UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
BACHELOR'S DEGREE

Research and Development

BACHELOR THESIS

by

NGO The Dzung (BA10-010)

Information and Communication Technology

Title:

Face Verification on Edge Device

Supervisors: Dr. GIANG Anh Tuan

Lab name: USTH ICTLab

Hanoi, 2021

DECLARATION

NGO The Dzung, hereby declare that my thesis does not contain plagiarism from any other source. I certified that the work and the results are entirely of my own.

In case of plagiarism, I acknowledge the consequences and I accept that my thesis will not be evaluated. In this case, my bachelor’s thesis will be marked as "Failed".

August 25th, 2023

NGO The Dzung

ACKNOWLEDGMENTS

First and foremost, I want to extend my gratitude to Dr. GIANG Tuan Anh for helping me on this project, for being my supervisor and for the invaluable guidance he has provided throughout my internship.

I also wish to convey my deepest thanks to my family for their encouragement from the very beginning of my journey at USTH.

August 25th, 2023

NGO The Dzung

LIST OF ACRONYMS

CNNs Convolution Neural Network

SSD Single Shot MultiBox Detector

YOLO You Only Look Once

GPU Graphics Processing Unit

TFLite TensorFlowLite

UI User Interface

FLOPs Floating-point operations per second

SPP Spatial Pyramid Pooling

CSP Cross-Stage Pyramid Network

PANet Path Aggregation Network

SiLU Sigmoid Linear Unit

ReLU Rectified Linear Unit

GDConv Global Depthwise Convolution

SGD Stochastic Gradient Descent

BCE Binary Cross Entropy

mAP mean Average Precision

LFW Labelled Face in Wild

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Beside conventional security methods, facial biometrics are increasingly being used to enhance user information security. Facial recognition not only enhances security but also facilitates more convenient, cost-effective, and quicker human resource management compared to conventional methods. This project aims to create a real-time facial verification model using Deep Learning techniques on data comprising human facial images. Furthermore, the intention is to embed this model on mobile devices, allowing the application to operate without an internet connection. The core of this product consists of two Deep Learning models: The first model processes each real-time frame to detect the presence of a human face, and the detected face is forwarded to the second model to determine its similarity to stored facial information. Each model uses distinct data labeling and is evaluated using different datasets and metrics. Finally, the product undergoes testing in a real-world environment.

Keywords: *facial biometrics, real-time, facial verification, Deep Learning, embed, mobile devices,*

I. INTRODUCTION

1.1. Motivation

Nowadays, we rely on smartphones and other devices for a wide range of tasks, from securing our personal information to simplifying our daily routines. However, many of these devices require an internet connection to perform tasks like facial recognition.

Imagine a scenario where you want your smartphone to recognize your face to unlock it or to access sensitive information, but you're in a place with no internet access. This is where embedded face verification on edge devices becomes crucial. It means that your device can recognize your face and provide access without internet connection.

This approach uses advanced techniques like deep learning to process facial features directly on your device. It's not only convenient but also enhances security by ensuring that sensitive facial data stays on your device, rather than being sent over the internet. So, even in remote areas or during network outages, your device can still verify you, providing a seamless and secure user experience.

Furthermore, from a manager’s perspective, using a face verification tool directly on phones offers numerous advantages. For example, let's consider a company that needs to calculate employee salaries based on their working hours. In the traditional method, they would have to purchase an external machine for fingerprint scanning. This approach not only incurs additional expenses, time but is also less accurate. Face recognition on mobile devices can provide a cost-effective solution for manager.

1.2. Background

This section provides an overview of all the technologies used in this project, serving as the background for all the methods and results presented. Face Detection and Face Verification are the two primary tasks in this project. The Face Detection model is responsible for determining whether a face appears in front of the camera. Once a face is detected, it is then processed through the Face Verification model to confirm the person's identity as valid or not.

Face detection has a rich history dating back to the 1960s when researchers first began exploring the concept. Early methods relied on simple heuristics and templates to identify faces, often with limited success. However, significant advancements came in the 2000s with the introduction of more sophisticated techniques like Viola-Jones [1]. These methods laid the groundwork for modern face detection algorithms. Today, face detection has reached remarkable levels of accuracy and speed, thanks to deep learning techniques like Convolutional Neural Networks (CNNs). CNN-based models, such as Single Shot MultiBox Detector (SSD) [2] and You Only Look Once (YOLO) [3], have revolutionized face detection. These models can detect faces in real-time and under various conditions, including variations in lighting, poses, and facial expressions.

Face verification is a subset of facial recognition technology that focuses on confirming whether two facial images or patterns belong to the same individual. It has evolved significantly over time, moving from traditional methods (Face Recognition Using Eigenface [4]) to cutting-edge techniques (Tripet Loss [5] or FaceNet [6]). Today, face verification has seen a remarkable transformation, largely due to the advent of Deep Learning and Convolutional Neural Networks (CNNs). Modern face verification systems utilize Deep Learning algorithms to extract and compare facial features. These algorithms can handle a wide range of variations and are exceptionally accurate. Contemporary face verification systems offer remarkable levels of precision and speed. They are utilized in various applications, including access control, identity verification for financial transactions, secure document access, and even unlocking mobile devices. These systems have gained widespread acceptance and have become a cornerstone of modern security and authentication practices.

1.3. Technology frameworks

PyTorch Lightning is an open-source Python library built on top of PyTorch, designed to simplify and streamline the process of training complex deep learning models. It provides a high-level interface and a set of abstractions that allow researchers and engineers to focus on their model's core logic and experiments rather than the details of training loops and GPU management.

A TFLite model, short for TensorFlow Lite model, is a compact and optimized version of a machine learning model designed specifically for mobile and embedded devices. TensorFlow Lite is an open-source framework developed by Google that enables efficient deployment of Machine Learning models on resource-constrained platforms.

Flutter is an open-source UI software development toolkit created by Google. It allows developers to build natively compiled applications for mobile, web, and desktop from a single codebase. Flutter is known for its "write once, run anywhere" philosophy, making it a popular choice for cross-platform app development. It offers a rich set of customizable widgets, a fast development cycle, and high-performance rendering, resulting in visually appealing and responsive user interfaces. Flutter uses the Dart programming language and has gained widespread adoption for creating visually stunning and fast mobile applications.

Swift is a powerful and user-friendly programming language developed by Apple for building iOS, macOS, watchOS, and tvOS applications. It was introduced as a replacement for Objective-C and has quickly become the preferred language for iOS app development. Swift is known for its clean syntax, safety features, and performance improvements, which make app development more efficient and less error-prone. It has a growing community of developers, extensive libraries, and support from Apple, making it an excellent choice for creating native applications on Apple platforms.

II. OBJECTIVE

The primary objective of this thesis is to design, train, and deploy a highly efficient face verification model, specifically optimized for edge computing environments. This encompasses the development of a robust deep learning-based face verification system that can accurately authenticate individuals in real-time while operating seamlessly on resource-constrained edge devices. The project aims to bridge the gap between state-of-the-art face verification techniques and practical deployment on edge hardware, ultimately contributing to advancements in the fields of edge computing and computer vision.

III. MATERIAL AND METHODS

3.1. Face Detection

3.1.1. Data Acquisitions

The research utilized the CelebA dataset, generously made available by The Chinese University of Hong Kong. This extensive dataset comprises over 200,000 celebrity face images, annotated with attributes such as gender, age, and facial landmarks. Its diversity and rich annotations make it an excellent choice for training and evaluating face detection models.

