visible in RGB images. They provide more detail and can better identify changes in skin colour and texture. Using both types of images can yield an efficient and accurate hand recognition algorithm that can then be used in a variety of applications.

This system can be used to improve image processing techniques for hand recognition and identification. It can be useful not only for hand identification but for many other tasks like gesture recognition and sign language recognition. It could be, also, a starting point for future improvement of hand identification system. In addition, this system can be used in a variety of applications, including security authentication, biometrics recognition, and privacy improvement.

1.3. Research aims and objectives

The four research objectives of this project are: 1) to study and understand the biometrics required for hand recognition; 2) to find and acquire images of hands from people of different ages, genders, and skin; 3) to explore RGB image processing techniques and for hand recognition using hyperspectral imaging; 4) and to develop an accurate hand recognition system that can be used in a variety of applications.

In order to study and understand the biometric techniques required for hand recognition, the research needs to delve into the principles of RGB image processing techniques used for different tasks, such as hand, face, and object detection and recognition. This includes reading and understanding research papers that discuss the most important features that can be used to identify a person. In addition, to find and acquire hand images from people of different ages, genders, and skins, the research used images of hands from publicly available sources. This can help us to develop an algorithm capable of analysing these images and automatically identifying key features of hand images. With a working algorithm for RGB images, it can be implemented into hyperspectral images so that higher evaluation metrics can be achieved.

To achieve these goals, a commitment to extensive research on biometrics necessary for hand recognition is required. For the development of image processing algorithms, existing hand recognition solutions need to be researched, tested and evaluated. Online resources on hand recognition are limited, so there is also a need to study other detection and recognition systems that have similar techniques and structures that require the use of hands to identify people.

Finally, the project hopes to develop an accurate hand recognition system that can be used for a variety of applications. In addition, it may be a valuable starting point for future improvements and implementation of hardware applications with a user-friendly graphical user interface.

1.4. Overview of the proposed solution

The proposed solution relies on a Convolutional Neural Network (CNN), specifically the pre-trained VGG16 model. This gets as input pre-processed image that contain hand features, ideal to recognise people from hand.

For RGB images, the proposed pre-processing step involves applying a guided filter to the original image, and then applying a high pass filtering. This helps obtain high-detail views of hands which are then fed into CNN's network. By doing so, more features on each hand become visible for detection and classification by the system, thus improving its accuracy and dependability when identifying and classifying hands.

For hyperspectral images, the proposed solution involves applying Principal Component Analysis (PCA) to only keep the first three principal bands of the original image. This reduces its dimensionality, which is then fed into the same CNN network as RGB images. By doing so, this network can learn to recognise and classify hand features for both RGB images and hyperspectral images.

This project utilised a CNN network with pre-trained VGG16 model. A flatten layer was applied to the output of the VGG16 model, followed by dense ReLU and another dense layer for classes. This architecture has been proven highly effective for image recognition tasks, making it ideal for our proposed solution.

The above image processing algorithm for hand identification is expected to be highly accurate and dependable, capable of recognising and classifying hand features in both RGB and hyperspectral images. Using a pretrained VGG16 model in combination with the appropriate preprocessing of RGB images and hyperspectral images should significantly boost its precision and efficiency, making it an effective solution for hand identification tasks.

1.5. Project Plan

The development of the proposed biometric hand identification system lasted two semesters. The first semester focused on research, testing, and evaluation of existing solutions. Research was also done in deep learning and especially for CNN, in order to understand how they work. In weeks 6-10, existing hand recognition solution were comprehended, evaluated with different datasets. In the end of semester 1, research papers on biometric hand identification were studies. And during weeks 11 and 12, the interim report was written.

The second semester focused on developing the image processing algorithm for hand identification. In week 1, the biometrics needed were decided, and started developing the network model for hand recognition. Four weeks were given for the development of the initial design of the RGB algorithm. After that, two weeks were spent on improving and enhancing the algorithm, by using a pretrained CNN model, meanwhile trying other pre-processing techniques. The algorithm was not as accurate as expected, and in order to improve it, more time was needed. So, between weeks 7-9 it was decided to move to the development of the algorithm for hyperspectral images. Thereafter, two weeks were given to write the report and complete some evaluation test for the algorithm.

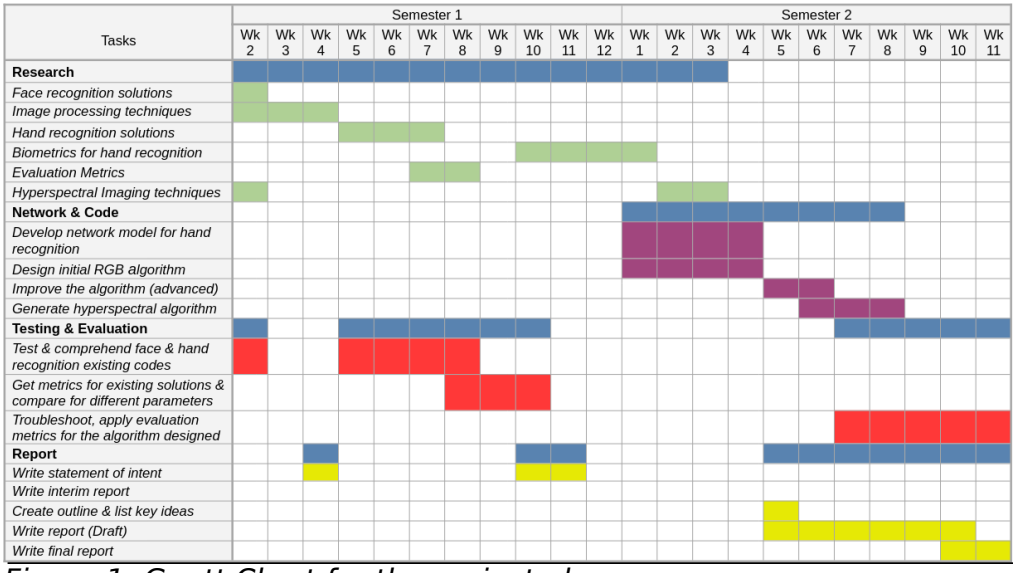


Figure 1: Gantt Chart for the project plan

2. Literature Review

2.1. Overview of image processing algorithms for hand identification

Hand identification is a very complex technique with applications ranging from biometric identification to gesture recognition. In recent years, several algorithms have been developed to accurately identify and analyse hand images.

One such algorithm is the "11k hands" [10]. This algorithm utilises a large dataset of hand images containing over 11,000 images of hands, captured in different poses and under varying lighting conditions. The hand images were pre-processed using a high pass filter from a normalised and a guided filtered image. The algorithm uses a two-stream CNN network to extract features from the images, which are then used for gender recognition and biometric identification. The first stream focuses on learning global features of the hand, and second stream deals with the local features of the hand. In order to classify new hand images according to their gender and biometric characteristics, the algorithm first trains the CNN on a subset of the hand images. It has been demonstrated that the algorithm performs well in both gender recognition and biometric identification activities.

The second algorithm discovered was developed by the Satoshi Tsutsui, Yanwei Fu, David Crandall [11]. This algorithm uses egocentric hand gestures, such as pointing or waving, to identify the person to whom the hand belongs. To extract valuable information from the hand gesture, such as the size, shape, and movement of the hand, the algorithm combines deep learning and hand-crafted features. As a result, the algorithm was able to achieve an accuracy of 88.2% in identifying the person to whom the hand belongs.

Another system for automatic person identification based on the examination of fingernail and knuckle patterns, was proposed by Mona Alghamdi, Plamen Angelov, and Lopez Pellicer Alvaro [4]. The suggested

framework uses bounding boxes to localise different hand image components, such as the base knuckle, major and minor knuckles, thumb knuckle, and fingernails. The system uses various deep learning neural networks to identify similarities and extract features. The findings obtained using the 11k hands and the Hong Kong Polytechnic University Contactless Hand Dorsal Images (HKPU), show the value of fingernail patterns and knuckle patterns in the identification process. The fingernails patterns outperformed other hand parts in terms of identification scores. The findings on the left hand are greater to those on the right, and the PolyU dataset receives a 100% score in the fingernail of the thumb finger.

2.2. Review of hyperspectral imaging and its applications

Hyperspectral imaging has been found to be useful for biometric identification, and specifically in the recognition of dorsal hand vein images and hand biometrics using SWIR imaging. Two recent papers provided valuable insights into the potential applications of hyperspectral imaging in biometric identification. Those papers used an adaptive ROI extraction for hyperspectral dorsal hand vein images [12] and an approach to SWIR hyperspectral hand biometrics [13]. In addition, face recognition techniques were explored to obtain more material about the use of hyperpsectral images in biometric identification [14].

The first paper focused on the use of hyperspectral imaging for the extraction of dorsal hand vein images, which have gained popularity as a promising biometric property for identity recongition. The authors suggested a reliable and flexible region of interest extraction algorithm that significantly boosts the reliability and precision of the dorsal hand vein recognition system. The support vector machine (SVM) classifier and morphological processes were used in the algorithm to categorise pixels as vein or non-vein regions. The findings demonstrated that, in terms of accuracy and robustness, the suggested method works better than several cutting-edge algorithms.

The second paper examined the application of hand biometric identification using short-wave infrared (SWIR) hyperspectral imaging. The authors suggested a new method for obtaining features from SWIR

hand images that blends hyperspectral imaging with machine learning methods. Principal component analysis (PCA) and linear discriminant analysis are used to extract features after a number of preprocessing stages, including noise reduction and image enhancement. The findings showed that the suggested method, even in poor lighting and with few training examples, achieves high accuracy in hand identification.

Face recognition methods for hyperspectral images were investigated by the Laboratory for Applications of Remote Sensing from Purdue University. According to their research, the most biased absorption bands for facial recognition were those connected to haemoglobin. Two feature band subsets were chosen based on the physical absorption properties of the skin on the face. On a database of 25 people, three techniques for hyperspectral face recognition are suggested and put to the test. According to the results, multispectral face recognition with chosen feature bands outperforms face recognition with all bands, a single band, or traditional RGB colour bands.

2.3. Main findings

The main finding from the above papers that were beneficial for the development of the proposed solution are the followings:

- Applying a combination of guided filtering and high pass filtering to preprocess hand images, can produce high detailed hand images appropriate for feature extraction.
- Hand features like hand size, shape, and knuckle patterns are crucial in identifying individuals.
- Vein patterns can be used as biometric identification in hyperspectral imaging.
- Principal Component Analysis can be used to reduce the dimensionality of hyperspectral images, but also for feature extraction.

3. Technical Background

3.1. Pre-processing

Because preprocessing is used in image processing for accurate and efficient analysis of image dataset. Hence, for this project, preprocessing techniques such as normalization, guided filter, background removal, and principal component analysis (PCA) are crucial for enhancing the quality and utility of the image data.

3.1.1. Normalisation

One of the most used pre-processing step in image processing, is normalisation. In more detail, it adjusts the pixel values of an image to a certain range, making sure they are ideal for processing. Normalisation can be accomplished using various techniques like min-max normalisation, z-score normalisation and scaling.

Min-max normalisation is used to scale the pixel values of an image to a specific range. It rescales the minimum and maximum values of pixels within an image in a new range, usually between 0 and 1 [15]. The equation below shows how the min-max method is applied in mathematical terms, where xi is the input image, xmin and xmax are the minimum and maximum values accordingly.

$$x_{normallised} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (1)$$

Z-score normalisation is used to modify the pixel values in an image. Based on this, the mean and standard deviation of each pixel intensity are calculated, then scaled, so they have a mean of zero and standard deviation of one [15].

$$x_{normallised} = \frac{x_i - \mu_i}{\sigma_i} \tag{2}$$

Scaling is simpler process for normalisation, which reduces the pixel values of an image to a specific range. In this process, the pixel values are divided by a specific number to bring them within that specified range [16]. This approach works best when the pixel values already fall within this specific range. For example, in RGB images dividing the pixel values by 255, scales the image to a normalised range of 0-1 [17], something that can be observed in the equation below.

$$x_{scalled} = \frac{x_i}{255} \tag{3}$$

In an image processing system for hand identification in RGB and hyperspectral images, normalisation can be applied to ensure the pixel values of the images fall within a range suitable for processing. This technique works on both RGB and hyperspectral images prior to feature extraction and classification stages. By applying normalisation, the system became more effective at recognising hand features in images.

3.1.2. Guided filter

The guided filter is a nonlinear type of filter commonly used in image processing tasks like edge-preserving smoothing. Guided filters work by minimizing the difference between a filtered image and its original equivalent while keeping edges intact. They are particularly beneficial when smoothing an image while leaving visible contour lines intact.

The Figure 2 below shows the application process of the guided filter, in which an input image and a guidance image are used to produce an output filtered image. Each pixel is processed by multiplying its corresponding pixel in the input picture by a weight calculated from the guidance image unique to that pixel [18]. The output value at that pixel is the output of the filtered image.

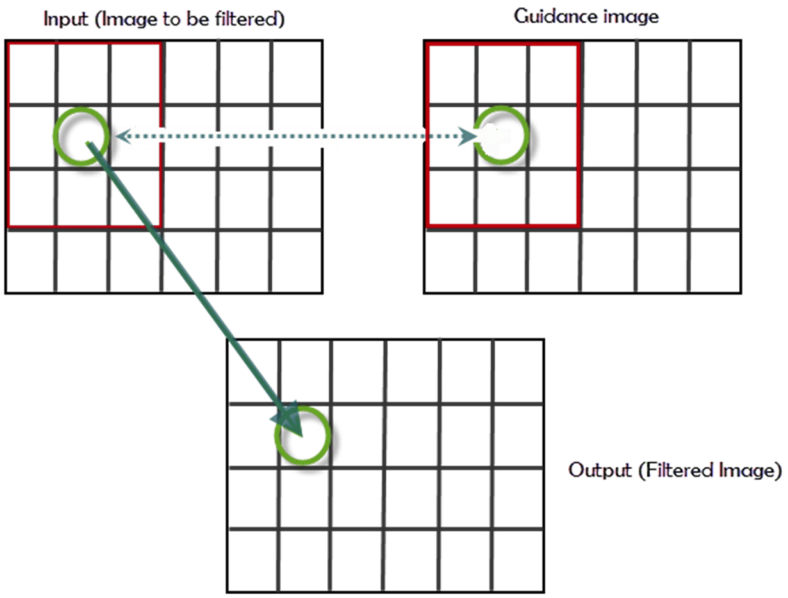


Figure 2: Input image, a guidance image and an output filtered image [18]

There are several advantages of using guided filter over other smoothing filters, like bilateral filter. It preserves edges in an image while simultaneously smoothing it, which makes it ideal for image processing tasks where maintaining edge details is crucial. While, the edge-preserving filter in the guided filter is a local linear model, the gradient reversal artefacts caused by the bilateral filter cause false edges to appear in the picture, something that can be observed in the Figure 3 below.

Figure 3: Difference in edges of guided and bilateral filters [18]

Performance is another significant distinction between bilateral and guided filtration. As we can see in the Figure 4 below, in the case of a bilateral filter, as the radius (computes the filter weights) increases, so does the amount of time it takes for the filter to smooth the picture. However, in the case of a guided filter, it happens instantly.

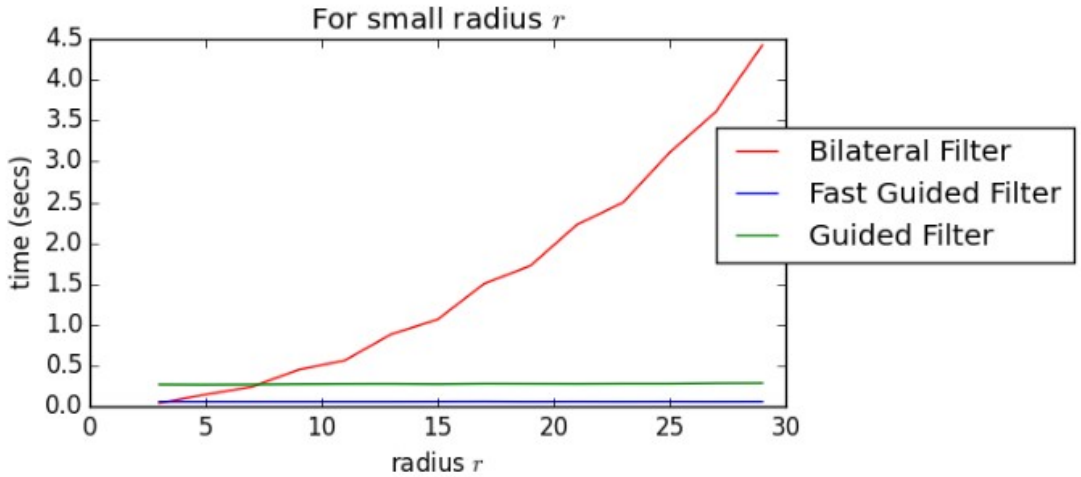


Figure 4: Performance of guided filter [18]

3.1.3. Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a technique that reduces the dimensions of a dataset, and keeps the most important information. It is especially useful in image processing tasks where the input data has a high dimensionality, such as hyperspectral images. In these images, each pixel is represented by values representing its intensity across each spectral band. With such large datasets it may be difficult to process them efficiently, and that is the reason that PCA is used [19].

