HIGH-VALUE FRUIT BIOMETRIC IDENTIFICATION VIA TRIPLET-LOSS TECHNIQUE

BY

PLAIFAH LAIMEK

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING (ARTIFICIAL INTELLIGENCE AND INTERNET OF THINGS)
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY
THAMMASAT UNIVERSITY
ACADEMIC YEAR 2022

THAMMASAT UNIVERSITY
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THESIS

BY

PLAIFAH LAIMEK

ENTITLED

HIGH-VALUE FRUIT BIOMETRIC IDENTIFICATION VIA TRIPLET-LOSS TECHNIQUE

was approved as partial fulfillment of the requirements for
the degree of Master of Engineering (Artificial Intelligence and Internet of Things)

On June 9, 2023

Chairperson _____
(Teera Phatrapornnant, Ph.D.)

Member and Advisor _____
(Associate Professor Waree Kongprawechnon, Ph.D.)

Member _____
(Associate Professor Toshiaki Kondo, Ph.D.)

Member _____
(Professor Tsuyoshi Isshiki, Ph.D.)

Director _____
(Professor Pruettha Nanakorn, D.Eng.)

Thesis Title	HIGH-VALUE FRUIT BIOMETRIC IDENTIFICATION VIA TRIPLET-LOSS TECHNIQUE
Author	Plaifah Laimek
Degree	Master of Engineering (Artificial Intelligence and Internet of Things)
Faculty/University	Sirindhorn International Institute of Technology/Thammasat University
Thesis Advisor	Associate Professor Waree Kongprawechnon, Ph.D.
Academic Years	2022

ABSTRACT

This thesis proposes a novel method for biometric authentication of fruits based on their distinctive rind patterns, similar to fingerprint identification. Luxury fruits, highly valued in Japan, currently rely on serial numbers, QR codes, and RFID tags for authentication, which can be forged or replicated. By implementing biometric authentication using rind patterns, the trust and value of these fruits can be significantly enhanced, while also preventing fraud and counterfeiting in the agricultural industry. The study introduces a melon identification system that utilizes a convolutional neural network (CNN) with a triplet loss function, enabling accurate identification even with variations in lighting, shadows, and angle. The proposed method overcomes the limitations of previous approaches by capturing important features through CNN's automatic feature identification. This research contributes to the field of agricultural product authentication, providing a secure and reliable method that can be extended to other products, increasing customer trust and market value.

Keywords: Agriculture, Authentication, Fruits, Melon, Rind Pattern, Identification, Verification, Computer Vision, Machine learning

ACKNOWLEDGEMENTS

This thesis was supported by the Thailand Advanced Institute of Science and Technology (TAIST), National Science and Technology Development Agency (NSTDA), Tokyo Institute of Technology, Sirindhorn International Institute of Technology (SIIT), and Thammasat University (TU) under the TAIST Tokyo Tech scholarship program.

Plaifah Laimek

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

With the increasing attention on artificial intelligence (AI) and its successful implementation in various domains, including culinary, transportation, e-commerce, and security, Biswal (2022) the potential for implementing AI-based technologies in product authentication is immense. The use of machine learning to improve the reliability of digital image authentication has sparked the development of biometric authentication methods, such as fingerprint scanner for personal electronics devices security Stanley et al. (2009) and facial recognition used for online banking Yogalakshmi et al. (2020).

In this thesis, our focus lies on the application of biometric authentication in the agricultural industry, particularly in authenticating fruits based on their unique rind patterns. While biometric identification methods have gained significant attention in various sectors, their utilization in the authentication of luxury fruits remains largely unexplored. In countries like Japan, luxury fruits such as melons, watermelons, grapes, apples, and white strawberries hold immense cultural value and are commonly exchanged as prestigious gifts. Among them, the Yūbari king melon is the most valuable and popular, retail and auctioned prices ranging from $30 to $30,000 Kim (2022). The value of these luxury fruits largely stems from the limited supply as they are specifically cultivated with specific process, environment and nutrients. Especially the Yūbari king melon has to be grown in Yūbari city, Hokkaido province, the traceability of the product is highly essential for the consumer confidence and its market value. Although currently, the authentication practices for these luxury fruits heavily rely on a combination of serial numbers, QR codes, and RFID tags Kumar et al. (2009). However, these conventional methods are prone to vulnerabilities, as they can be easily forged or replicated, thus jeopardizing the security and integrity of these valuable products.

To address these limitations and enhance customer trust, we propose a novel approach: biometric authentication of fruits using their rind patterns. The concept draws inspiration from the success of fingerprint identification in human biometrics. By utilizing the uniqueness and intricacy of each fruit's rind pattern, similar to a person's fingerprint, we aim to develop a secure and reliable authentication method that can significantly increase the value and trustworthiness of luxury fruits while deterring fraudulent practices in the agricultural industry.

Previous research on melon identification by Ishiyama et al. (2012) has incorporated minutiae feature extraction, a technique commonly used in fingerprint matching. While these studies have achieved promising results in controlled image acquisition environments, minutiae features are used to identify human fingerprint and may not account for the distinct characteristics of melon rind patterns Jain et al. (2006), which often contain features beyond simple bifurcations. In our research, we aim to overcome this limitation by utilizing a convolutional neural network (CNN) architecture with a triplet loss function. This approach enables the network to automatically identify relevant features in the training process, capturing not only bifurcations but also trifurcations and other unidentified features that contribute to accurate melon pattern matching.

CHAPTER 2
BACKGROUND KNOWLEDGE

The research community has given limited attention of the topic of agricultural product authentication. However, a notable study conducted by Ishiyama Rui in 2012 Ishiyama et al. (2012) which not only proved the viability of minutiae features recognition on melon rind pattern but this study also implemented the pose-normalization technique. As the image data of each melon was acquired using a handheld camera, each melon stem may not be centered in the middle of the image. These variations of melon angles can reduce the matching accuracy, though it can be negated using the pose-normalization technique.

Minutiae extraction techniques typically focus on specific features prominent in fingerprints, such as terminations, ridges, and bifurcations Ali et al. (2016) and Jain et al. (2006). Ishiyama's study demonstrated the effectiveness of using minutiae extraction for melon identification. However, the unique characteristics of melon rind patterns and fingerprints indicate that relying solely on minutiae features might overlook crucial elements required for accurate melon pattern matching. This observation was supported by the dataset analyzed in this study which revealed the presence of not only bifurcations but also trifurcations in melon rind patterns as visually depicted in Fig. 2.1. To address this limitation, this thesis proposes a novel method that utilizes a triplet loss function integrated with a convolutional neural network (CNN) architecture to identify similarities among inputted melon images. CNNs have demonstrated their ability to automatically capture relevant features during the training process Alzubaidi et al. (2021). By leveraging the power of CNNs, the proposed method aims to capture trifurcations and other critical features that might have been overlooked by relying solely on minutiae features.

The triplet loss function, a supervised learning technique, plays a pivotal role in the proposed method. It minimizes the distance between positive pairs of melon images while pushing negative pairs further apart, thus optimizing the margin in the embedding space. This technique has shown exceptional efficacy in tasks like face recognition Schroff et al. (2015b), which shares similarities with melon rind recognition, where the model needs to identify individual melons accurately.

By combining the unique characteristics of melon rind patterns, the power of CNNs in feature extraction, and the effectiveness of the triplet loss function, this thesis aims to establish a robust and accurate authentication system for luxury fruits. The proposed method

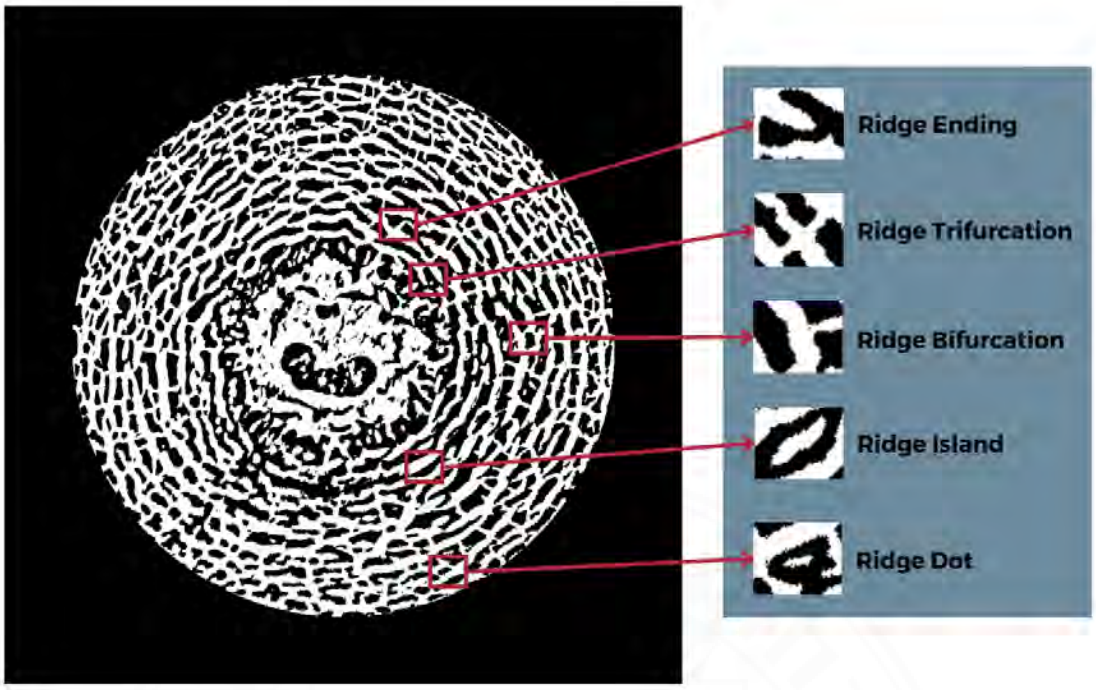


Figure 2.1 Fingerprint minutiae-like pattern on melon rind.

has the potential to revolutionize agricultural product authentication, enhance customer trust, and mitigate fraud and counterfeiting in the industry.

2.1 Minutiae Feature

In the field of fingerprint recognition, minutiae features refer to the unique and distinctive local characteristics or patterns found within fingerprint images. These distinct features are widely used in fingerprint matching and identification Hollingum (1992). Fingerprints, much like the patterns on a melon rind, are highly individualistic and exhibit intricate ridge structures. These ridge structures form unique patterns that are present on the surface of the fingertip. Minutiae features are specific points where the ridge structures exhibit changes or distinct properties, just as the irregularities, bumps, and patterns on a melon rind form its distinct features. There are two primary types of minutiae features: Ridge Endings: Ridge endings occur when a ridge segment terminates abruptly, forming a point or a small loop. They represent the points where the end of the ridge, indicating the termination of a particular ridge structure. Ridge Bifurcations: Ridge bifurcations occur when a ridge splits into two separate branches, forming a Y-shaped structure. They represent the points where the fingerprint ridges diverge in multiple directions. Other types of minutiae features may include dots, short ridges, or ridge crossings, but ridge endings and ridge bifurcations are the most commonly utilized and reliable minutiae features in fingerprint recognition sys-

tems Hong et al. (1998). These types of patterns can also be found on the melon pattern as shown in Fig. 2.1. The process of extracting minutiae features involves several steps. First, a fingerprint image is acquired using a sensor such as an optical or capacitive scanner. The image is then pre-processed to enhance its quality, remove noise, and improve contrast. Next, the ridges in the fingerprint image are thinned to obtain a skeletonized representation, where only the central ridge lines remain Jain et al. (1997). Thinning simplifies the image and separates the ridge structures for further analysis. Once the ridge thinning is complete, minutiae points are detected by examining the ridge structure. Various algorithms, such as crossing number, ridge tracing, or ridge orientation-based methods, are employed to identify and extract the minutiae features. The location, orientation, and other relevant attributes of each minutia point are recorded, forming a set of minutiae descriptors Zaeri (2011). During the matching process, the minutiae features extracted from a captured fingerprint are compared with the minutiae features stored in a database of reference fingerprints. The matching algorithms assess the spatial relationships, distances, and orientations between the minutiae points to determine the similarity or dissimilarity between the two fingerprints. Multiple matching algorithms can be used to match fingerprint image Jain et al. (1997) which is chosen according to the type of feature used in that task. Though this study mainly uses accidental coincidence probability (ACP) to match detected minutiae features, to compare the proposed method with the method incorporated by Ishiyama et al. (2012). If a sufficient number of matching minutiae features are found within defined thresholds, a positive identification or verification is established.