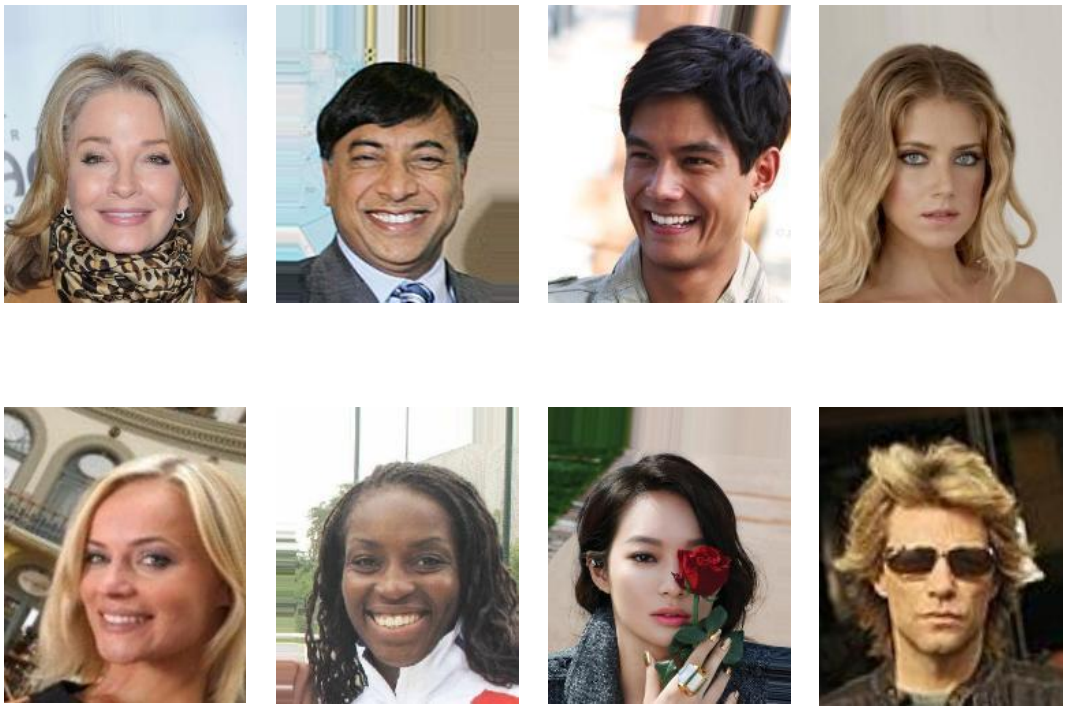


Figure 1: CelebA dataset overview

3.1.2. Data Preprocessing

Unfortunately, most of the annotations were incorrect; therefore, I had to annotate the dataset from scratch. Each image was processed by using a pretrained model from OpenCV to obtain the bounding boxes of faces until they reach 30k images for the new dataset. Afterward, coordinates were converted to YOLO format and organized into

folders, maintaining a training-to-validation ratio of 5:1. All images will be resized to 640x640 before feeding in the model.

Figure 2: YOLO label format

```
0  0.481719 0.634028 0.690625 0.713278
0  0.741094 0.524306 0.314750 0.933389
27 0.364844 0.795833 0.078125 0.400000
```

Figure 3: How to store coordinate in annotation file (NO class, x, y, w, h)

Dataset were applied below augmentation techniques:

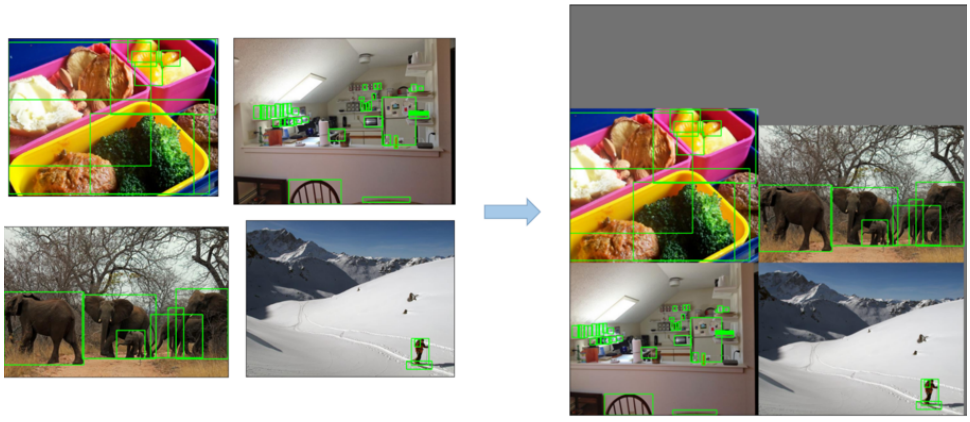


Figure 4: Mosaic Technique

Figure 5: Copy Paste technique

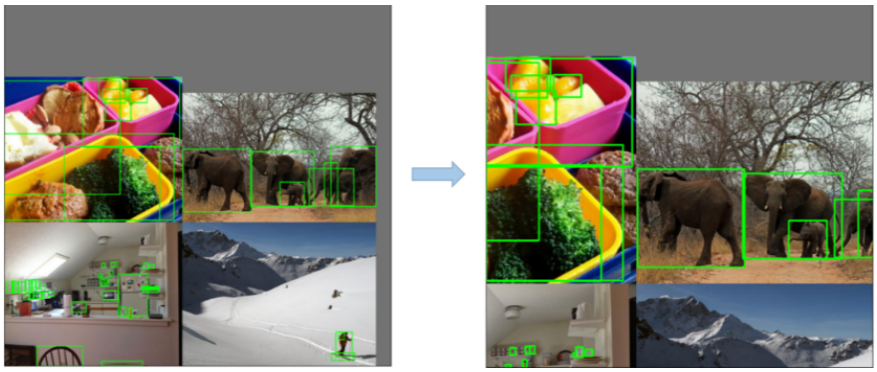


Figure 6: Random affine (Rotation, Scale, Translation and Shear)

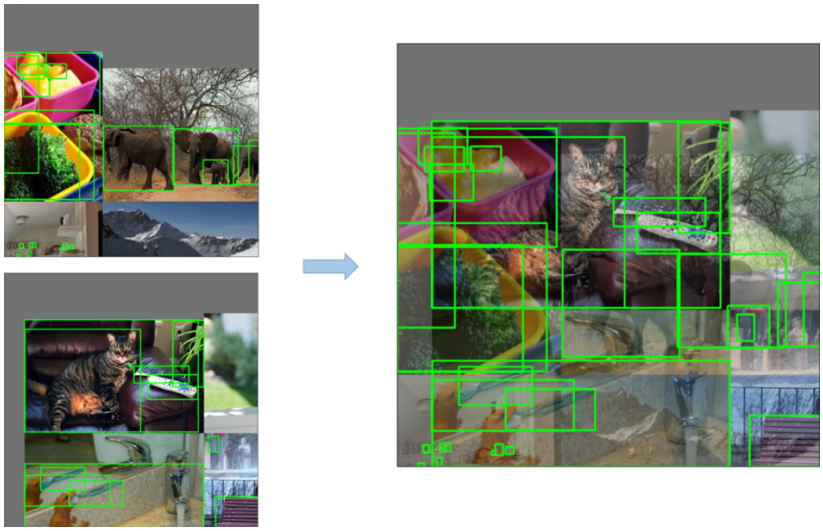


Figure 7: MixUp

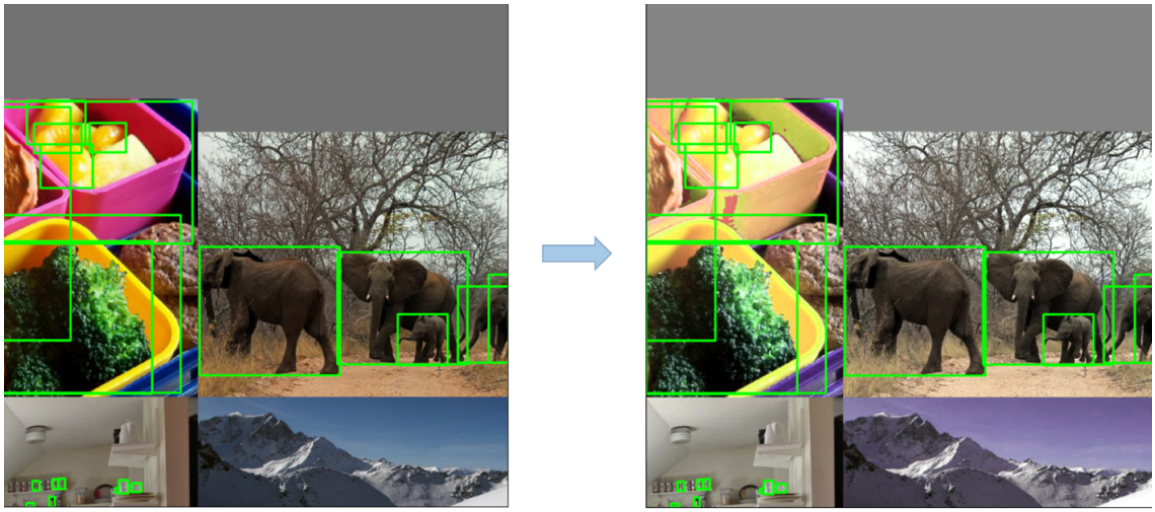


Figure 8: Augment HSV (Hue, Saturation, Value)