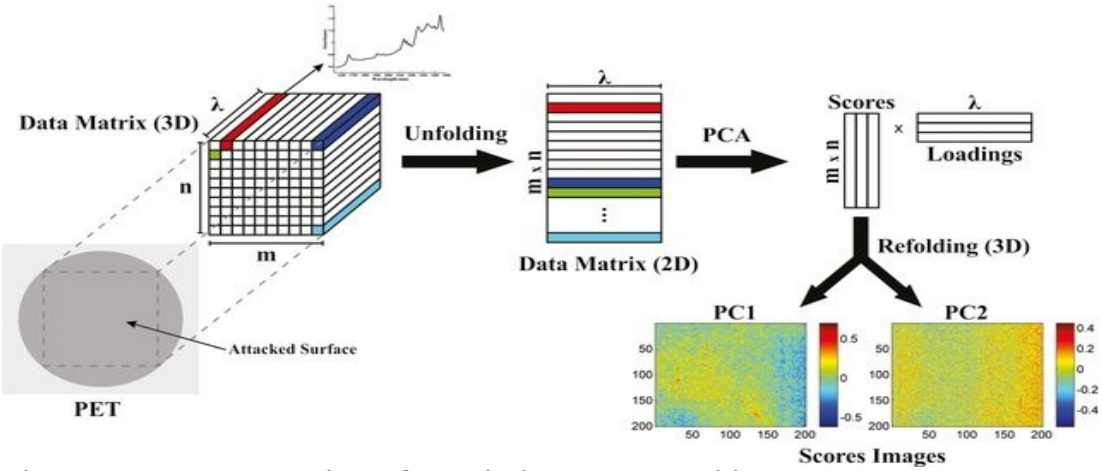


Figure 5: Representation of PCA in hyperspectral images

The process of applying PCA in hyperspectral images can be observed in Figure 5 above. Firstly, the 3D data cube of the spectral data is converted into a 2D data matrix, where each pixel is considered as a sample (mxn) and each spectral band as a variable (λ). After that, PCA is applied to the data matrix, to obtain the principal components (PCs), which represent the most important information and variation in the data. Scores and loadings are produced as a result of the PCA analysis. The scores show how each sample is projected into the principal components and reveal the similarities and variations between samples. The loadings show the proportional contribution of each initial variable (spectral band) to each principal component. The PCs can be visualized as images, where each pixel represents the value of the corresponding PC, and can provide information about the variation and patterns in the data [20].

3.2. Image Types

3.2.1. RGB Images

RGB images are digital images composed of three primary colour channels: red, green, and blue. Each pixel represents a combination of intensity values for these three colours. They are commonly used in object recognition, classification and segmentation. When it comes to hand identification tasks, RGB images can be utilised to extract colour information from the hand region which could then be used as a feature in classification algorithms.

3.2.2. Hyperspectral Images

Hyperspectral images are images composed of hundreds or thousands of spectral bands. Unlike traditional RGB images, which only contain three colour channels (red, green and blue), hyperspectral images capture data across a wider range of wavelengths, enabling more detailed characterisations of objects in the picture which can be useful in various applications [21]. Furthermore, it breaks down the light captured by each pixel into multiple spectral bands, which allows for more detailed information about the pictured scenario [22].

The Figure 6 below, shows the difference of RGB and hyperspectral images in terms of the wavelength range.

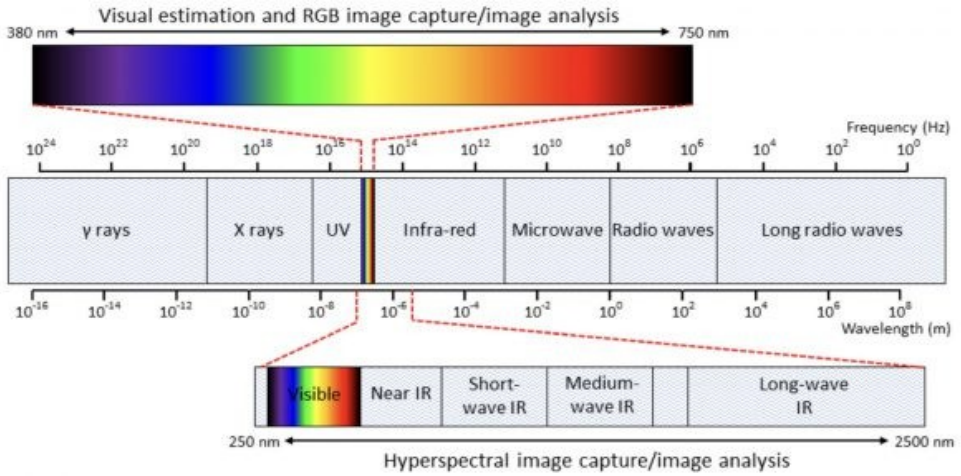


Figure 6: Wavelength range for RGB and hyperspectral imaging [22]

3.3. Deep Learning Network

Deep learning networks are a class of machine learning algorithms that have revolutionised computer vision. These networks are designed to learn hierarchical representations of data and have demonstrated remarkable performance in various computer vision tasks. Deep learning networks consist of multiple non-linear transformations that extract features from input data [23].

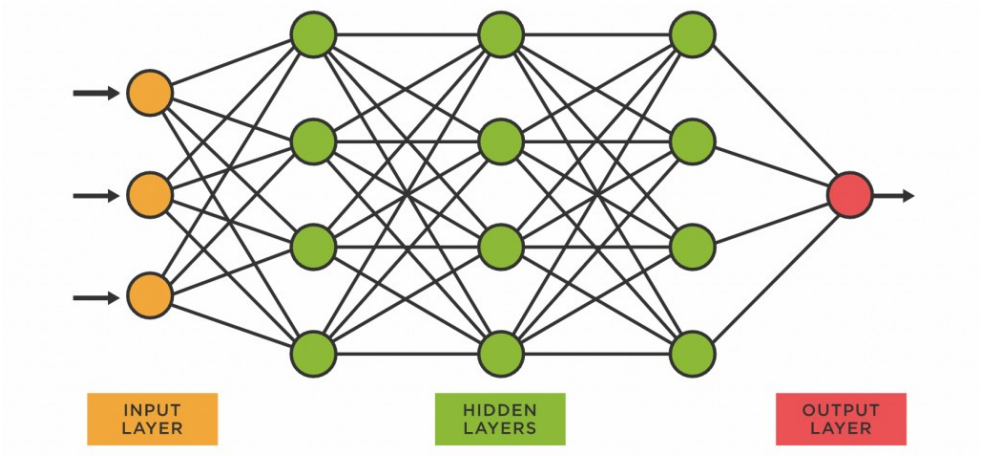


Figure 7: Deep Learning Neural Network [23]

Deep learning networks are composed of multiple layers, as it can be seen in Figure 7 above, with each layer responsible for processing a particular type of information [24]. The input layers are the first layers of the network and are responsible for receiving input data and passing it to the next layers for processing. The hidden layers are not visible to the user and perform complex calculations on the input data. Predictions are made using the appropriate characteristics that these layers learn to extract from the data. The network's depth and the complexity of the features it can learn, both depend on the number of hidden layers. Finally, the output layer of the network produces the final outputs based on the information processed by the hidden layers.

3.3.1. Convolutional Neural Network

Convolutional Neural Networks (CNN) are a type of deep learning neural networks widely used for image processing tasks such as hand identification in RGB and hyperspectral images. CNNs consist of multiple layers, such as convolutional layers, pooling layers, and fully connected ones [25].

Convolutional layers extract features from an input image by applying convolutional filters, which are learned during training and used to identify different patterns and features within it. Pooling layers reduce spatial dimensions while still preserving essential information. The flattening layer converts 2D feature maps from the pooling layer into 1D feature vectors that can be fed into the fully connected layers. Those layers perform classification tasks by mapping extracted features onto their corresponding classes. The dense layer is a fully connected layer, which utilises all the input features to produce an output. The final layers, known as the output layer, is in charge of making predictions using the input features and learned parameters. The number of nodes in this layer varies based on the nature of the problem being solved, as for a classification problem, like hand identification, the layers should correspond to the number of individuals in the database.

The CNN architecture can be studied in the Figure 8 below.

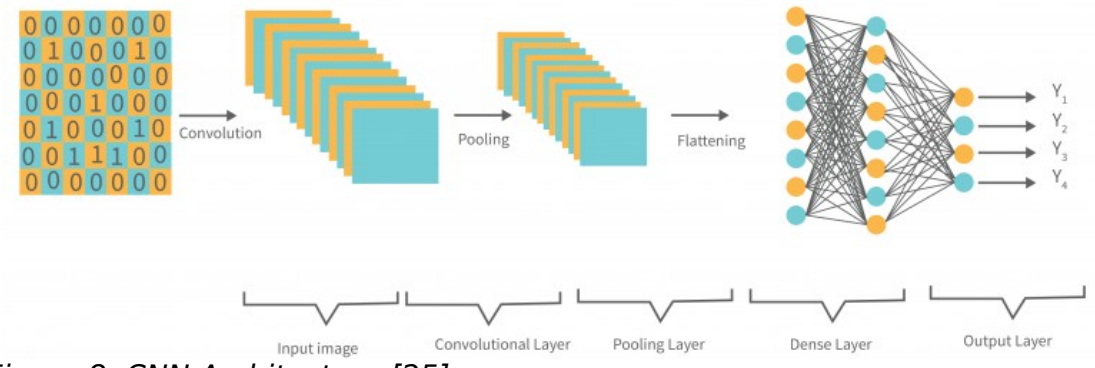


Figure 8: CNN Architecture [25]

3.3.2. Transfer Learning

Transfer learning is an approach in deep learning where a pre-trained model trained on a large dataset is used as a starting point for another task. The concept behind transfer learning is that the new model can use the learned characteristics to enhance performance. It can be especially advantageous when the new dataset is small and training a model from scratch might be challenging.

3.3.2.1. VGG16

VGG16 is a deep convolutional neural network architecture developed by the Visual Geometry Group at Oxford University [26]. This architecture consists of 16 convolutional and fully connected layers with approximately 138 million trainable parameters [27]. This model has demonstrated impressive success in testing across a range of computer vision tasks, including segmentation, object detection, and image classification.

This pre-trained VGG16 model was trained on large dataset called ImageNet and can be fine-tuned for the proposed hand identification task by extracting features from the new dataset while replacing fully connected layers with new ones specific to that task. Utilising transfer learning with VGG16, new models achieve high accuracy with less training data and computational resources.

The architecture of the VGG16 can be seen in Figure 9 below.

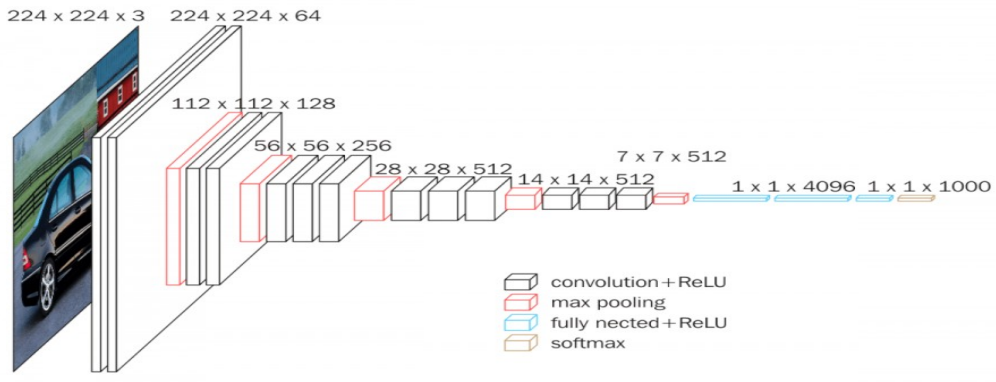


Figure 9: VGG16 Architecture [27]

3.4. Evaluation Metrics

Identifying a person from unique hand features such as shape and other biometric patterns is a difficult task and requires extensive testing. Evaluating the performance of these systems requires the use of specific metrics that can help quantify their accuracy and reliability. Some of the most used metrics are the accuracy, precision, recall, F1-Score, loss, and also the training – validation graphs.

Before showing the evaluation metrics there is a need to comprehend the variables that the evaluation metrics formulas contain. Those can be observed above.

- True positives (TP): The number of correctly identified hand images belonging to a specific person ID.
- True negatives (TN): The number of correctly identified hand images that do not belong to the person ID being considered.
- False positives (FP): The number of incorrectly identified hand images as belonging to the person ID being considered, when in fact they do not.
- False negatives (FN): The number of incorrectly identified hand images as not belonging to the person ID being considered, when in fact they do.

3.4.1. Accuracy

Accuracy is one of the most common evaluation metrics used in machine and deep learning. For the proposed hand identification system, it measures the proportion of correctly classified hand images out of the total number of hand images in the dataset. In hand identification systems, accuracy is crucial as it determines the reliability of the model in real-world scenarios. A high accuracy score indicates that the model is successful in identifying hand images with a high level of precision. The formula shows below is used to calculate the accuracy of the model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

3.4.2. Precision & Recall

Precision and recall are two other evaluation metrics for hand identification systems. Precision measures the proportion of true positive hand images out of all positive predictions made by the model. Recall measures the proportion of true positive hand images out of all actual positive cases in the dataset. A high precision score indicates that the model is making accurate predictions, while a high recall score indicates that the model is detecting most of the relevant hand images in the dataset.

$$Precision = \frac{TP}{TP + FP} \quad (5)$$

$$Recall = \frac{TP}{TP + FN} \quad (6)$$

3.4.3. F1-Score

The F1-score is a harmonic mean of precision and recall, which is used to balance these two evaluation metrics. It provides a single score that represents the performance of the hand identification system. A high F1-score indicates that the model is performing well in both precision and recall.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (7)$$

3.4.4. Loss

Loss is an evaluation metric that measures the difference between the predicted and actual values of the hand identification system. It is calculated using a loss function, which penalizes the model for incorrect predictions. The lower the loss value, the better the performance of the model.

4. Hand Datasets & Biometrics

4.1. Hand Datasets

Three hand datasets were used for the creation of a hand identification system. The first two datasets were specifically for RGB hand images, while the third one was for hyperspectral hand images.

4.1.1. RGB Datasets

The images used for the RGB algorithm consist of two different datasets, the “11k hands” [20], and the “The Hong Kong Polytechnic University Contactless Hand Dorsal Images Database” [21]. The first one contains 11,076 dorsal and palm images from both right and left hands, with a resolution of 1900x1200. The hand images were taken from people with different age, gender, and skin colour. The second dataset, contains 2505 hand dorsal images from the right hand of 501 different subjects, with three knuckle patterns illustrated in each of the four fingers from each subject. For development of the initial algorithm that was identifying 32 participants, 251 images were used from the “11k hands” dataset. The improved algorithm uses 300 images merged from two data sets, evenly distributed and suitable for 30 users.

Figure 10: Sample of hand from the datasets, the two on the left are from the "11k hands" and the two on the right are from "The Hong Kong Polytechnic University"

4.1.2. Hyperspectral Dataset

The images used for the hyperspectral algorithm consist of one dataset, acquired from “The Grupo de Procesado Digital de la señal” which is a digital signal processing group from “Instituto para el Desarrollo Tecnológico y la Innovación en Comunicaciones IdeTIC” [28].

The database is a collection of hyperspectral images of hands from 154 people, with each sample consisting of 350 images acquired by a hyperspectral device. The images were taken in 256 bands between 900nm and 1600nm, and users were allowed to wear wristwatches or wristlets. The age of the users ranged from 18 to 60 years, with 86% of them between 18 and 30. Approximately 70% of the users are students and teachers from the university, and the remaining 30% are administration and cleaning staff.

The hyperspectral camera used is a Xenics Xeva-1.7-320 camera. The camera provides 256 gray level images with a resolution of 320 by 256 pixels, and it is used in conjunction with a SPECIM Imspector N17E optical spectrograph with numerical aperture f/2.0 to transform the SWIR camera into a line spectral imaging device.

Working with the hyperspectral images in Google Colab, which has a small storage capacity, can be difficult due to the hyperspectral images' big size. A smaller portion of the data was used for the algorithm's training and testing in order to get around this restriction. It is reasonable in this situation to use 10 users, each with 10 images. As it provides for a diverse collection of images while keeping the whole size manageable.

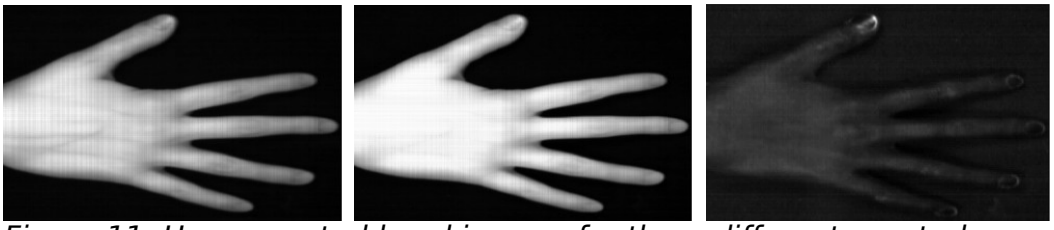


Figure 11: Hyperspectral hand images for three different spectral bands

4.2. Hand Biometrics

Hand biometrics are feature on hand, that are used for identification purposes. In this project, focus is given on four key features of a human's hand: hand shape, skin texture, knuckle pattern, and vein patterns.

4.2.1. Shape

The shape of the hand can vary greatly from person to person because it is determined by the proportions and relative sizes of the fingers, and from the length and width of the overall dorsal hand [29], [30].

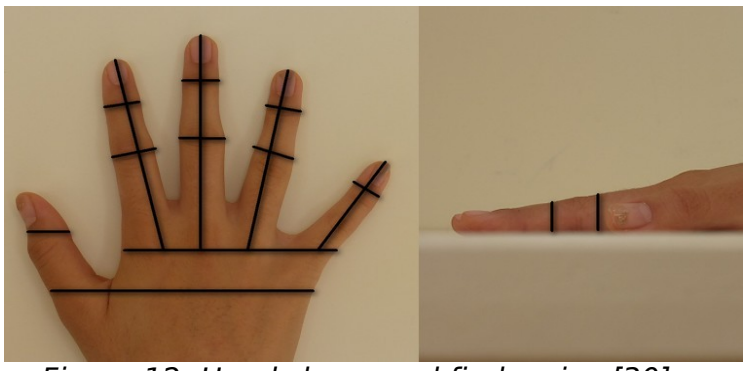


Figure 12: Hand shape and finder size [30]

4.2.2. Skin Texture

The texture of the skin and the pattern that it creates, is a unique characteristic to each individual and can be used for identification purposes [31]. This feature can be affected by a variety of factors, including age, genetics, and environmental factors [32].