2.2 Triplet Loss

Triplet-loss is widely known as a great loss function to train a Siamese network. A Siamese network is a neural network with two or more identical subnetworks used to generate embedding vectors for each input and compare them. A predecessor Siamese network uses a contrasive loss function to compare the embedding result of two input images. It operates on pairs of face images, categorizing them as either positive (same identity) or negative (different identity). The goal is to learn an embedding space where the distance between positive pairs is minimized while the distance between negative pairs is maximized Tanveer et al. (2021). Although a famous study put forward by Schroff et al. (2015a) introduces the use of triplet loss function on a three-input Siamese network, which ultimately avoids the drawback of contrastive loss at the cost of increased computation resource Kertész (2021). As opposed to contrastive loss, which considers the predefined margin only when dealing with the negative pair, triplet loss keeps track of the margin between the anchor and positive

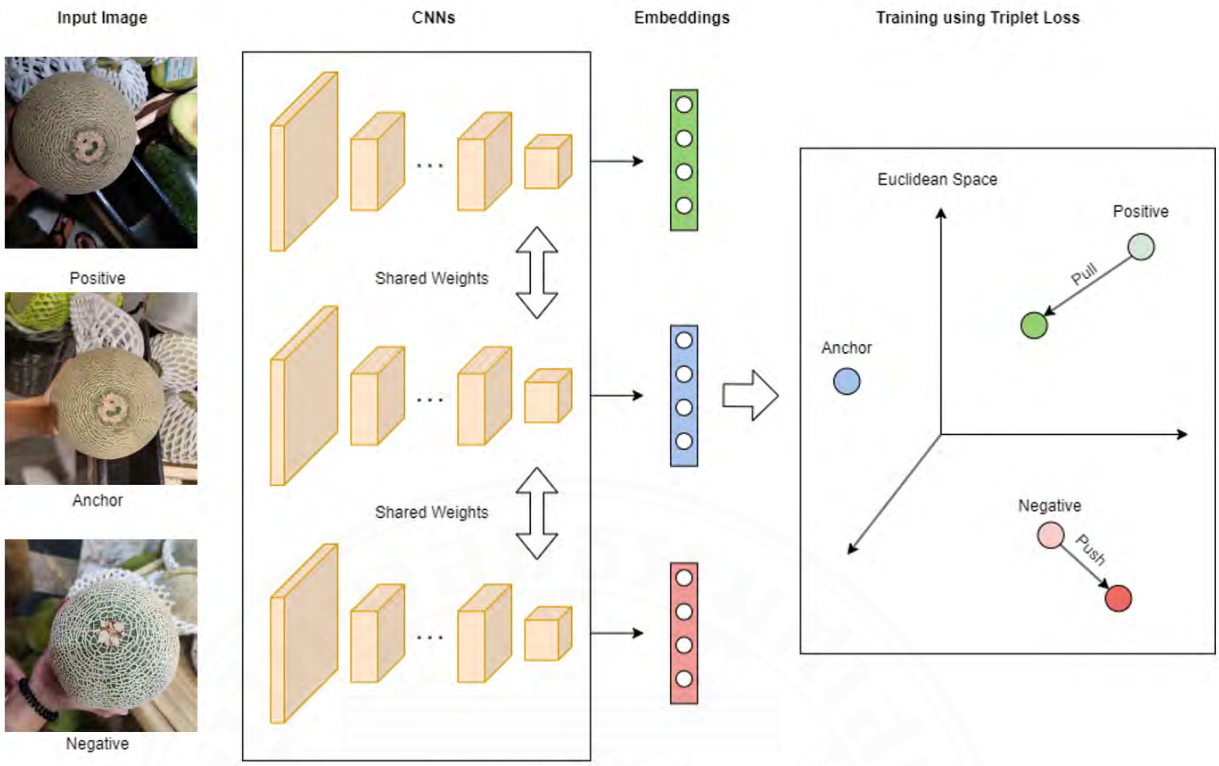


Figure 2.2 Triplet-loss training process diagram.

and the anchor and negative. The triplet loss function operates on sets of triplets, each consisting of an anchor face image, a positive face image (same identity as the anchor), and a negative face image (different identity from the anchor). The objective is to learn an embedding space where the distance between the anchor and the positive image is minimized while the distance between the anchor and the negative image is maximized. The loss function can be formulated as follows:

$$Loss = max(d(a,p) - d(a,n) + margin, 0) \tag{2.1}$$

where d(a, p) represents the distance metric between the anchor (a) and positive (p) embeddings, and d(a, n) represents the distance between the anchor (a) and negative (n) embeddings, the margin is a hyperparameter that determines the desired separation between positive and negative pairs. The loss function encourages the positive pairs to have smaller distances than the negative pairs by at least the margin value.

An illustration of the training process is shown in Fig. 2.2. The face recognition system learns the optimal parameters by iteratively sampling triplets from the training dataset and optimizing the triplet loss function. The network updates the feature embeddings such

that the distance between the anchor and positive pairs decreases while increasing the distance between the anchor and negative pairs. Various techniques can be employed to enhance the effectiveness of the triplet loss function, such as online triplet mining, which dynamically selects informative triplets during training, and batch hard mining, which focuses on the most challenging triplets within a mini-batch to optimize the loss function efficiently Shrivastava et al. (2016).

Triplet loss-based methods have demonstrated significant success in face recognition tasks, achieving high accuracy and robustness. They have been widely adopted in both commercial and research face recognition systems. By learning discriminative face embeddings through the triplet loss function, these systems can effectively handle challenging scenarios, such as variations in pose, illumination, and facial expressions, and provide reliable identification and verification capabilities in real-world applications Haider et al. (2023). The triplet loss-base Siamese network can be adapted to the melon rind pattern recognition scenario in a similar way as it is employed for face recognition. Just like human faces, melon also has a unique rind pattern that is intricate enough to be used as identification means.

CHAPTER 3
METHODOLOGY

The objective of this study is to improve upon the melon identification process described in the work of Ishiyama et al. Ishiyama et al. (2012). Our approach builds upon their study by automating the process and eliminating the need for manual intervention, making it more convenient and potentially attractive for use in the agricultural industry. Our workflow consists of four main steps: semantic segmentation, image preprocessing, training, and identity matching, as depicted in Fig. 3.1. Throughout the process, we will highlight similarities and differences with the approach described in Ishiyama et al.

3.1 Image Acquisition

Four images with slightly different angles were acquired from 56 individual melons using four smartphones. Each phone took a single photo of the melon with the stem approximately centered in the middle of the image. These images were taken free-handed, without using any platform or background, and under varying lighting conditions. As shown in Figure 3.2, the resulting images often contain shadows, non-uniform backgrounds, and misaligned stem placements. The uniformity of the image acquired differs from the approach used in the previous study of Ishiyama et al. Ishiyama et al. (2012), in which melons were placed on a flat table and photographed under controlled lighting conditions as shown in (Ishiyama et al. (2012), Fig. 5). A total of 496 images of 124 melons from varying angles and backgrounds were captured.

3.2 Semantic Segmentation

In the previous study by Ishiyama et al. Ishiyama et al. (2012), the background was manually cropped out of the images to retain only the pixels containing the melon. Semantic segmentation can eliminate this manual process. First, the ground truth mask image is created by labeling regions of the acquired images with two classes: the melon and stem. The semantic segmentation model is trained using the U-Net architecture with a VGG-16-based encoder on labeled ground truth images. Further details about the semantic segmentation model and configuration can be found in divamgupta (2021). The trained semantic segmentation model is then used to generate a mask image of the detected melon and stems. The background can be automatically cropped out using the mask image, resulting in images that

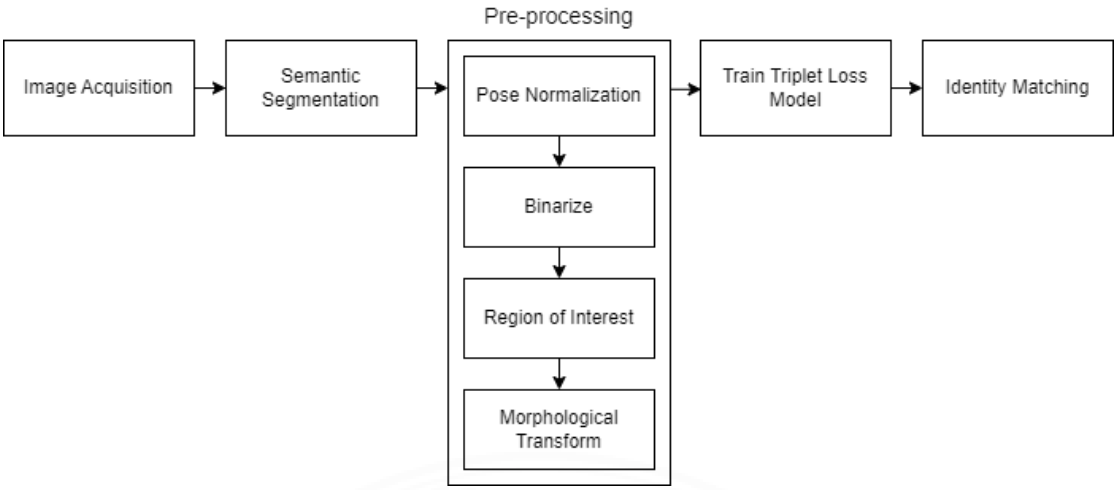


Figure 3.1 Workflow diagram

Figure 3.2 Original image from image acquisition.

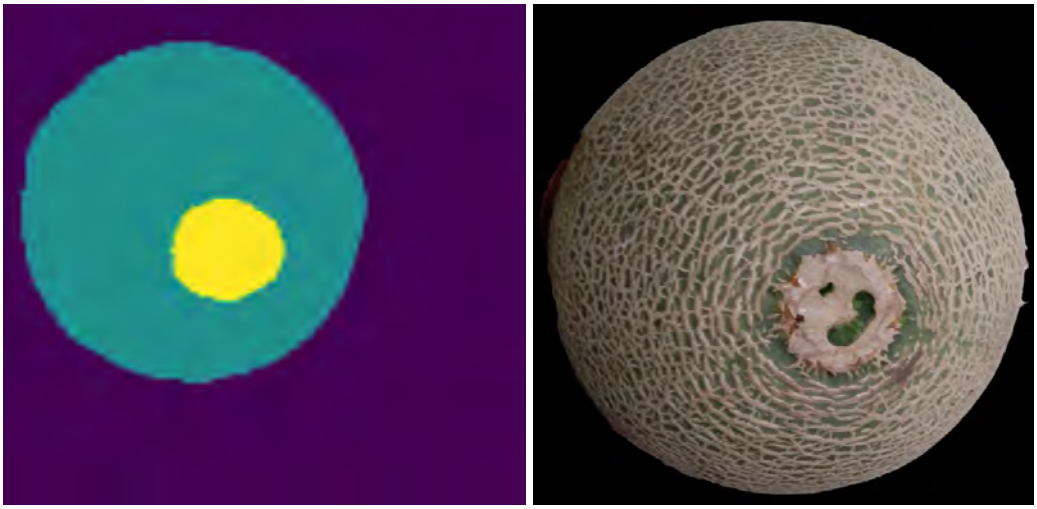


Figure 3.3 Semantic segmentation mask and cropped image.

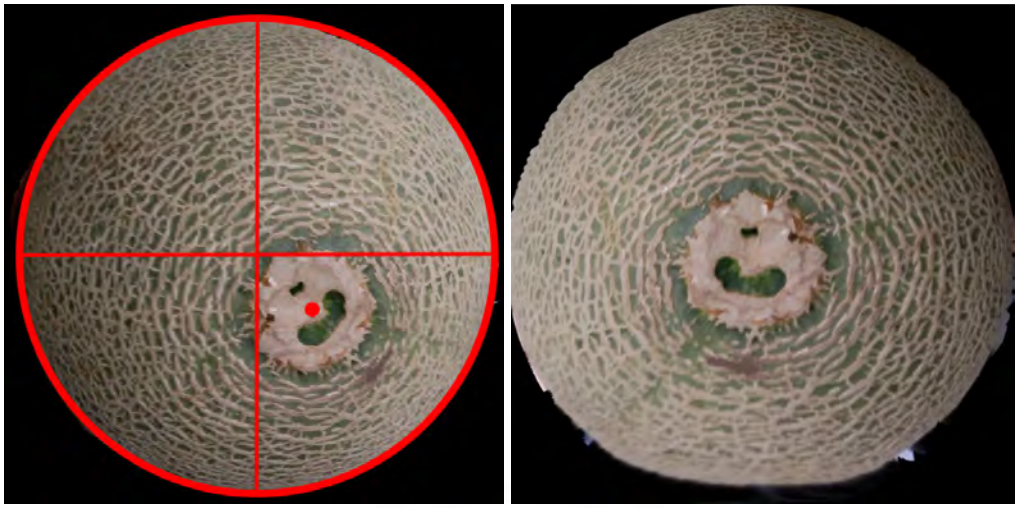


Figure 3.4 Pose normalization visualization.

contain only the melon, as shown in Fig. 3.3. This process eliminated the need for manual cropping, making the identification process more efficient.

3.3 Image Preprocessing

After obtaining a background-less image from the semantic segmentation process, the image is passed through four main preprocessing steps. Preprocessing is done to remove redundant information keeping only the relevant melon rind pattern to be trained in the triplet loss model.

3.3.1 Pose Normalization

Having the stem even slightly off-centered can result in lower matching accuracy. As the images used in this study were taken free-handed with no platform, the images contain a significant deviation from the center in most images. The pose normalization technique can be applied to center the images by translating the pixels in the image so that a desired point (e.g., the stem) is brought to the center while minimizing distortion. In the previous study by Ishiyama et al. Ishiyama et al. (2012), the stem was manually located in the image and used to guide the pose normalization process. In contrast, the semantic segmentation process already detected both the melon and stem in the image, allowing us to retrieve the stem location from the detected mask and perform pose normalization without needing manual selection. An example of the pose normalization process and its result is shown in Fig. 3.4. The result of pose normalization helps to improve the accuracy of image matching by reducing variations in the placement of the stem.

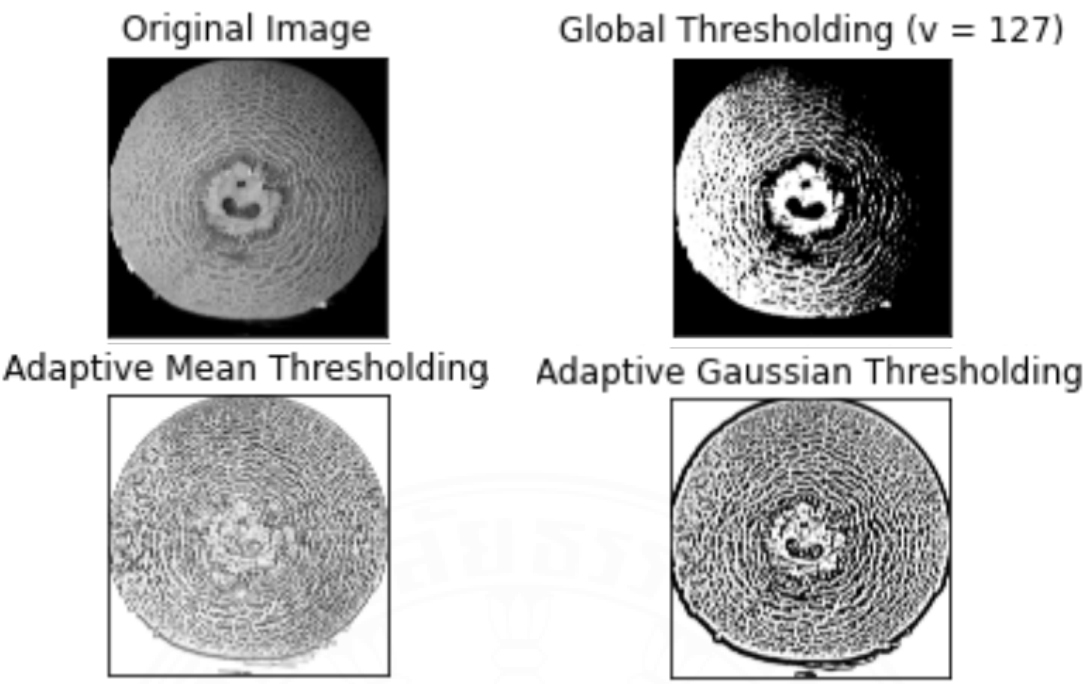


Figure 3.5 Comparison of thresholding methods.