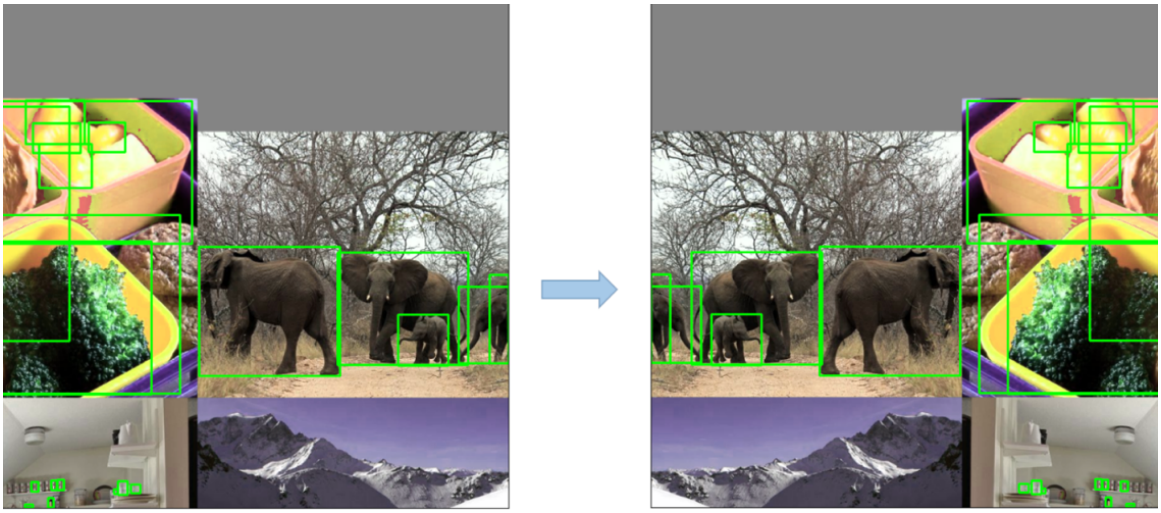


Figure 9: Horizontal flip

3.1.3. Model

The model for Face Detection was chosen based on four criteria: Performance, Availability (whether there is any public source code that can be easily modified), Lightweight (whether the model is light enough to be embedded in an edge device), and

High Speed (whether the model is fast enough to run in real-time). Below is overview of Object Detection benchmark:

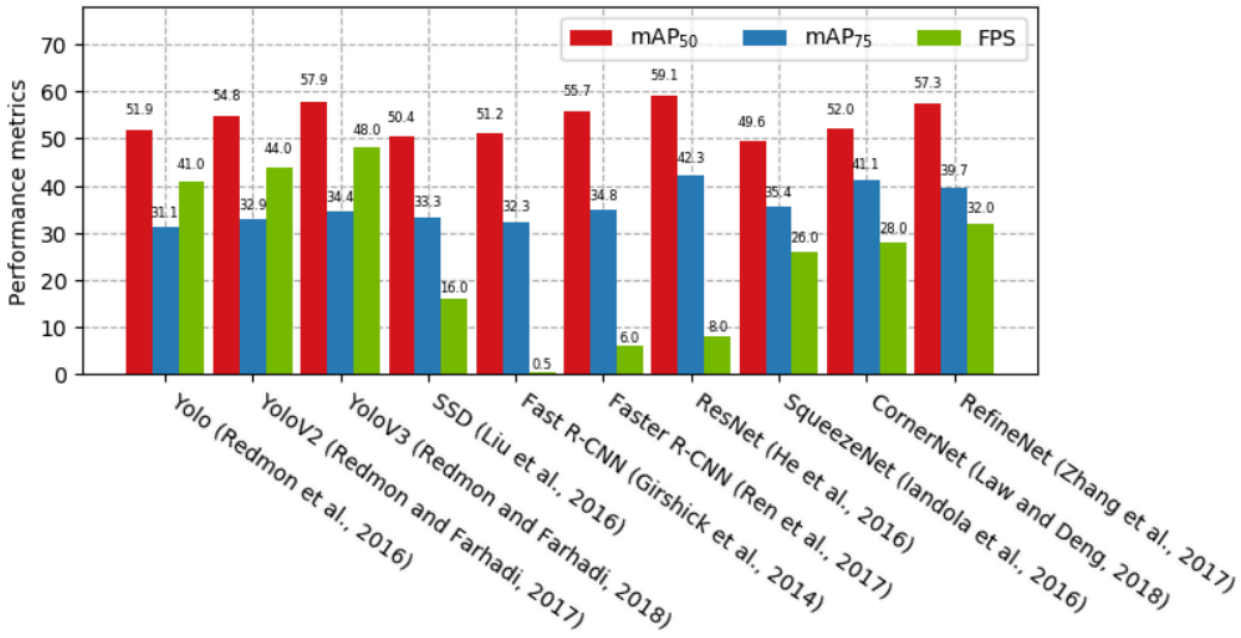


Figure 10: Pascal VOC 2012 Benchmarking [7]

All the models with a frame rate of 30 FPS or lower have been eliminated because the project requires real-time performance. Among the remaining options, the YOLO family dominates in terms of refresh rate, with YOLOv3 achieving the highest mAP_{50} and slightly lower mAP_{75} when compared to RefineNet [8]. Additionally, since there is no reliable public source code available for RefineNet, YOLO is the best choice. Now, let's determine which version of YOLO should be used.

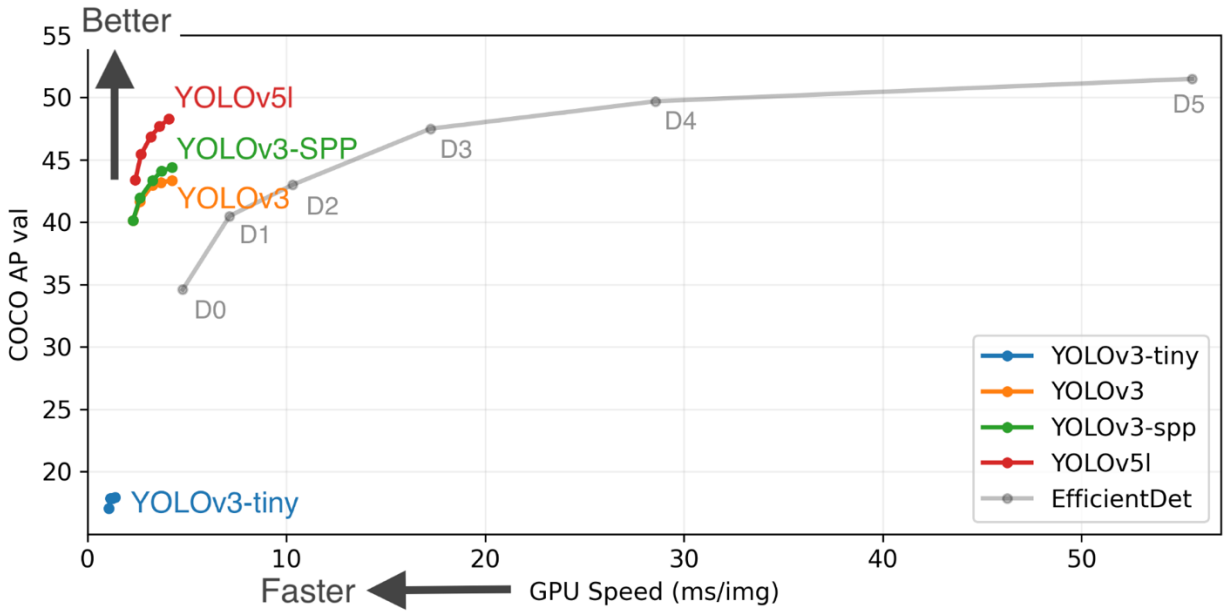


Figure 11: Comparison between YOLOv3 variants, YOLOv5l and EfficientDet from Ultralytics

Based on the above benchmark, YOLOv5l is both faster and more precise than YOLOv3[9]. However, let's take a look at below figure:

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Figure 12: Pretrained Checkpoints on COCO val2017 dataset on 300 epochs from Ultralytics

- n for nano (extra small) size model
- s for small size model
- m for medium size model
- l for large size model
- x for extra large size model

Although YOLOv5l's metric scores are better than YOLOv5n, the number of parameters and FLOPs (Floating Point Operations Per Second) and speed of YOLOv5n are extremely higher than the large version. This is a trade-off between Accuracy and Speed. I will try the nano version first because it is the lightest version. If the performance is acceptable, we can use YOLOv5n for Face Detection task.