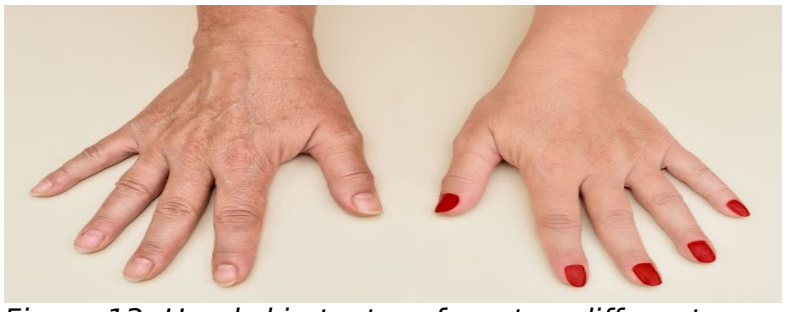


Figure 13: Hand skin texture from two different people [32]

4.2.3. Knuckle Patterns

Knuckle patterns are another hand feature that can be used for identification. The patterns of the skin around the knuckles are unique to each individual and can be used to distinguish each human being. There are three significant knuckles: a distal interphalangeal (DIP) joint, a proximal interphalangeal (PIP) joint, and a metacarpophalangeal (MCP) joint [33]. Those can be shown in the Figure 14 below.

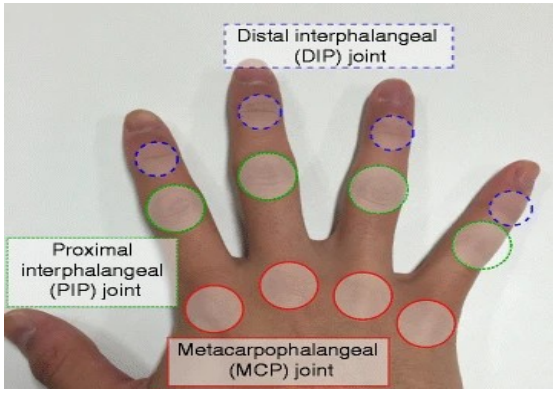


Figure 14: Knuckle Patterns [33]

4.2.4. Vein Patterns

Vein patterns are also an important feature of the hand. The veins in the hand are arranged in a unique pattern that can be used for identification purposes [34], [35]. Vein patterns are formed by the blood vessels underneath a person's skin [36]. Unlike other hand biometrics, vein patterns are not affected by changes in skin or lighting conditions. However, vein patterns can not be captured by RGB digital cameras, and they require either infrared cameras or hyperspectral cameras [37].

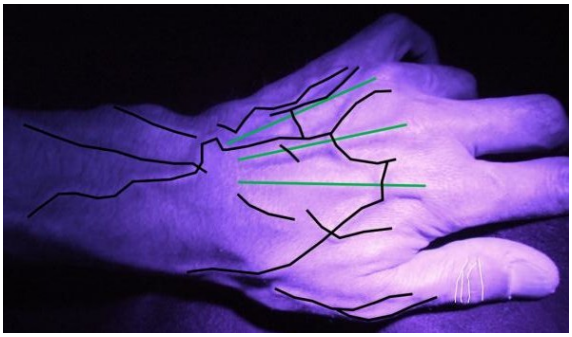


Figure 15: Hand vein patterns [35]

5. System Requirements

Deep learning networks need certain requirements to be trained properly. Those requirements depend on the complexity of the algorithm, as well as the computational power required for pre-processing the images. In the case of the hand identification system, the current laptop’s CPU and GPU were not strong enough for those tasks, so a solution was necessary.

So, it was agreed to use Google Colab and design the algorithm in Python rather than Matlab to get around this problem. This was a great decision because Google Colab gives a cloud-based environment with GPU acceleration and lots of RAM for running complex algorithms. The project was finished without the need for an expensive hardware or software purchase, thanks to the use of Google Colab.

When it came to evaluating the performance of the hand identification system, Matlab was used to evaluate existing methods. Those evaluations were time-consuming and required huge computational cost. However, for the proposed solution, the training and evaluation were done in Python, and specifically in Google Colab, with the GPU and RAM of Google. This made the system evaluation process quicker and more effective.

Depending on how complicated the algorithm is and how much computational power is needed to process the images, a hand identification system using RGB and hyperspectral images have different system requirements. In this instance, the project was successfully finished thanks to the use of Google Colab, which offered the required computational power to run the algorithm effectively.

6. Methodology

The methodology of this system involved several steps, including preprocessing, feature extraction, picking the optimal hyperparameters for training, and applying evaluation metrics. Those steps can be observed in the Figure 16 below.

Figure 16: Block diagram of the hand identification system

The pre-processing step consists of preparing the hand image for feature extraction. In this step, the image was modified, enhanced and converted to a computational format acceptable to the CNN.

Feature extraction is the next step, where the unique features of the hand are identified. This can be done by extracting features such as the hand shape, knuckle patterns, skin texture, and vein patterns if possible. Those features were extracted with the help of the convolution and pooling layers of the CNN model.

The model was then trained in labelled images using the VGG16 pre-trained network. A dataset of labelled hand images were used to train the model, where these labels identify the individual in each image. In order to recognise and identify hands correctly, the model needed be trained to associate the extracted features with the appropriate labels.

For the model to be accurate, it is essential to choose the optimal hyperparameters. Hyperparameters are adjustable parameters that determine how the model works and learns. Through trial and experimentation, the ideal values for these parameters were chosen.

Last but not least, the model's performance was tested using appropriate evaluation metrics. The metrics were used to evaluate the performance of the hand identification system, which are the accuracy, precision, recall, F1 score, and also the learning curved during training.

6.1. Image pre-processing

6.1.1. RGB Images

The preprocessing method used in hand identification based on RGB hand images datasets involved two primary steps: background removal as well as image modification and enhancement.

6.1.1.1. Background Removal

The background removal method requires the use of the "mediapipe" selfie segmentation to separate the foreground, which is the hand, from the background. The Figure 17 below represents the input image and the hand with the white background.

Figure 17: Process of removing the background from the input image

In more details, the first step of the background removal process was to extract the segmentation mask of the input image, using the 'SelfieSegmentation' module from the mediapipe library. This mask was then converted into a binary mask, where the foreground was represented by white and the background by black.

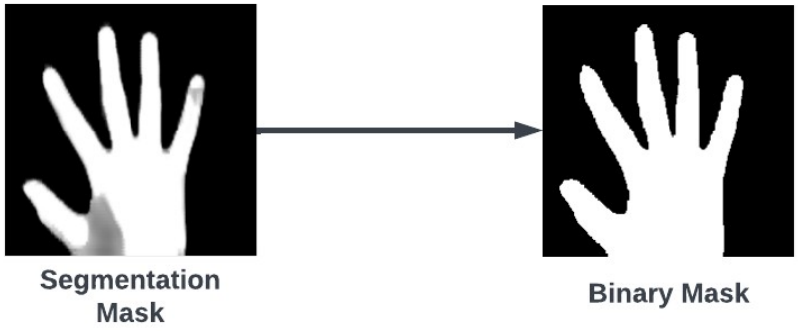


Figure 18: Representation of segmentation and binary mask

After that, the binary mask was inverted to get a mask where the background was white, and the foreground was black. Additionally, a white background image of the same size as the original image was created. To obtain the background image, the inverted mask was applied to the white background.

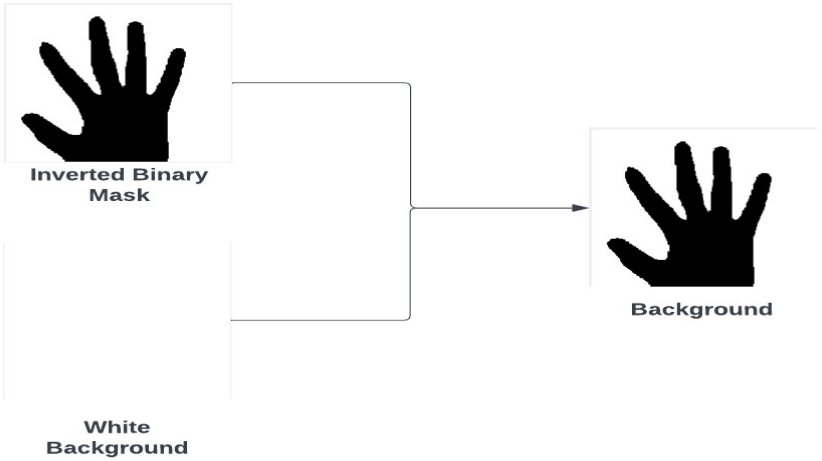


Figure 19: Process of obtaining the background image

The binary mask was then applied to the original image to obtain the foreground.

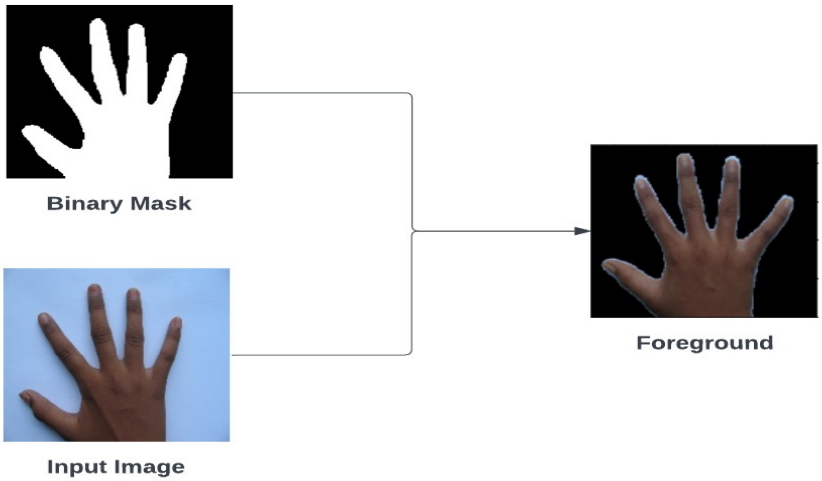


Figure 20: Process of obtaining the foreground image

Finally, the foreground and background images were combined to obtain the final result.

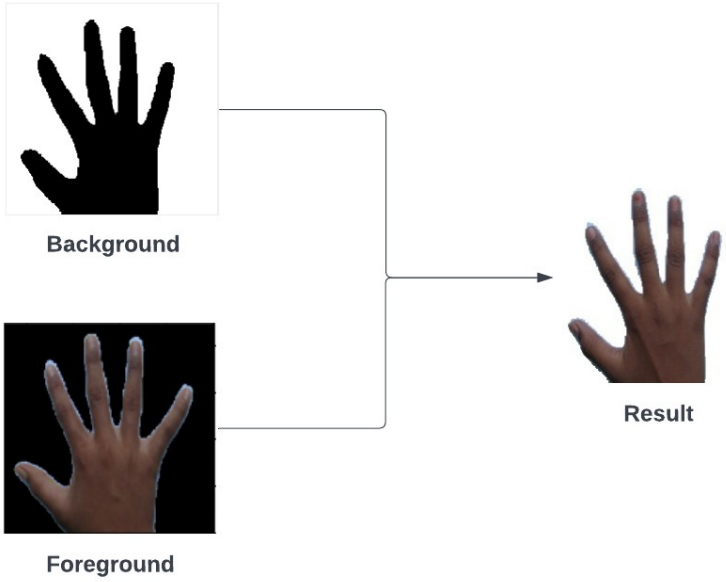


Figure 21: Process of obtaining the final image with no background

6.1.1.2. Image Modification

The second step was image modification and enhancement, which involved improving the details of the input image to make it more suitable for the CNN algorithm. The first technique used was normalisation, which involved scaling the pixel values of the image to a range between 0 and 1. In more detail, scaling was applied, as a simple normalisation process, by dividing the pixels of the hand image to 255. This step is crucial because deep learning algorithms, especially CNNs, work best when the input data is normalised to avoid bias towards specific pixel values, while also reducing computational costs.

After that a guided filter was applied, which is a smoothing technique that reduces noise and enhances the image's edges. This method preserved the edges of the picture while averaging the pixel values in a nearby neighbourhood. The result was a smoother image with enhanced edges.

The parameters used for the guided filter were a regularisation parameter (epsilon) of 0.05 and a radius of 10. Those values seemed optimal resulting in a hand image where hand features were visible.

The final technique used in the pre-processing is high-pass filtering, which amplifies the differences between the input image and the smoothed image, enhancing the image's details. It divided the normalised image by the smoothed image. This highlighted the edges of the hand, making it easier for the CNN to extract features from the hand images, and then training a more accurate network.

The whole pre-processing step can be observed in the Figure 22 below, where the output images of each technique is included.

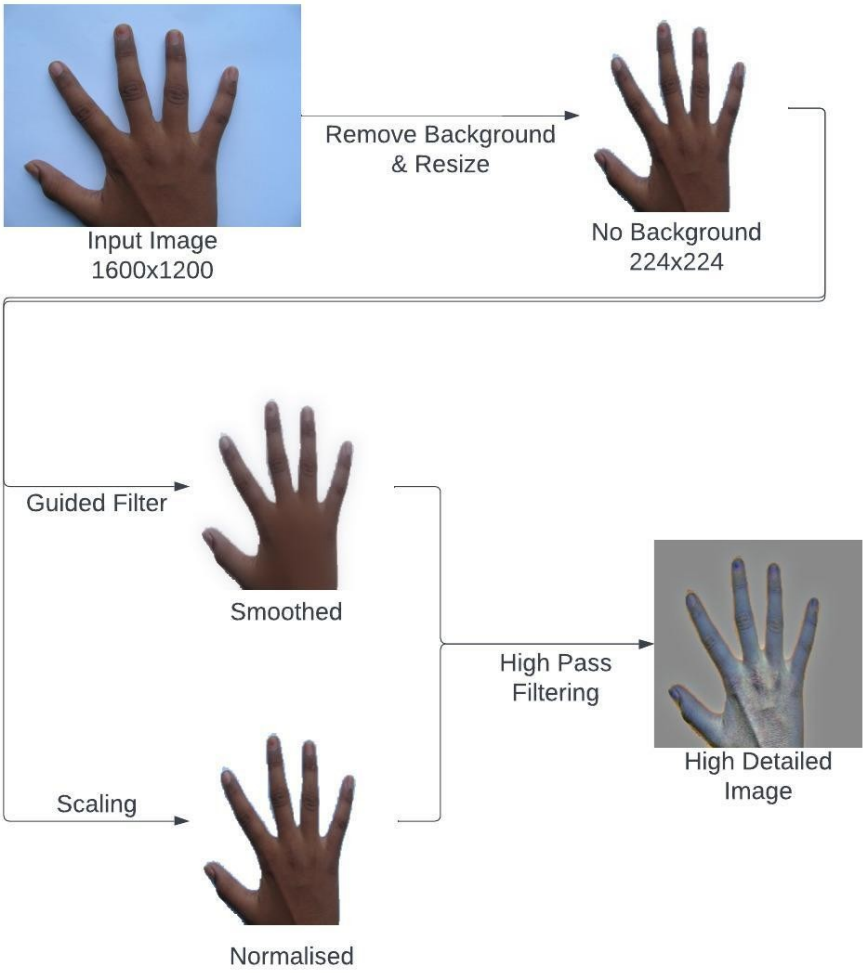


Figure 22: Pre-processing of RGB hand images

6.1.2. Hyperspectral Images

Hyperspectral data are high-dimensional and complex, which makes it challenging to analyse and understand them. The pre-procesing of the hyperspectral images consists of two steps, image modification and the application of the PCA in the first 3 principle components, as three channels are required for the CNN model.

In the image modification the image was cropped to the region of interest to exclude irrelevant regions, thereby reducing the amount of data and improving the speed and accuracy of processing. Next, the images were normalised using min-max normalisation. Min-max normalisation was preferred over z-score for hyperspectral hand images as it scales the pixel values to a specific range (0 to 1) rather than making them zero-mean and unit variance, which helps to preserve the spectral information.

The reason for choosing the first 3 principal components for the hyperspectral dataset was due to the high eigenvalues associated with each component, after applying PCA. Higher eigenvalues signified that a larger amount of the variance in the data is captured by that specific component. In the Figure 23 below, it can be observed that the first 3 principal components have significantly larger eigenvalues than the rest, indicating that they captured most of the variation in the data.

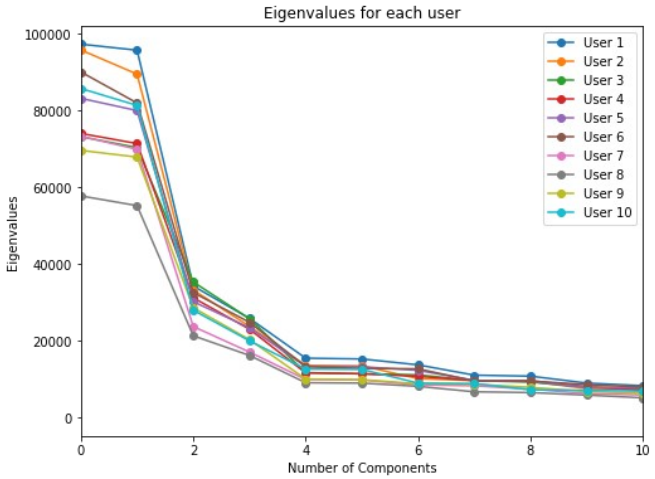


Figure 23: Plot of eigenvalues for 10 users

In order to apply PCA, the data dimension had to be reshaped, as PCA operates on a matrix of data where each row represented a pixel and each column represents a spectral channel. This format of data was achieved by reshaping the original 3D hyperspectral image to a 2D array, where each row represents a pixel and each column represents a spectral channel.