3.3.2 Binarize

After pose normalization, we processed the images using a thresholding function to reduce the dimensionality of the data and transform the RGB images into binary images. As shown in Fig. 3.5 compared to the other thresholding methods, adaptive Gaussian thresholding can accurately transform and emphasize the relevant rind pattern. Adaptive Gaussian thresholding is used to binarize the images to simplify the training image and remove redundant information, significantly reducing the model's training resource to identify and match the melons.

3.3.3 Region of Interest

The drawback of using semantic segmentation to crop the images is that the outer perimeter of each melon image may not be uniform, which could negatively impact the accuracy of our matching model. The binary image is cropped, retaining only each image's central 60% circular area. Removal of the irregular perimeter helped to ensure that the images were more consistent in terms of their size and shape, improving the performance of our model.

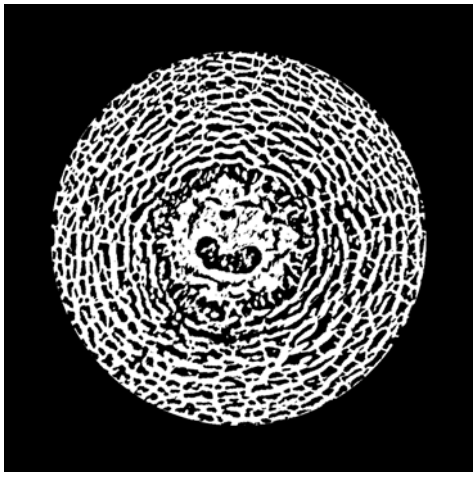


Figure 3.6 ROI cropped binary image.

Figure 3.7 Triplet loss training visualization.

3.3.4 Morphological transform

One issue that can arise during the thresholding process is the introduction of salt and pepper noise, which can negatively impact the model's performance. The morphological opening followed by closing can be used to eliminate these noises. These operations are known to smooth out sharp edges on the contours of the shapes, fill in any gaps or holes, and remove stray pixels, resulting in a clearer and sharper image.

3.4 Triplet Loss

Triplet loss was first put forward by Schroff, Florian in 2015 Schroff et al. (2015b) as a method for face recognition. As opposed to regular classification models, triplet loss excels in recognizing a large number of classes. Triplet loss does not directly classify each input image as each designated class. It determines the similarity of each image by the distance between each image after encoding them into the embedding space.

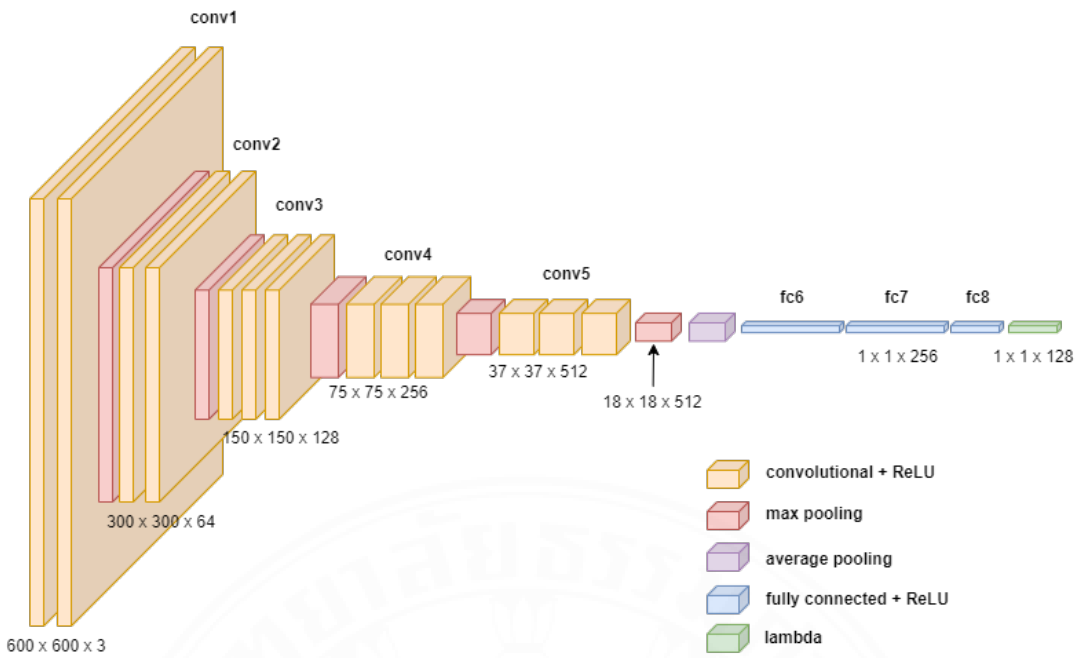


Figure 3.8 VGG-16 layers configuration.

3.4.1 Model Configuration

The triplet-loss model is based on the VGG-16 model, a well-known model pre-trained on the ImageNet dataset, by removing the classification layer and using the feature learning layer to recognize melon rind patterns. This approach, known as transfer learning, allowed the model to achieve better performance in a shorter training time. The modified VGG-16 model is shown in Fig. 3.8.

3.4.2 Training

The Triplet loss model is trained by simultaneously using three inputs: anchor, positive and negative image. In the training process, the loss of the model is calculated using the equation:

$$L = max(d(a,p) - d(a,n) + m, 0) \tag{3.1}$$

where:

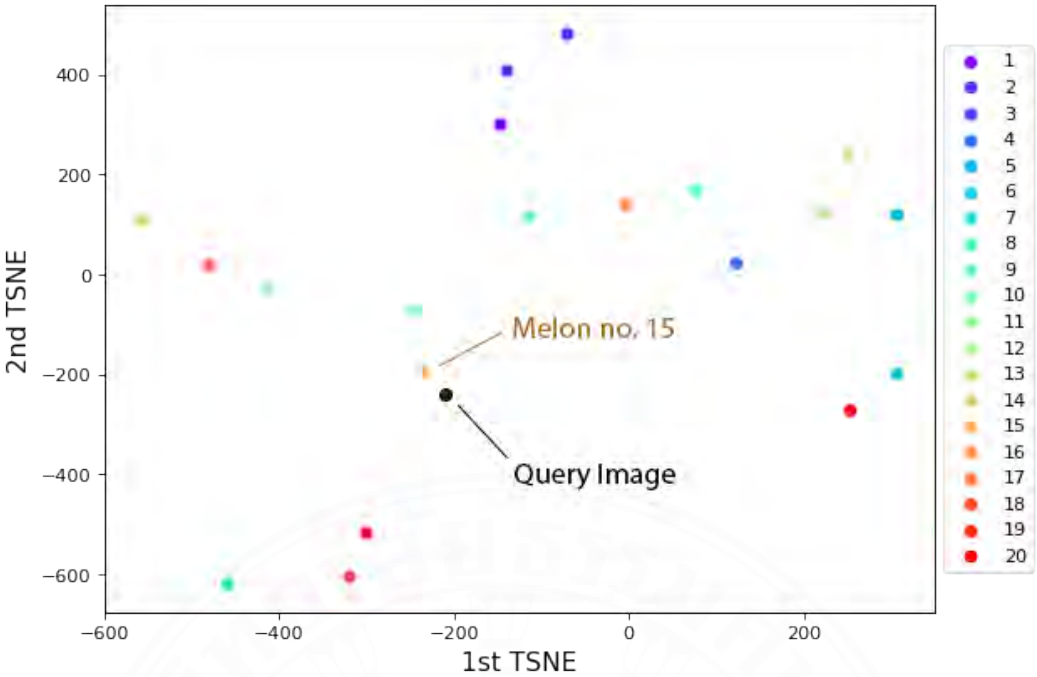


Figure 3.9 Query image identification on embedding space.

a = anchor, focal sample.

p = positive, sample in the same class as the anchor.

n = negative, sample in different class of the anchor.

d = euclidean distance function to measure the distance between the samples in embedding space.

m = margin, the distance between the negative and positive sample.

During training, the model is fed with multiple batches of triplets consisting of an anchor, a positive, and a negative sample. After transforming the input images into embedding vectors, the loss for each triplet is calculated. The model's weights are then adjusted to increase the distance between the negative and anchor samples while decreasing the distance between the anchor and positive samples. In each epoch, we use two types of triplet batches: a random batch, which consists of triplets selected randomly, and a hard batch, which consists of triplets with high loss values. While training, the model will modify its weights until the negative sample is farther away from the anchor and the positive sample closer. The distance between the negative and positive samples is also optimized to minimize the model's cost and prevent overfitting.

3.4.3 Identity Matching

After training, the model is able to transform binary melon images into matrices of embedding values. These values can be used to identify the melon in each image, as the dataset includes four images for each unique melon. One image of each melon is transformed into an embedding matrix and kept as a database of known melons, while the remaining images are used as query images. To identify a query image, it is first converted into an embedding matrix, and the Euclidean distance between it and the other images in the database is calculated. The image with the minimum distance is identified as the matching image and, therefore, the identity of the query image. This identity matching process is illustrated in Fig. 3.9. In the example shown, the image in the database with the minimum distance is identified as melon number 15.

CHAPTER 4
RESULT AND DISCUSSION

The model and training configurations were verified using a 10-fold cross-validation method. The melon image dataset was divided into ten parts, with nine parts used for training and the remaining part used as test images for each fold of validation. Out of the total 124 melon identities, 12 melons were selected as test images, rotating them until all parts of the dataset were used as test images. The matching process involved the following steps: (1) One image per melon from the test set served as the identity database, while the remaining three images per melon were treated as query images. (2) Each query image was compared with all identities in the database using a distance metric. (3) The identity with the embedding vector having the least Euclidean distance to the query image's embedding vector was identified as the matched identity. The matching process resulted in a total of 432 instances checked per validation fold. As the embedding vector of both the query and database image has large dimensions, dimensionality reduction technique is used to visualize relationship between each embedding matrix and visualize the model's ability to characterize and group the images of the same melon identity. Principal component analysis (PCA) a well-known dimensionality reduction technique was first applied, although the model's matching performance is high the visualized embedding space shows overlapping of multiple melon identity in the same space. As PCA is a linear technique that captures the maximum variance of the data, PCA is not effective in preserving relationships between individual data points. The visualized cluster does not correctly visualize the embedding space of triplet loss model which pushes the negative samples away while the positive samples should be positioned close together. Instead by using t-Distributed Stochastic Neighbor Embedding (TSNE) to visualize the embedding space, TSNE is a non-linear method that prioritizes preserving the local structure of the data which results in much better visualization for this study. The TSNE representation of the embedding space for each fold is depicted in Figure [4.1-4.10]. In each fold, the model successfully clustered the embeddings of the same melon identity, although some query images were positioned far from their corresponding identity cluster. Errors in identity matching arose when images were captured with the melon stem significantly off-centered, and even with pose normalization, some nuances in the rind texture were not accurately captured in the image. The model's performance is presented in Figure 4.11. To evaluate the matching performance, metrics such as false acceptance rate (FAR), false

