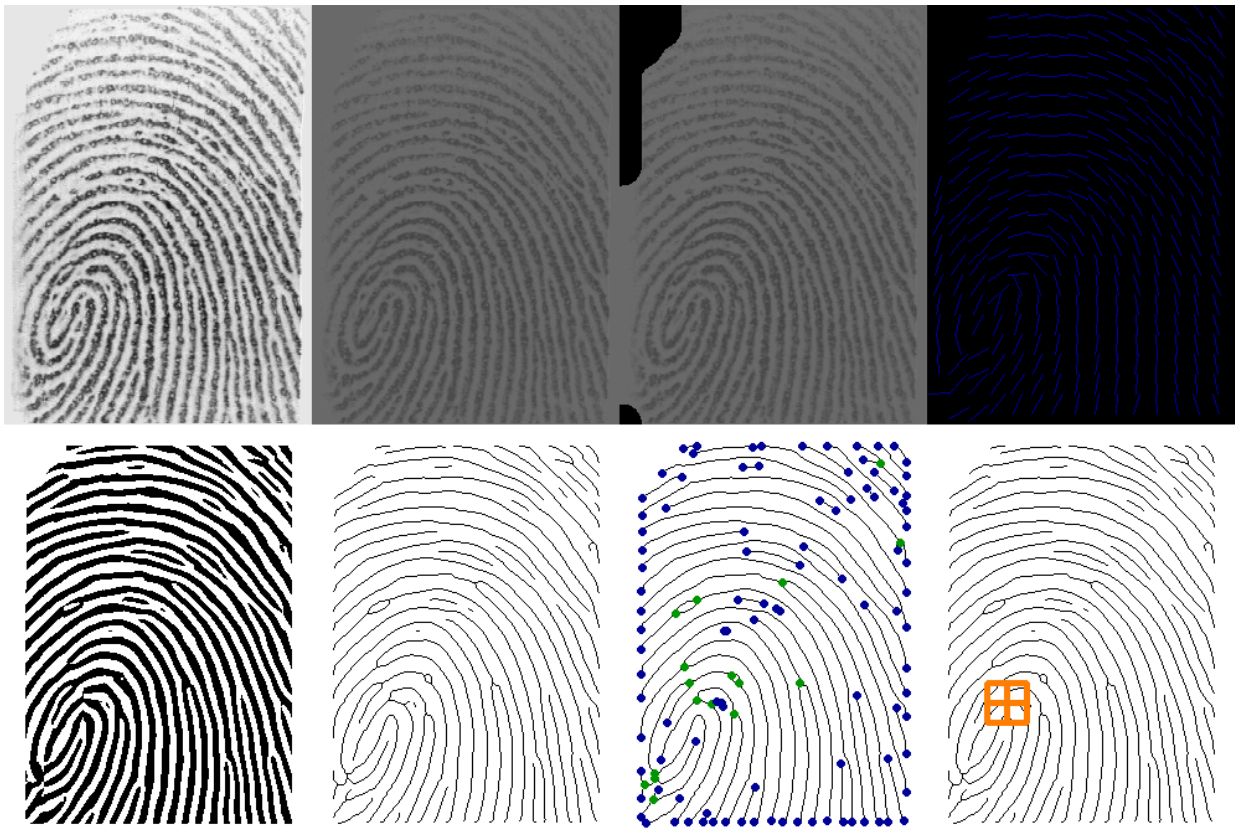
FINGERPRINT IDENTIFICATION



STUDENT: Sindi Ziu

EPOKA UNIVERSITY
DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

CONTENTS

1. INTRODUCTION

One of the most important components of contemporary security and identity systems is fingerprint recognition. The precision and effectiveness of fingerprint recognition are essential for many uses, including forensic investigations and smartphone unlocking. This report details our attempt to use Python to create an identification program from scratch.

Our main goal is to use various machine learning methods to build a reliable fingerprint identification system. We will investigate many algorithms along the process, including Convolutional Neural Networks (CNN), Multi-Layer Perceptrons (MLP), k-Nearest Neighbors (k-NN), and Support Vector Machines (SVM). We want to assess these algorithms' efficiency and precision in fingerprint recognition by putting them through a rigorous testing process with different parameters.

A significant portion of the process was focused on image preprocessing and minutiae extraction, in addition to testing and implementing the methods. The quality of fingerprint images must be optimized using preprocessing methods including image enhancement, noise reduction, and normalization before being sent into the recognition system. Minutiae extraction is the process of locating and obtaining important characteristics, such as bifurcations and ridge ends, that are essential for fingerprint comparison and matching.

Through this project, we seek to gain insights into the underlying mechanisms of fingerprint identification while also striving to enhance the performance of our system. By documenting our journey and findings, we hope to contribute to the advancement of fingerprint recognition technology and its practical applications.

2. METHODOLOGY

The goal of this project is to achieve fingerprint identification by applying a variety of supervised learning techniques to a dataset of fingerprint images. There are three main steps to the project:

1) Image preprocessing, which includes using different normalization techniques and feature extraction, minutiae detection and extraction.
2) Training every model with distinct structures or parameters (e.g., experimenting with MLP with varying numbers of nodes in the hidden layer).
3) Evaluating the model and computing accuracy metrics to determine which model performs better and under what circumstances.

Dataset

The dataset used contains images of different fingerprints. There are 16 unique fingers for each of which we have 8 different images; 8 fingerprint captures of the same finger. So in total there are 128 images. Their dimensions are 248px width and 338px height. There is a naming convention where first we have the **"label"** (ex. "10") of the images, signifying which finger it is, followed by an **"_"** and then the specific fingerprint capture number, from 1 to 8. Below we have a few of them visualized accompanied by the file names (labels).

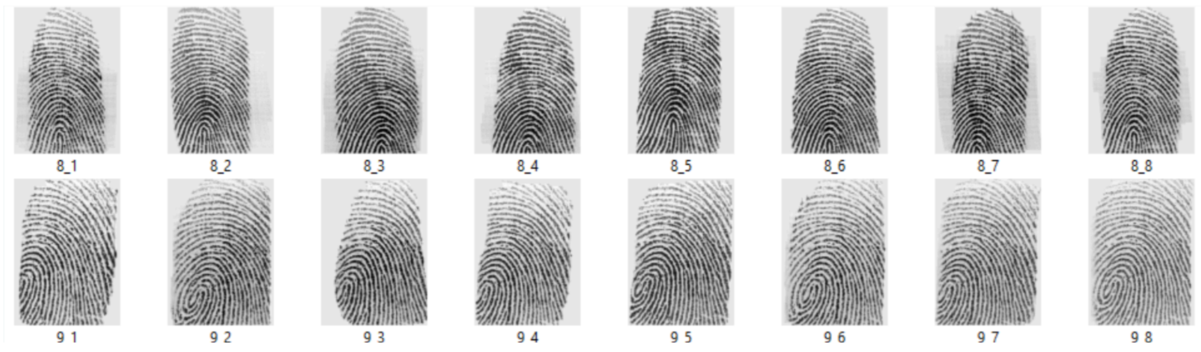


Fig 1

3. STATE OF THE ART

Methodology	Author	Accuracy
SVM	Xuan Xu	90.24%
SVM	S. Adebayo Daramola	80%
CNN	Bhavesh Pandya	98.21%
SVM	Youssef Elmir	68.4%
MLP	Terje Kristensen	88.8%

Most of the papers regarding the subject are on Convolutional Neural Networks and SVM, although not many papers actually explain and implement each step of the way towards implementation. Some only focus on feature extraction, a crucial process which will determine the rest of the performance; others deal with very advanced mathematical concepts which are hard to grasp and digest as a student. So we have taken some inspiration from these authors, but have tried to still keep it authentic and true to our subject.