After applying PCA to the hyperspectral image dataset and identifying the first three principal components with the highest eigenvalues, the dimensionality of the dataset was reduced by selecting only these components. The projections of each pixel onto the first three main components were contained in the two-dimensional array that makes up the resulting PCA image. However, it is necessary to transform this PCA image back into a three-dimensional format with dimensions that correspond to the original height, width, and spectral channels of the hyperspectral image in order to use it as input to a convolutional neural network (CNN) model. This reshaped PCA image could then be fed into the CNN model as input, allowing the model to learn and extract features from the reduced-dimensionality image representation.

The Figure 24 above shows each channel (channel 1, 2, 3) of the hyperspectral image after applying PCA to the first 3 principal components.

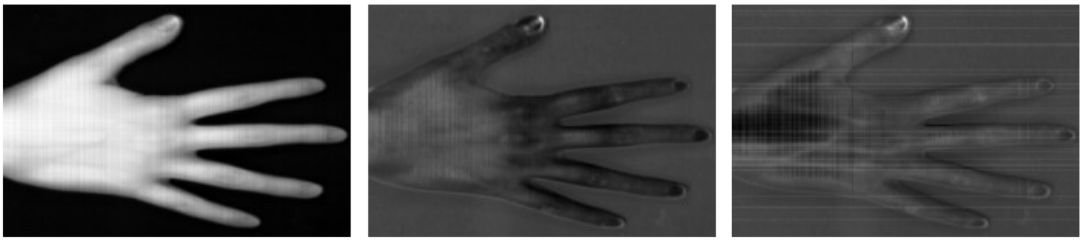


Figure 24: Representation of each channel after PCA was applied for the first 3 principle components

6.2. Model Selection

The choice of the CNN model was done by exploring the most popular pretrained models for images classification like VGG16, AlexNet, and ResNet. VGG16 is a popular CNN model that has been used in various image recognition tasks. Compared to other popular CNN models like AlexNet and ResNEt, VGG16 has several advantages that make it an ideal for the proposed hand identification system.

The improved accuracy of VGG16 over AlexNet was one of its key benefits. VGG16 has 13 convolutional layers as opposed to 5 convolutional layers in AlexNet, making it deeper. With more features being extracted from the input images thanks to this depth, VGG16 achieved a better accuracy rate [38]–[40].

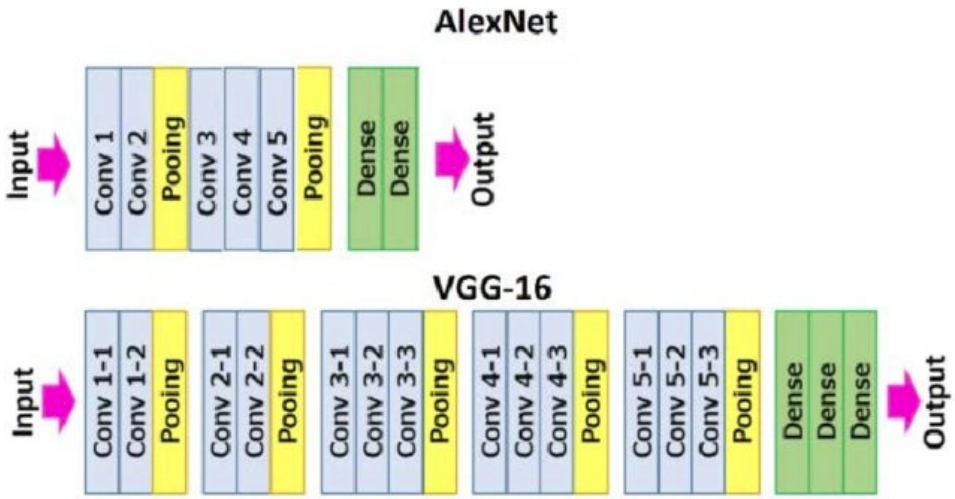


Figure 25: Difference between AlexNet and VGG16 [40]

Better feature extraction skills are another benefit of VGG16. Convolutional layers were followed by more max-pooling layers in the design of VGG16, which resulted in a more reliable feature extraction procedure. In contrast, AlexNet had less convolutional and max pooling layers, which could lead to problems with underfitting (the model was unable to capture feature patterns).

The Residual Neural Network (ResNet) is the latest deep learning network model compared to AlexNet and VGG16. It has been widely used in many image recognition applications, due to each complex achitecture and the number of layers [41]. It is more complicated model than VGG16, as it used skip connections between layers, which make it more challenging and requires more computationan cost [42]. In contrast, the VGG16 model is more effective because of its simple architecture, which has no skip connections and makes training simpler while requiring less memory.

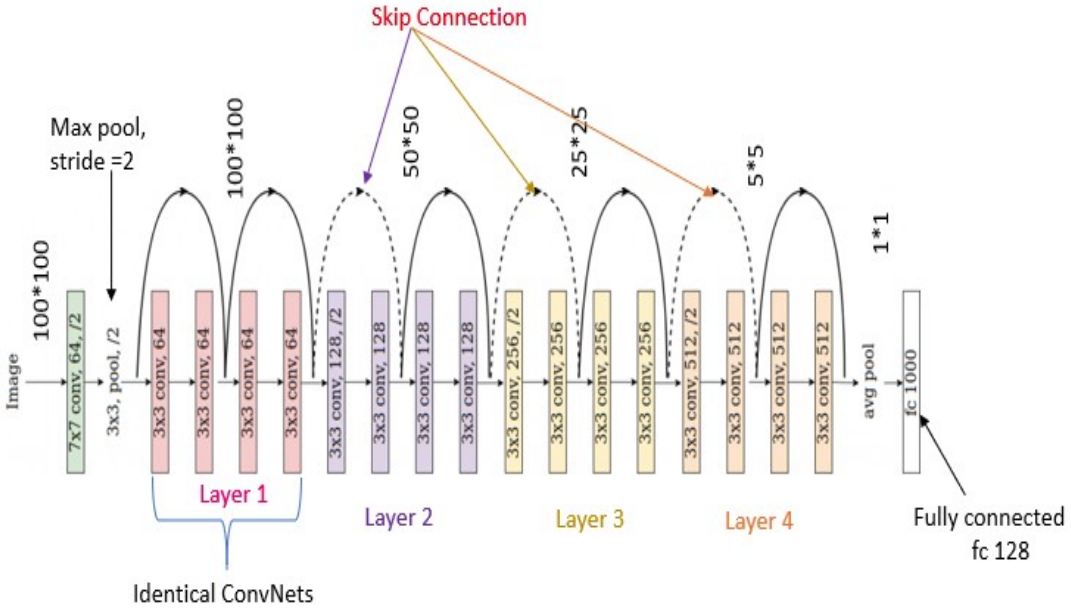


Figure 26: ResNet model architecture example [42]

From the points above it can be observed that VGG16 is an ideal choice for the proposed hand identification system due to its higher accuracy, better feature extraction capabilities compared to AlexNet, and simpler architecture, efficient memory use compared to ResNet. Those are the reasons that VGG16 was chosen as a pretrained model for the proposed hand identification solution.

6.3. Feature extraction and model training

Identifying significant features from input images and using those features to build a classification model are essential stages in the development of an image processing algorithm for a hand identification system. Therefore, is it essential to use feature extraction, and train the model on those features.

Feature extraction is the process of extracting important features from an image that can be used for further analysis. For the proposed hand identification system, the features that were extracted were the hand shape, knuckle patterns, skin texture, and vein patterns, if possible. The features were extracted using the VGG16 pre-trained model.

The input of the model was 224x224x3 for RGB images, and 210x310x3 for the hyperspectral images. The first two dimensions represented the size of the image, while the third was the number of channels. RGB images had three colour channels, one for red, one for green and one for blue. Hyperspectral images were also represented with three channels, as they were derived from the first 3 principal components after applying PCA.

From previous sections, it is known that the VGG16 mode consists of convolution and pooling layers. The convolutional layers of the network are responsible for detecting low-level features such as hand shape, and the max pooling layers downsample the feature maps to reduce computational complexity and increase robustness to variations in hand images, thereby enabling the network to learn hand-specific features that can be used for hand identification tasks.

Figure 27: CNN architecture containing the VGG16, and the last 3 layers

Once the features were extracted from the input images using the pre-trained VGG16 model, the last three layers of the model were modified to suit the specific task of the hand identification.

The last three layers were:

i. Flatten layer: This layer converted the output of the VGG16 model to a 1-dimensional array, which could be used as input to the dense layer.

ii. Dense layer (ReLu): This layer had 4096 neurons and used the ReLU activation function, which is a non-linear activation function used to introduce non-linearity in the network. It helped to improve the ability of the model to learn non linear relationships between input and output.

iii. Dense layer (Softmax): This layer used the Softmax activation function, which converted the outputs of the previous layer into probability scores for each class. This layer was added to classify the input image and the neurons used depend on the classes that the dataset has.

After modifying the VGG16 model, the data was split into training and validation sets. From the 10 images for each user, 6 of them were used for training and 2 for validation, and the other 2 for testing the model in unseen data. By doing so, overfitting was avoided making the model able to generalise to new data.

Finally, the model was compiled using the Adam optimiser with a learning rate of 0.00001, and a loss function of sparse categorical cross-entropy. Adam optimiser was an adaptive optimisation method that determined unique adaptive learning rates for each neural network parameter.

6.4. Hyperparameters

Hyperparameters selection is a necessary step when training a deep learning network model. CNN required careful tuning of hyperparameters to achieve optimal performance. Therefore, selecting the appropriate learning rate, batch size, and epochs is essential to develop a reliable and robust hand identification system.

6.4.1. Learning rate

It is required to balance convergence speed and efficiency when choosing the ideal learning rate. In this instance, the experimentation with lower learning rates did not result in any significant performance increase, indicating that the model was already performing at its best. Additionally, the use of this value in other effective image classification systems for hands [10], [43], [44] impacted the decision to use 0.0001 as the learning rate. The past success of comparable systems offered a compelling reason to choose this learning rate, even though it may not always guarantee the optimal performance for every model or application.

6.4.2. Batch Size

Selecting the ideal batch size is important in order to create the optimal image processing system for hand identification. The batch size was the number of training examples used in one iteration of the optimisation algorithm during training. A model's performance could be significantly impacted by selecting the right batch number.

The evaluation metrics varied across different training runtimes. Some possible reasons for that could be the model architecture, and the initialization of the model's parameters. As a result, it was difficult to check each evaluation metric for every train run. By combining multiple evaluation metrics into a composite score, it was possible to identify the optimal batch size much quicker and more accurate.

The composite score, which combined accuracy, precision, recall, F1 score, and loss, could be used to evaluate the performance of a model trained with a particular batch size. The formula above shows the

calculation of the composite score (CS), where the goal is to keep it as small as possible.

$$CS = (Accuracy + Precision + Recall + F1\ Score) - Loss \quad (8)$$

The table below includes the evaluation metrics and the composite score for different batch sizes (10, 12 ,24). The system was trained in three different runtimes for each batch size in order to get reliable results. The following results were obtained using the “11k hand” dataset for 32 users and 15 epochs.

Table 1: Evaluation metrics for different batch sizes

Batch Size	10			12			24		
Training Runtime	1st	2nd	3rd	1st	2nd	3rd	1st	2nd	3rd
Accuracy	83%	80%	73%	83%	85%	81%	80%	78%	75%
Loss	18	15	23	18	32	20	32	24	37
Precision	80%	81%	73%	81%	82%	84%	78%	75%	69%
Recall	83%	80%	73%	83%	85%	82%	80%	78%	75%
F1-Score	80%	77%	70%	80%	82%	80%	76%	74%	69%
Composite Score	-142	-143	-120	-143	-132	-145	-122	-125	-101
Average CS	-135			-140			-116		

The results above show that a batch size of 12 consistently gave the best composite score across three training runs. The average composite score for batch size 12 was -140, which was lower than the scores for batch sizes 10 and 24.

Additionally, the individual evaluation metrics showed relatively consistent performance for batch size 12 across multiple training runs. This indicated that this batch size might be well-suited for the proposed hand identification task.

6.4.3. Epochs

Epoch is a single iteration of the entire dataset through a neural network during the training process. The number of training epoch for a CNN model relies on a number of variables, including the model's complexity, the size of the dataset, and the available computational resources. For the hand identification model, the three epochs were picked (15, 25, 30) to evaluate the performance of the model across different numbers of training iterations.

The following table 2 includes the evaluation metrics and the composite score for different batch epochs, with a batch size of 12, as this was the optimal.

Table 2: Evaluation metrics for different number of epochs

Epochs	15			25			30		
Training Runtime	1st	2nd	3rd	1st	2nd	3rd	1st	2nd	3rd
Accuracy	83%	85%	81%	83%	86%	83%	78%	83%	78%
Loss	18	32	20	15	17	16	29	23	30
Precision	81%	82%	84%	82%	86%	82%	83%	86%	81%
Recall	83%	85%	82%	83%	87%	83%	78%	83%	78%
F1-Score	80%	82%	80%	81%	84%	81%	77%	82%	77%
Composite Score	-143	-132	-145	-148	-154	-147	-131	-145	-128
Average CS	-140			-149			-134		

From the table above, it appears that the model achieved the lowest average composite score of -149 when trained for 25 epochs, compared to 15 epochs (-140) and 30 epochs (-134). This suggested that the model's performance on the evaluation metrics was highest when trained for 25 epochs. Also, the individual evaluation metrics were more consistent performance for epochs 15 and 25. Therefore, the 25 epochs were chosen for the proposed model.

6.5. Evaluations

Evaluation measures like accuracy, loss, precision, recall, and F1-score are used, in this case, to assess the model's performance for RGB and hyperspectral images. These measures evaluated the model's accuracy in classifying the image's pixels while also accounting for true positive and false positive rates. While the precision and recall metrics gave information on how well the model was recognising true positives and true negatives, the accuracy metric show how well the model was performing. The following evaluations were executed using a batch size 12, 25 epochs for the RGB images, and batch size 13, 15 epochs for the hyperspectral images. The RGB images used are only from the “11k hands” database, and only for the dorsal right hand.

6.5.1. RGB Images

The RGB image processing system for hand identification was evaluated in three different pre-processing techniques, for only the 30 users of the “11k hands”. The 1st method was simple normalisation and resizing of the images. The 2nd method produced a more detailed image in both BGR and grey formats, as described in the previous sections. By comparing the performance of the model for these three pre-processing methods, a insight was gained about how different image processing techniques can affect the accuracy of the model.

The following Figure 28 shows the different pre-processing methods used. The left one shows the simple pre-processing, the middle one the high-detailed in bgr, and the right one the high-detailed in grey format.

Figure 28: Three different preprocessing methods for the RGB algorithm

Table 3: Evaluation metrics for 3 different pre-processing methods

Pre-Processing	Simple	High-Detailed BGR	High-Detalied Gray
Accuracy	83.95%	83.85%	80.8%
Loss	17.2	9.1	9.89
Precision	82.25%	82.37%	77.5%
Recall	84.25%	82.25%	80.75%
F1-Score	81.5	82.3%	77%

Based on the results, from the Table 3 above, it can be seen that the high-detailed BGR pre-processing method has more consistent metrics compared to the other pre-processing methods. The accuracy and precision values were almost identical for both high-detailed BGR. Even though the simple pre-processing method had slightly better recall, the high-detailed BGR method had higher F1-Score. Additionally, the high-detailed BGR pre-processing method had the lowest loss value, showing that the model performed better when this pre-processing method was used.

In order to understand how well the model was able to learn the features and patterns in the training data and how well it could generalise to new, unseen data, the model loss graphs had to be plotted for each pre-processing method.

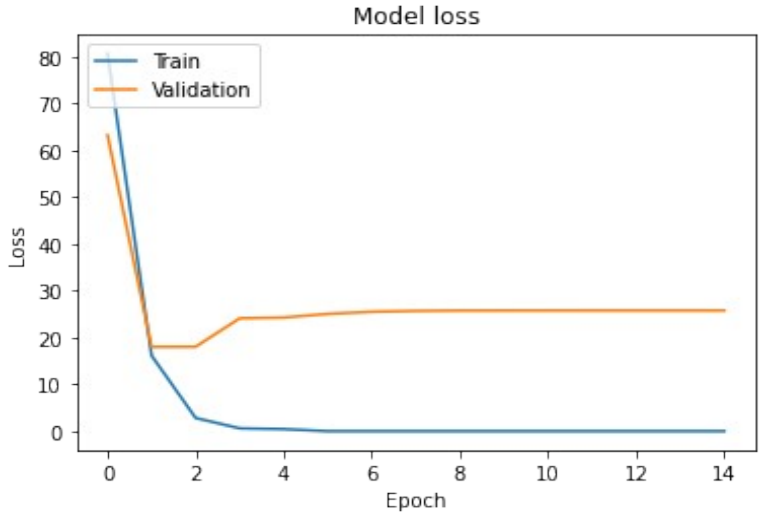


Figure 29: Model loss plot for the simple pre-processing method

The loss plot in the Figure 29 above, shows that the model was overfitting to the training data. This is because the model cannot generalise well to new data because the validation loss became straight before hitting the training loss. The lack of detail in the images produced by this pre-processing makes it challenging for the model to extract useful features.

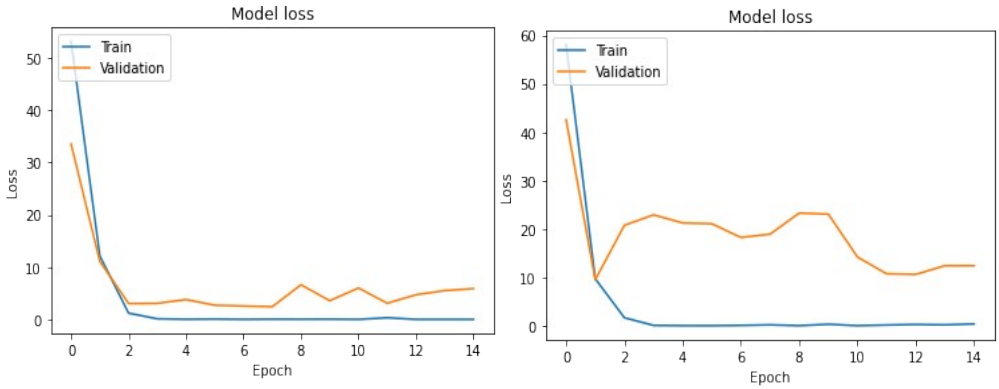


Figure 30: Model loss plots for the high detailed bgr and grey methods accordingly

The model loss of the high detailed BGR image can be seen on the left side of the Figure 30 above. In more details, it demonstrates that the validation loss curve approaches the training loss curve, showing that the model generalised to new data more effectively. The gap between the training and validation loss curves did, however, still show some overfitting. This might be because the images produced by this pre-processing technique are complicated, making it challenging for the model to pick up on the patterns and features in the training data.