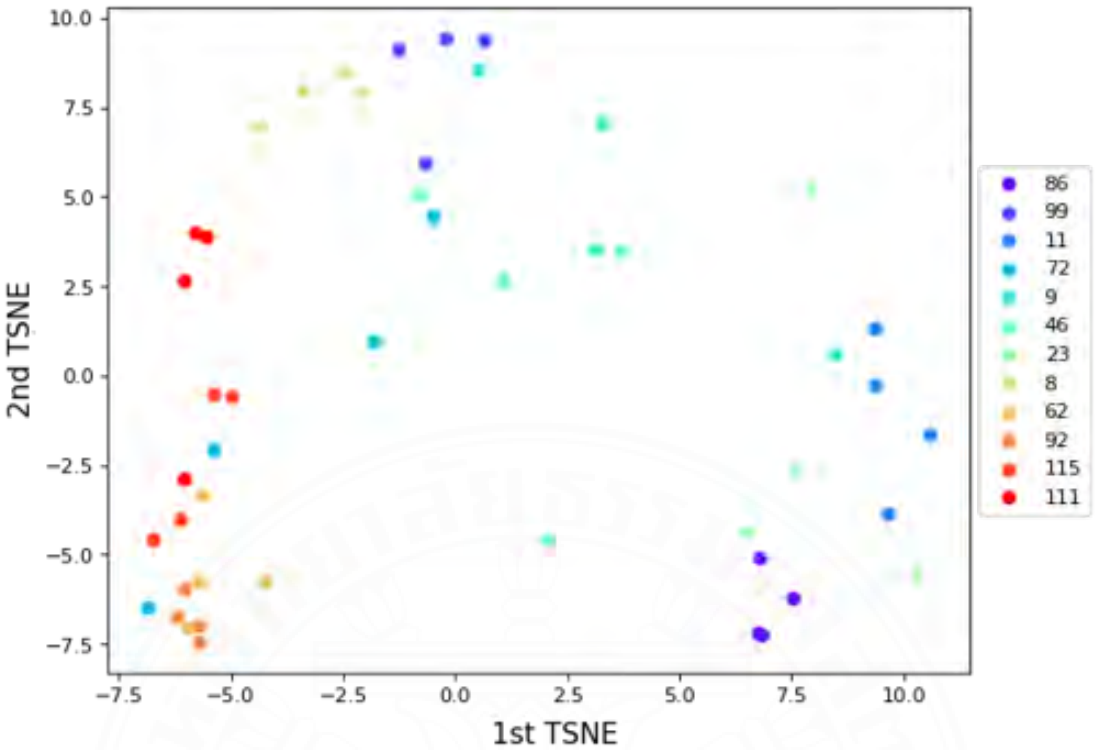


Figure 4.1 First Fold

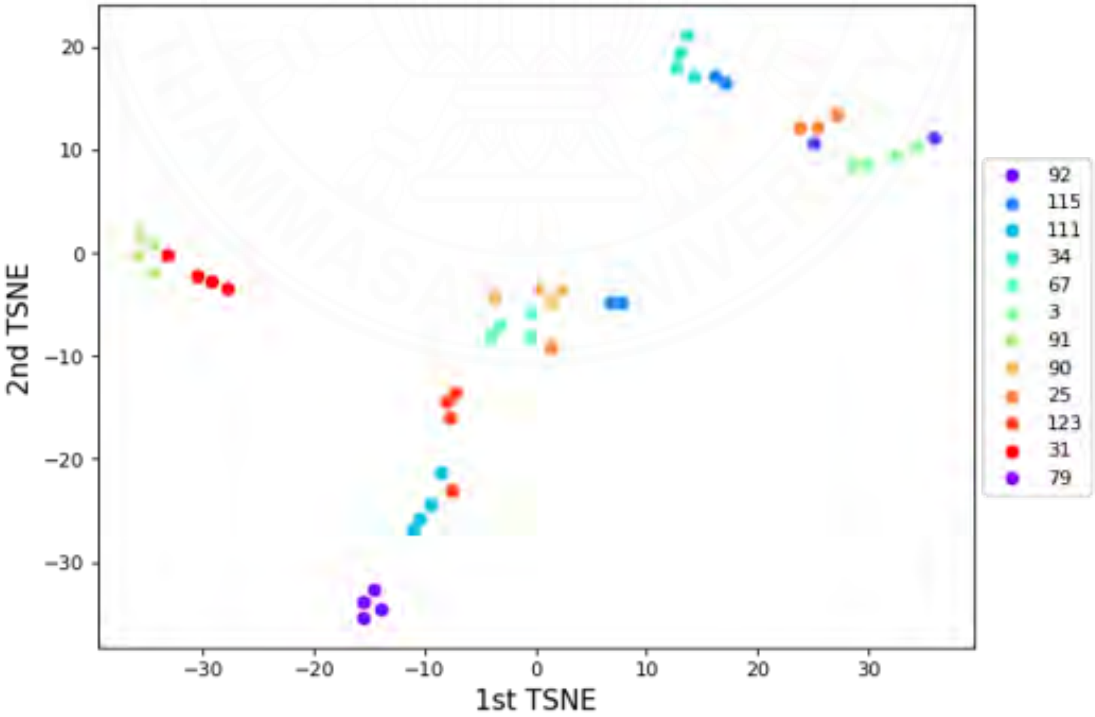


Figure 4.2 Second Fold

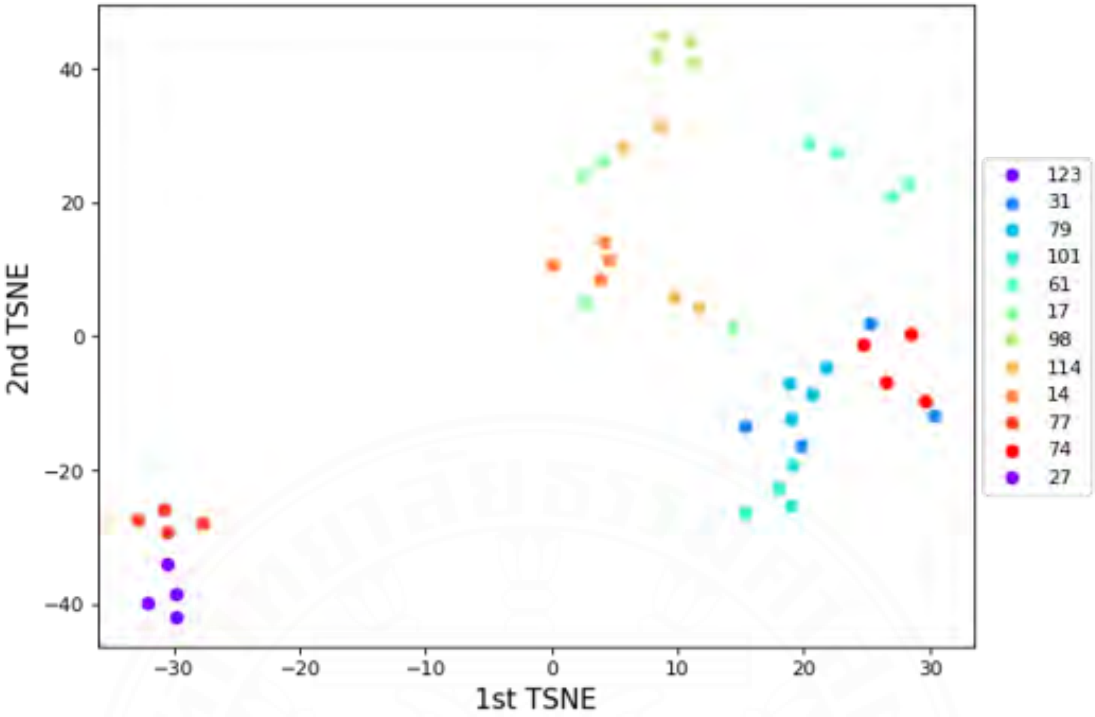


Figure 4.3 Third Fold

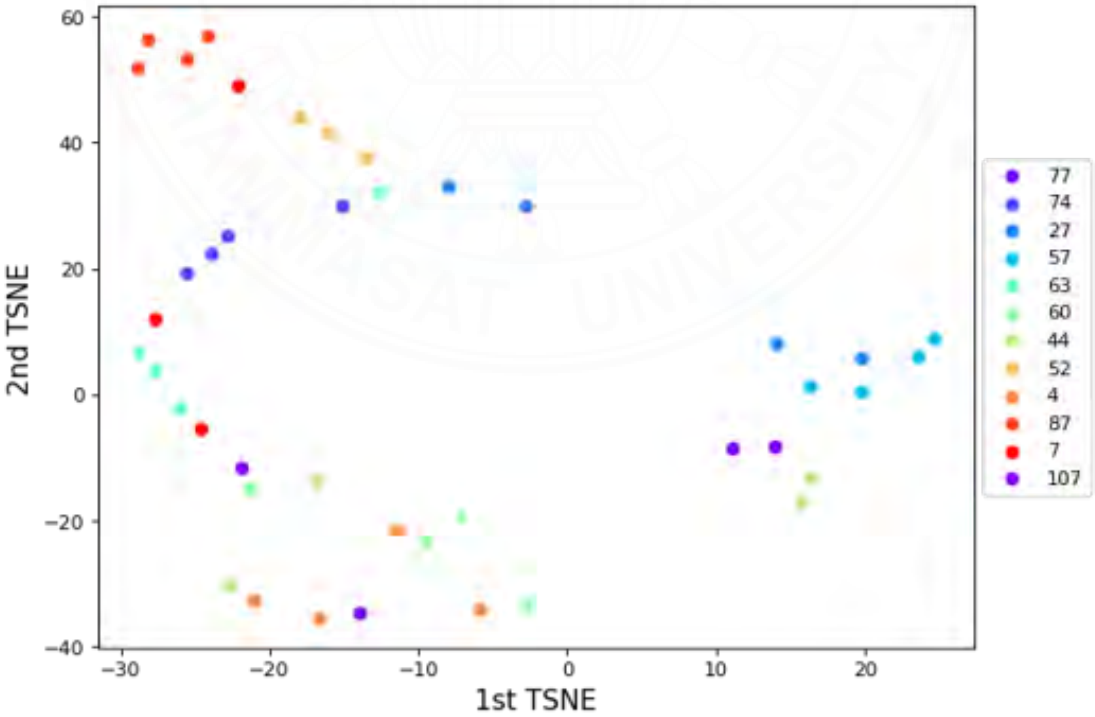


Figure 4.4 Forth Fold

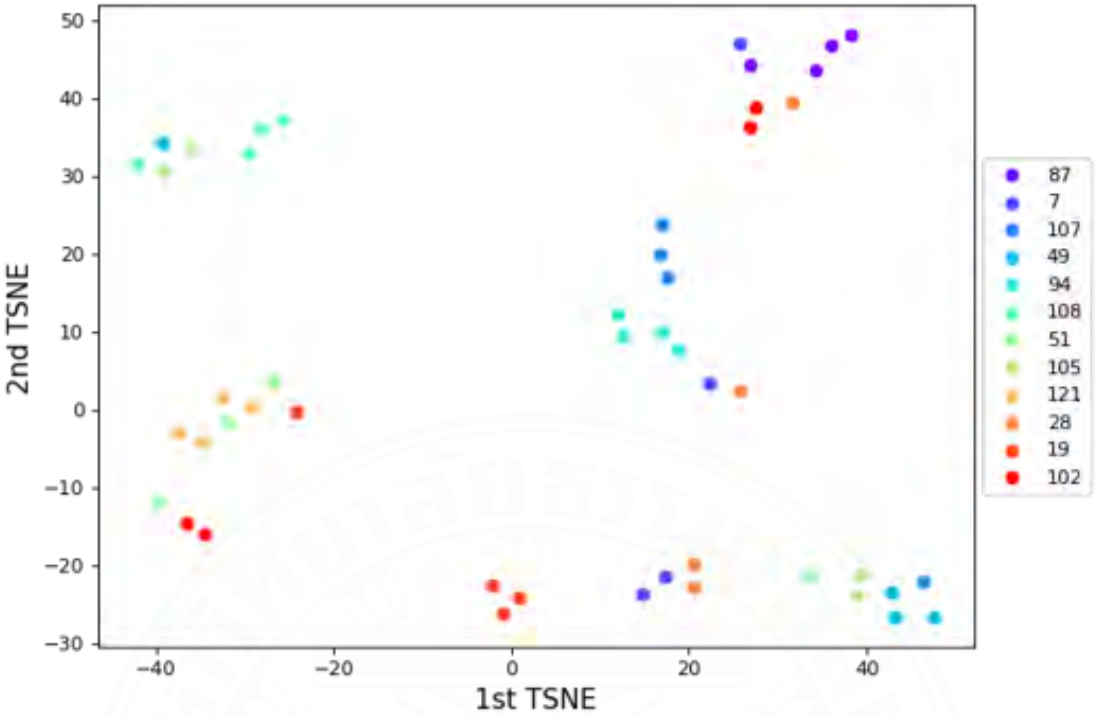


Figure 4.5 Fifth Fold

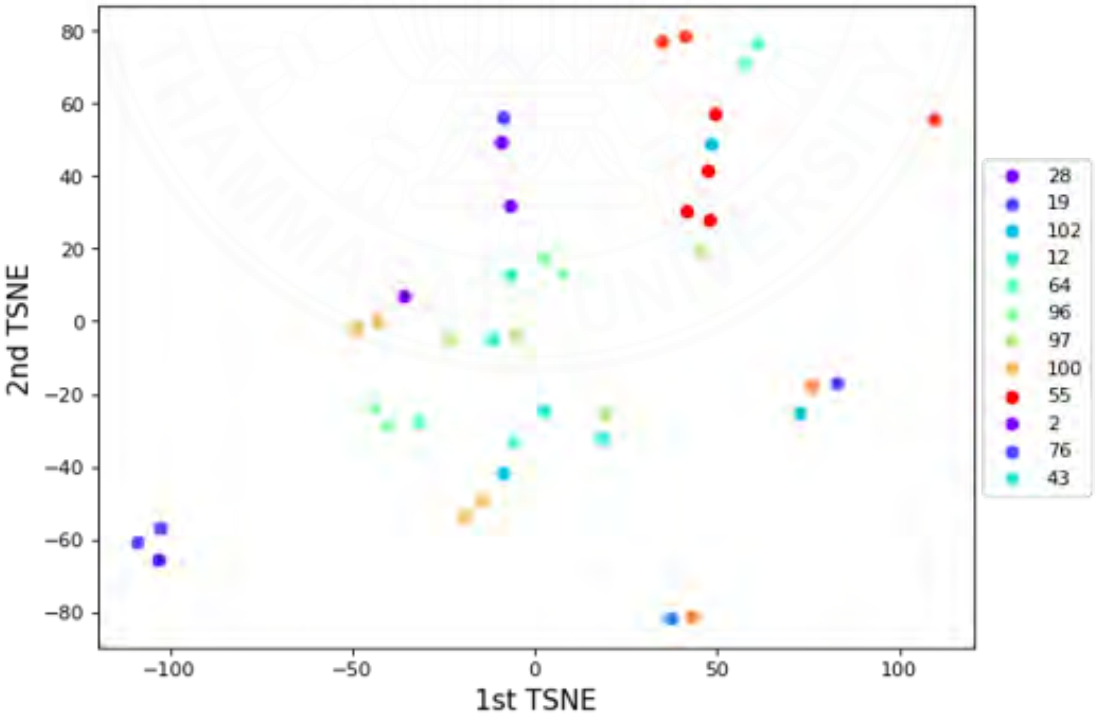


Figure 4.6 Sixth Fold

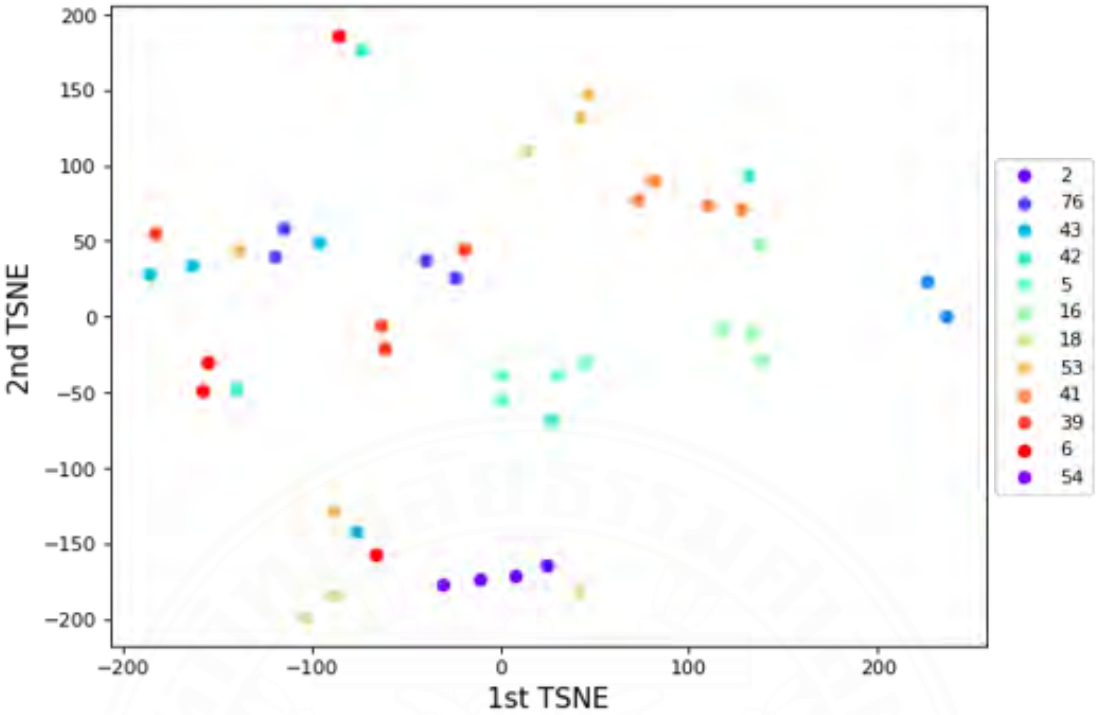


Figure 4.7 Seventh Fold

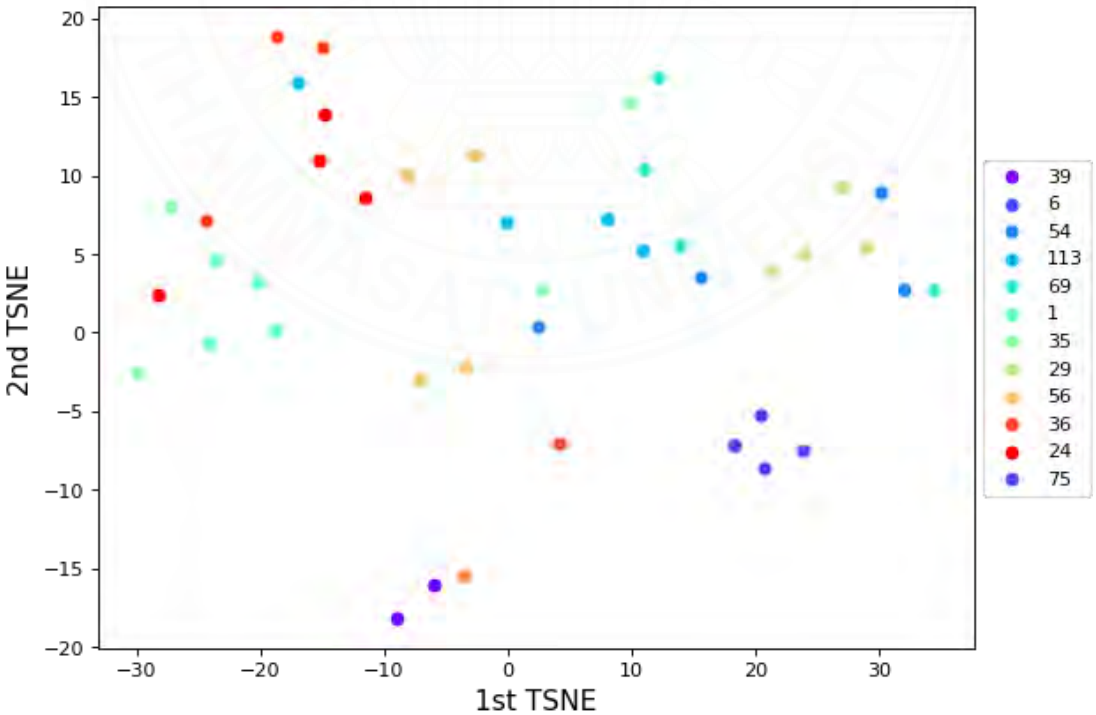


Figure 4.8 Eighth Fold

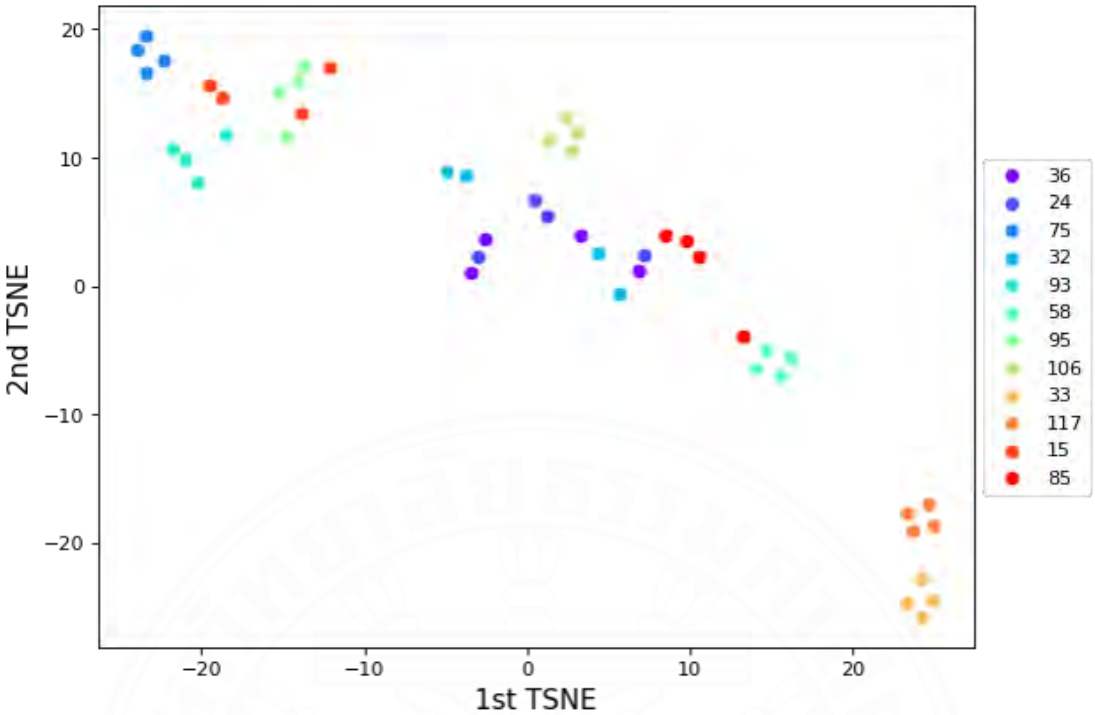


Figure 4.9 Ninth Fold

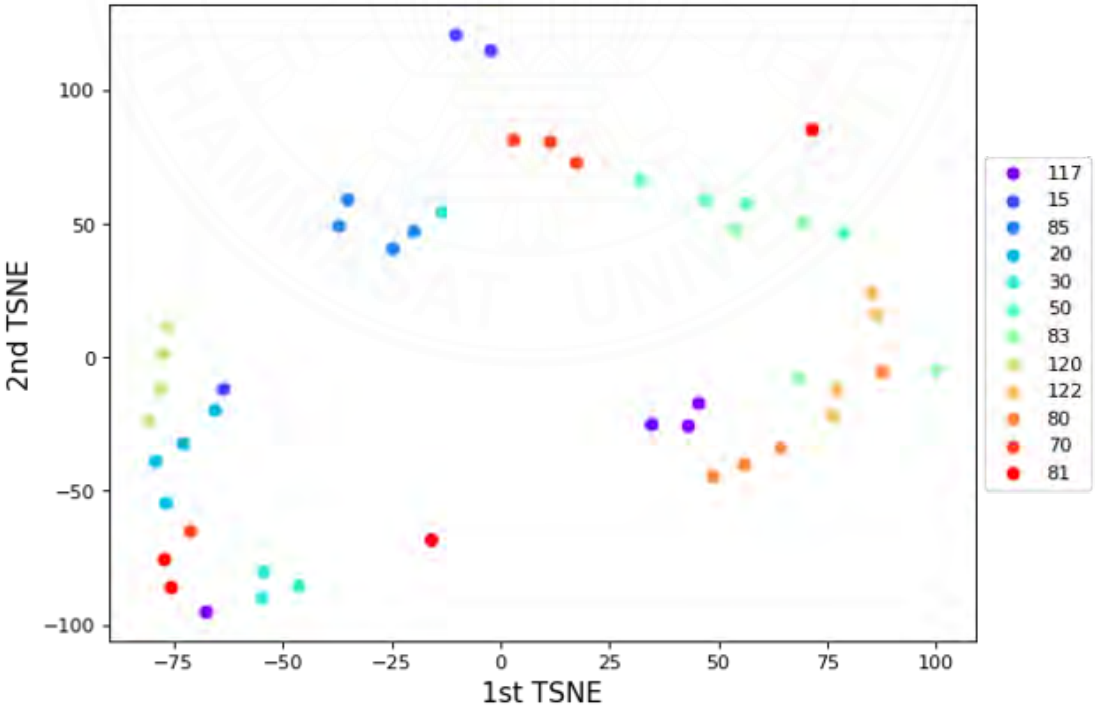


Figure 4.10 Tenth Fold

	False Acceptance Rate (FAR)	False Rejection Rate (FRR)	Top Rank ID Error
Fold 1	0.23%	6.94%	3.47%
Fold 2	0.23%	5.32%	2.55%
Fold 3	0.00%	6.71%	3.70%
Fold 4	0.00%	7.41%	4.17%
Fold 5	0.46%	6.94%	4.40%
Fold 6	0.23%	7.18%	4.86%
Fold 7	1.11%	5.56%	3.70%
Fold 8	1.39%	5.09%	4.40%
Fold 9	0.00%	5.09%	3.01%
Fold 10	0.46%	5.79%	2.78%
AVG	**0.41%**	**6.20%**	**3.70%**

Figure 4.11 System performance in 10-fold cross-validation.

rejection rate (FRR), and Top Rank ID Error were utilized, which are commonly employed for evaluating the performance of biometric systems Natarajan and Shanthi (2018).