All the YOLOv5 models from Ultralytics are built from 3 components: CSP-Darknet53 [10] as a backbone for feature extraction, SPP [11] (in new version, SPP was replaced by SPPF - Spatial Pyramid Pooling Fast to speed up the process) and PANet [12] as a neck and the head is the same as YOLOv4 [13].

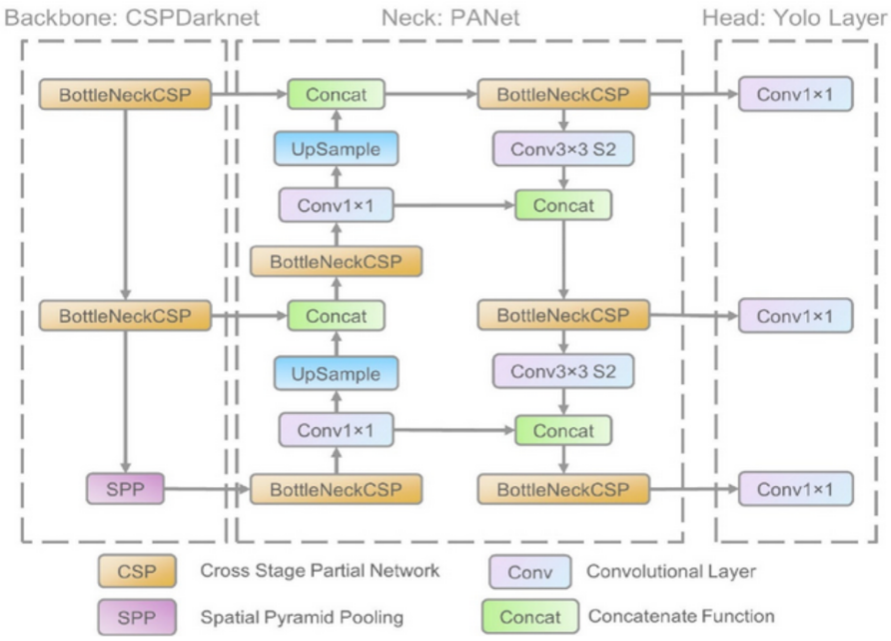
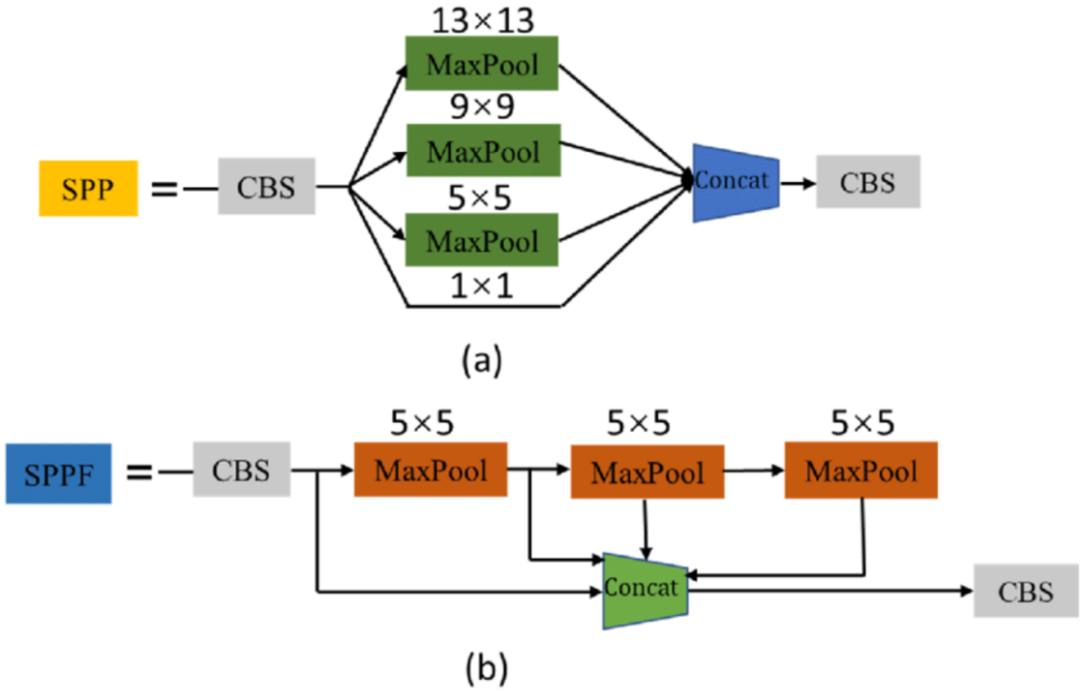


Figure 13: Simplified YOLOv5 Architecture [14]

"SPPF is an optimized version of SPP I created myself that is mathematically identical with less FLOPs" – according to Glenn Jocher who is one of the authors of YOLOv5 source code from Ultralytics.



(a) SPP structure diagram and (b) SPPF structure diagram.

Figure 14: Comparison between SPP and SPPF architecture [15]

```
[Running] python -u "/Users/dzungngo/Desktop/dsa_python/thesis.py"
Input Tensor Shape:  torch.Size([8, 32, 16, 16])
SPP time: 0.9967648983001709
SPPF time: 0.42037391662597656

[Done] exited with code=0 in 3.696 seconds
```

Figure 15: Comparison speed of SPP and SPPF

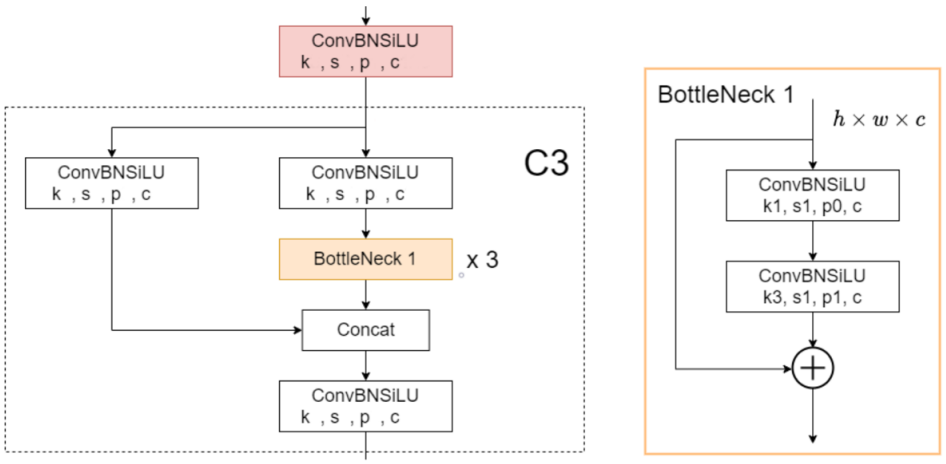


Figure 16: BottleNeck CSP module architecture from Ultralytics

YOLO is a deep network; as a consequence, it has to use residual blocks in order to enable the information to go to the deepest layers and to deal with the vanishing gradient problem. However, it leads to a new problem - redundant gradients. CSPNet solves this issue by truncating the gradient flow. According to the authors: "*The partial transition layer is a hierarchical feature fusion mechanism, which uses the strategy of truncating the gradient flow to prevent distinct layers from learning duplicate gradient information*" – p.5 [10]