The model loss of the high detailed gray image can be seen on the right side of the Figure 30 above. It shows that the validation loss curve went up and down, reaching a little closer to the training loss curve in comparison with the simple preprocessing method, but not as close as with the high detailed BGR method. This indicates that the high detailed BGR method might be more suitable for training the model than this pre-processing technique.

6.5.2. Hyperspectral Images

For the hyperspecrtal images, the same evaluation metrics were used with the RGB ones. A focus must be given in the use of a softmax layer with 11 neurons instead of 10 suggests, as the model was not accurate when the last layer was set to 10 outputs.

Table 4: Evaluation metrics for rgb and hyperspectral images

Images	RGB	Hyperspectral
Accuracy	83.85%	85%
Loss	9.1	0.95
Precision	82.37%	85%
Recall	82.25%	85%
F1-Score	82.3%	83%

The Table 4 above show that the hyperspectral images outperformed the RGB images in every metric. The model trained on hyperspectral images has learned more important features and patterns from the data, as demonstrated by the lower loss value for these images, leading to improved performance. This is probably because hyperspectral images had greater spectral resolution and more spectral information.

The model loss of the hyperspectral images can be seen in the Figure 31 below, showing similar graphs with the high detailed BGR format, described before.

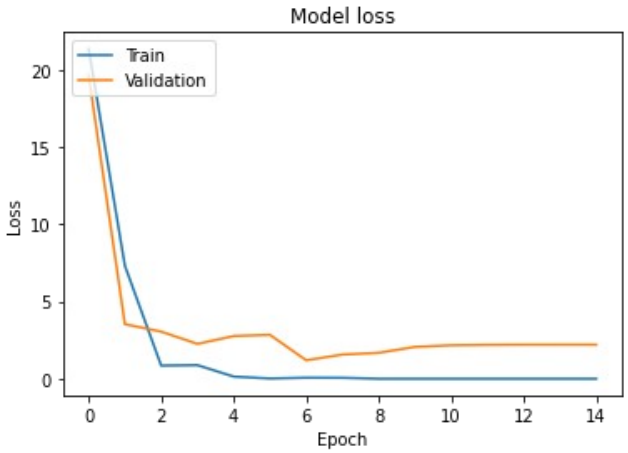


Figure 31: Model loss plot for hyperspectral images

7. Experimental Results

7.1. Comparison of the results with different datasets

In this section, a comparison was done in the performance of the proposed image processing algorithm trained on the 11k Hands Dataset, and after that, trained on a combination of the 11k Hands Dataset and The Hong Kong Polytechnic University (HKPU) Contactless Hand Dorsal Images Database. This is done to observe how the algorithm performs to different datasets with various backgrounds and light conditions.

Table 5: Evaluation metrics for different datasets

Datasets	“11k hands”	“11k hands” & “HKPU”
Accuracy	83.85%	83%
Loss	9.1	15
Precision	82.37%	82%
Recall	82.25%	83%
F1-Score	82.3%	81%

The results in the Table 5, show that the algorithm trained on the 11k hands dataset performs a slightly better than the model trained on the 11k hands and HKPU datasets alone. It should be noted that there is not much of a performance difference between the two datasets.

The combination of the two datasets show greater loss, meaning that the algorithm might have had more trouble adapting to the new and varied hand images that the HKPU dataset introduced. This might be due to the variations in lighting, skin tone, something that has not been seen in the 11k hands dataset.

To sum up, the difference between these two datasets underline the importance of having different datasets for developing an optimal algorithm. So, it is crucial to continuously expand and enhance the available datasets in order to advance this area.

7.2. Testing the trained model

Three experiments were done for the testing of the RGB model, resulting in one correct prediction for the 11k hands dataset, one correct prediction for the HKPU dataset, and one incorrect prediction.

The following Figure 31, represents the hand images of the person with ID 0 that the model was trained with. It is worth noting that user 0 belongs to the 11k hands dataset.

Figure 32: Hand images belonging to user 0, which were used to train the model

The prediction made by the model for user with ID 0 was correct, indicating that the model was able to classify correctly the hand image of user 0. By correctly predicting the user's identity, it suggests that the model had learned to recognise hand images, belonging to a specific person, from this dataset. Below, there are hand images of the user with ID 0, that the model was tested with.

Figure 33: Hand images belonging to user 0, which were used to test the model

The hand images of the individual with ID 1 whose hands were used to train the model are shown in Figure 34 below. It is important to note that user 1 is a part of the dataset for HKPU.

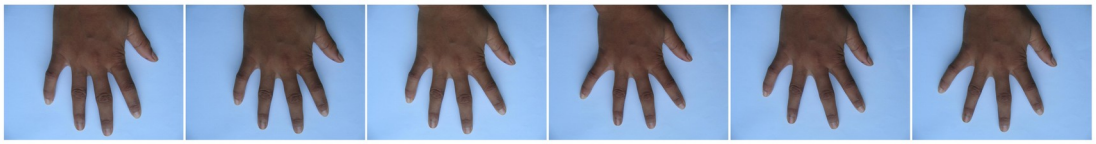


Figure 34: Hand images belonging to user 1, which were used to train the model

The model made a correct prediction for the user with ID 1. The user's hands with ID 1 hands are shown in the Figure 35 below, which were used to examine the model.

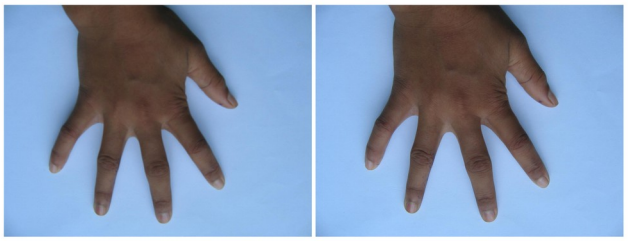


Figure 35: Hand images belonging to user 1, which were used to test the model

It is worth noting that user 1 has hand images that were similar to those of user 3. Although the hand images are similar, the model was still able to correctly predict the user's identity. Those results might indicate that the model was able to identify human beings from distinguishable hand features, and was not biased on matching images based on overall similarity. The hand images of user with 3 can be observed in the following Figure 36.

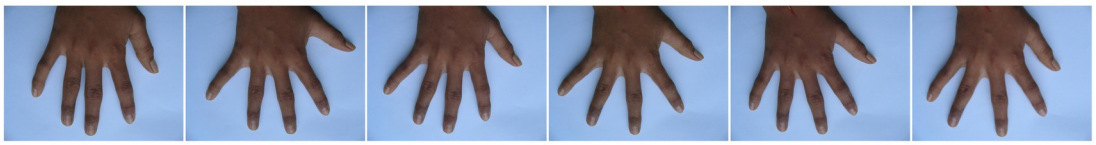


Figure 36: Hand images belonging to user 3, which were used to train the model

Figure 37 below displays the hand images of the person with ID 6 whose hands were used to train the model. It is essential to keep in mind that user 6 is included in the data set for 11k hands.

Figure 37: Hand images belonging to user 6, which were used to train the model

The model's prediction for user with ID 6 was incorrect, as the hand images, shown in Figure 38 below, were identified as user 16. This suggests that the model might have difficulty distinguishing between hand images with similar positions and postures.

Figure 38: Hand images belonging to user 6, which were used to test the model

It is vital to note that both user 6 and user 16 have similar postures and hand positions, which may have influenced the inaccurate prediction. This might suggest that there was a small bias in the model towards particular hand postures or positions, which could be explored further to increase the model's accuracy in future work.

Figure 39: Hand images belonging to user 16, which were used to train the model

7.3. Comparison of the proposed system with existing algorithms

The proposed model was compared with the CNN model of the “11k hands”. The comparisons were done both for the 11k hands dataset alone and for the combination of the 11k hands and the HKPU datasets.

However, it was not possible to compare the proposed solution with the biometric identification performance of the 11k hands, due to some errors in the Matlab environment. Instead, the proposed solution was compared with the 11k hands model for gender recognition, which is a related task.

Table 6: Comparison between the proposed algorithm and the 11k hands, using the 11k hands dataset

Algorithms	11k Hands	Proposed solution
Accuracy	94%	83.85%
Precision	94.5%	82.37%
Recall	93.4%	82.25%
F1-Score	93.9%	82.3%

The results from the Table 6 above show that the 11k Hands algorithm was having greater performance from the proposed hand identification solution.

Table 7: Comparison between the proposed algorithm and the 11k hands, using the combined dataset

Algorithms	11k Hands	Proposed solution
Accuracy	50%	83%
Precision	50%	82%
Recall	92.6%	83%
F1-Score	64.9%	81%

However, when combining the two datasets, it can be observed from the Table 7 above that the proposed algorithm outperforms the 11k hands algorithm.

8. Conclusion

The goal of this project was to create an algorithm that identifies individuals based on RGB and hyperspectral hand images. A different approach to existing solutions is to develop a less complex algorithm that was more accurate for different data sets with various backgrounds and lighting conditions.

The first step of this project was to acquire both RGB and hyperspectral hand datasets. The RGB images were pre-processed by first removing the background, and using a guided filter in combination with a high pass filter, in order to get a high detailed image. For the hyperspectral images, PCA was applied for the first 3 principal components, which had the higher eigenvalues. Those images were then fed to a pretrained CNN, called VGG16, where hand features like hand shape, skin texture, knuckle patterns, and vein patterns, were extracted and trained. Lastly, the model was evaluated by applying evaluation metrics, to unused images. The optimal hyperparameters were found and used to evaluate the model in different preprocessing techniques. The results showed that the high-detailed BGR pre-processing produced the best results in terms of accuracy and loss. However, the algorithm showed some bias towards tha hand posture and position. Even though, the 11k hands algorithm outperformed the proposed solution in the 11k hands dataset, it was not as accurate for the combined dataset. In addition, the algorithm showed greater performance when tested with hyperspectral images, with higher evaluation metrics and lower loss.

In conclusion, the project showed the potential of pre-processing methods and pre-trained models for hand recognition and offered insightful data on how various algorithms work. The project's results can be useful to other related tasks like gesture recognition and hand pose estimation as well as computer vision and hand identification research and practise.

9. Future work

The proposed solution can be improved by designing a better CNN network that provides more stable evaluations. This can involve exploring different architectures and hyperparameters to optimise the model's performance. Additionally, we can investigate the use of different pre-trained models that may be more suitable for hand recognition tasks.

Also, we can evaluate the model with different datasets, having different backgrounds, lighting conditions, and image resolutions. In doing so, the effectiveness and adaptability of the model can be evaluated in more depth. Additionally, it may be advantageous to try the model with more users to see how well it scales with more identities. The algorithm would be tested, also, on palm identification, expect from the right dorsal hands. This will increase the accuracy and reliability of the system in identifying people. Also, the hand posture bias observed in the current person hand identification algorithm could be eliminated by using a Support Vector Machine (SVM) model, which can improve classification accuracy and reduce the impact of the bias.

Finally, a hardware tool with a user-friendly graphical interface can be developed for a system that automatically identifies individuals based on hand images. A hardware tool with a camera could be used, to instantly capture and display images of the hands on a computer screen. Numerous applications could be made of this, including in patient identity in healthcare and security and authentication systems.

As it can be observed, the proposed solution provides a strong starting point for further research and advancement in this area, where there are many unexplored possibilities.

10. References

[1] O. Bowcott and O. B. L. affairs correspondent, ‘UK’s facial recognition technology “breaches privacy rights”’, *The Guardian*, Jun. 23, 2020. Accessed: Mar. 29, 2023. [Online]. Available: https://www.theguardian.com/technology/2020/jun/23/uks-facial-recognition-technology-breaches-privacy-rights

[2] L. Alzubaidi *et al.*, ‘Review of deep learning: concepts, CNN architectures, challenges, applications, future directions’, *Journal of Big Data*, vol. 8, no. 1, p. 53, Mar. 2021, doi: 10.1186/s40537-021-00444-8.

[3] T. Savič and N. Pavešić, ‘Personal recognition based on an image of the palmar surface of the hand’, *Pattern Recognition*, vol. 40, no. 11, pp. 3152–3163, Nov. 2007, doi: 10.1016/j.patcog.2007.03.005.

[4] M. Alghamdi, P. Angelov, and L. P. Alvaro, ‘Person identification from fingernails and knuckles images using deep learning features and the Bray-Curtis similarity measure’, *Neurocomputing*, vol. 513, pp. 83–93, Nov. 2022, doi: 10.1016/j.neucom.2022.09.123.

[5] R. Castro-Ortega, C. Toxqui-Quitl, G. Cristobal, J. V. Marcos, A. Padilla-vivanco, and R. Pérez, *Analysis of the hand vein pattern for people recognition*. 2015. doi: 10.1117/12.2188817.

[6] J. M. Amigo, ‘Chapter 1.1 - Hyperspectral and multispectral imaging: setting the scene’, in *Data Handling in Science and Technology*, J. M. Amigo, Ed., in Hyperspectral Imaging, vol. 32. Elsevier, 2019, pp. 3–16. doi: 10.1016/B978-0-444-63977-6.00001-8.

[7] C. Mingtsung and L. Cai, *Research on the Application of Face Recognition System*. 2020. doi: 10.2991/assehr.k.200727.057.

[8] ‘What makes our hands unique?’, Jun. 07, 2022. https://erc.europa.eu/projects-figures/stories/what-makes-our-hands-unique (accessed Mar. 29, 2023).

[9] ‘Color Spaces, clarkvision.com’. https://clarkvision.com/articles/color-spaces/ (accessed Mar. 27, 2023).

[10] M. Afifi, ‘11K Hands: Gender recognition and biometric identification using a large dataset of hand images’, *Multimed Tools Appl*, vol. 78, no. 15, pp. 20835–20854, Aug. 2019, doi: 10.1007/s11042-019-7424-8.

[11] S. Tsutsui, Y. Fu, and D. Crandall, ‘Whose hand is this? Person Identification from Egocentric Hand Gestures’. arXiv, Nov. 17, 2020. Accessed: Mar. 25, 2023. [Online]. Available: http://arxiv.org/abs/2011.08900

[12] W. Nie and B. Zhang, ‘Robust and adaptive ROI extraction for hyperspectral dorsal hand vein images’, *IET Computer Vision*, vol. 13, no. 6, pp. 595–604, 2019, doi: 10.1049/iet-cvi.2018.5732.

[13] M. A. Ferrer, A. Morales, and A. Díaz, ‘An approach to SWIR hyperspectral hand biometrics’, *Information Sciences*, vol. 268, pp. 3–19, Jun. 2014, doi: 10.1016/j.ins.2013.10.011.

[14] D. Wei, L. Zhang, D. Zhang, and Q. Pan, ‘Studies on Hyperspectral Face Recognition in Visible Spectrum With Feature Band Selection’, *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 40, pp. 1354–1361, Nov. 2010, doi: 10.1109/TSMCA.2010.2052603.

[15] G. Aksu, C. O. Güzeller, and M. T. Eser, ‘The Effect of the Normalization Method Used in Different Sample Sizes on the Success of Artificial Neural Network Model’, *International Journal of Assessment Tools in Education*, pp. 170–192, Apr. 2019, doi: 10.21449/ijate.479404.

[16] T. A. Team, ‘Scaling vs. Normalizing Data – Towards AI’. https://towardsai.net/p/data-science/scaling-vs-normalizing-data-5c3514887a84, https://towardsai.net/p/data-science/scaling-vs-normalizing-data-5c3514887a84 (accessed Mar. 21, 2023).

[17] A. Ponraj, ‘A Tip A Day — Python Tip #8: Why should we Normalize image pixel values or divide by 255?’, *Analytics Vidhya*, Feb. 22, 2021. https://medium.com/analytics-vidhya/a-tip-a-day-python-tip-8-why-should-we-normalize-image-pixel-values-or-divide-by-255-4608ac5cd26a (accessed Mar. 21, 2023).

[18] B. Hiremath, ‘All You Need to Know About Guided Image Filtering’, *Analytics India Magazine*, Dec. 22, 2021. https://analyticsindiamag.com/all-you-need-to-know-about-guided-image-filtering/ (accessed Mar. 25, 2023).

[19] R. Dutt, ‘PCA on HyperSpectral Data’, *Medium*, Aug. 24, 2021. https://towardsdatascience.com/pca-on-hyperspectral-data-99c9c5178385 (accessed Mar. 30, 2023).

[20] A. Dantas de Oliveira, V. Da Silva, M. Pimentel, G. Vinhas, C. Pasquini, and Y. Almeida, ‘Use of Infrared Spectroscopy and Near Infrared Hyperspectral Images to Evaluate Effects of Different Chemical Agents on PET Bottle Surface’, *Materials Research*, vol. 21, Jul. 2018, doi: 10.1590/1980-5373-MR-2017-0949.

[21] ‘Hyperspectral Imaging - an overview | ScienceDirect Topics’. https://www.sciencedirect.com/topics/medicine-and-dentistry/hyperspectral-imaging (accessed Mar. 27, 2023).

[22] ‘What is Hyperspectral Imaging? How does it work?’, *HAIP Solutions*. https://www.haip-solutions.com/what-is-hyperspectral-imaging/ (accessed Mar. 30, 2023).

[23] P. Ingle, ‘Top Neural Network Architectures For Machine Learning Researchers’, *MarkTechPost*, Sep. 23, 2022. https://www.marktechpost.com/2022/09/23/top-neural-network-architectures-for-machine-learning-researchers/ (accessed Mar. 25, 2023).