4. DATA PREPROCESSING

Fingerprint recognition heavily relies on the quality of the input images. Therefore, preprocessing the fingerprint images is a crucial step to ensure optimal performance of the subsequent identification algorithms. In this section, we outline the detailed preprocessing steps we have explored to enhance the quality of fingerprint images for our identification program. While continuously experimenting, we went through a few different pipelines until we were finally successful.

First pipeline:

- Resizing Images (Normalization of Image Dimensions): Standardizing the dimensions of all fingerprint images is essential to ensure uniformity, facilitating the input for machine learning models.
- Applying CLAHE (Contrast Limited Adaptive Histogram Equalization): CLAHE enhances the local contrast of an image, particularly beneficial for improving the clarity of fingerprint images where ridge-valley contrast is crucial. We added this to try to balance the contrast of the grayscale between the images and also recover any info that could potentially be lost because of a lower contrast on some images.
- Noise Reduction: Fingerprint images often contain various types of noise, which can be mitigated using techniques like Gaussian Blur to smooth out the images.
- Edge Enhancement: Emphasizing the boundaries of ridges can aid in fingerprint analysis. The Canny edge detector is a popular choice for this purpose.
- Binary Thresholding: Converting grayscale images to binary images highlights ridges against valleys distinctly, facilitating feature extraction algorithms in fingerprint analysis.
- Morphological Operations: These operations further clean up the images, closing small holes or connecting breaks in ridges for better analysis.

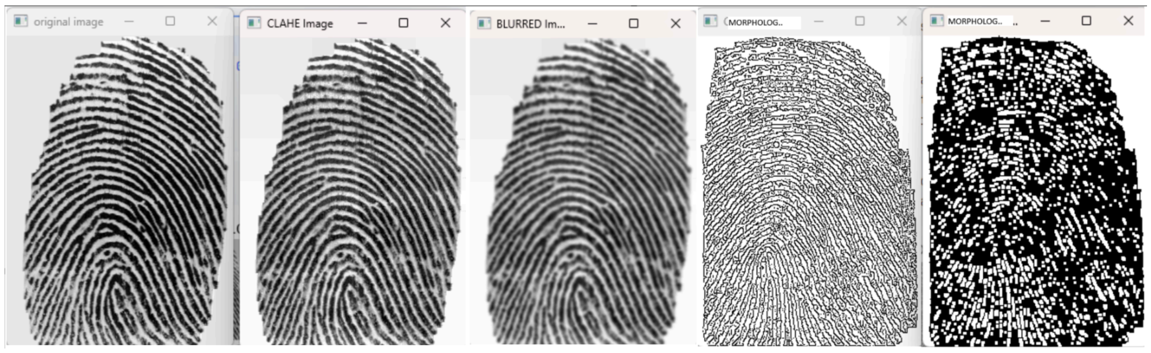
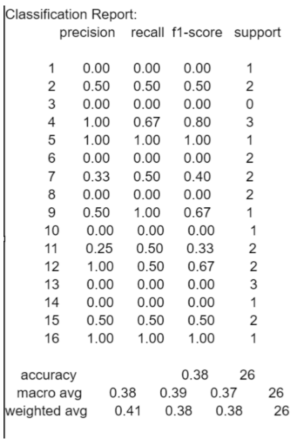


Fig 2

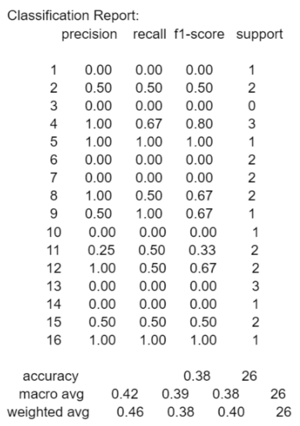
We tested directly with the CLAHE image as after blur the details only worsened, and the image after our morphological analysis seems not readable. We tried the SVM algorithm with this pipeline and the results were not satisfactory at all, the accuracy when using the processed data was almost identical with

the raw data. Below is the info for each classification, reprocessed to the left and original right.
As a second approach we tried to go for something more simple, crafting it from the bottom up. The pipeline goes like this:

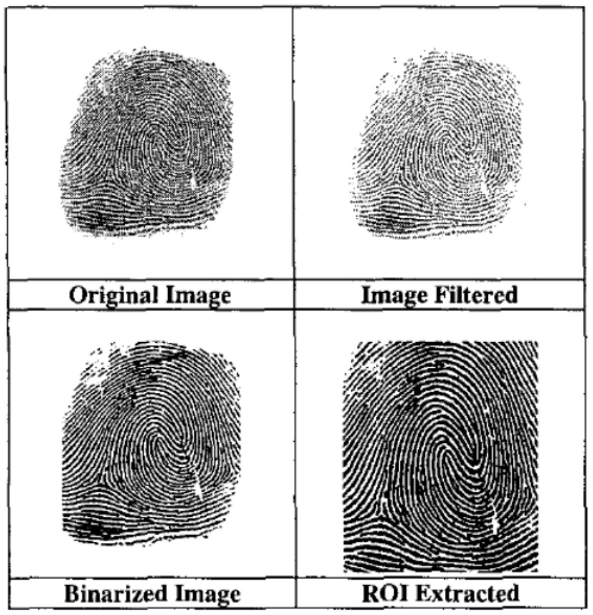
Second pipeline:
1. CLAHE application →
2. Gaussian blur (3x3 size (lowest)) →
3. Binarize through Otsu's method →
4. Thinning to enhance ridges→
5. Region of Interest ROI



```
Classification Report:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.50      0.50      0.50         2
           3       0.00      0.00      0.00         0
           4       1.00      0.67      0.80         3
           5       1.00      1.00      1.00         1
           6       0.00      0.00      0.00         2
           7       0.33      0.50      0.40         2
           8       0.00      0.00      0.00         2
           9       0.50      1.00      0.67         1
          10       0.00      0.00      0.00         1
          11       0.25      0.50      0.33         2
          12       1.00      0.50      0.67         2
          13       0.00      0.00      0.00         3
          14       0.00      0.00      0.00         1
          15       0.50      0.50      0.50         2
          16       1.00      1.00      1.00         1

    accuracy                           0.38        26
   macro avg       0.38      0.39      0.37        26
weighted avg       0.41      0.38      0.38        26
```