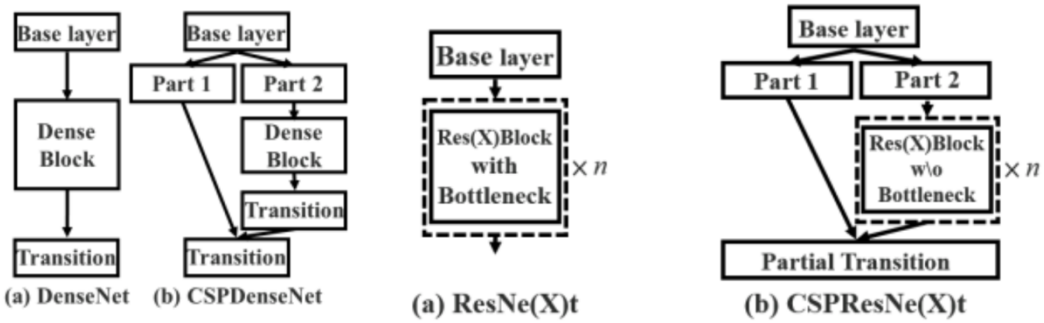


Figure 17: Apply CSPNet to Res(X)t and DenseNet [12]

In YOLOv5, the authors chose Sigmoid Linear Units (SiLU) as an activation function:

$$SiLU(x) = x\left(\frac{1}{1 + e^{-x}}\right) \tag{3.1}$$

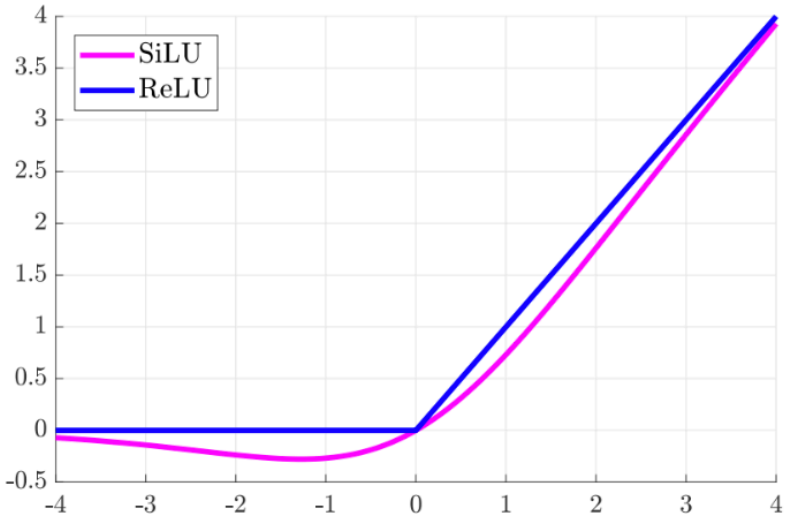


Figure 18: Compare SiLU and ReLU activation functions [16]

SiLU is a smooth and differentiable activation function, which means its gradient is defined everywhere. This property aids in the training process using backpropagation and gradient-based optimization methods like stochastic gradient descent (SGD). In contrast, ReLU has a non-differentiable point at zero, which can lead to convergence issues in some cases

3.1.4. Training

The model followed AutoAnchor strategy which is recommended for custom datasets. AutoAnchor is a technique used when training on custom datasets for object detection. It involves automatically determining anchor box priors (sizes and aspect ratios) based on the characteristics of the objects in the custom dataset. Custom anchor boxes are

calculated to better match the object size distribution in the dataset, improving detection accuracy.

The YOLOv5 loss consists of three components:

- **Classification Loss** (BCE Loss)
- **Objectness Loss** (BCE Loss)
- **Location Loss** (CIoU Loss)

$$Loss = \lambda_1 L_{cls} + \lambda_2 L_{obj} + \lambda_3 L_{loc} \tag{3.2}$$

Where:

- λ_1, λ_2, λ_3 are hyperparameters used to control the relative importance of different components in the total loss function. You can use them to adjust how much each component contributes to the overall loss during training.

- L_{cls}, L_{obj} can be calculated by Binary Cross Entropy Loss function:

$$L_{BCE} = -\sum_{i=1}^{N_{cls}} [p_i \log(\hat{p_i}) + (1 - p_i) \log(1 - \hat{p_i})] \tag{3.3}$$

- L_{loc} can be calculated by Complete IoU Loss Function:

$$L_{CIoU} = 1 - IoU + \frac{\rho^2(b,\ b^{gt})}{c^2} + \alpha v \tag{3.4}$$

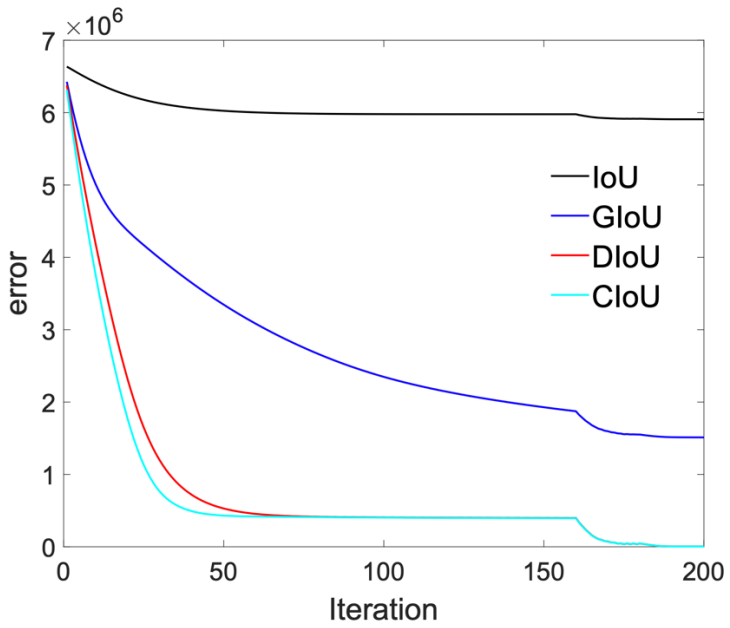


Figure 19: Compare between variants of IoU losses [17]

3.1.5. Validation

In the context of the object detection task, model performance will be evaluated using the mean Average Precision (mAP) metric. Additionally, for Face Verification, which demands high precision, the Precision metric will also be applied.

