TOWARD A FLEXIBLE FACIAL ANALYSIS FRAMEWORK IN OPENISS FOR VISUAL EFFECTS

Yiran Shen

A thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science at

Concordia University

Montréal, Québec, Canada

September 2019

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Yiran Shen**

Entitled: **Toward a Flexible Facial Analysis Framework in OpenISS for Visual Effects**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Rajagopalan Jayakumar

_____ Examiner
Dr. Gregory Butler

_____ Examiner
Dr. Weiyi Shang

_____ Supervisor
Dr. Joey Paquet

_____ Supervisor
Dr. Serguei A. Mokhov

Approved by _____
Chair of Department or Graduate Program Director

_____ 20 _____ _____
Dr. Amir Asif, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Toward a Flexible Facial Analysis Framework in OpenISS for Visual Effects

Yiran Shen

Facial analysis, including tasks such as face detection, facial landmark detection, and facial expression recognition, is a significant research domain in computer vision for visual effects. It can be used in various domains such as facial feature mapping for movie animation, biometrics/face recognition for security systems, and driver fatigue monitoring for transportation safety assistance. Most applications involve basic face and landmark detection as preliminary analysis approaches before proceeding into further specialized processing applications. As technology develops, there are plenty of implementations and resources for each task available for researchers, but the key missing properties among them all are flexibility and usability. The integration of functionality components involves complex configurations for each connection joint which is typically problematic with poor reusability and adjustability. The lack of support for integrating different functionality components greatly impact the research effort and cost for individual researchers, which also leads us to the idea of providing a framework solution that can help regarding the issue once and for all. To address this problem, we propose a user-friendly and highly expandable facial analysis framework solution. It contains a core that supports fundamental services for the framework, and a facial analysis module composed of implementations for facial analysis tasks. We evaluate our framework solution and achieve our goals of instantiating the facial analysis specialized framework, which essentially perform tasks in face detection, facial landmark detection, and facial expression recognition. This framework solution as a whole, solves the industry problem of lacking an execution platform for integrated facial analysis implementations and fills the gap in visual effects industry.

Acknowledgments

During my studies, I was very lucky to get inspiration, suggestions and help from many people. I would like to express my very great appreciation to my supervisor Dr. Joey Paquet and co-supervisor Dr. Serguei Mokhov for their valuable and constructive advice during the planning and development of this research work. I am gratefully indebted to them for their valuable time and input on this thesis. I would also like to thank my labmates and good friends, Haotao Lai and Jashanjot Singh for their valuable suggestions and encouragement on my work.

Secondly, I must express my profound gratitude to my parents, Yiping Zhang and Xiangyu Shen, and to my girlfriend Xitong Zhang for providing me with unfailing love, support and continuous encouragement throughout the process of researching and writing this thesis.

Last but not least, I need to thank my friends, Qinwei Luo, Bo Li and Jingye Hou. I'm very fortunate to have them around during the difficult times. This accomplishment would not have been possible without them. Thank you.

Contents

List of Figures

List of Tables

Acronyms

AAM Active Appearance Model Methods. vi, 22

BN Batch Normalization. 32

CECLM Convolutional Experts Constrained Local Model. vi, 24, 68, 102, 111, 117, 118

CGI Computer Generated Imagery. 18

CLM Constrained Local Methods. 22, 23, 24

CLNF Constrained Local Neural Fields. 23, 68, 85, 117

CNN Convolution Neural Network. 19, 21, 31, 34, 40

CRIFST the China Research Institute of File Science & Technology. 3, 5, 7, 9, 121

FAM Facial Animation Movie. 3, 16, 100

FPS Frame Per Second. 13, 14, 25, 97, 101, 103, 104, 108, 109, 110, 111, 113

HOG Histograms of Oriented Gradients. 19, 20, 42, 65, 93, 101, 108

IoU Intersection over Union. 105, 106, 107

LBF Local Binary Features. vi, 25, 83, 88, 97, 101, 111, 117, 118

LNF Local Neural Field. 23

mAP Mean Average Precision. 104, 107, 108

OpenISS Open Illimitable Space System. 42, 46, 47, 48, 50, 52, 53, 100, 119, 120, 121

SSD Single Shot MultiBox Detector. 21, 65, 66, 85, 88, 102, 108, 117, 118

Chapter 1

Introduction

In this chapter, we will first introduce the facial animation background of our thesis. We summarize our research problems by identifying and analyzing two concrete problems in the facial animation movie production procedure. Following this, we state the goal of this thesis, define our usage scenarios and extract both functional and non-functional requirements. Finally, we list our contributions, followed by a brief overview of the structure of the thesis.

1.1 Motivation

The primary motivation for this research work stems from a research and development problem faced with by our partner in China — the China Research Institute of File Science & Technology (CRIFST). CRIFST is a research lab that aims to introduce visual effects in the film-making process. As a part of our collaboration with CRIFST, they are looking for a solution to match actor-based facial expressions to animated characters, ideally in real-time, to save editing and artists costs during production and post-production in the creation of facial animation motion pictures.

Facial animation movie (FAM) is a kind of movie that only focuses on creating facial actions of a character. Facial animation movie could be used in clips in an animated movie, a cinematic opening trailer of a game or the facial details of any animated characters. In our thesis, we are going to focus on two specific and concrete problems in the production process of the facial animation movie.