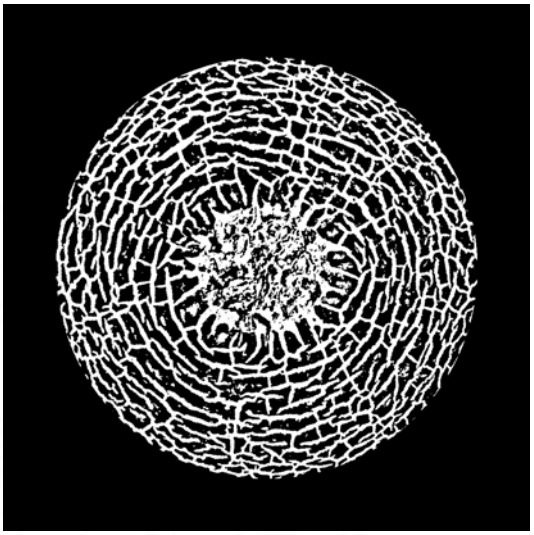
$$FAR = \frac{FalseAcceptance}{TotalCheck} \tag{4.1}$$

$$FRR = \frac{FalseRejection}{TotalCheck} \tag{4.2}$$

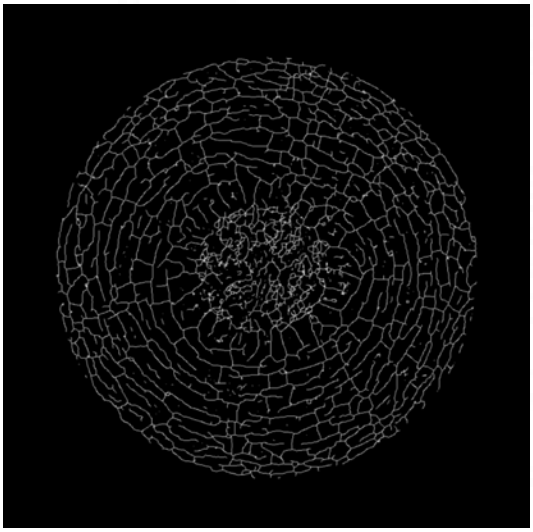
$$TopIDErr = \frac{NegativePair}{TotalCheck} \tag{4.3}$$

Each melon image is queried in the evaluation process, and its embedding is used to calculate the Euclidean distance with all identities in the database. Suppose the minimum distance exceeds the specified threshold. In that case, the queried melon is considered a counterfeit melon. It is counted towards false rejections, contributing to the false rejection rate (FRR) calculation since all melons used are present in the database. On the other hand, false acceptances, which contribute to the false acceptance rate (FAR), occur when the minimum distance falls below the threshold, but the matched identity is incorrect. The top-rank ID error (TopIDErr) is determined by instances where the minimum distance pair has a mis-

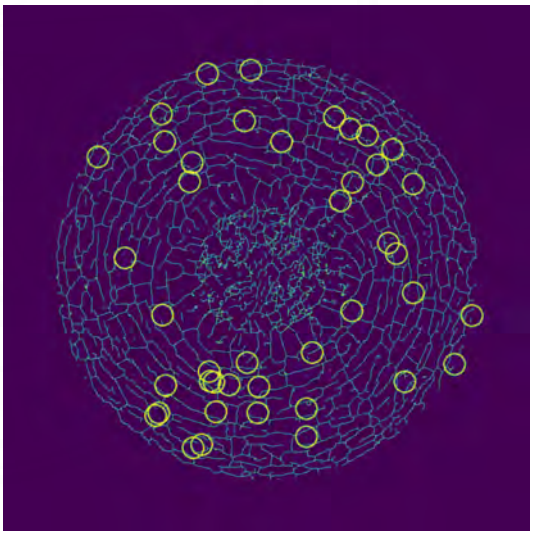
(a) Cropped and pose-normalized



(b) Binarization and ROI cropping.