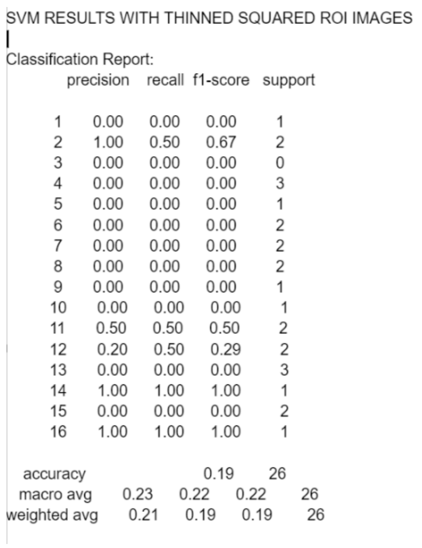


```
Classification Report:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.50      0.50      0.50         2
           3       0.00      0.00      0.00         0
           4       1.00      0.67      0.80         3
           5       1.00      1.00      1.00         1
           6       0.00      0.00      0.00         2
           7       0.00      0.00      0.00         2
           8       1.00      0.50      0.67         2
           9       0.50      1.00      0.67         1
          10       0.00      0.00      0.00         1
          11       0.25      0.50      0.33         2
          12       1.00      0.50      0.67         2
          13       0.00      0.00      0.00         3
          14       0.00      0.00      0.00         1
          15       0.50      0.50      0.50         2
          16       1.00      1.00      1.00         1

    accuracy                           0.38        26
   macro avg       0.42      0.39      0.38        26
weighted avg       0.46      0.38      0.40        26
```

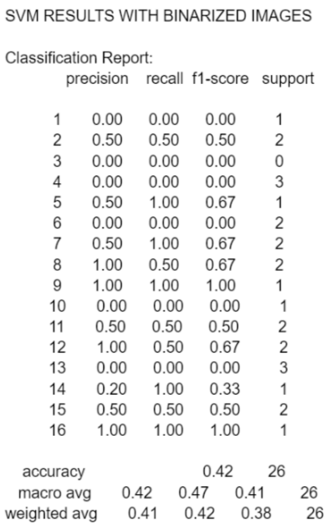


Unfortunately as we see this last delivered very poor results, with the ROI version of SVM having an accuracy of 0.19. Although it is surprising to see how much better the binarized images perform, and even better the raw images with no preprocessing.

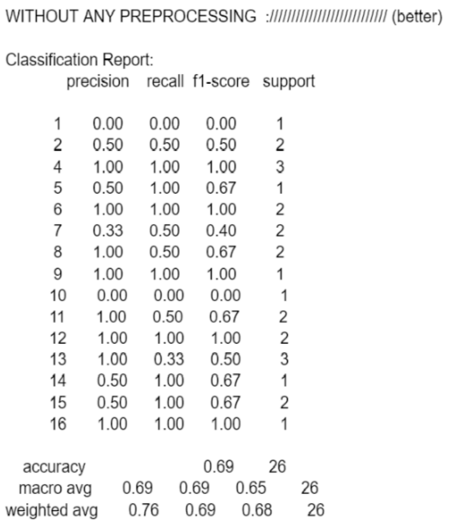
Model SVM *Fig 3*



```
SVM RESULTS WITH THINNED SQUARED ROI IMAGES
|
Classification Report:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       1.00      0.50      0.67         2
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         3
           5       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         2
           7       0.00      0.00      0.00         2
           8       0.00      0.00      0.00         2
           9       0.00      0.00      0.00         1
          10       0.00      0.00      0.00         1
          11       0.50      0.50      0.50         2
          12       0.20      0.50      0.29         2
          13       0.00      0.00      0.00         3
          14       1.00      1.00      1.00         1
          15       0.00      0.00      0.00         2
          16       1.00      1.00      1.00         1

    accuracy                           0.19        26
   macro avg       0.23      0.22      0.22        26
weighted avg       0.21      0.19      0.19        26
```



```
SVM RESULTS WITH BINARIZED IMAGES

Classification Report:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.50      0.50      0.50         2
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         3
           5       0.50      1.00      0.67         1
           6       0.00      0.00      0.00         2
           7       0.50      1.00      0.67         2
           8       1.00      0.50      0.67         2
           9       1.00      1.00      1.00         1
          10       0.00      0.00      0.00         1
          11       0.50      0.50      0.50         2
          12       1.00      0.50      0.67         2
          13       0.00      0.00      0.00         3
          14       0.20      1.00      0.33         1
          15       0.50      0.50      0.50         2
          16       1.00      1.00      1.00         1

    accuracy                           0.42        26
   macro avg       0.42      0.47      0.41        26
weighted avg       0.41      0.42      0.38        26
```



```
WITHOUT ANY PREPROCESSING  ://///////////////////////// (better)

Classification Report:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.50      0.50      0.50         2
           4       1.00      1.00      1.00         3
           5       0.50      1.00      0.67         1
           6       1.00      1.00      1.00         2
           7       0.33      0.50      0.40         2
           8       1.00      0.50      0.67         2
           9       1.00      1.00      1.00         1
          10       0.00      0.00      0.00         1
          11       1.00      0.50      0.67         2
          12       1.00      1.00      1.00         2
          13       1.00      0.33      0.50         3
          14       0.50      1.00      0.67         1
          15       0.50      1.00      0.67         2
          16       1.00      1.00      1.00         1

    accuracy                           0.69        26
   macro avg       0.69      0.69      0.65        26
weighted avg       0.76      0.69      0.68        26
```

After looking through some previous successful works of other engineers we decided to use the work of Manuel Cuevas [4] for our full feature extraction. Minutiae extraction is a critical component of fingerprint recognition. Here are the stages of the preprocessing pipeline:

Third (and final) pipeline:

1. Grayscale Transformation → 2. Image Normalization → 3. Segmentation → 4. Directional Map → 5. Frequency Map → 6. Gabor's Filtering → 7. Skeletonization → 8. Minutiae Detection → 9. Singularity Detection

Grayscale Transformation - This step makes it possible to optimize on the general appearance of the image and facilitates biometric processing.
Image Normalization - The main goal of normalization is to reduce the variance of the gray level value along the ridges to facilitate subsequent processing steps
Segmentation - Segmentation is necessary in order to eliminate the edges of the image and areas that are too noisy. This step makes it possible to reduce the size of the useful part of the image and subsequently optimize the minutiae extraction phase.
Directional map - The directional map defines the local orientation of the striates in the fingerprint impression.
Frequency Map - In addition to the directional map, we must have the local estimation of the frequency map to be able to construct the Gabor's filter. The frequency map is an image of the same size as the fingerprint image and represents the local frequency of the streaks
Gabor Filter - Gabor filters have both frequency-selective and orientation-selective properties and have an optimal joint resolution in both spatial and frequency domains. The resulting image will be the spatial convolution of the original normalized image and one of the base filters in the direction and local frequency from the two-directional and frequency maps
Skeletonization - The process involves using morphological erosion operations to remove redundant pixels from the ridges, leaving only one-pixel wide ridges. The resulting skeleton can be used to extract the minutiae points of the fingerprint for identification purposes.
Minutiae Detection - There are two types of minutiae points in a fingerprint: ridge endings and bifurcations. In this project, we use cross number method for detecting minutiae points. The method involves analyzing 3 x 3 pixel blocks and calculating the crossing number.
Singularity Detection - The Poincaré index method is used to detect singularities in fingerprints. The method involves computing the orientation field (G) and position ([i,j]) of elements in the fingerprint.