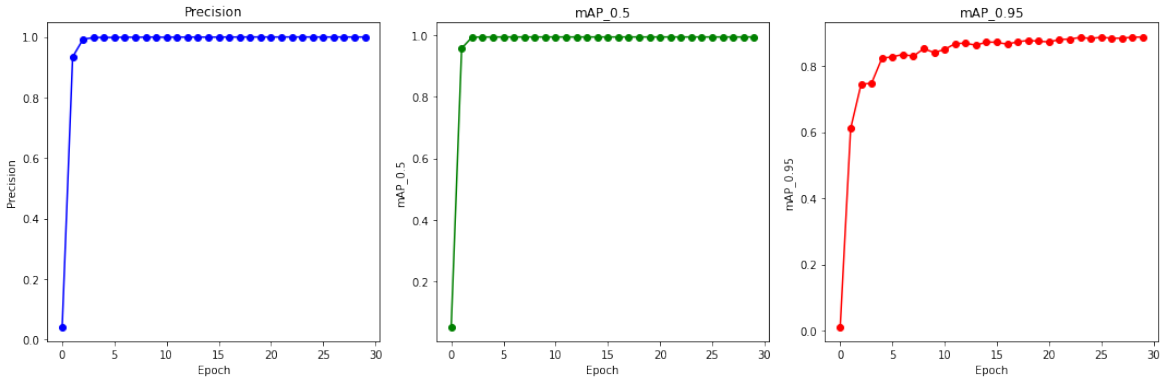


Figure 20: Face Detection Evaluation Scores

3.1.6. Result

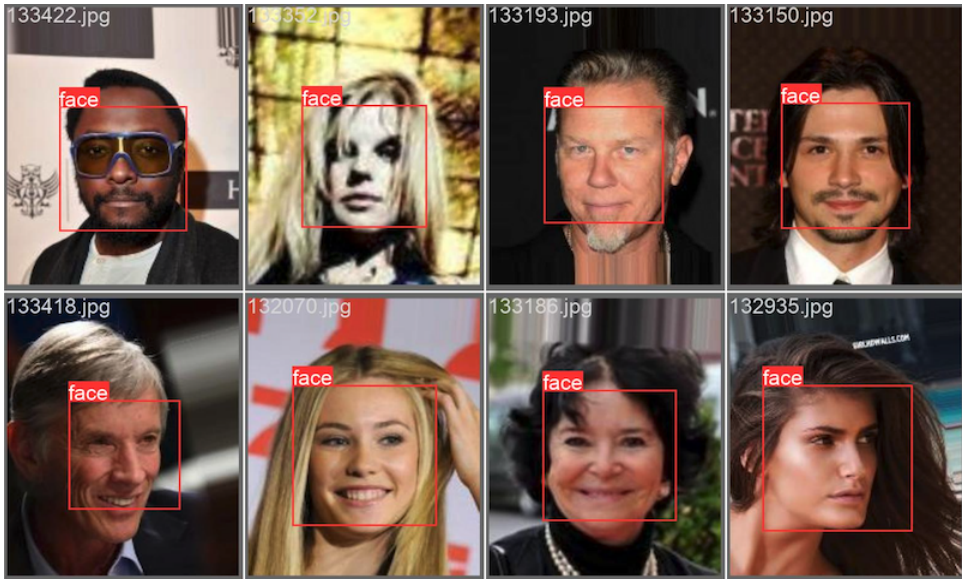


Figure 21: Result on server

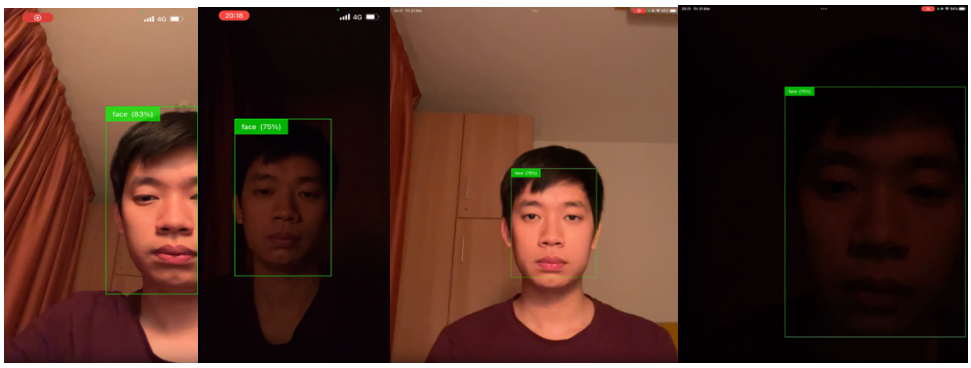


Figure 22: Result on Iphone 14 Pro Max and Ipad Pro in different light conditions

3.2. Face Verification

3.2.1. Data Acquisitions

I used two datasets for this model: MS1Mv3 (also called MS1M-RetinaFace), which consists of almost 90,000 identities for training, and LFW (deep funneled), which contains 5,000 identities for validation. Both dataset were normalized follow the paper [18] then cropped to (112 x 112).

Figure 23: MS1M-RetinaFace dataset overview

Figure 24: LFW dataset overview

3.2.2. Model

At the moment this thesis was written (15/09/2023), ArcFace [19] is top 1 ranking on paperswithcode website. It is a SOTA model and reaches 99.83% accuracy. As a result, ArcFace is the first candidate for Face Verification without any doubt. ArcFace is not a specific model, ArcFace actually is a Loss Function which is used to train a model how to embed an image into a vector. Instead of directly comparing two faces, the model will use two embedding vectors and calculate their distance to determine whether they belong to the same person or not. Because the model will be implemented on edge-devices which requires high performance, MobileFaceNet [20] was chosen to be a backbone.

Input	Operator	t	c	n	s
$112^2 \times 3$	Conv3x3	-	64	1	2
$56^2 \times 64$	Depthwise conv3x3	-	64	1	1
$56^2 \times 64$	Bottleneck	2	64	5	2
$28^2 \times 64$	Bottleneck	4	128	1	2
$14^2 \times 128$	Bottleneck	2	128	6	1
$14^2 \times 128$	Bottleneck	4	128	1	2
$7^2 \times 128$	Bottleneck	2	128	2	1
$7^2 \times 128$	Conv1x1	-	512	1	1
$7^2 \times 512$	Linear GDConv7x7	-	512	1	1
$1^2 \times 512$	Linear conv1x1	-	128	1	1

Table 1: MobileFaceNet Architecture [20]

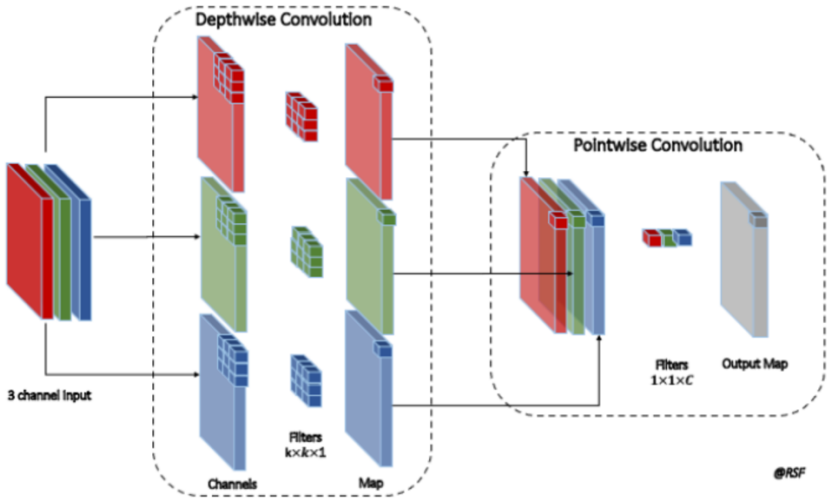


Figure 25: Depthwise separable convolution of the MobileNet [21]

According to the authors: A GDConv layer is a depthwise convolution layer with kernel size equaling the input size, pad = 0, and stride = 1. The output for global depthwise convolution layer is computed as:

$$G_m = \sum_{i,j} K_{i,j,m} \cdot F_{i,j,m} \tag{3.5}$$

where F is the input feature map of size $W \times H \times M$, K is the depthwise convolution kernel of size $W \times H \times M$, G is the output of size $1 \times 1 \times M$, the m^{th} channel in G has only one element G_m, (i,j) denotes the spatial position in F and K, and m denotes the channel index.

Global depthwise convolution has a computational cost of:

$$\mathrm{W} \cdot \mathrm{H} \cdot \mathrm{M} \tag{3.6}$$

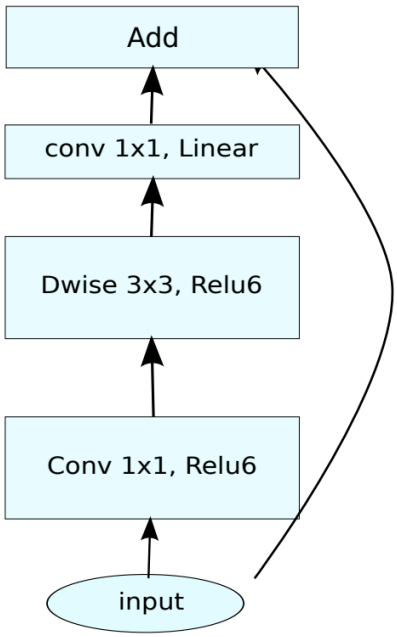


Figure 26: Bottleneck structure [22]

3.2.3. Training

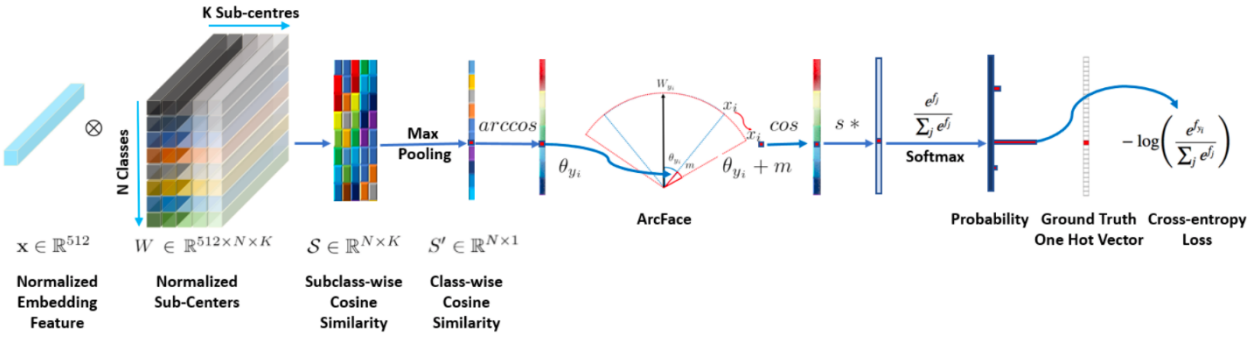


Figure 27: ArcFace Loss [19]

ArcFace Loss Formula:

$$L = -\frac{e^{s \cdot \cos(\theta_{y_i} + m)}}{e^{s \cdot \cos(\theta_{y_i} + m)} + \sum_{j=1, j \neq y_i}^{N} e^{s \cdot \cos(\theta_j)}} \tag{3.7}$$

Where:

- $\theta_j = \arccos(max_k(W_{j_k}^T)),\ k \in \{1, 2, \ldots, K\}$.