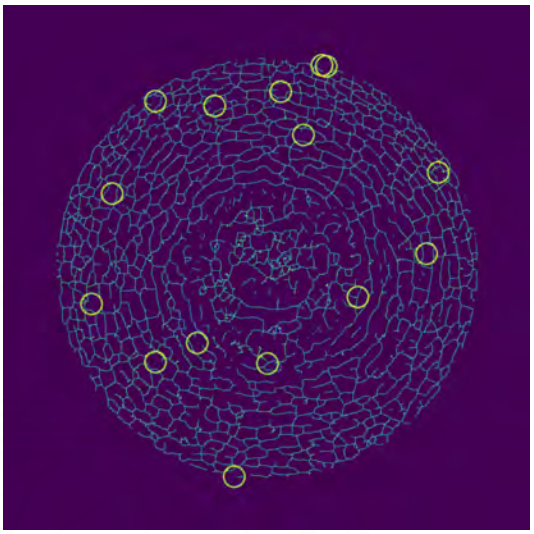
(c) Ridge thinning to obtain skeletonized structure of melon rind pattern.



(d) Detected Minutiae feature on the texture of melon.

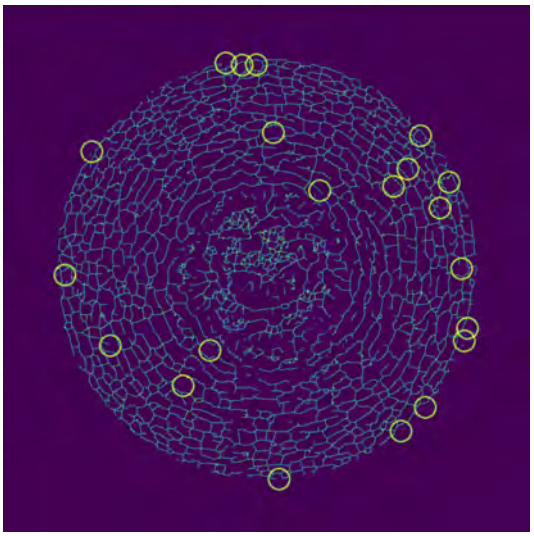
Figure 4.12 Minutiae feature extraction method.

(a) Melon #9 taken with phone A.



(b) Detected minutiae feature of melon #9A.

(c) Melon #9 taken with phone B.



(d) Detected minutiae feature of melon #9B.

Figure 4.13 Example of inconsistent detected melon pattern of the same melon identity.

matched identity, regardless of whether the distance value surpasses the threshold.

The performance of the proposed method is satisfactory. However, it falls short when compared to the results of the study by Ishiyama et al. Ishiyama et al. (2012), which achieved a top-rank ID error of only 0.06%. It is important to note that the previous study utilized a larger dataset of 1,776 melons, resulting in 3,154,200 matching instances. As shown in Equations (4.1) - (4), a larger sample population can lead to a reduction in both FAR and FRR. The significant difference in the number of total checking instances highlights the need for a much larger dataset to compare performance with the previous study comprehensively.

In contrast to the method used in the Ishiyama study, the identical technique was applied to the same dataset in this study. The process is illustrated in Figure 4.12, starting with background removal and pose normalization to center the melon stem. The image is then binarized and ridge thinning to obtain the ridge skeleton. Finally, minutiae-like features of the melon rind pattern are extracted to identify each melon identity.

However, attempting to apply the same procedure to the dataset used in this study resulted in inconsistent results. The dataset exhibited significant dissimilarity among the four images of the same melon, primarily due to significant camera angle variations and inconsistent lighting conditions for each sample. Although minutiae features were successfully detected using the accidental coincidence probability (ACP) approach on the same image, achieving a 100% match as expected, the results were disappointing when matching different images of the same melon identity, with a match rate of less than 1%. The match rate remained unacceptably low even when considering the pair of images with the least angle variation and lighting conditions. An example of the inconsistency in detected minutiae features is shown in Figure 4.13. Therefore, employing minutiae feature extraction and ACP on this dataset is not viable.

On the other hand, the triplet loss method proves to be much more reliable, even when faced with high variations in image quality. In practical use cases where queries are made from the customer's side, it is inevitable that some query images will have poor lighting and variations in angle with respect to the center of the melon.

Furthermore, additional studies could be conducted to compare the proposed method with other minutiae-based techniques, including directional minutiae combined with matching algorithms other than ACP. These experiments allow for a comparison of the trade-offs between accuracy and efficiency. It is worth noting that the proposed triplet loss method requires significantly higher computational resources compared to other traditional techniques used for fingerprint recognition tasks.

REFERENCES

Ali, M. M., Mahale, V. H., Yannawar, P., & Gaikwad, A. T. (2016). Overview of fingerprint recognition system. *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 1334–1338. https://doi.org/10.1109/ICEEOT.2016.7754900

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1), 53. https://doi.org/10.1186/s40537-021-00444-8

Biswal, A. (2022). Artificial intelligence (ai) applications in 2023: Simplilearn. https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/artificial-intelligence-applications

Divamgupta. (2021). Image-segmentation-keras: Implementation of segnet, fcn, unet , pspnet and other models in keras.

Haider, I., Yang, H.-J., Lee, G.-S., & Kim, S.-H. (2023). Robust human face emotion classification using triplet-loss-based deep cnn features and svm. *Sensors*, *23*(10). https: //doi.org/10.3390/s23104770

Hollingum, J. (1992). Automated fingerprint analysis offers fast verification. *Sensor Review*, *12*(3), 12–15. https://doi.org/10.1108/eb007878

Hong, L., Wan, Y., & Jain, A. (1998). Fingerprint image enhancement: Algorithm and performance evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(8), 777–789. https://doi.org/10.1109/34.709565

Ishiyama, R., Nakamura, Y., Monden, A., Huang, L., & Yoshimoto, S. (2012). Melon authentication by agri-biometrics: Identifying individual fruits using a single image of rind pattern. *1*.

Jain, A., Hong, L., Pankanti, S., & Bolle, R. (1997). An identity-authentication system using fingerprints. *Proceedings of the IEEE*, *85*(9), 1365–1388. https://doi.org/10.1109/5.628674

Jain, A., Ross, A., & Pankanti, S. (2006). Biometrics: A tool for information security. ieee tran inform forensics secur. *Information Forensics and Security, IEEE Transactions on*, *1*, 125–143. https://doi.org/10.1109/TIFS.2006.873653

Kertész, G. (2021). Different triplet sampling techniques for lossless triplet loss on metric similarity learning. *2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 000449–000454. https://doi.org/10.1109/SAMI50585.2021.9378628

Kim, G. (2022). Most expensive fruits in japan. https://www.bokksu.com/blogs/news/7-most-expensive-fruits-in-japan

Kumar, A., Hanmandlu, M., Madasu, V. K., & Lovell, B. C. (2009). Biometric authentication based on infrared thermal hand vein patterns. *2009 Digital Image Computing: Techniques and Applications*, 331–338. https://doi.org/10.1109/DICTA.2009.63

Natarajan, A., & Shanthi, N. (2018). A survey on multimodal biometrics authentication and template protection. *2018 International Conference on Intelligent Computing and Communication for Smart World (I2C2SW)*, 64–71. https://doi.org/10.1109/I2C2SW45816.2018.8997125

Schroff, F., Kalenichenko, D., & Philbin, J. (2015a). Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 815–823. https://doi.org/10.1109/CVPR.2015.7298682

Schroff, F., Kalenichenko, D., & Philbin, J. (2015b). FaceNet: A Unified Embedding for Face Recognition and Clustering. *arXiv e-prints*, Article arXiv:1503.03832, arXiv:1503.03832.

Shrivastava, A., Gupta, A., & Girshick, R. (2016). Training region-based object detectors with online hard example mining. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 761–769. https://doi.org/10.1109/CVPR.2016.89

Stanley, P., Jeberson, W., & Klinsega, V. (2009). Biometric authentication: A trustworthy technology for improved authentication. *2009 International Conference on Future Networks*, 171–175. https://doi.org/10.1109/ICFN.2009.49

Tanveer, M., Tan, H.-K., Ng, H.-F., Leung, M. K., & Chuah, J. H. (2021). Regularization of deep neural network with batch contrastive loss. *IEEE Access*, *9*, 124409–124418. https://doi.org/10.1109/ACCESS.2021.3110286

Yogalakshmi, S., Megalan, L. L., & Jerrin Simla, A. (2020). Review on digital image processing techniques for face recognition. *2020 International Conference on Communication and Signal Processing (ICCSP)*, 1633–1637. https://doi.org/10.1109/ICCSP48568.2020.9182091

Zaeri, N. (2011). Minutiae-based fingerprint extraction and recognition. In J. Yang (Ed.), *Biometrics*. IntechOpen. https://doi.org/10.5772/17527

APPENDIX

APPENDIX A
PYTHON CODES

```
from wfmread import wfmread
from IPython.display import clear_output
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import clear_output
import os
from scipy.optimize import curve_fit
from scipy.stats import gaussian_kde
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
from PIL import Image
import PIL.ImageOps
import random
import math
ch1_dict = {
    '1':'wfm/InternalNew3_N5_300mV_200M_1us_200_Ch1.wfm',
    '2':'wfm/Surface_N5_500mV_200M_1us_200_Ch1.wfm',
    '3':'wfm/Positive2_N5_10mV_5G_2us_10000sa_Ch1.wfm',
    '4':'wfm/FreeMoving1_N5_50mV_200M_1us_200_Ch1.wfm',
    '5':'wfm/FloatingParticle_N5_2V_1G_1us_1000_Ch1.wfm',
    '6':'wfm/1-Internal_45mm33_Ch1.wfm',
    '7':'wfm/6kV_internal_oil_89pF_surface_69pF_N6_L43_HV_plates_Ch1.wfm'
}
ch2_dict = {
    '1':'wfm/InternalNew3_N5_300mV_200M_1us_200_Ch2.wfm',
    '2':'wfm/Surface_N5_500mV_200M_1us_200_Ch2.wfm',
    '3':'wfm/Positive2_N5_10mV_5G_2us_10000sa_Ch2.wfm',
    '4':'wfm/FreeMoving1_N5_50mV_200M_1us_200_Ch2.wfm',
    '5':'wfm/FloatingParticle_N5_2V_1G_1us_1000_Ch2.wfm',
    '6':'wfm/1-Internal_45mm33_Ch2.wfm',
    '7':'wfm/6kV_internal_oil_89pF_surface_69pF_N6_L43_HV_plates_Ch2.wfm'
```

```
}

# print("Creating dataset with classes: 1.Internal 2.Surface 3.Corona 4.
    Freemoving 5.Floating")

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import Normalize
from scipy.interpolate import interpn

def density_scatter( x , y, ax = None, sort = True, bins = 20, **kwargs ) :
    """
    Scatter plot colored by 2d histogram
    """
    if ax is None :
        fig , ax = plt.subplots()
        fig.set_figheight(6)
        fig.set_figwidth(6)
    data , x_e, y_e = np.histogram2d( x, y, bins = bins, density = True )
    z = interpn( ( 0.5*(x_e[1:] + x_e[:-1]) , 0.5*(y_e[1:]+y_e[:-1]) ) , data
         , np.vstack([x,y]).T , method = "splinef2d", bounds_error = False)

    #To be sure to plot all data
    z[np.where(np.isnan(z))] = 0.0

    # Sort the points by density, so that the densest points are plotted last
    if sort :
        idx = z.argsort()
        x, y, z = x[idx], y[idx], z[idx]

    ax.scatter( x, y, c=z, **kwargs )

internal = {}
internal2 = {}
surface = {}
corona = {}
```

```
insur = {}
insur2 = {}

internal['ch1'] = wfmread(ch1_dict['1']).wflist
internal['ch2'] = wfmread(ch2_dict['1']).wflist
internal2['ch1'] = wfmread(ch1_dict['6']).wflist
internal2['ch2'] = wfmread(ch2_dict['6']).wflist
surface['ch1'] = wfmread(ch1_dict['2']).wflist
surface['ch2'] = wfmread(ch2_dict['2']).wflist
corona['ch1'] = wfmread(ch1_dict['3']).wflist
corona['ch2'] = wfmread(ch2_dict['3']).wflist
insur['ch1'] = np.concatenate((internal['ch1'],surface['ch1']),axis = 0)
insur['ch2'] = np.concatenate((internal['ch2'],surface['ch2']),axis = 0)
insur2['ch1'] = wfmread(ch1_dict['7']).wflist
insur2['ch2'] = wfmread(ch2_dict['7']).wflist
for i in range(len(insur2['ch2'])):
    insur2['ch2'][i] = insur2['ch2'][i]*(-1)
data_list = [internal,internal2,surface,corona,insur,insur2]
dat_name = ['Internal','Internal','Surface','Corona','InternalSurface','
    InternalSurface']
for i in range(len(data_list)):
    ch1 = data_list[i]['ch1']
    ch2 = data_list[i]['ch2']
    Nframes = len(ch1)
    data_name = dat_name[i]

    print('Generating '+data_name+' | '+ str(Nframes)+' pulses')

    rand_idx = np.random.choice(int(Nframesrames),int(Nframes), replace=False
        ).tolist()


    # p_per_img = int(math.floor(Nframes/max_img_amnt))

    p_per_img = [20,100,500]
    n_level = ['mild','moderate','severe']
    for p_in in p_per_img:
```

```
        level = n_level[p_per_img.index(p_in)]
        data_name = dat_name[i]+"_"+level
        print(f'Generating {level}')
        rand_idx = np.random.choice(int(Nframes),int(Nframes), replace=False)
            .tolist()
        # max_img_amnt = int(math.floor(Nframes/p))
        # print(str(max_img_amnt)+ ' images |' +str(p)+' points')
        n = 0;
        while len(rand_idx) > p_in*1.3:
            # print(str(n+1),end = ', ')
            phase_lst = []
            peak_lst = []
            dif = random.uniform(0.7, 1.4)
            p = math.floor(p_in*dif)
            for _ in range(p):
                s = rand_idx.pop()
                curr_ch1 = ch1[s]
                curr_ch2 = ch2[s]
                peak_Idx = np.argmax(abs(curr_ch2))
                peak_V = curr_ch2[peak_Idx]
                peak_Phase = curr_ch1[peak_Idx]
                peak_lst.append(peak_V)
                phase_lst.append((360/4.54)*peak_Phase)
            x = np.array(phase_lst)
            y = np.array(peak_lst)
            if len(x) <5:
                break


            fig = plt.figure(frameon=False)
            fig.set_size_inches(2.56,2.56)
            ax = plt.Axes(fig, [0., 0., 1., 1.])
            ax.set_axis_off()
            abs_peak = list(map(abs, peak_lst))
            yrange = (round(max(abs_peak)*100)/100)
            fig.add_axes(ax)
```

```
        x = np.array(phase_lst)
        y = np.array(peak_lst)
        try:
            posP = max(y[x<180])
            posP = np.format_float_scientific(posP, precision = 2,
                exp_digits=1)
        except:
            posP = '0'
        try:
            negP = max(y[x>=180])
            negP = np.format_float_scientific(negP, precision = 2,
                exp_digits=1)
        except:
            negP = '0'
        ax.hist2d(x, y, (60,100), cmap=plt.cm.binary)
        ax.set_xlim([0,360])
        ax.set_ylim([-yrange*1.5,yrange*1.5])
        # fname = data_name+"("+str(per*100)+")[p"+posP+"][n"+negP+"]_"+
            str(int(np.random.rand(1)*10**8))+'.png';
        uID = str(int(np.random.rand(1)*10**8))
        # fname = str(p)+_+data_name+"[p"+posP+"][n"+negP+"]_"+ str(int(np
            .random.rand(1)*10**8))+'.png';
        fname = (f"{str(p)}_{data_name}[p{posP}][n{negP}]_{uID}.png")




        # if n >= datasetAmount[data_name]:
        #     path = '2ddata/.unused/'
        #     if not os.path.exists(path+data_name):
        #         os.makedirs(path+data_name)
        # else:
        #     path = '2ddata/.dataset/'
        #     if not os.path.exists(path+data_name):
        #         os.makedirs(path+data_name)
        path = '2ddata/.dataset/'
```

```
            # if not os.path.exists(path+data_name):
            #     os.makedirs(path+data_name)
            if not os.path.exists(f'{path}{data_name}'):
                os.makedirs(f'{path}{data_name}')
            fig.savefig('%s%s/%s' %(path,data_name,fname), bbox_inches='tight'
                , pad_inches = 0)
            image = Image.open('%s%s/%s' %(path,data_name,fname))
            image = PIL.ImageOps.grayscale(image)
            image = PIL.ImageOps.invert(image)

            image.save('%s%s/%s' %(path,data_name,fname))
            #

            plt.close('all')
            plt.clf()
            plt.cla()
            n+=1
        print(f'{n} images generated')
print('file generation completed')
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
import os
import numpy as np
np.random.seed(0)
import matplotlib.pyplot as plt
%matplotlib inline
from pylab import *
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.layers import Conv2D, ZeroPadding2D, Activation, Input,
    concatenate
from keras.layers import GlobalAveragePooling2D,Dropout
from keras.models import Model
from keras.datasets import mnist

from tensorflow.keras.layers import BatchNormalization
from keras.layers.pooling import MaxPooling2D
```

```
from tensorflow.keras.layers import concatenate
from keras.layers.core import Lambda, Flatten, Dense
from keras.initializers import glorot_uniform,he_uniform

from tensorflow.keras.layers import Layer
from keras.regularizers import l2
from keras import backend as K
from tensorflow.keras.utils import normalize
from keras.utils.vis_utils import plot_model
import keras

from sklearn.metrics import roc_curve,roc_auc_score

from keras.applications.vgg16 import VGG16
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

""" unitity functions """
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
import os

def read_image(path):
    """ function to read single image at the given path
        note: the loaded image is in B G R format
    """
    return cv.imread(path)


def BGR2RGB(image):
    """ function to transform image from BGR into RBG format """
    return cv.cvtColor(image, cv.COLOR_BGR2RGB)


def BGR2Gray(image):
    """ function to transofrm image from BGR into Gray format """
```

```
    return cv.cvtColor(image, cv.COLOR_BGR2GRAY)


def show_image(image, img_format='RGB', figsize=(8, 6)):
    """ function to show image """
    if img_format == 'RGB' or img_format == 'Gray':
        pass
    elif img_format == 'BGR':
        image = BGR2RGB(image)
    else:
        raise ValueError('format should be "RGB", "BGR" or "Gray"')

    fig, ax = plt.subplots(figsize=figsize)
    if format == 'Gray':
        ax.imshow(image, format='gray')
    else:
        ax.imshow(image)
    return fig


def detect_finger(image, face):
    """ function to denote location of finger on image """
    img = image.copy()
    for (x, y, w, h) in face:
        cv.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

    return img


def crop_finger(image, face, scale_factor=1.0, target_size=(128, 128)):
    """ crop finger at the given positons and resize to target size """
    rows, columns, channels = image.shape
    x, y, w, h = face[0]
    mid_x = x + w // 2
    mid_y = y + h // 2

    # calculate the new vertices
```

```
    x_new = mid_x - int(w // 2 * scale_factor)
    y_new = mid_y - int(h // 2 * scale_factor)
    w_new = int(w * scale_factor)
    h_new = int(h * scale_factor)

    # validate the new vertices
    left_x = max(0, x_new)
    left_y = max(0, y_new)
    right_x = min(columns, x_new + w_new)
    right_y = min(rows, y_new + h_new)

    # crop and resize the facial area
    cropped = image[left_y:right_y, left_x:right_x, :]
    resized = cv.resize(cropped, dsize=target_size, interpolation=cv.
        INTER_LINEAR)

    return resized

def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img = cv.imread(os.path.join(folder,filename))
        if img is not None:
            images.append(img)
    images = np.array(images)
    return images

def img_to_encoding(image_path, model):
    img1 = cv.imread(image_path, 1)
    img1 = cv.resize(img1, (600,600))
    img = img1[...,::-1]
    img = np.around(img/255.0, decimals=12)
    #img = np.around(img/255.0)
    x_train = np.array([img])
    embedding = model.predict_on_batch(x_train)
    return embedding