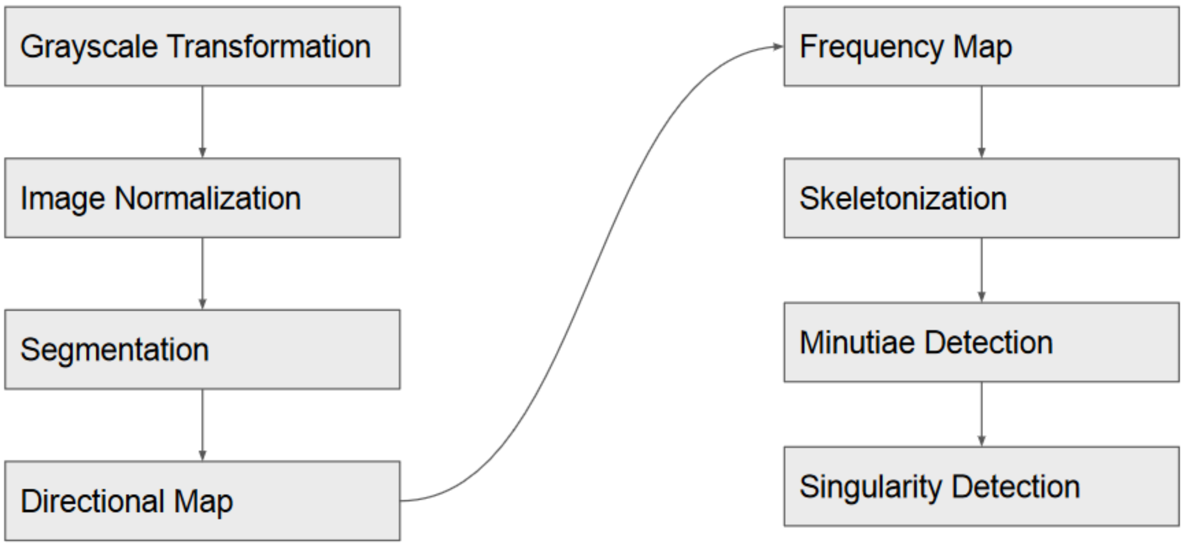


Fig 4. Final Preprocessing Pipeline

To implement this algorithm we have turned to python as it offers all the commodities needed. Through each step an image is generated on top of the original image, showcasing the changes each step has made, then in the end they are all merged together to show us the full progress in Fig 5. As we can see here the images are properly processed in order to turn out as clear as possible while still holding all the necessary information for identification. These points of information are seen in the minutia part, where all the endings and bifurcations are noted with either a blue or green color on the image. In the background in our code i also attempted to save all the coordinates and types of minutiae, in order to convert them into a big features vector to input into the models that we will build, together with the singularities information. Although after many attempts, I couldn't figure out a way of representing the data gracefully and kept on hitting dead ends. So I resorted into using the minutiae images directly and inputing them as training and testing data for our algorithms. After going through all the preprocessing steps I saved the "minutiae" images into a separate folder for direct access for all the algorithms we would implement.

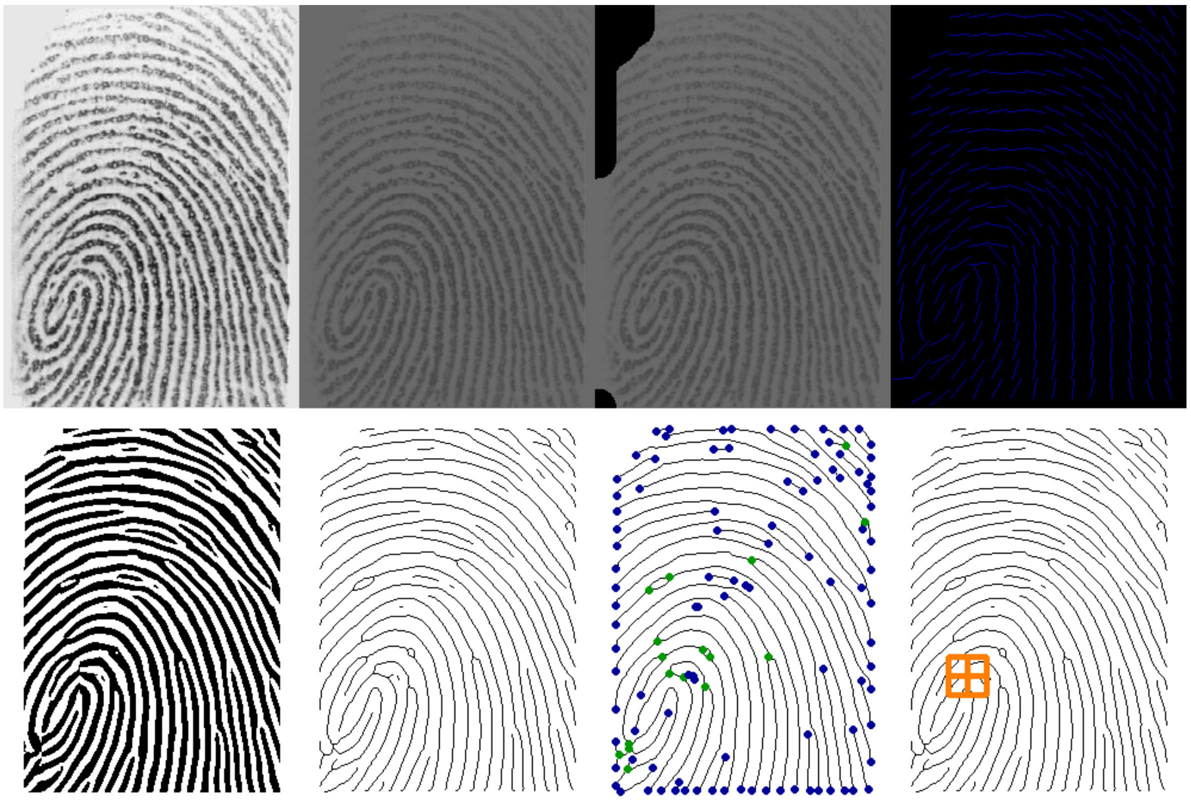


Fig 5. The result of the final preprocessing applied on our images:

5. SVM

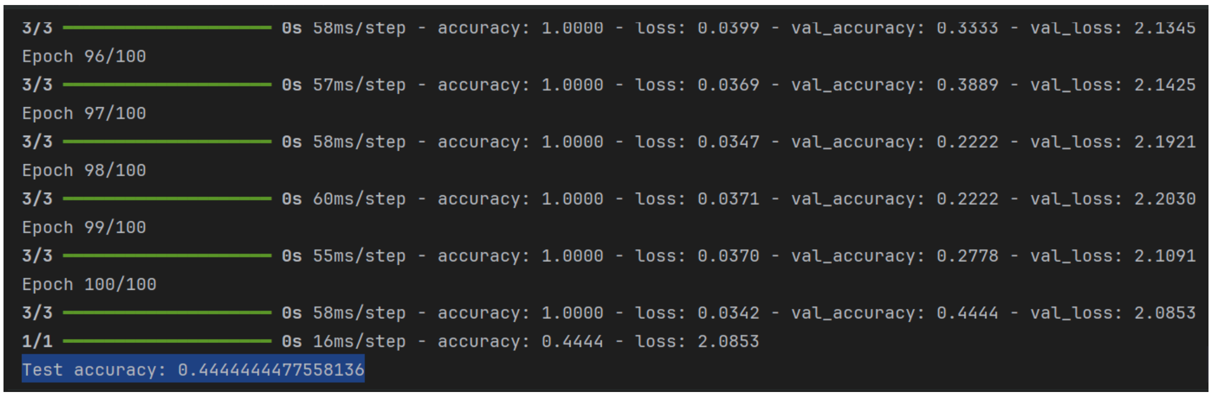
The Support vector machine model has two parameters that can be switched in order to achieve the highest accuracy: the kernel and the degree (if the kernel is polynomial). The kernels chosen were: polynomial to the third degree and linear. Results are as follow:

	Accuracy	**Precision**	**Recall**	**F1 Score**
First preprocessing, linear kernel	0.38	0.39	0.37	0.38
First preprocessing, poly kernel	0.0	0.0	0.0	0.0
Second preprocessing, ROI images	0.19	0.22	0.19	0.19
Second preprocessing, binarized images	0.42	0.47	0.41	0.38
Original Images, linear kernel	0.69	0.76	0.68	0.65
Third preprocessing, Minutiae images	0.44	0.54	0.44	0.45
Third preprocessing, Original images	0.55	0.77	0.55	0.59

The peak accuracy of this model was 0.69 achieved when using the linear kernel on the original images. This leads us to believe that the chosen preprocessing was not ideal for this algorithm, further proven when we see the second highest accuracy, once again reached with the original images Runner ups are the values from the minutiae images with the third preprocessing, followed by the binarized images using the second preprocessing. The poly kernel is completely non valid for this case as it gave a 0.0 accuracy.

6. MLP

When implementing a Multilayered Perceptron Neural Networks, there are a few parameters we can adjust that can influence our results. That is the number of nodes in the hidden layer as well as the activation (transfer) function used, in our case **"ReLu"**. Activation function is "softMax". We have trained the MLP model on both the Minutiae images as well as the original images, for a different number of hidden layer nodes and took note of their results. We have run the algorithm with 100 epochs due to limited computational power.



```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 1.0000 - loss: 0.0399 - val_accuracy: 0.3333 - val_loss: 2.1345
Epoch 96/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 1.0000 - loss: 0.0369 - val_accuracy: 0.3889 - val_loss: 2.1425
Epoch 97/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 1.0000 - loss: 0.0347 - val_accuracy: 0.2222 - val_loss: 2.1921
Epoch 98/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 1.0000 - loss: 0.0371 - val_accuracy: 0.2222 - val_loss: 2.2030
Epoch 99/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 1.0000 - loss: 0.0370 - val_accuracy: 0.2778 - val_loss: 2.1091
Epoch 100/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 1.0000 - loss: 0.0342 - val_accuracy: 0.4444 - val_loss: 2.0853
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.4444 - loss: 2.0853
Test accuracy: 0.4444444477558136
```

	128 nodes in hidden layer	**256 nodes**	**512 nodes**
Original Images	0.11	0.28	0.39
Minutiae Images	0.22	0.33	0.28

*figured out mistake i was making when making the train test split, I didn't shuffle the data beforehand, to ensure the possible inclusion of classes in a balanced way, instead I was leaving one class almost not represented at all. Tested with this change the 128 node architecture and got an accuracy of 0.44 , with the minutiae data. Meanwhile on the original data we got an accuracy of 0.39 for 128 nodes and 0.5 for 256 nodes.

7. K-NEAREST NEIGHBORS (KNN)

For this next algorithm, the only parameter that we can try adjusting is the number of neighbors (k).

	3-neighbors	5-neighbors	7-neighbors	9-neighbors
Minutiae Images	0.28	0.33	0.33	0.38
Original Images	0.17	0.17	0.11	0.05

The accuracy is still pretty low, but we can see that finally here the processed images perform better than the original images, peaking at the case of k=9 neighbors.

8. RANDOM FOREST

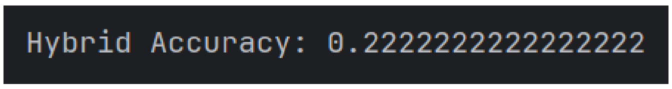
Tried playing around with the “number of estimators” and got the best result (which is very bad) when using 50 and 200 as the number.

	Nr. of Estimators	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Minutiae Images	**100**	0.17	0.34	0.17	0.18
	200	0.22	0.31	0.22	0.21
	250	0.28	0.48	0.28	0.29
Original Images	**100**	0.44	0.32	0.44	0.34
	200	0.5	0.46	0.5	0.43
	250	0.5	0.46	0.5	0.43

We see that the highest accuracy for this algorithm is once again when using the original images.

9. HYBRID: KNN and Random Forest

In this case we tried to combine the results of our KNN and Random Forest models, into a hybrid model. In order to do this we have combined the predictions of both using majority voting and then continue to choose the most common label as the final. We kept the versions of our base models with the highest accuracy to combine together; the KNN with value of k=9, using the minutiae images and Random Forest with nr of estimators = 200, using the original images. Unfortunately this performed worse than both our previous algorithm with an accuracy of :

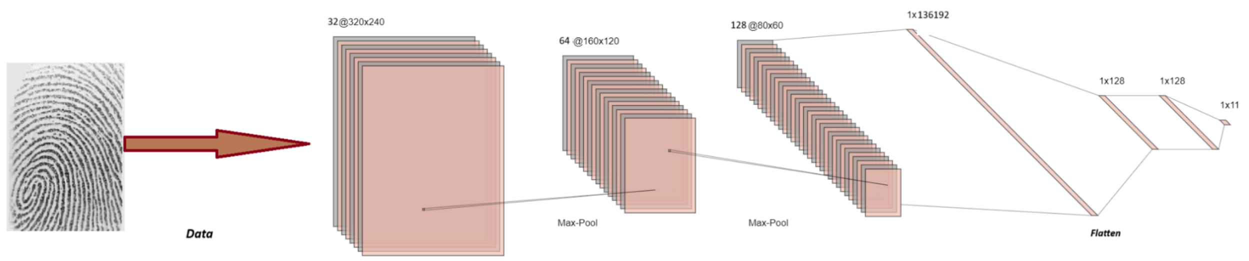


```
Hybrid Accuracy: 0.2222222222222222
```

10. Convolutional Neural Network (CNN)

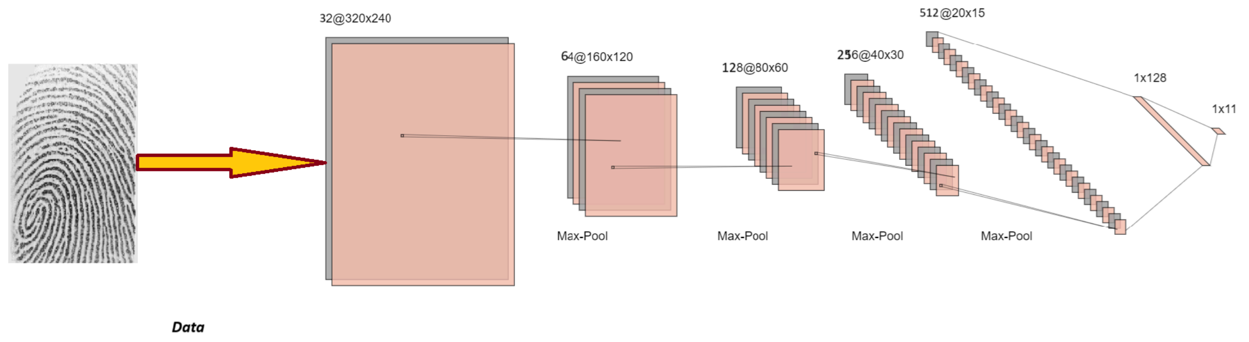
Finally we had to implement a Convolutional Neural Network, the model which is most suitable for image classification. First I made an implementation with 3 convolutional layers, each of them with 32, 64 & 128 nodes respectively as well as two dense hidden layers with 128 nodes each. Windowing size is 3x3. I have used the ReLu activation function and on the last classification layer we have a "softmax" activation function. Adam optimizer.