[24] H. Singh, ‘Deep Learning 101: Beginners Guide to Neural Network’, *Analytics Vidhya*, Mar. 01, 2021. https://www.analyticsvidhya.com/blog/2021/03/basics-of-neural-network/ (accessed Mar. 25, 2023).

[25] ‘CNN Architecture - Detailed Explanation’, *InterviewBit*, Jun. 10, 2022. https://www.interviewbit.com/blog/cnn-architecture/ (accessed Mar. 25, 2023).

[26] ‘Visual Geometry Group - University of Oxford’. https://www.robots.ox.ac.uk/~vgg/research/very_deep/ (accessed Mar. 25, 2023).

[27] M. ul Hassan, ‘VGG16 - Convolutional Network for Classification and Detection’, Nov. 20, 2018. https://neurohive.io/en/popular-networks/vgg16/ (accessed Mar. 25, 2023).

[28] ‘GPDS Group’. https://gpds.ulpgc.es/ (accessed Mar. 22, 2023).

[29] M. Bača, P. Grd, T. Fotak, M. Bača, P. Grd, and T. Fotak, *Basic Principles and Trends in Hand Geometry and Hand Shape Biometrics*. IntechOpen, 2012. doi: 10.5772/51912.

[30] ‘Hand geometry’, *Wikipedia*. Mar. 09, 2023. Accessed: Mar. 30, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hand_geometry&oldid=1143668128

[31] J. Xie, L. Zhang, J. You, D. Zhang, and X. Qu, ‘A study of hand back skin texture patterns for personal identification and gender classification’, *Sensors (Basel)*, vol. 12, no. 7, pp. 8691–8709, 2012, doi: 10.3390/s120708691.

[32] D. B. Fischer, ‘7 Signs of Aging Hands and How to Treat Them’. https://www.beverlyfischer.net/blog/prevent-aging-hands (accessed Mar. 30, 2023).

[33] D. Kusanagi, S. Aoyama, K. Ito, and T. Aoki, ‘A practical person authentication system using second minor finger knuckles for door security’, *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, p. 8, Mar. 2017, doi: 10.1186/s41074-017-0016-5.

[34] A. Yüksel, L. Akarun, and B. Sankur, ‘Biometric Identification through Hand Vein Patterns’, in *2010 International Workshop on Emerging*

Techniques and Challenges for Hand-Based Biometrics, Aug. 2010, pp. 1–6. doi: 10.1109/ETCHB.2010.5559295.

[35] ‘Lancaster leads on pioneering hand identification research’, *EurekAlert!* https://www.eurekalert.org/news-releases/815306 (accessed Mar. 29, 2023).

[36] J. R. G. Neves and P. L. Correia, ‘Hand veins recognition system’, in *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, Jan. 2014, pp. 122–129.

[37] H. Akbari, Y. Kosugi, K. Kojima, and N. Tanaka, ‘Blood vessel detection and artery-vein differentiation using hyperspectral imaging’, in *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Sep. 2009, pp. 1461–1464. doi: 10.1109/IEMBS.2009.5332920.

[38] ‘Understanding VGG16: Concepts, Architecture, and Performance’, *Datagen*. https://datagen.tech/guides/computer-vision/vgg16/ (accessed Mar. 24, 2023).

[39] ‘11k Hands’. https://sites.google.com/view/11khands (accessed Mar. 22, 2023).

[40] S. N. Mohd Rum and F. Az, ‘FishDeTec: A Fish Identification Application using Image Recognition Approach’, *International Journal of Advanced Computer Science and Applications*, vol. 12, Jan. 2021, doi: 10.14569/IJACSA.2021.0120312.

[41] K. He, X. Zhang, S. Ren, and J. Sun, ‘Deep Residual Learning for Image Recognition’, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[42] ‘Transfer Learning with ResNet in PyTorch | Pluralsight’. https://www.pluralsight.com/guides/introduction-to-resnet (accessed Mar. 30, 2023).

[43] V. dos Santos, ‘Hand-gesture recognition neural network analysis’, *Medium*, Nov. 16, 2020. https://vitorgabo.medium.com/hand-gesture-recognition-neural-network-analysis-bfe2fef78161 (accessed Mar. 25, 2023).

[44] A. R. Asif *et al.*, ‘Performance Evaluation of Convolutional Neural Network for Hand Gesture Recognition Using EMG’, *Sensors (Basel)*, vol. 20, no. 6, p. 1642, Mar. 2020, doi: 10.3390/s20061642.

11. Appendix

11.1. rgb_notebook.ipynb

Before you start

Move to the directory of the notebook - only for Google Colab

```
cd /content/drive/MyDrive/Colab Notebooks/rgb_hand
```

```
/content/drive/MyDrive/Colab Notebooks/rgb_hand
```

Load Libraries

```
import pandas as pd
import cv2
import numpy as np
from cv2.ximgproc import guidedFilter
import mediapipe as mp
from keras import Sequential
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.metrics import classification_report
```

Load the CSV file

```
ds = pd.read_csv("dorsal_right_30_comb.csv")
```

Split train and test datasets

```
# Create a dictionary to store the count of images for each ID
id_counts = {}

# Iterate through the DataFrame to count the number of images
for each ID
for idx, row in ds.iterrows():
    id_num = row["ID"]
    if id_num in id_counts:
        id_counts[id_num] += 1
    else:
        id_counts[id_num] = 1
```

```
# Create a list to store the indices of the test data
test_indices = []

# Iterate through the DataFrame to find the last 2 images for
each ID
for idx, row in ds.iterrows():
    id_num = row["ID"]
    if id_counts[id_num] > 2:
        id_counts[id_num] -= 1
    else:
        test_indices.append(idx)

# Split the DataFrame into training and testing datasets
train_ds = ds.drop(test_indices)
test_ds = ds.iloc[test_indices]
```

Pre-Processing

Remove Background from Hand Image

```
def remove_background(img_path):
  mp_selfie_segmentation = mp.solutions.selfie_segmentation
  selfie_segmentation =
mp_selfie_segmentation.SelfieSegmentation(model_selection=1)

  # Read image
  image = cv2.imread(img_path)

  # Resize to 244x244
  image = cv2.resize(image, (224, 224))

  # Convert image from BGR to RGB
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

  # Get the segmentation mask
  results = selfie_segmentation.process(image)

  # Extract the segmentation mask
  mask = results.segmentation_mask

  # Create a binary mask where foreground is white and the
background is black
  mask = np.stack((mask,)*3, axis=-1)
  mask = np.where(mask > 0.1, 255, 0).astype('uint8')

  # Invert the mask
  mask_inv = cv2.bitwise_not(mask)
```

```
  # Create a white background image of the same size as the
original image
  background = np.ones_like(image)*255

  # Apply the mask to the original image to get foreground
  foreground = cv2.bitwise_and(image, mask)

  # Apply the inverted mask to the white background to get
background
  background = cv2.bitwise_and(background, mask_inv)

  # Combine the foreground and background to get the final
result
  result = cv2.add(foreground, background)

  return result
```

High Detailed BGR Pre-Processing

```
def high_bgr(img_path):

  # Normalise the image
  norm = np.array(img_path, dtype=np.float32) / 255.0

  # Apply the guided filter to get a smoothed - low frequency
image
  low_img = guidedFilter(norm, norm, 10, 0.05)

  # Convert the image to BGR
  bgr = cv2.cvtColor(low_img, cv2.COLOR_RGB2BGR)

  # Divide normalised image to low frequency image
  high_bgr =
np.uint8(cv2.normalize(np.divide(cv2.cvtColor(norm,
cv2.COLOR_RGB2BGR), bgr), None, 0, 255, cv2.NORM_MINMAX))

  return high_bgr
```

Function that applies background removal and high detail image conversion

```
def preprocess_image(img_path):

  # Remove the background
  image = remove_background(img_path)

  # Get high detailed image
```

```
  output_img = high_bgr(image) # Change this to apply
different pre-processing

  return output_img
```

Prepare the train and test data

```
# Prepare the train data
train_data = []
for i, row in train_ds.iterrows():
    img_path = "/content/drive/MyDrive/Colab
Notebooks/rgb_hand/hands_comb/" + row["ImageName"]
    output_img = preprocess_image(img_path)
    train_data.append(output_img)

train_data = np.array(train_data)
train_labels = train_ds["ID"].values

# Prepare test data
test_data = []
for i, row in test_ds.iterrows():
    img_path = "/content/drive/MyDrive/Colab
Notebooks/rgb_hand/hands_comb/" + row["ImageName"]
    output_img = preprocess_image(img_path)
    test_data.append(output_img)

test_data = np.array(test_data)
test_labels = test_ds["ID"].values
```

Feature extraction and Training

Split the train data into training and validation sets

```
train_data, val_data, train_labels, val_labels =
train_test_split(train_data, train_labels, test_size=0.25,
random_state=42)
```

Model architecture

```
# Load the VGG16 model
vgg_model = VGG16(weights='imagenet', include_top=False,
input_shape=(224,224,3))

# Freeze the layers
for layer in vgg_model.layers:
    layer.trainable = False

# Add the fully connected layers
model = Sequential()
model.add(vgg_model)
```

```
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(30, activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(lr=0.00001), metrics=['accuracy'])

# Display the model structure
model.summary()
WARNING:absl:`lr` is deprecated, please use `learning_rate`
instead, or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.Adam.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)
14714688


 flatten (Flatten)           (None, 25088)             0



 dense (Dense)               (None, 4096)
102764544


 dense_1 (Dense)             (None, 30)                122910



=================================================================
Total params: 117,602,142
Trainable params: 102,887,454
Non-trainable params: 14,714,688
_________________________________________________________________
```

Train model and save it

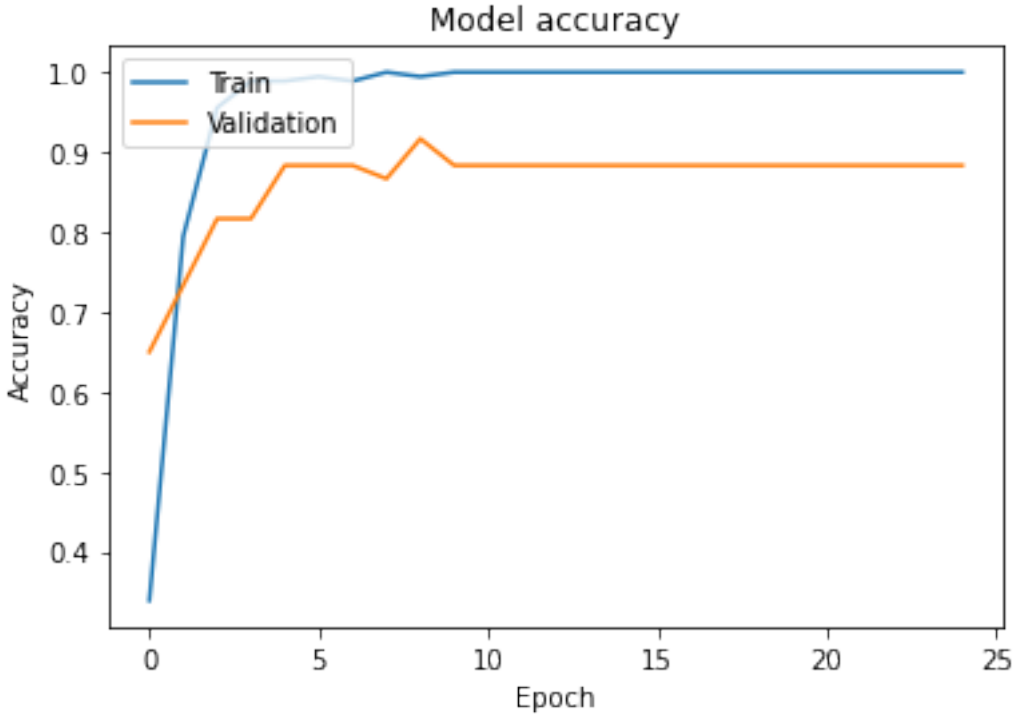
```
model_tr = model.fit(train_data, train_labels, epochs=25,
batch_size=12, shuffle=True, validation_data=(val_data,
val_labels))

model.save('trained_model.h5')
```

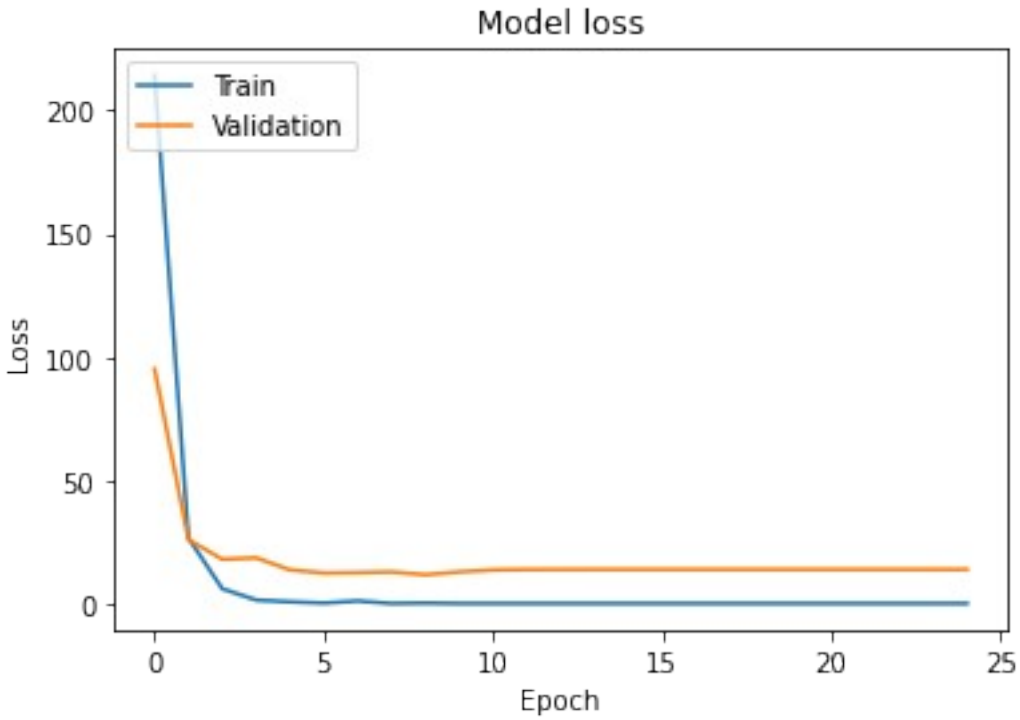
```
Epoch 1/25
15/15 [==============================] - 6s 112ms/step - loss:
214.3158 - accuracy: 0.3389 - val_loss: 95.1121 -
val_accuracy: 0.6500
Epoch 2/25
15/15 [==============================] - 1s 91ms/step - loss:
26.5355 - accuracy: 0.7944 - val_loss: 25.9139 - val_accuracy:
0.7333
Epoch 3/25
15/15 [==============================] - 1s 90ms/step - loss:
5.9778 - accuracy: 0.9556 - val_loss: 17.9059 - val_accuracy:
0.8167
Epoch 4/25
15/15 [==============================] - 1s 90ms/step - loss:
1.3852 - accuracy: 0.9889 - val_loss: 18.5049 - val_accuracy:
0.8167
Epoch 5/25
15/15 [==============================] - 1s 87ms/step - loss:
0.6897 - accuracy: 0.9889 - val_loss: 13.6361 - val_accuracy:
0.8833
Epoch 6/25
15/15 [==============================] - 1s 90ms/step - loss:
0.1680 - accuracy: 0.9944 - val_loss: 12.3573 - val_accuracy:
0.8833
Epoch 7/25
15/15 [==============================] - 1s 87ms/step - loss:
1.0267 - accuracy: 0.9889 - val_loss: 12.5091 - val_accuracy:
0.8833
Epoch 8/25
15/15 [==============================] - 1s 87ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 12.8061 -
val_accuracy: 0.8667
Epoch 9/25
15/15 [==============================] - 1s 91ms/step - loss:
0.1565 - accuracy: 0.9944 - val_loss: 11.6607 - val_accuracy:
0.9167
Epoch 10/25
15/15 [==============================] - 1s 92ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 12.8111 -
val_accuracy: 0.8833
Epoch 11/25
15/15 [==============================] - 1s 89ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.6073 -
val_accuracy: 0.8833
Epoch 12/25
15/15 [==============================] - 1s 91ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.7802 -
val_accuracy: 0.8833
Epoch 13/25
15/15 [==============================] - 1s 87ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8174 -
val_accuracy: 0.8833
```

```
Epoch 14/25
15/15 [==============================] - 1s 87ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8254 -
val_accuracy: 0.8833
Epoch 15/25
15/15 [==============================] - 1s 91ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8271 -
val_accuracy: 0.8833
Epoch 16/25
15/15 [==============================] - 1s 91ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8275 -
val_accuracy: 0.8833
Epoch 17/25
15/15 [==============================] - 1s 91ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8276 -
val_accuracy: 0.8833
Epoch 18/25
15/15 [==============================] - 1s 93ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8276 -
val_accuracy: 0.8833
Epoch 19/25
15/15 [==============================] - 1s 94ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8276 -
val_accuracy: 0.8833
Epoch 20/25
15/15 [==============================] - 1s 93ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8276 -
val_accuracy: 0.8833
Epoch 21/25
15/15 [==============================] - 1s 96ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8276 -
val_accuracy: 0.8833
Epoch 22/25
15/15 [==============================] - 1s 94ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8276 -
val_accuracy: 0.8833
Epoch 23/25
15/15 [==============================] - 1s 92ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8276 -
val_accuracy: 0.8833
Epoch 24/25
15/15 [==============================] - 1s 92ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8276 -
val_accuracy: 0.8833
Epoch 25/25
15/15 [==============================] - 1s 92ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 13.8276 -
val_accuracy: 0.8833
```