```

```
def get_data(path):
    data = {}
    for files in os.listdir(path):
        keys,_ = files.split('_')
        if keys in data:
            data[keys].append(files)
        else:
            data[keys] = [files]
    return data

def train_test_split(data, ratio = 0.2):
    train = {}
    test = {}
    for key in data.keys():
        vals = data[key]
        split = int(len(vals)*ratio)
        train[key] = vals[split:]
        test[key] = vals[:split]
    return train,test

def get_data_label(path,ratio = 0.2):
    """
    Given path returns tran and test images and label associated with it
    """
    _data = get_data(path)
    _train,_test = train_test_split(_data, ratio = ratio)

    train_image = []
    train_labels = []
    test_image = []
    test_labels = []
    for keys, vals in _train.items():
        train_image += [np.asarray([cv.imread(os.path.join(path,files))/255.
            \
                        for files in vals])]
        train_labels += [keys]

```

```
    for keys, vals in _test.items():
        test_image += [np.asarray([cv.imread(os.path.join(path,files))/255. \
                    for files in vals])]
        test_labels += [keys]
    return np.asarray(train_image), np.asarray(train_labels), \
np.asarray(test_image), np.asarray(np.asarray(test_labels)),_test

def DrawPics(tensor,nb=0,template='{}',classnumber=None):
    if (nb==0):
        N = tensor.shape[0]
    else:
        N = min(nb,tensor.shape[0])
    fig=plt.figure(figsize=(16,2))
    nbligne = floor(N/20)+1
    for m in range(N):
        subplot = fig.add_subplot(nbligne,min(N,20),m+1)
        axis("off")
        plt.imshow(tensor[m,:,:,0],vmin=0, vmax=1,cmap='Greys')
        if (classnumber!=None):
            subplot.title.set_text((template.format(classnumber)))

excluded_list = fold[10]
# excluded_list = []
import itertools
imgNames = []
imgList = {}
nb_classes = 124
for p in ["a", "b", "c", "d"]:
  for i in range(1,nb_classes+1):
    if i in excluded_list:
      continue
    name = f"{i}_{p}"
    imgNames.append(name)
    imgList[name] = cv.imread(f"img_digitize/{name}.png",cv.IMREAD_GRAYSCALE)

c = 0

```

```
img_width=600

for phone in ["a", "b", "c", "d"]:
  for i in range(1, nb_classes+1):

    if i in excluded_list: continue
    name = f"{i}_{p}"
    im = imgList[name]
    # try:
    im = cv.resize(im, (img_width,img_width), interpolation = cv.INTER_AREA)
    cv.imwrite(f"DB1_A/{i}_{phone}.png", im)
    c += 1
    # except: excluded_list.append(i)
nb_classes -= len(excluded_list)
imgList = {}
print("total img data: ", c)

datapath = './DB1_A/'
x_train,y_train,x_test,y_test,testfiles = get_data_label(datapath,ratio =
    0.0)
x_train = np.asarray(x_train)
type(x_train[0])

class TripletLossLayer(Layer):
    def __init__(self, alpha, **kwargs):
        self.alpha = alpha
        super(TripletLossLayer, self).__init__(**kwargs)

    def triplet_loss(self, inputs):
        anchor, positive, negative = inputs
        p_dist = K.sum(K.square(anchor-positive), axis=-1)
        n_dist = K.sum(K.square(anchor-negative), axis=-1)
        return K.sum(K.maximum(p_dist - n_dist + self.alpha, 0), axis=0)

    def call(self, inputs):
        loss = self.triplet_loss(inputs)
        self.add_loss(loss)
```

```
        return loss

def build_model(input_shape, network, margin=0.2):
    '''
    Define the Keras Model for training
        Input :
            input_shape : shape of input images
            network : Neural network to train outputing embeddings
            margin : minimal distance between Anchor-Positive and Anchor-
                Negative for the lossfunction (alpha)

    '''
     # Define the tensors for the three input images
    anchor_input = Input(input_shape, name="anchor_input")
    positive_input = Input(input_shape, name="positive_input")
    negative_input = Input(input_shape, name="negative_input")

    # Generate the encodings (feature vectors) for the three images
    encoded_a = network(anchor_input)
    encoded_p = network(positive_input)
    encoded_n = network(negative_input)

    #TripletLoss Layer
    loss_layer = TripletLossLayer(alpha=margin,name='triplet_loss_layer')([
        encoded_a,encoded_p,encoded_n])

    # Connect the inputs with the outputs
    network_train = Model(inputs=[anchor_input,positive_input,negative_input
        ],outputs=loss_layer)

    # return the model
    return network_train


def fingerRecoModel(input_shape,embeddingsize):
    X_input = Input(input_shape)
    # base = keras.applications.inception_resnet_v2.InceptionResNetV2(weights
```

```
        ='imagenet', input_tensor = X_input,input_shape = input_shape,
        include_top=False)
    # base=keras.applications.mobilenet_v2.MobileNetV2(weights='imagenet',
        input_tensor = X_input,input_shape = input_shape,include_top=False)
    base=VGG16(weights='imagenet', input_tensor = X_input,input_shape =
        input_shape,include_top=False)
    #imports the VGG16 model and discards the last 1000 neuron layer.
    X=base.output
    X=GlobalAveragePooling2D()(X)
    # X=Dense(1024,activation='relu')(X) #we add dense layers so that the
        model can learn more complex functions and classify for better
        results.
    X=Dense(512,activation='relu')(X) #dense layer 2
    X=Dense(256,activation='relu')(X) #dense layer 3
    X=Dense(embeddingsize, name = 'dense_layer')(X)

    # L2 normalization
    X = Lambda(lambda x: K.l2_normalize(x,axis=1))(X)

    # Create model instance
    model = Model(inputs = X_input, outputs = X, name='FingerRecoModel')

    return model

    input_shape=(img_width, img_width, 3)
FRmodel = fingerRecoModel(input_shape=(img_width, img_width, 3),
    embeddingsize =128)
network_train = build_model(input_shape,FRmodel)
# optimizer = Adam(lr = 0.00006, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
    decay=0.0)
optimizer = Adam(learning_rate = 0.00006, beta_1=0.9, beta_2=0.999, epsilon
    =1e-08)
network_train.compile(loss=None,optimizer=optimizer)
network_train.summary()
plot_model(network_train,show_shapes=True, show_layer_names=True, to_file='
    02 model.png')
print(network_train.metrics_names)
```

```
n_iteration=0

def get_batch_random(batch_size,s="train"):
    """
    Create batch of APN triplets with a complete random strategy

    Arguments:
    batch_size -- integer

    Returns:
    triplets -- list containing 3 tensors A,P,N of shape (batch_size,w,h,c)
    """
    if s == 'train':
        X = x_train
    else:
        X = x_test

    m, w, h,c = X[0].shape


    # initialize result
    triplets=[np.zeros((batch_size,h, w,c)) for i in range(3)]

    for i in range(batch_size):
        #Pick one random class for anchor
        anchor_class = np.random.randint(0, nb_classes)
        nb_sample_available_for_class_AP = X[anchor_class].shape[0]

        #Pick two different random pics for this class => A and P
        [idx_A,idx_P] = np.random.choice(nb_sample_available_for_class_AP,
            size=2,replace=False)

        #Pick another class for N, different from anchor_class
        negative_class = (anchor_class + np.random.randint(1,nb_classes)) %
            nb_classes
        nb_sample_available_for_class_N = X[negative_class].shape[0]