Arcface used Cosine Similarity to calculate the closeness between two face embedding vectors:

$$\cos(\theta) = \frac{A \cdot B}{||A||\,||B||} \tag{3.8}$$

Where:

- $A \cdot B$ represents the dot product of vectors A and B
- $||A||, ||B||$ are Euclidean norm of vectors A and B, calculated as:

$$||A|| = \sqrt{{A_1}^2 + {A_1}^2 + \ldots + {A_n}^2} \quad (3.9)$$

3.2.4. Validation

Since the output of the model is in the form of probabilities; therefore, choosing the AUROC metric is optimal because it eliminates the need to set a threshold.

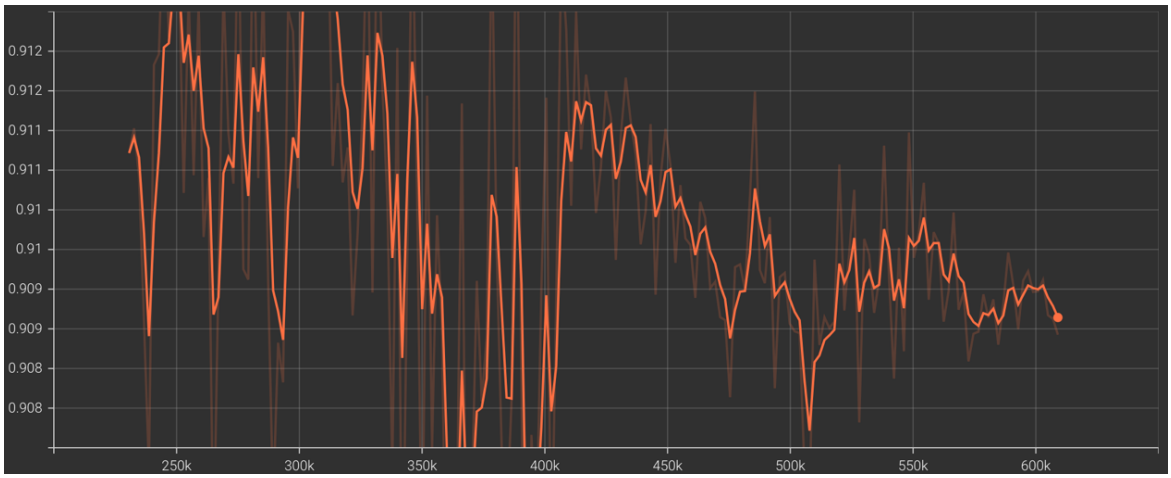


Figure 28: AUROC score

To choose threshold, I applied below strategy:

- Split validation dataset into 10 folds
- Test each threshold (0.1, 0.15, 0.2, … 1.0) on each fold
- Choose the best threshold on each fold by precision
- Final threshold is mean of threshold on each fold
- Test again on test set (1:5 ratio of LWF dataset)

3.2.5. Result

Figure 29: True Positive Results

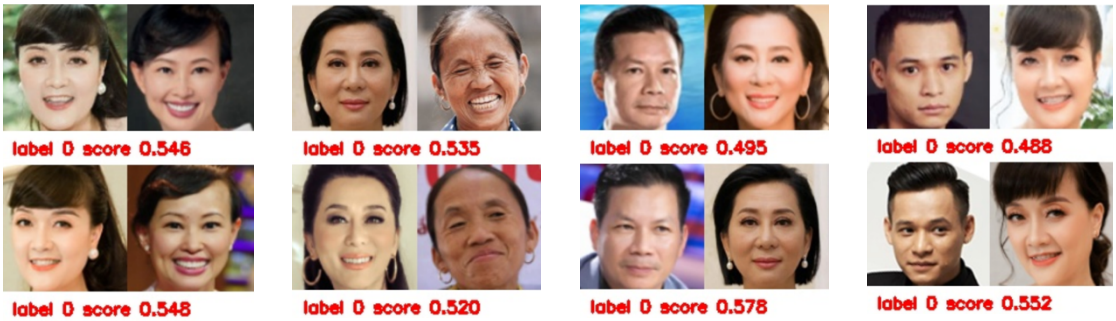


Figure 30: True Negative Results

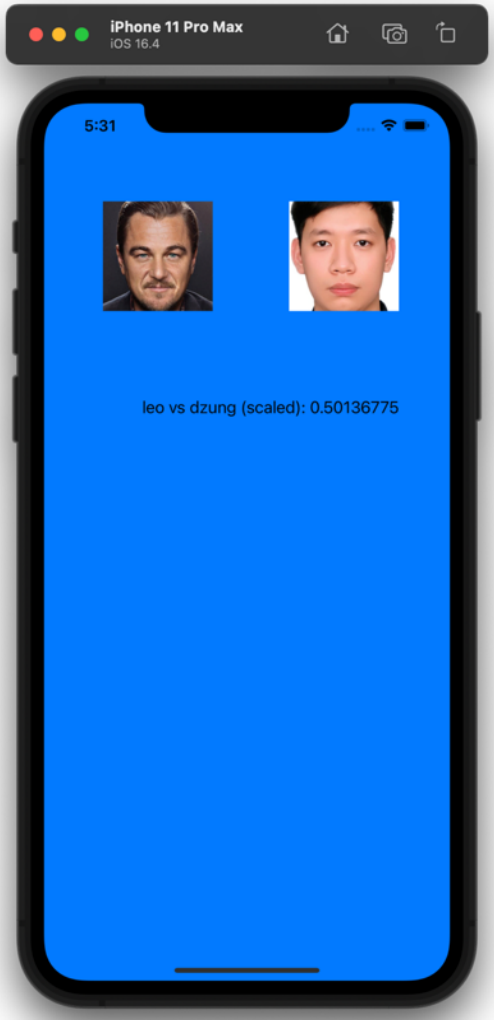


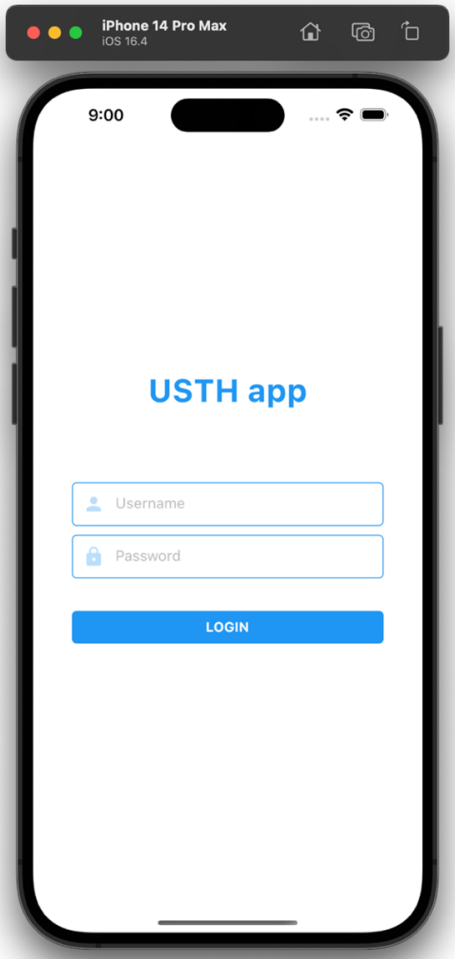
Figure 31: Result on Iphone 11 Pro Max Simulator

3.3. Mobile Development

3.3.1. User Interface

The models will be deployed on both iOS and Android devices. Therefore, Flutter is the best choice because we can build a UI for cross-platform development with only one source code. About the camera, it can not operate the camera with Flutter because it is relevant to AI models which must be handled by native platforms. As a result, Event

Channel and Method Channel APIs were used as a bridge between Flutter and Swift/Android code bases.