Plot learning curves

```
# Plot accuracy
plt.plot(model_tr.history['accuracy'])
plt.plot(model_tr.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot loss
plt.plot(model_tr.history['loss'])
plt.plot(model_tr.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```





Evaluation and Testing

Evaluation Metrics

```
# Predict the labels for test data
predictions = model.predict(test_data)
predicted_labels = np.argmax(predictions, axis=1)

# Calculate loss
test_loss, test_accu = model.evaluate(test_data, test_labels)
print('Accuracy:', test_accu)
print('Loss:', test_loss)

# Calculate precision, recall, and F1-score for each class
target_names = [f"ID_{i}" for i in range(30)]
report = classification_report(test_labels, predicted_labels,
target_names=target_names)
print(report)
```

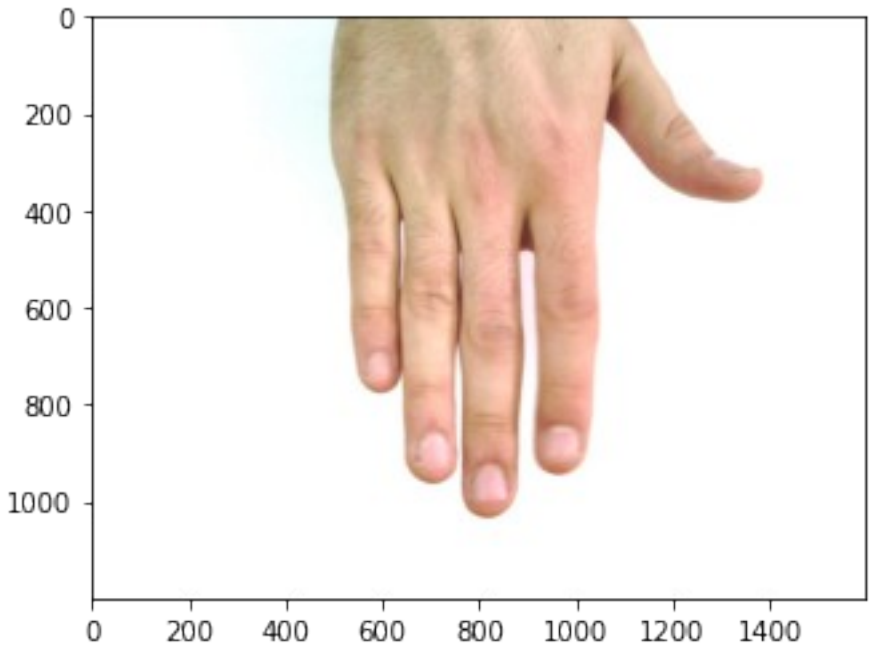
```
2/2 [==============================] - 5s 2s/step
2/2 [==============================] - 1s 142ms/step - loss:
28.5314 - accuracy: 0.8167
Accuracy: 0.8166666626930237
Loss: 28.531368255615234
              precision    recall  f1-score   support

        ID_0       1.00      1.00      1.00         2
        ID_1       1.00      1.00      1.00         2
```

```
        ID_2       1.00      1.00      1.00         2
        ID_3       1.00      0.50      0.67         2
        ID_4       1.00      1.00      1.00         2
        ID_5       1.00      1.00      1.00         2
        ID_6       0.00      0.00      0.00         2
        ID_7       0.40      1.00      0.57         2
        ID_8       0.50      0.50      0.50         2
        ID_9       1.00      1.00      1.00         2
       ID_10       0.67      1.00      0.80         2
       ID_11       1.00      1.00      1.00         2
       ID_12       1.00      1.00      1.00         2
       ID_13       1.00      0.50      0.67         2
       ID_14       1.00      1.00      1.00         2
       ID_15       1.00      1.00      1.00         2
       ID_16       0.25      0.50      0.33         2
       ID_17       1.00      1.00      1.00         2
       ID_18       0.00      0.00      0.00         2
       ID_19       0.67      1.00      0.80         2
       ID_20       1.00      1.00      1.00         2
       ID_21       1.00      1.00      1.00         2
       ID_22       0.67      1.00      0.80         2
       ID_23       1.00      0.50      0.67         2
       ID_24       1.00      0.50      0.67         2
       ID_25       1.00      1.00      1.00         2
       ID_26       0.50      0.50      0.50         2
       ID_27       1.00      1.00      1.00         2
       ID_28       1.00      1.00      1.00         2
       ID_29       1.00      1.00      1.00         2

    accuracy                           0.82        60
   macro avg       0.82      0.82      0.80        60
weighted avg       0.82      0.82      0.80        60

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-
score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-
score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Testing

Correctly Predicted from "11k hands"

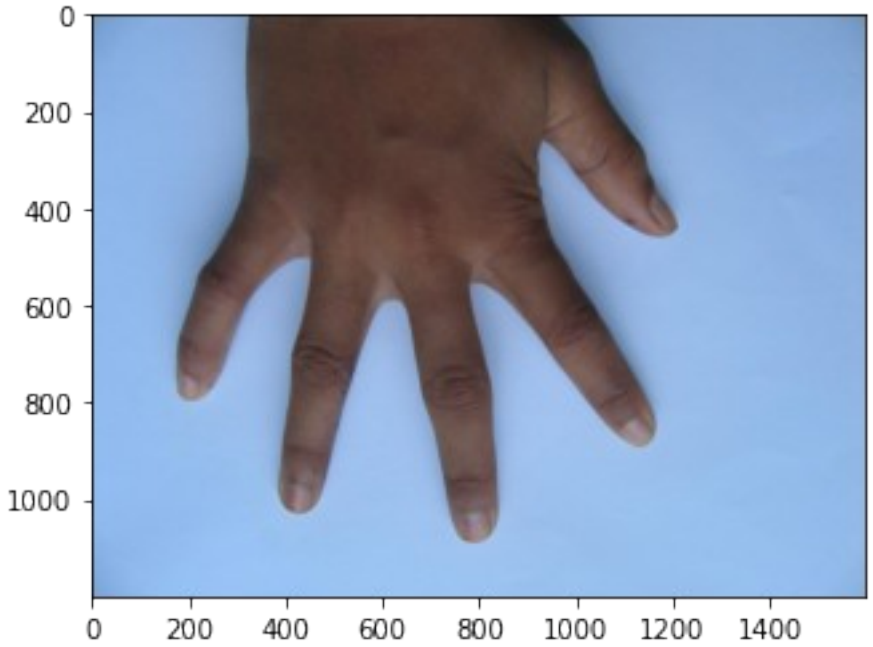
```
# Load the saved model
loaded_model = tf.keras.models.load_model('trained_model.h5')

# Input image path
img_path = "/content/drive/MyDrive/Colab
Notebooks/rgb_hand/hands_comb/Hand_0000010.jpg"

# Input image ID
img_ID = 0

# Preprocess the input image
output_img = preprocess_image(img_path)

# Expand dimension from (224,224,3) to (1, 224, 224, 3) for
the prediction
input_image = np.expand_dims(output_img, axis=0)

# Pass the preprocessed image through the loaded model
prediction = loaded_model.predict(input_image)


# Extract the predicted class with the highest probability
predicted_class = np.argmax(prediction)


# Show the input image with its ID and the predicted ID
plt.imshow(cv2.cvtColor(cv2.imread(img_path),
cv2.COLOR_BGR2RGB))
print('Input image ID:', img_ID)
print('Predicted label:', predicted_class)
1/1 [==============================] - 1s 805ms/step
Input image ID: 0
Predicted label: 0
```



Correctly Predicted from "HKPU"

```
# Load the saved model
loaded_model = tf.keras.models.load_model('trained_model.h5')

# Input image path
img_path = "/content/drive/MyDrive/Colab
Notebooks/rgb_hand/hands_comb/195_IMG_4688.jpg"

# Input image ID
img_ID = 1

# Preprocess the input image
output_img = preprocess_image(img_path)

# Expand dimension from (224,224,3) to (1, 224, 224, 3) for
the prediction
input_image = np.expand_dims(output_img, axis=0)

# Pass the preprocessed image through the loaded model
prediction = loaded_model.predict(input_image)


# Extract the predicted class with the highest probability
predicted_class = np.argmax(prediction)


# Show the input image with its ID and the predicted ID
plt.imshow(cv2.cvtColor(cv2.imread(img_path),
```

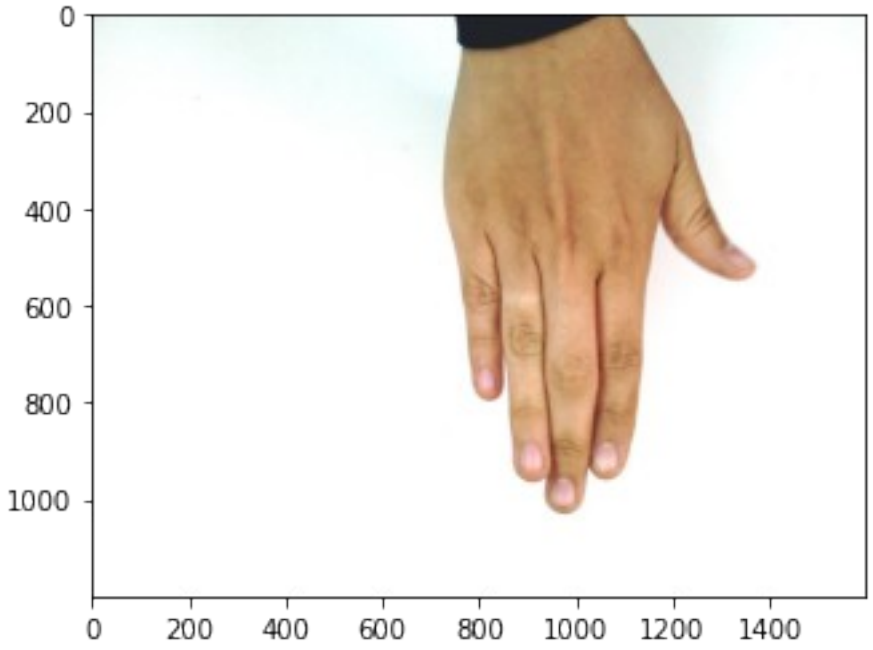
```
cv2.COLOR_BGR2RGB))
print('Input image ID:', img_ID)
print('Predicted label:', predicted_class)
1/1 [==============================] - 0s 172ms/step
Input image ID: 1
Predicted label: 1
```



Incorrect Prediction

```
# Load the saved model
loaded_model = tf.keras.models.load_model('trained_model.h5')

# Input image path
img_path = "/content/drive/MyDrive/Colab
Notebooks/rgb_hand/hands_comb/Hand_0000717.jpg"

# Input image ID
img_ID = 3

# Preprocess the input image
output_img = preprocess_image(img_path)

# Expand dimension from (224,224,3) to (1, 224, 224, 3) for
the prediction
input_image = np.expand_dims(output_img, axis=0)

# Pass the preprocessed image through the loaded model
prediction = loaded_model.predict(input_image)


# Extract the predicted class with the highest probability
```

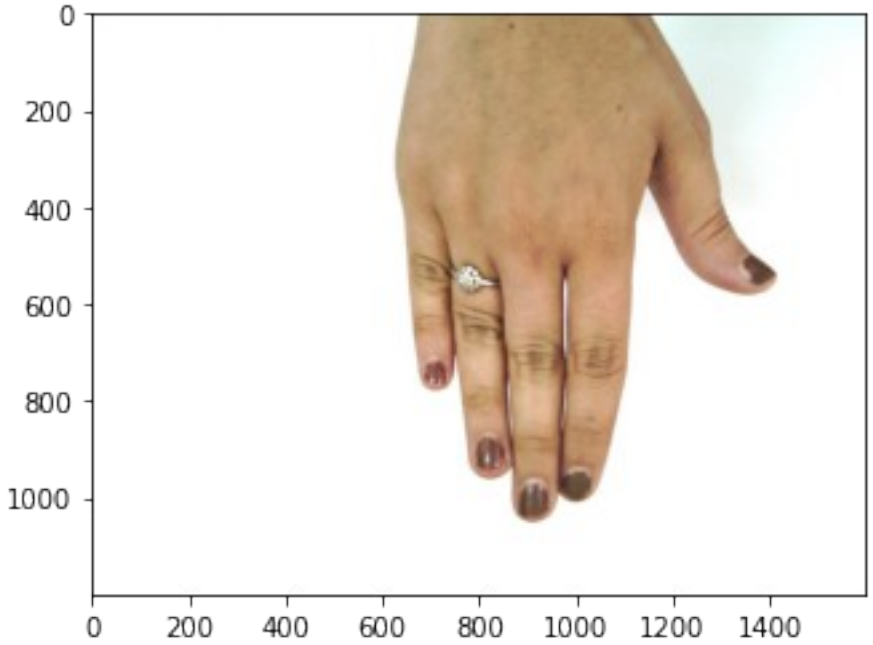
```
predicted_class = np.argmax(prediction)


# Show the input image with its ID and the predicted ID
plt.imshow(cv2.cvtColor(cv2.imread(img_path),
cv2.COLOR_BGR2RGB))
print('Input image ID:', img_ID)
print('Predicted label:', predicted_class)
WARNING:tensorflow:5 out of the last 6 calls to <function
Model.make_predict_function.<locals>.predict_function at
0x7fbaa87b1820> triggered tf.function retracing. Tracing is
expensive and the excessive number of tracings could be due to
(1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects
instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has
reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracin
g and https://www.tensorflow.org/api_docs/python/tf/function
for  more details.
1/1 [==============================] - 0s 459ms/step
Input image ID: 6
Predicted label: 16
```



Hand Image with ID 16

```
# Input image path
img_path = "/content/drive/MyDrive/Colab
Notebooks/rgb_hand/hands_comb/Hand_0001099.jpg"

# Input image ID
```

```
img_ID = 16

# Show the input image with its ID and the predicted ID
plt.imshow(cv2.cvtColor(cv2.imread(img_path),
cv2.COLOR_BGR2RGB))
print('Input image ID:', img_ID)
Input image ID: 16
```



11.2. hsi_notebook.ipynb

Move to the directory of the notebook - only for Google Colab

```
cd drive/MyDrive/Colab Notebooks/hyperspectral_hands/
/content/drive/MyDrive/Colab Notebooks/hyperspectral_hands
```

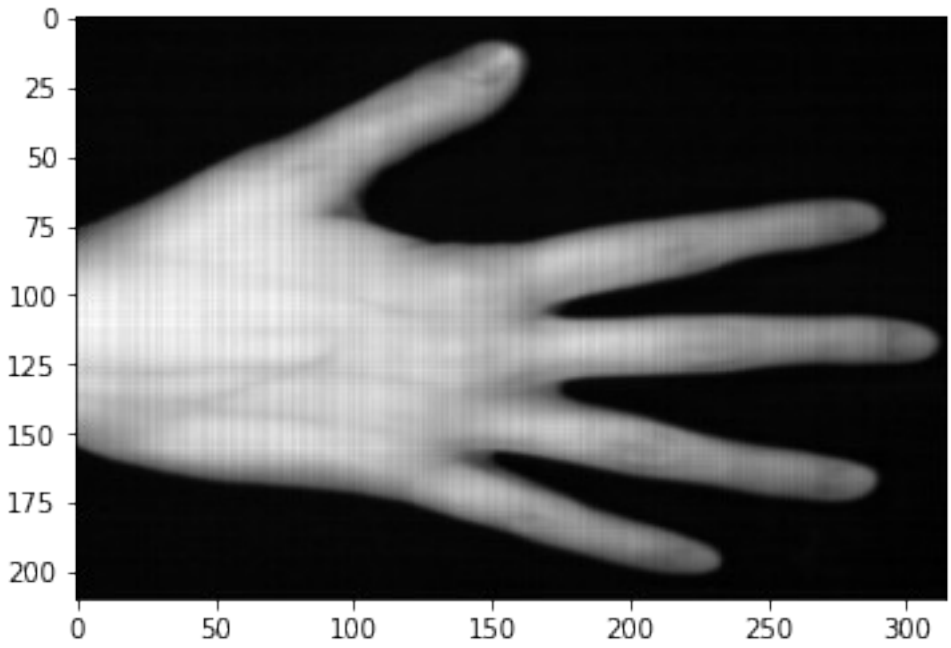
Load Libraries

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
import scipy.io as sio
import matplotlib.pyplot as plt
from skimage import color, data, restoration
from sklearn.decomposition import PCA
from keras import Sequential
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
```

Show Hyperspectral Hand Images and Eigenvalues

Hyperspectral Hand Image band 10