```

```
        #Pick a random pic for this negative class => N
        idx_N = np.random.randint(0, nb_sample_available_for_class_N)

        triplets[0][i,:,:,:] = X[anchor_class][idx_A,:,:,:]
        triplets[1][i,:,:,:] = X[anchor_class][idx_P,:,:,:]
        triplets[2][i,:,:,:] = X[negative_class][idx_N,:,:,:]

    return triplets

def drawTriplets(tripletbatch, nbmax=None):
    """display the three images for each triplets in the batch
    """
    labels = ["Anchor", "Positive", "Negative"]

    if (nbmax==None):
        nbrows = tripletbatch[0].shape[0]
    else:
        nbrows = min(nbmax,tripletbatch[0].shape[0])

    for row in range(nbrows):
        fig=plt.figure(figsize=(16,2))

        for i in range(3):
            subplot = fig.add_subplot(1,3,i+1)
            axis("off")
            plt.imshow(tripletbatch[i][row,:,:,0],vmin=0, vmax=1,cmap='Greys')
            subplot.title.set_text(labels[i])
def compute_dist(a,b):
    return np.sum(np.square(a-b))

def get_batch_hard(draw_batch_size,hard_batchs_size,norm_batchs_size,network
    ,s="train"):
    """
    Create batch of APN "hard" triplets

    Arguments:
    draw_batch_size -- integer : number of initial randomly taken samples
```

```
    hard_batchs_size -- interger : select the number of hardest samples to
        keep
    norm_batchs_size -- interger : number of random samples to add

    Returns:
    triplets -- list containing 3 tensors A,P,N of shape (hard_batchs_size+
        norm_batchs_size,w,h,c)
    """
    if s == 'train':
        X = x_train
    else:
        X = x_test

    m, w, h,c = X[0].shape


    #Step 1 : pick a random batch to study
    studybatch = get_batch_random(draw_batch_size,s)

    #Step 2 : compute the loss with current network : d(A,P)-d(A,N). The
        alpha parameter here is omited here since we want only to order them
    studybatchloss = np.zeros((draw_batch_size))

    #Compute embeddings for anchors, positive and negatives
    A = network.predict(studybatch[0])
    P = network.predict(studybatch[1])
    N = network.predict(studybatch[2])

    #Compute d(A,P)-d(A,N)
    studybatchloss = np.sum(np.square(A-P),axis=1) - np.sum(np.square(A-N),
        axis=1)

    #Sort by distance (high distance first) and take the
    selection = np.argsort(studybatchloss)[::-1][:hard_batchs_size]

    #Draw other random samples from the batch
    selection2 = np.random.choice(np.delete(np.arange(draw_batch_size),
```

```
        selection),norm_batchs_size,replace=False)

    selection = np.append(selection,selection2)

    triplets = [studybatch[0][selection,:,:,:], studybatch[1][selection
        ,:,:,:], studybatch[2][selection,:,:,:]]

    return triplets

def compute_probs(network,X,Y):
    '''
    Input
        network : current NN to compute embeddings
        X : tensor of shape (m,w,h,3) containing pics to evaluate
        Y : tensor of shape (m,) containing true class

    Returns
        probs : array of shape (m,m) containing distances

    '''
    m = X.shape[0]
    nbevaluation = int(m*(m-1)/2)
    probs = np.zeros((nbevaluation))
    y = np.zeros((nbevaluation))

    #Compute all embeddings for all pics with current network
    embeddings = network.predict(X)

    size_embedding = embeddings.shape[1]

    #For each pics of our dataset
    k = 0
    for i in range(m):
            #Against all other images
            for j in range(i+1,m):
                #compute the probability of being the right decision : it
                    should be 1 for right class, 0 for all other classes
```

```
            probs[k] = -compute_dist(embeddings[i,:],embeddings[j,:])
            if (Y[i]==Y[j]):
                y[k] = 1
                #print("{3}:{0} vs {1} : {2}\tSAME".format(i,j,probs[k],k))
            else:
                y[k] = 0
                #print("{3}:{0} vs {1} : \t\t\t{2}\tDIFF".format(i,j,probs[
                    k],k))
            k += 1
    return probs,y
#probs,yprobs = compute_probs(network,x_test_origin[:10,:,:,:],y_test_origin
    [:10])

def compute_metrics(probs,yprobs):
    '''
    Returns
        fpr : Increasing false positive rates such that element i is the
            false positive rate of predictions with score >= thresholds[i]
        tpr : Increasing true positive rates such that element i is the true
            positive rate of predictions with score >= thresholds[i].
        thresholds : Decreasing thresholds on the decision function used to
            compute fpr and tpr. thresholds[0] represents no instances being
            predicted and is arbitrarily set to max(y_score) + 1
        auc : Area Under the ROC Curve metric
    '''
    # calculate AUC
    auc = roc_auc_score(yprobs, probs)
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(yprobs, probs)

    return fpr, tpr, thresholds,auc

def compute_interdist(network):
    '''
    Computes sum of distances between all classes embeddings on our reference
        test image:
        d(0,1) + d(0,2) + ... + d(0,9) + d(1,2) + d(1,3) + ... d(8,9)
```

```
        A good model should have a large distance between all theses embeddings

    Returns:
        array of shape (nb_classes,nb_classes)
    '''
    res = np.zeros((nb_classes,nb_classes))

    ref_images = np.zeros((nb_classes,img_rows,img_cols,1))

    #generates embeddings for reference images
    for i in range(nb_classes):
        ref_images[i,:,:,:] = dataset_test[i][0,:,:,:]
    ref_embeddings = network.predict(ref_images)

    for i in range(nb_classes):
        for j in range(nb_classes):
            res[i,j] = dist(ref_embeddings[i],ref_embeddings[j])
    return res

def draw_interdist(network,n_iteration):
    interdist = compute_interdist(network)

    data = []
    for i in range(nb_classes):
        data.append(np.delete(interdist[i,:],[i]))

    fig, ax = plt.subplots()
    ax.set_title('Evaluating embeddings distance from each other after {0} iterations'.format(n_iteration))
    ax.set_ylim([0,3])
    plt.xlabel('Classes')
    plt.ylabel('Distance')
    ax.boxplot(data,showfliers=False,showbox=True)
    locs, labels = plt.xticks()
    plt.xticks(locs,np.arange(nb_classes))

```

```
    plt.show()

def find_nearest(array,value):
    idx = np.searchsorted(array, value, side="left")
    if idx > 0 and (idx == len(array) or math.fabs(value - array[idx-1]) <
        math.fabs(value - array[idx])):
        return array[idx-1],idx-1
    else:
        return array[idx],idx

def draw_roc(fpr, tpr,thresholds):
    #find threshold
    targetfpr=1e-3
    _, idx = find_nearest(fpr,targetfpr)
    threshold = thresholds[idx]
    recall = tpr[idx]


    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.')
    plt.title('AUC: {0:.3f}\nSensitivity : {2:.1%} @FPR={1:.0e}\nThreshold
        ={3})'.format(auc,targetfpr,recall,abs(threshold) ))
    # show the plot
    plt.show()

def DrawTestImage(network, images, refidx=0):
    '''
    Evaluate some pictures vs some samples in the test set
        image must be of shape(1,w,h,c)

    Returns
    scores: resultat des scores de similarites avec les images de base => N

    '''
    N=4
```

```
    _, w,h,c = x_test[0].shape
    nbimages=images.shape[0]

    #generates embedings for given images
    image_embedings = network.predict(images)

    #generates embedings for reference images
    ref_images = np.zeros((nb_classes,w,h,c))
    for i in range(nb_classes):
        ref_images[i,:,:,:] = x_test[i][refidx,:,:,:]
    ref_embedings = network.predict(ref_images)

    for i in range(nbimages):
        #Prepare the figure
        fig=plt.figure(figsize=(16,2))
        subplot = fig.add_subplot(1,nb_classes+1,1)
        axis("off")
        plotidx = 2

        #Draw this image
        plt.imshow(images[i,:,:,0],vmin=0, vmax=1,cmap='Greys')
        subplot.title.set_text("Test image")

        for ref in range(nb_classes):
            #Compute distance between this images and references
            dist = compute_dist(image_embedings[i,:],ref_embedings[ref,:])
            #Draw
            subplot = fig.add_subplot(1,nb_classes+1,plotidx)
            axis("off")
            plt.imshow(ref_images[ref,:,:,0],vmin=0, vmax=1,cmap='Greys')
            subplot.title.set_text(("Class {0}\n{1:.3e}".format(y_test[ref],
                dist)))
            plotidx += 1

# Hyper parameters
evaluate_every = 10 # interval for evaluating on one-shot tasks
batch_size = 24
```

```
hard_batch_size=10
rand_batch_size=2
n_iter = 800 # No. of training iterations prevous 300
#n_val = 250 # how many one-shot tasks to validate on

print("Starting training process!")
print("-------------------------------------")
t_start = time.time()
for i in range(1, n_iter+1):
    triplets = get_batch_hard(batch_size,hard_batch_size,rand_batch_size,
        FRmodel)
    loss = network_train.train_on_batch(triplets, None)
    n_iteration += 1
    if i % evaluate_every == 0:
        print("\n ------------- \n")
        print("[{3}] Time for {0} iterations: {1:.1f} mins, Train Loss: {2}".
            format(i, (time.time()-t_start)/60.0,loss,n_iteration))

network_train.save_weights("/content/drive/MyDrive/MINE_MELON/src/3Triplet/
    fold10_10.h5")

trained_weights =[]
import os
path = list(os.walk("/content/drive/MyDrive/MINE_MELON/src/3Triplet/"))[0]
for file in path[2]:
  if ".h5" in file:
    trained_weights.append(os.path.join(path[0],file))
trained_weights

from sklearn.preprocessing import StandardScaler

import random
from sklearn.decomposition import PCA
import matplotlib.cm as cm
import io

db = []
```

```
s = 12
st = 57
target = []
randomSample = random.sample(excluded_list,len(excluded_list));
# randomSample = random.sample(range(st, st+s), s);
for p in ['a','b','c','d']:
  for i in randomSample:
      try:
        name = f"{i}_{p}"
        im = cv.imread(f"img_digitize/{name}.png")
        im = cv.resize(im, (600,600),interpolation = cv.INTER_AREA)
        db.append(im)
        target.append(i)
      except: continue

dbe = FRmodel.predict(np.array(db))
#save projector.tensorflow
np.savetxt("vecs.tsv", dbe, delimiter='\t')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')
for i in target:
    out_m.write(str(i) + "\n")
out_m.close()

pca = PCA(n_components=2)
xs = StandardScaler().fit_transform(dbe)
pc = pca.fit_transform(xs)
colors = cm.rainbow(np.linspace(0, 1, s))
fig = figure(figsize=(8, 6), dpi=80)
# ax = fig.add_subplot(projection='3d')
ax = fig.add_subplot()

for i in range(len(dbe)):
  # ax.scatter(pc[i][0], pc[i][1], pc[i][2], color=colors[i % s])
  ax.scatter(pc[i][0], pc[i][1], color=colors[(i) % s])
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.95, box.height])
ax.set_xlabel("1st pca", fontsize=14)
```

```
ax.set_ylabel("2nd pca", fontsize=14)
# ax.set_zlabel("3rd pca", fontsize=14)

  # Put a legend to the right of the current axis
# ax.legend(range(st,st+s),loc='center left', bbox_to_anchor=(1, 0.5))
ax.legend(excluded_list,loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()


from sklearn.manifold import TSNE
import random
tsne = TSNE(n_components=2, verbose=1, random_state=0,perplexity=5,n_iter
    =1000)
pc = tsne.fit_transform(dbe)
colors = list(cm.rainbow(np.linspace(0, 1, s)))
# random.shuffle(colors)
fig = figure(figsize=(8, 6), dpi=80)
ax = fig.add_subplot()
# ax = fig.add_subplot(projection='3d')
mean = np.mean(pc)
standard_deviation = np.std(pc)

distance_from_mean = abs(pc - mean)
max_deviations = 2
for i in range(len(dbe)):
  distance_from_mean = abs(pc[i] - mean)
  # remove outlier
  if np.any(distance_from_mean > max_deviations * standard_deviation):
    continue
  # ax.scatter(pc[i][0], pc[i][1], pc[i][2], color=colors[i % s])
  ax.scatter(pc[i][0], pc[i][1], color=colors[i % s])#<<<<
  # ax.scatter(pc[i][0], pc[i][1], color=colors[10])

# ax.legend( range(st,st+s),loc='center left', bbox_to_anchor=(1, 0.5))
ax.legend(excluded_list,loc='center left', bbox_to_anchor=(1, 0.5))
ax.set_xlabel("1st TSNE", fontsize=14)
```

```
ax.set_ylabel("2nd TSNE", fontsize=14)
# ax.set_zlabel("3rd TSNE", fontsize=14)
plt.show()

# use excluded_list as database
database = {}
for i in excluded_list:
  database[i] = img_to_encoding(f"img_digitize/{i}_b.png",FRmodel)

res = {}
c = 0
FR = 0
CA = 0
FA = 0
IDE = 0
for p in ['a','c','d']:
  # for i in range(Start,Stop+1):
  for i in excluded_list:
    match = False
    name = f"{i}_{p}"
    test_dir = f"img_digitize/{name}.png"
    encoding = img_to_encoding(test_dir, FRmodel)
    temp_dist = 10
    # for j in range(Start,Stop+1):
    for j in excluded_list:
      dist = np.linalg.norm(encoding-database[j])
      c += 1
      if dist <= temp_dist:
        temp_dist = dist
        pre_iden = j
    if temp_dist > 0.4:
      FR +=1
      if pre_iden != i:
        IDE +=1
      else:
        match = True
    elif temp_dist <= 0.4 and pre_iden == i:
```

```
905         match = True
906         CA +=1
907       elif temp_dist <= 0.4 and pre_iden != i:
908         IDE +=1
909         FA +=1
910       # print(f"{i}_{p}: {match},{pre_iden},{temp_dist}")
911       res[f"{i}_{p}"] = [match,pre_iden,temp_dist]
912   print(c)
913
914   import pandas as pd
915   df=pd.DataFrame.from_dict(res).T
916   TP = len(df.loc[df[0]==True])
917   FP = len(df.loc[df[0]==False])
918   FN = FP
919   TN = c - (TP+FP+FN)
920   # print(f'FP: {FP}, FN: {FN}, TP: {TP}, TN: {TN}, Total: {c}')
921   acc = (TP + TN)/(TP+FP+TN+FN)*100
922   pre = TP/(TP+FP)*100
923   rec = TP/(TP+FN)*100
924   # print(f'Accuracy: %.2f%% Precision: %.2f%% Recall: %.2f%% ' %(acc,pre,rec)
        )
925   FAR = (FP/(c))*100
926   FRR = (FN/(c))*100
927   # print("FAR = %.3f%% FRR = %.3f%%" %(FAR,FRR))
928
929   CR = c-(FR+CA+FA)
930   print(f"Total Checks = {(CR+CA+FR+FA)}")
931   print(f"Correct Acceptance = {CA} Correct Rejection = {CR}")
932   print(f"False Acceptance = {FA} False Rejection = {FR}")
933   print("FAR = %.2f%% FRR = %.2f%%" %((FA/c)*100,(FR/c)*100))
934   print(f"Top-Rank ID Error = %.2f%%" %((IDE/c)*100))
```