First CNN Architecture (Fig. 6)



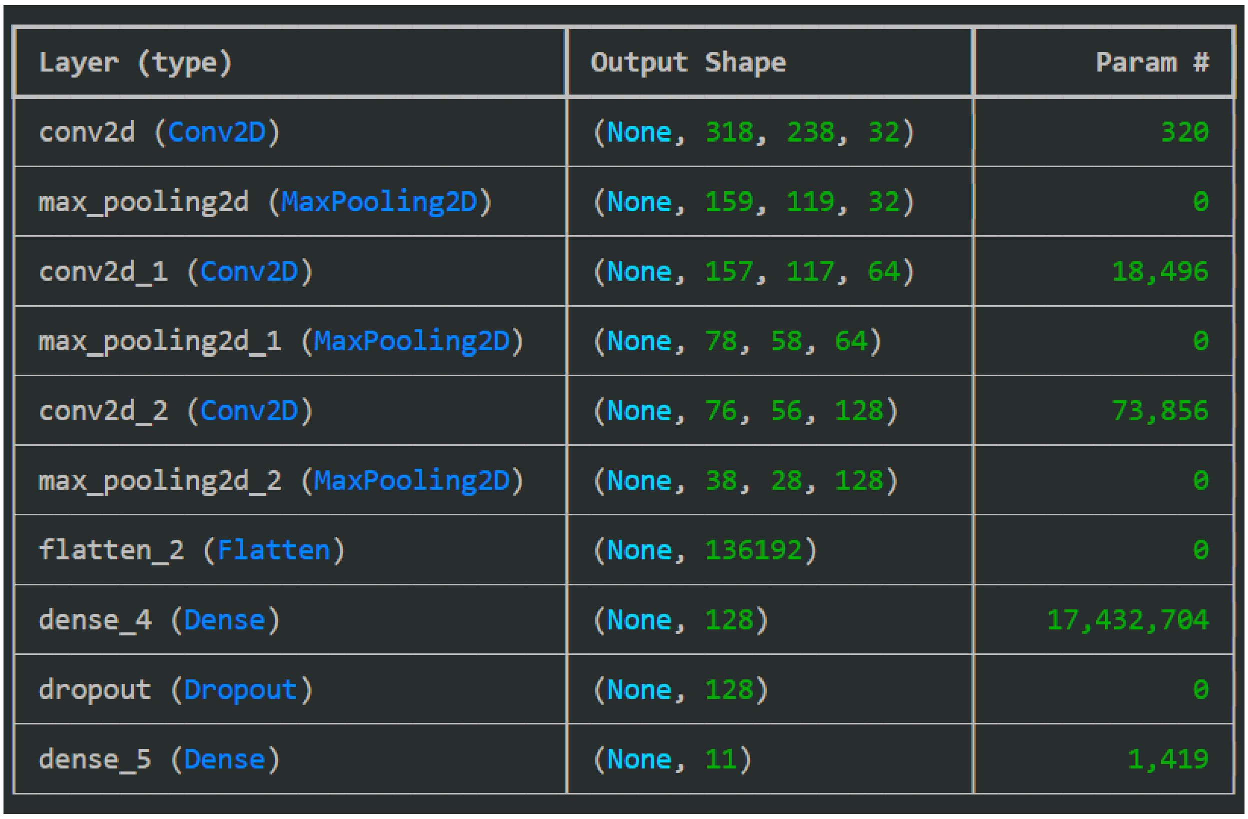
On the second implementation I increased the number of convolutional layers adding 2 more, with 256 and 512 nodes. The results this time looked very promising, trained with 100 epochs.

Second CNN Architecture (on Fig. 7)