```
# Load dataset
data_train = pd.read_csv('hands_train.csv')

img_path = "/content/drive/MyDrive/Colab
Notebooks/hyperspectral_hands/users/dorsal/cubo001_1.mat"
image = sio.loadmat(img_path)
image = image['cubo']
image = np.transpose(image, (1, 2, 0))
image = image[45:255, 10:325, 10]

plt.imshow(image, cmap="gray")
plt.show()
```



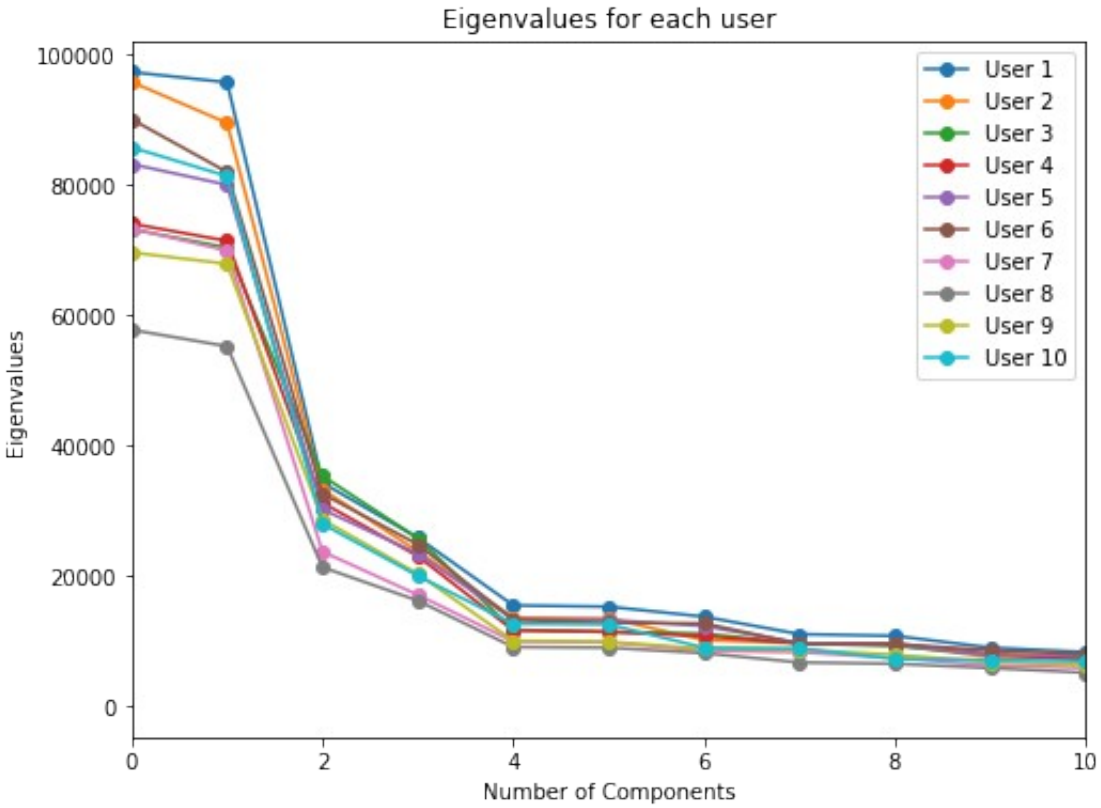
Apply PCA to every user (each 1 image)

```
# Load data
data_train = pd.read_csv('hands_pca.csv')

# Apply PCA for each user
pca_results = []
for i, row in data_train.iterrows():
    img_path = "/content/drive/MyDrive/Colab
Notebooks/hyperspectral_hands/users/dorsal/" +
row['ImageName']
    img_tr = sio.loadmat(img_path)
    img_tr = img_tr['cubo']
    img = np.transpose(img_tr, (1, 2, 0))
    img = img[45:255, 10:325, :]
    spectral_data = img.reshape(256, -1)
    pca = PCA()
    pca.fit(spectral_data.T)
    pca_results.append(pca)
```

Plot Eigenvalues for each user

```
# Plot eigenvalues for each user
plt.figure(figsize=(8, 6))
for i, pca in enumerate(pca_results):
    plt.plot(pca.explained_variance_, label=f'User {i+1}',
marker='o', markevery=1)
plt.xlabel('Number of Components')
plt.ylabel('Eigenvalues')
plt.xlim(0, 10)  # set x-axis limits
```

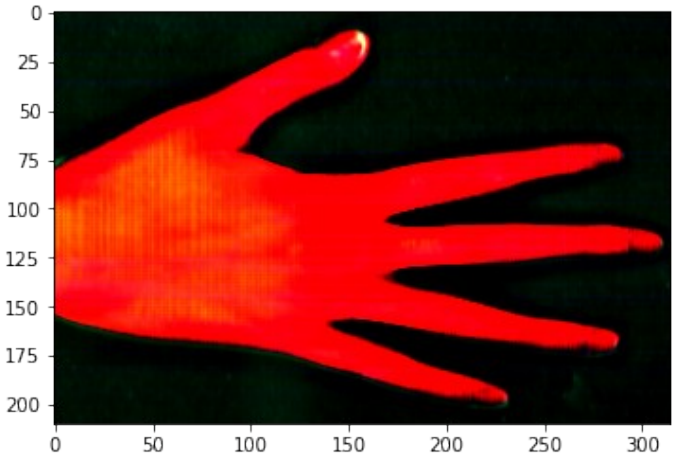
```
plt.title("Eigenvalues for each user")

plt.legend()
plt.show()
```



Apply PCA to the first 3 principal components

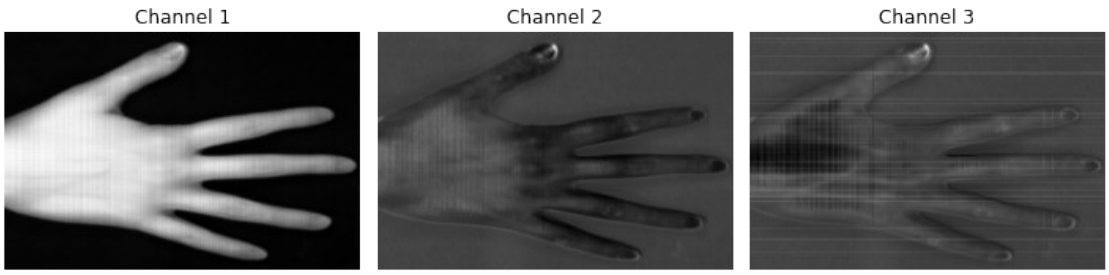
```
# Load data
data_train = pd.read_csv('hands_train.csv')

img_path = "/content/drive/MyDrive/Colab
Notebooks/hyperspectral_hands/users/dorsal/cubo001_1.mat"
img = sio.loadmat(img_path)
img = img['cubo']
img = np.transpose(img, (1, 2, 0))
img = img[45:255, 10:325, :]

# Normalisation
img = (img - np.min(img)) / (np.max(img) - np.min(img))

# Reshape to a 2D array
h, w, b = img.shape
img_reshaped = img.reshape((h*w, b))

# Apply PCA to reduce the dimensionality of the hyperspectral
image
pca = PCA(n_components=3)
```

```
img_pca = pca.fit_transform(img_reshaped)

# Reshape PCA image back to the original shape
img_pca_reshaped = img_pca.reshape((h, w, 3))

plt.imshow(img_pca_reshaped, cmap="gray")
plt.show()
WARNING:matplotlib.image:Clipping input data to the valid
range for imshow with RGB data ([0..1] for floats or [0..255]
for integers).
```



Plot the 3 channels after PCA for the first 3 principal components

```
fig, axs = plt.subplots(1, 3, figsize=(10, 5))
axs = axs.flatten()
for i in range(3):
    axs[i].imshow(img_pca_reshaped[:,:,i], cmap='gray')
    axs[i].set_title(f"Channel {i+1}")
    axs[i].axis('off')

plt.tight_layout()
plt.show()
```



Pre-Processing

Apply PCA to the first 3 principal components

```
def preprocess_image(img_path):

  image = sio.loadmat(img_path)
  image = image['cubo']
  image = np.transpose(image, (2, 1, 0))
  image = image[45:255, 10:325, :]

  # Normalisation
  image = (image - np.min(image)) / (np.max(image) -
np.min(image))

  # Reshape hyperspectral image to a 2D array
  h, w, b = image.shape
  img_reshaped = image.reshape((h*w, b))

  # Apply PCA to reduce the dimensionality of the
hyperspectral image
  pca = PCA(n_components=3)
  img_pca = pca.fit_transform(img_reshaped)

  # Reshape PCA image back to the original shape
  output_img = img_pca.reshape((h, w, 3))

  return output_img
```

Prepare the train and test data

```
# Load datasets
train_ds = pd.read_csv('hands_train.csv')
test_ds = pd.read_csv('hands_test.csv')

# Prepare train data
train_data = []
for i, row in train_ds.iterrows():
    img_path = "/content/drive/MyDrive/Colab
Notebooks/hyperspectral_hands/users/dorsal/" +
row['ImageName']
    output_img = preprocess_image(img_path)
    train_data.append(output_img)

train_data = np.array(train_data)
train_labels = train_ds["ID"].values

# Prepare test data
test_data = []
for i, row in test_ds.iterrows():
    img_path = "/content/drive/MyDrive/Colab
Notebooks/hyperspectral_hands/users/dorsal/" +
row['ImageName']
    output_img = preprocess_image(img_path)
    test_data.append(output_img)
```

```
test_data = np.array(test_data)
test_labels = test_ds["ID"].values
```
Split the train data into training and validation sets

```
train_data, val_data, train_labels, val_labels =
train_test_split(train_data, train_labels, test_size=0.25,
random_state=42)
```
Model architecture

```
# Load the VGG16 model
vgg_model = VGG16(weights='imagenet', include_top=False,
input_shape=(210,310,3))

# Freeze the layers
for layer in vgg_model.layers:
    layer.trainable = False

# Add the fully connected layers
model = Sequential()
model.add(vgg_model)
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(11, activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(lr=0.0001), metrics=['accuracy'])

# Display the model structure
model.summary()
WARNING:absl:`lr` is deprecated, please use `learning_rate`
instead, or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.Adam.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 6, 9, 512)
14714688


 flatten (Flatten)           (None, 27648)             0



 dense (Dense)               (None, 4096)
113250304
```

```
 dense_1 (Dense)             (None, 11)                45067


=================================================================
Total params: 128,010,059
Trainable params: 113,295,371
Non-trainable params: 14,714,688
_________________________________________________________________
```

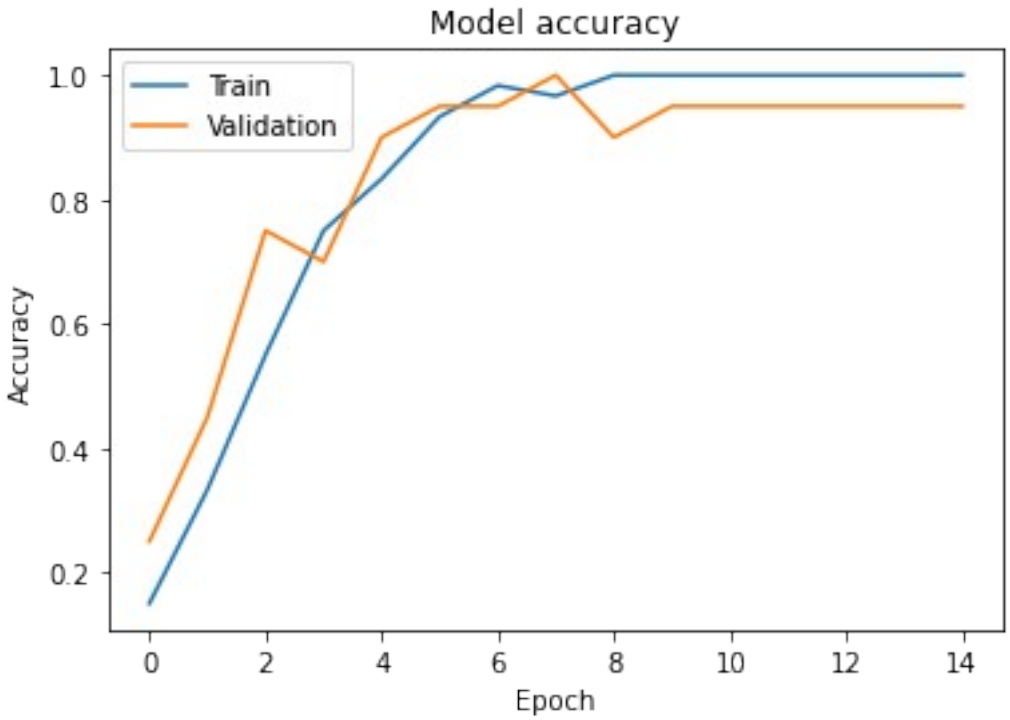
Train model

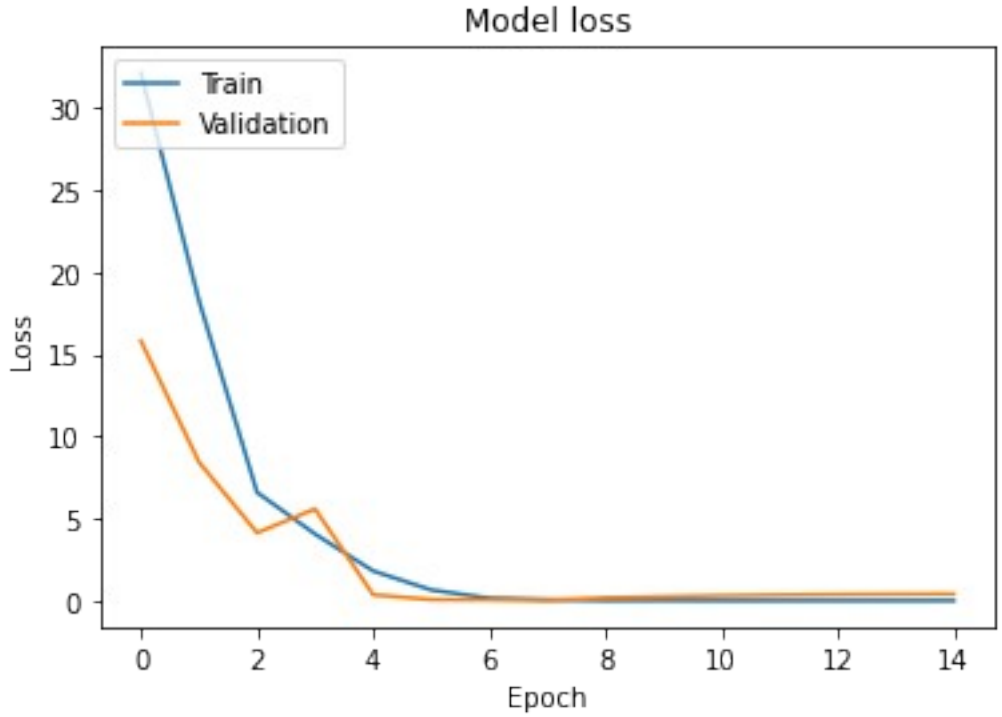
```
history = model.fit(train_data, train_labels, epochs=15,
batch_size = 12, validation_data=(val_data, val_labels))

model.save('trained_model.h5')
Epoch 1/15
5/5 [==============================] - 6s 448ms/step - loss:
32.0867 - accuracy: 0.1500 - val_loss: 15.7915 - val_accuracy:
0.2500
Epoch 2/15
5/5 [==============================] - 1s 131ms/step - loss:
18.2383 - accuracy: 0.3333 - val_loss: 8.4222 - val_accuracy:
0.4500
Epoch 3/15
5/5 [==============================] - 1s 131ms/step - loss:
6.5854 - accuracy: 0.5500 - val_loss: 4.1258 - val_accuracy:
0.7500
Epoch 4/15
5/5 [==============================] - 1s 132ms/step - loss:
4.0495 - accuracy: 0.7500 - val_loss: 5.5775 - val_accuracy:
0.7000
Epoch 5/15
5/5 [==============================] - 1s 132ms/step - loss:
1.8056 - accuracy: 0.8333 - val_loss: 0.3574 - val_accuracy:
0.9000
Epoch 6/15
5/5 [==============================] - 1s 132ms/step - loss:
0.6529 - accuracy: 0.9333 - val_loss: 0.0646 - val_accuracy:
0.9500
Epoch 7/15
5/5 [==============================] - 1s 131ms/step - loss:
0.1517 - accuracy: 0.9833 - val_loss: 0.0498 - val_accuracy:
0.9500
Epoch 8/15
5/5 [==============================] - 1s 133ms/step - loss:
0.0636 - accuracy: 0.9667 - val_loss: 0.0089 - val_accuracy:
1.0000
Epoch 9/15
```

```
5/5 [==============================] - 1s 133ms/step - loss:
0.0013 - accuracy: 1.0000 - val_loss: 0.1753 - val_accuracy:
0.9000
Epoch 10/15
5/5 [==============================] - 1s 131ms/step - loss:
0.0063 - accuracy: 1.0000 - val_loss: 0.2411 - val_accuracy:
0.9500
Epoch 11/15
5/5 [==============================] - 1s 132ms/step - loss:
5.0687e-04 - accuracy: 1.0000 - val_loss: 0.2989 -
val_accuracy: 0.9500
Epoch 12/15
5/5 [==============================] - 1s 132ms/step - loss:
3.7220e-04 - accuracy: 1.0000 - val_loss: 0.3436 -
val_accuracy: 0.9500
Epoch 13/15
5/5 [==============================] - 1s 131ms/step - loss:
2.9462e-04 - accuracy: 1.0000 - val_loss: 0.3729 -
val_accuracy: 0.9500
Epoch 14/15
5/5 [==============================] - 1s 133ms/step - loss:
1.7849e-04 - accuracy: 1.0000 - val_loss: 0.3923 -
val_accuracy: 0.9500
Epoch 15/15
5/5 [==============================] - 1s 132ms/step - loss:
1.3091e-04 - accuracy: 1.0000 - val_loss: 0.4034 -
val_accuracy: 0.9500
```

Plot learning curves

```
# Plot accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```





Evaluation Metrics

```
# Predict the labels for test data
predictions = model.predict(test_data)
predicted_labels = np.argmax(predictions, axis=1)
```

```python
# Calculate loss
test_loss, test_accu = model.evaluate(test_data, test_labels)
print('Accuracy:', test_accu)
print('Loss:', test_loss)

# Calculate precision, recall, and F1-score for each class
from sklearn.metrics import classification_report
target_names = [f"ID_{i}" for i in range(10)]
report = classification_report(test_labels, predicted_labels,
target_names=target_names)
print(report)
```

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 152ms/step - loss:
0.8552 - accuracy: 0.8500
Accuracy: 0.8500000238418579
Loss: 0.8551972508430481
              precision    recall  f1-score   support

        ID_0       0.00      0.00      0.00         2
        ID_1       1.00      0.50      0.67         2
        ID_2       0.67      1.00      0.80         2
        ID_3       1.00      1.00      1.00         2
        ID_4       1.00      1.00      1.00         2
        ID_5       1.00      1.00      1.00         2
        ID_6       1.00      1.00      1.00         2
        ID_7       0.67      1.00      0.80         2
        ID_8       0.67      1.00      0.80         2
        ID_9       1.00      1.00      1.00         2

    accuracy                           0.85        20
   macro avg       0.80      0.85      0.81        20
weighted avg       0.80      0.85      0.81        20

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-
score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-
score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```