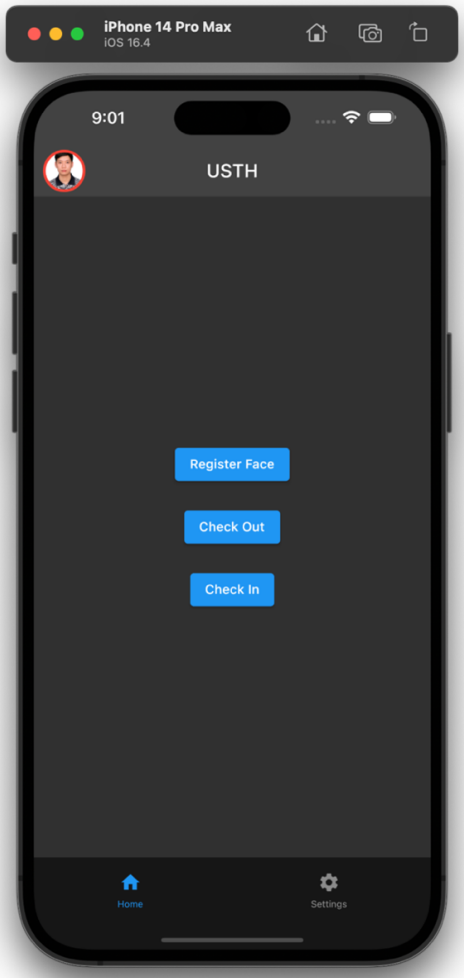


Figure 32: Login Screen

Figure 33: Home Screen

3.3.2. Model Embedding

As was said before, there wasn't a good enough package in Flutter (outdated, not for custom models). Native library is the only option to execute the model on the iOS environment. First of all, the model trained in Pytorch Lightning was converted to TensorFlowLite model. After that, that TFLite model called by TensorFlowLiteSwift.

IV. RESULT AND DISCUSSIONS

4.1. Results

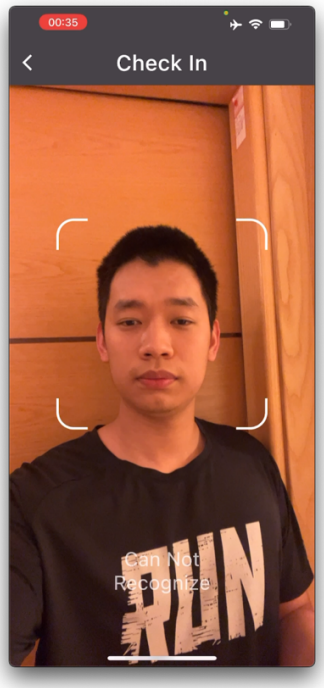


Figure 34: Can not recognize

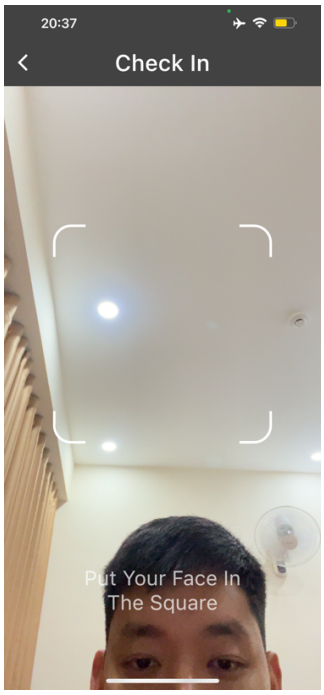


Figure 35: When face is detected but out of scanner square

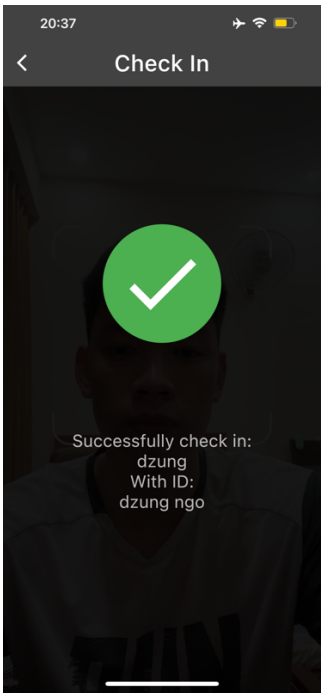


Figure 36: Verify Successfully

4.2. Discussion

Application can detect and verify faces in real-time. Therefore, there is no need to use a more advanced YOLO model than YOLOv5n. However, the application can be tricked by images of face as long as the image satisfies specific conditions (light, angle, blur, etc.) similar to those of real face.

V. CONCLUSION

5.1. Summary

In this project, I successfully collected data; I processed and fed them ready for training in Deep Learning models. The project is the combination of two tasks: Face Detection, once the face is detected, it will be gone through Face Verification model. Up to this point, the project has undergone successful testing on iOS and iPadOS devices. Even though the results are good, there are still some weaknesses to improve in the future.

5.2. Future works

As I stated above, here is a plan how the application will improved in the future:

First of all, the original Face Detection will be replaced by the Real Face Detection model. This method can eliminate the trick of using high quality images or video to verify. Secondly, this check-in feature can be applied to various applications for human resources management. Building an AI-integrated mobile application is one of the task I can do. In the near future, I plan to extend the project's compatibility to devices running the Android operating system as well. Last but not least, the application will be optimized for lighter and faster performance.

REFERENCE

[1]: Paul Viola and Michael Jones. “Rapid Object Detection using a Boosted Cascade of Simple Features”, 2001.

[2]: Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu and Alexander C. Berg. “SSD: Single Shot MultiBox Detector”, 2015.

[3]: Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”, 2015.

[4]: Matthew A.Turk and Alex P. Pentland. “Face Recognition Using EigenFace”, 1991.

[5]: Elad Hoffer, Nir Ailon. “Deep metric learning using Triplet network”, 2014.

[6]: Florian Schroff, Dmitry Kalenichenko, James Philbin. “FaceNet: A Unified Embedding for Face Recognition and Clustering”, 2015

[7]: Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias and Gigel Macesanu. “A Survey of Deep Learning Techniques for Autonomous Driving”, 2020

[8]: Guosheng Lin, Anton Milan, Chunhua Shen and Ian Reid. “RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation”

[9]: Joseph Redmon, Ali Farhadi. “YOLOv3: An Incremental Improvement”, 2018

[10]: Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen and Jun-Wei Hsieh. “CSPNet: A New Backbone that can Enhance Learning Capability of CNN”, 2019.

[11]: Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”, 2014.

[12]: Shu Liu, Lu Qi, Haifang Qin, Jianping Shi and Jiaya Jia. “Path Aggregation Network for Instance Segmentation”, 2018.

[13]: Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”, 2020

[14]: Renjie Xu, Heifeng Lin, Kangjie Lu, Lin Cao and Yunfei Liu. “A forest fire detection system based on ensemble learning”, 2021

[15]: Mulan Qiu, Liang Huang and Bo-Hui Tang. “ASFF-YOLOv5: Multielement Detection Method for Road Traffic in UAV Images Based on Multiscale Feature Fusion”. 2022

[16]: Stefan Elfwing, Eiji Uchibe and Kenji Doya. “Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning”, 2017

[17]: Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye and Dongwei Ren. “Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression”, 2019

[18]: Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj and Le Song. “SphereFace: Deep Hypersphere Embedding for Face Recognition”, 2017

[19]: Jiankang Deng, Jia Guo, Jing Yang, Niannan Xue, Irene Kotsia, and Stefanos Zafeiriou. “ArcFace: Additive Angular Margin Loss for Deep Face Recognition” , 2018

[20]: Sheng Chen, Yang Liu, Xiang Gao, Zhen Han. “MobileFaceNets: Efficient CNNs for Accurate Real-Time Face Verification on Mobile Devices”, 2018

[21]: César Cheuque, Marvin Querales, Roberto León, Rodrigo Salas. “An Efficient Multi-Level Convolutional Neural Network Approach for White Blood Cells Classification”, 2022

[22]: Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov and Liang-Chieh Chen. “MobileNetV2: Inverted Réiduals and Linear Bottlenecks”, 2018

APPENDICES