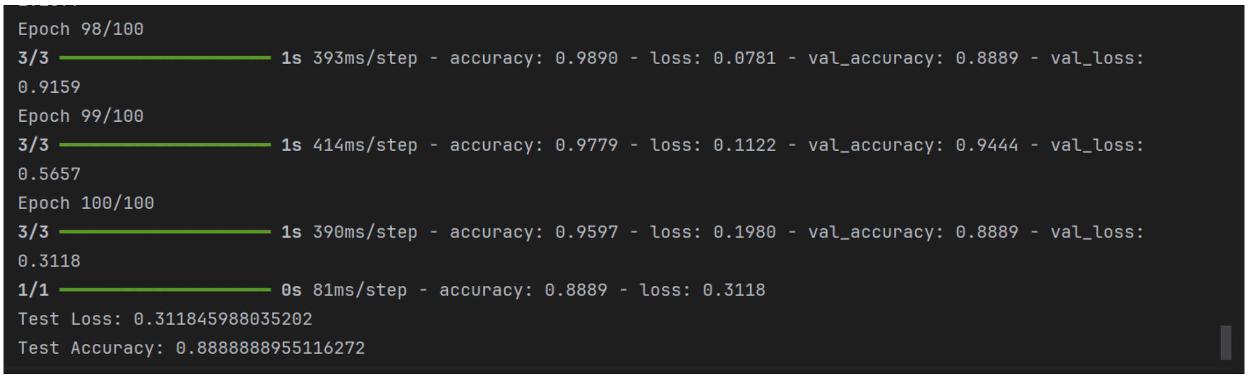


Using the Minutiae Images

First CNN architecture, Fig.6



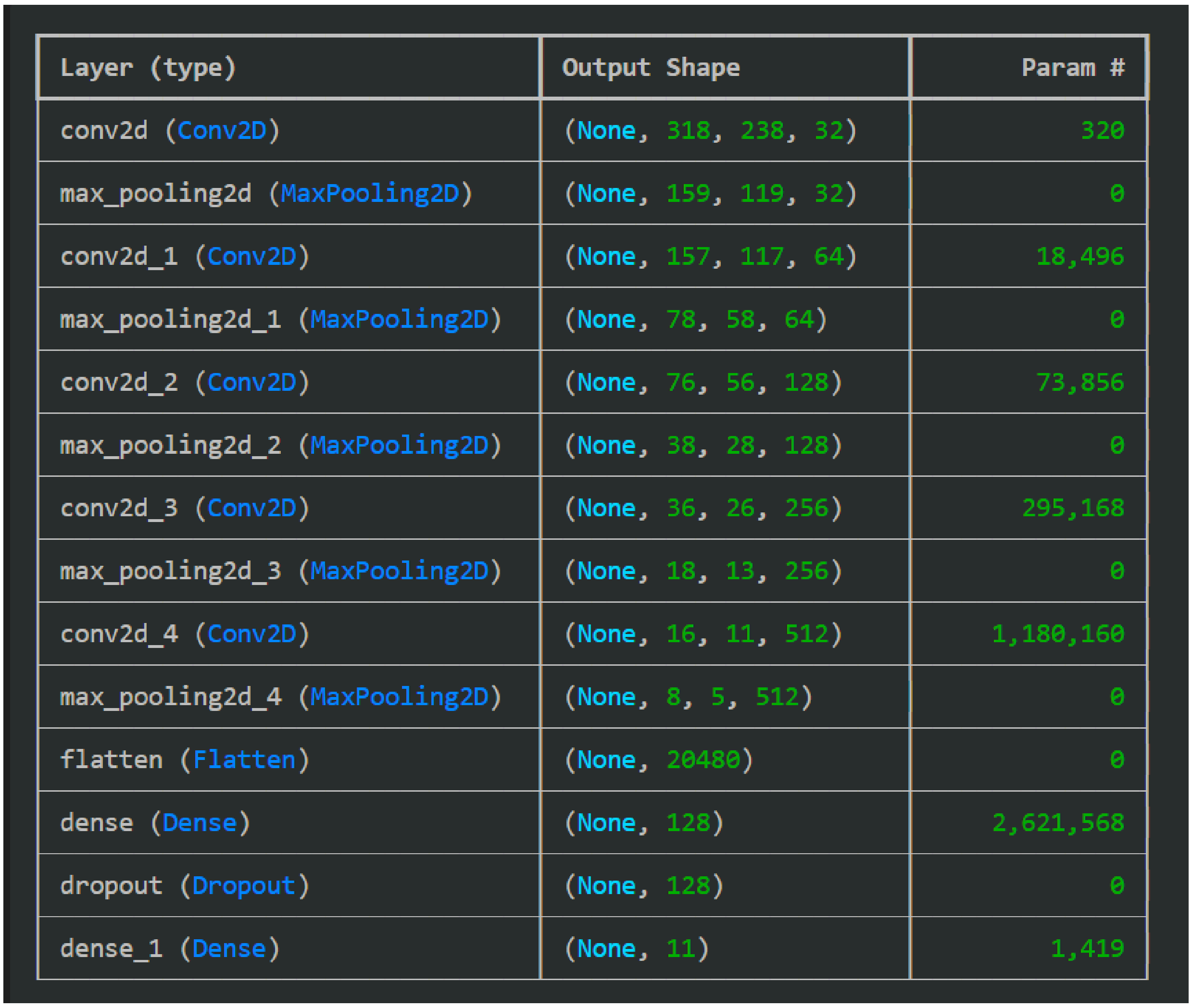
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 318, 238, 32)	320
max_pooling2d (MaxPooling2D)	(None, 159, 119, 32)	0
conv2d_1 (Conv2D)	(None, 157, 117, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 78, 58, 64)	0
conv2d_2 (Conv2D)	(None, 76, 56, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 38, 28, 128)	0
flatten_2 (Flatten)	(None, 136192)	0
dense_4 (Dense)	(None, 128)	17,432,704
dropout (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 11)	1,419



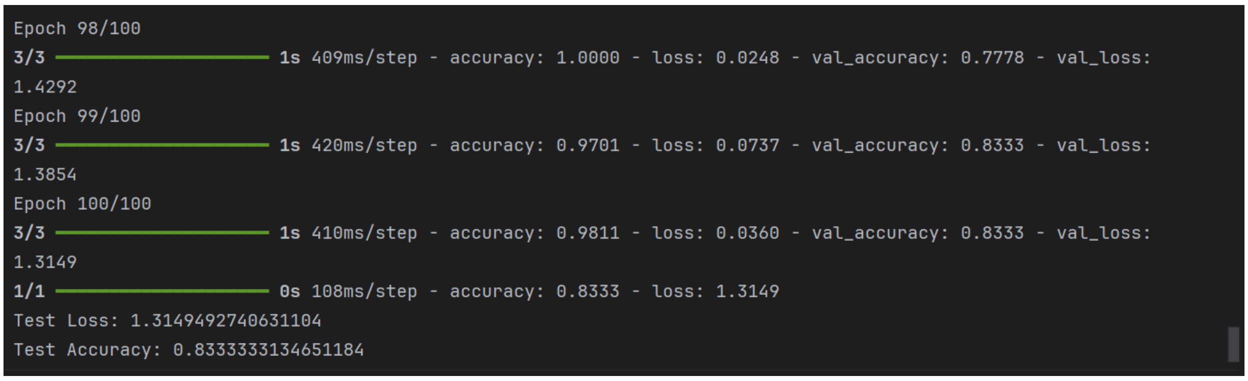
```
Epoch 98/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 393ms/step - accuracy: 0.9890 - loss: 0.0781 - val_accuracy: 0.8889 - val_loss:
0.9159
Epoch 99/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 414ms/step - accuracy: 0.9779 - loss: 0.1122 - val_accuracy: 0.9444 - val_loss:
0.5657
Epoch 100/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 390ms/step - accuracy: 0.9597 - loss: 0.1980 - val_accuracy: 0.8889 - val_loss:
0.3118
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.8889 - loss: 0.3118
Test Loss: 0.311845988035202
Test Accuracy: 0.8888888955116272
```

Test Accuracy: 0.89 (best)

Second CNN architecture, Fig.7



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 318, 238, 32)	320
max_pooling2d (MaxPooling2D)	(None, 159, 119, 32)	0
conv2d_1 (Conv2D)	(None, 157, 117, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 78, 58, 64)	0
conv2d_2 (Conv2D)	(None, 76, 56, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 38, 28, 128)	0
conv2d_3 (Conv2D)	(None, 36, 26, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 18, 13, 256)	0
conv2d_4 (Conv2D)	(None, 16, 11, 512)	1,180,160
max_pooling2d_4 (MaxPooling2D)	(None, 8, 5, 512)	0
flatten (Flatten)	(None, 20480)	0
dense (Dense)	(None, 128)	2,621,568
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 11)	1,419

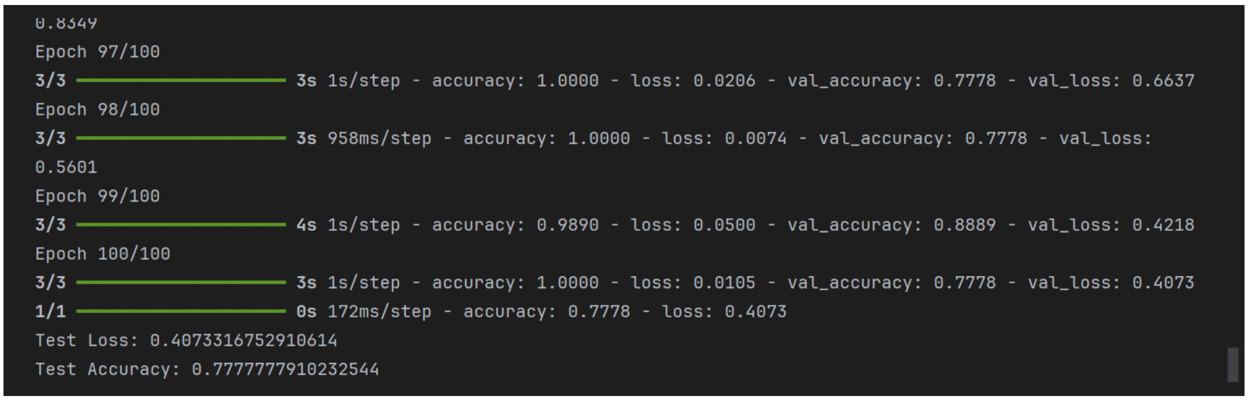


```
Epoch 98/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 409ms/step - accuracy: 1.0000 - loss: 0.0248 - val_accuracy: 0.7778 - val_loss:
1.4292
Epoch 99/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 420ms/step - accuracy: 0.9701 - loss: 0.0737 - val_accuracy: 0.8333 - val_loss:
1.3854
Epoch 100/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 410ms/step - accuracy: 0.9811 - loss: 0.0360 - val_accuracy: 0.8333 - val_loss:
1.3149
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.8333 - loss: 1.3149
Test Loss: 1.3149492740631104
Test Accuracy: 0.8333333134651184
```

Test Accuracy: 0.83

Using the Original Images

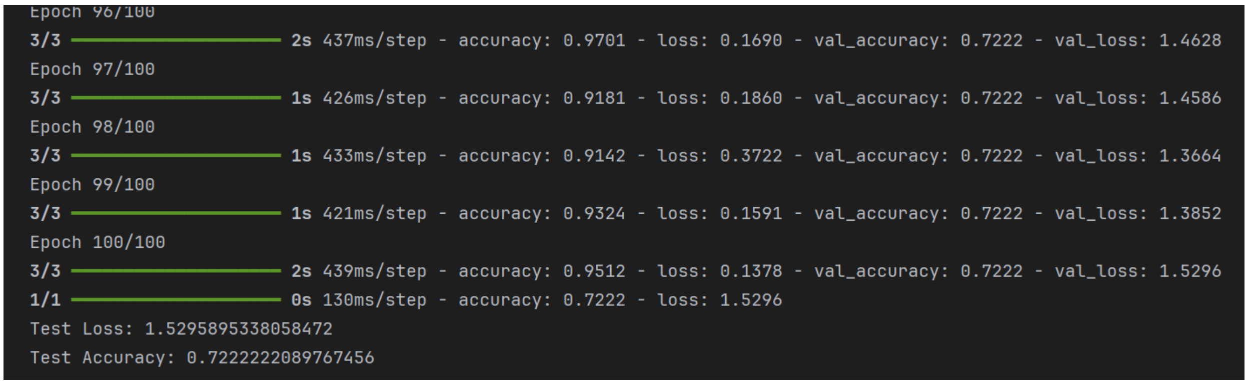
First CNN architecture, Fig.6



```
0.8349
Epoch 97/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step - accuracy: 1.0000 - loss: 0.0206 - val_accuracy: 0.7778 - val_loss: 0.6637
Epoch 98/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 958ms/step - accuracy: 1.0000 - loss: 0.0074 - val_accuracy: 0.7778 - val_loss:
0.5601
Epoch 99/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.9890 - loss: 0.0500 - val_accuracy: 0.8889 - val_loss: 0.4218
Epoch 100/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step - accuracy: 1.0000 - loss: 0.0105 - val_accuracy: 0.7778 - val_loss: 0.4073
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 172ms/step - accuracy: 0.7778 - loss: 0.4073
Test Loss: 0.4073316752910614
Test Accuracy: 0.7777777910232544
```

Test Accuracy: 0.7777777910232544

Second CNN architecture, Fig.7



```
Epoch 96/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 437ms/step - accuracy: 0.9701 - loss: 0.1690 - val_accuracy: 0.7222 - val_loss: 1.4628
Epoch 97/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 426ms/step - accuracy: 0.9181 - loss: 0.1860 - val_accuracy: 0.7222 - val_loss: 1.4586
Epoch 98/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 433ms/step - accuracy: 0.9142 - loss: 0.3722 - val_accuracy: 0.7222 - val_loss: 1.3664
Epoch 99/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 421ms/step - accuracy: 0.9324 - loss: 0.1591 - val_accuracy: 0.7222 - val_loss: 1.3852
Epoch 100/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 439ms/step - accuracy: 0.9512 - loss: 0.1378 - val_accuracy: 0.7222 - val_loss: 1.5296
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step - accuracy: 0.7222 - loss: 1.5296
Test Loss: 1.5295895338058472
Test Accuracy: 0.7222222089767456
```

Test Accuracy: 0.7222222089767456

11. CONCLUSIONS

Result table with all algorithms combined:

	Pre-processing	Specifications	Accuracy
SVM	None	Linear kernel	0.69
	Third (minutiae)	Linear kernel	0.44
	First	Linear kernel	0.38
MLP	Third (minutiae)	128 nodes in hidden layer	0.22
	Third (minutiae	256 nodes in hidden layer	0.5
	None	512 nodes in hidden layer	0.39
KNN	Third (minutiae)	9 - neighbors	0.38
	Third (minutiae)	5 - neighbors	0.33
	None	3 - neighbors	0.17
Random Forest	None	200 nr of estimators	0.5
	Third (minutiae)	200 nr of estimators	0.22
	None	100 nr of estimators	0.44
Hybrid		KNN + Random Forest	0.22
CNN	Third (minutiae)	3 convolutional layers	**0.89**
	Third (minutiae)	5 convolutional layers	0.83
	None	3 convolutional layers	0.77
	None	5 convolutional layers	0.72

Our best performing model was the Convolutional Neural Network with 3 hidden convolutional layers and 2 dense hidden layers, ReLu activation function and an Adam optimizer, with an accuracy of 89%. This was possible because of the preprocessing pipeline we put our images through in the beginning. As we can see, the original images without the preprocessing did not perform as well, with a max accuracy of 77%.

A surprise was Random Forest with a 50% accuracy because we have seen no papers about it and had thrown the assumption that it would perform very poorly.

The hybrid model also wasn't expected to perform that poorly but it requires further investigation.

REFERENCES

1. N. A. Alias and N. H. M. Radzi, "Fingerprint classification using Support Vector Machine," 2016 Fifth ICT International Student Project Conference (ICT-ISPC), Nakhonpathom, Thailand, 2016, pp. 105-108, doi: 10.1109/ICT-ISPC.2016.7519247. keywords: {Support vector machines;Fingerprint recognition;Classification algorithms;Testing;Biometric;Image Processing;Fingerprint;Support Vector Machine;Classification},

2. B. Pandya, G. Cosma, A. A. Alani, A. Taherkhani, V. Bharadi and T. M. McGinnity, "Fingerprint classification using a deep convolutional neural network," 2018 4th International Conference on Information Management (ICIM), Oxford, UK, 2018, pp. 86-91, doi: 10.1109/INFOMAN.2018.8392815. keywords: {Fingerprint recognition;Image matching;Feature extraction;Gabor filters;Histograms;Convolutional neural networks;CNN;Convolution Neural Network;Fingerprint Classification;Biometrics;Deep Learning},

3. W. F. Leung, S. H. Leung, W. H. Lau and A. Luk, "Fingerprint recognition using neural network," Neural Networks for Signal Processing Proceedings of the 1991 IEEE Workshop, Princeton, NJ, USA, 1991, pp. 226-235, doi: 10.1109/NNSP.1991.239519. keywords: {Fingerprint recognition;Neural networks;Feature extraction;Image matching;Multilayer perceptrons;Backpropagation;Fingers;Data models;Image converters;Cities and towns},

4. Manuel Cuevas, Fingerprint recognition, (2019), GitHub repository, https://github.com/cuevas1208/fingerprint_recognition.

5. Xuan Xu, Fingerprint Identification using SVM, (2019), https://cs229.stanford.edu/proj2016/report/Xu-FingerprintIdentificationUsingSVM-report.pdf