1.1.1 FAM Production Procedure Overview

Generally, producing any movie can be divided into three simplified steps:

- Pre-production — represents the period of writing scripts (storyline), finding resources, actors, locations, etc.
- Production — represents the period when the movie starts being shot until the shooting finishes. Shots can be traditional and/or computer-graphics based.
- Post-production — represents the period when the editor modifies the entire recorded shots to fit the director's vision for the movie until the end of the editing process. In here, different special effects and virtual scenes may be added, etc.

When it comes to the facial animation movie, there are two specific problems that need to be solved. They can be expressed in simple terms as the following:

Concrete Problem 1: During the production phase, when the actor needs to perform in the scene, the director will need to have a synchronized view of the generated animation imagery in order to provide feedback and instructions to the actor for each shot. A well-produced animation movie requires extensive attention to detail, which requires delicate body-parts movements, accurate facial expressions, and sounds. For the facial animation, which is our area of focus, the human actor needs to perform actions such as talking, appearing to be shocked, scared and so on while following the movie's script.

In order to allow the director to view the corresponding animation on a screen, a convenient real-time technology is needed to track the actor's facial actions and translate them into an animation sequence.

Concrete Problem 2: At the post-production phase, editors would review the entire movie with the director and animation artists and make adjustments, additions, or eliminations of certain scenes and shots, clips or frames. The director might expect a different result for some facial expression in certain

scenes than the initial version that was done through a motion capture after watching the resulting animation.

Therefore, the editors need a technology that can help the them to access more appropriate facial animation frames or sequences and do the replacement editing in the case where such sequences and frame are already available from the current or previous movie projects.

1.1.2 Existing Solutions

To address **Concrete Problem 1**, one of the mainstream solutions utilized for facial animation movie production process is 3D scanning [16]. The procedures for the animation movie production that use 3D scanning technology typically involve dozens of cameras arranged around the actor's face at different angles to perform simultaneous facial scan. Although this allows us to capture high resolution topology with extremely detailed texture maps, the overall process is very complicated. During the production phase of a facial animation movie, the facial capture is performed frame by frame, and each frame scan needs a considerable amount of processing time, this requires a tremendous cost budget for the movie producer, and it prolongs the shooting and production process. Due to its nature of frame-based scanning, it would not allow real-time generation and viewing on site for a complete scene that spans dozens of frames.

For the **Concrete Problem 2**, the traditional way to do it is that the editors use the keyword of the expected expression to search a database and find the appropriate expression. Editors might spend a prohibitively long time browsing through all facial animation sequences in a database for a replacement. However, existing matching technology only allows filtering the model database by facial expression keywords, and on many occasions, the results are not precisely accurate.

1.1.3 A Requested Need

CRIFST VFX Studio requests a need to address these two concrete problems. The need can be illustrated as Figure 1. The steps of the production phase shown by the blue box, can be described as following:

1. The movie shooter will capture the actor's performance streaming into the facial feature extractor1.

2. The extracted facial features then will flow into the animation translator which map those features to the digital model enabling the directors to see the synchronized animation character movie on their monitors.

As mentioned above, the editor may need to replace some of the frames within a recorded movie. For each unsatisfied frame, the steps of the post-production phase shown in brown box can be described as below.

1. The editor will input an expected expression image to indicate what kind of expression they need to replace the unsatisfied frame.

2. The facial feature extractor2 will extract the facial features of the given image. At the same time, the facial expression extractor will determine the expression category of the input image. All these two intermediate results will be sent to the database matcher.

3. The database matcher will use the extracted features and the detected expression to search among the well-defined expression database and return the most similar one.

4. The result frame from the database will be used to replace the unsatisfied frame within the animation movie.

For further references, we would like to define two terms to refer the techniques used in the two phases described above.

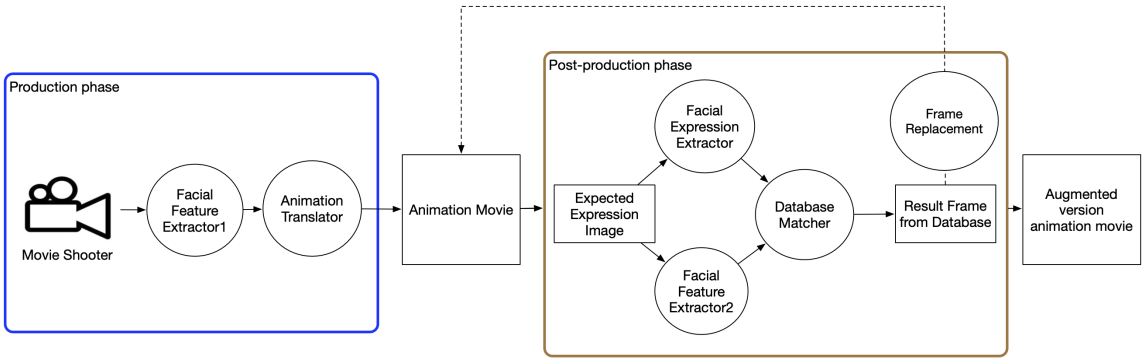

Figure 1: FAM production procedure diagram

- Landmark-based facial action mapping, refers to the techniques used in production phase.
- Human-face-based facial animation expression matching, refers to the techniques used in post-production phase.

1.2 Focus of Research

According to this requested need from CRIFST VFX Studio, we have performed a comprehensive survey and propose our own solution. We found that the animation translator in production phase can already be realized by some existing software like REALLUSION [2] and Autodesk Maya [17]. Assuming you already have your character model defined, what you need as input is just the facial landmarks for the actor's face. Under such consideration, we are not going to re-implement the animation translator but focus on the facial landmark extraction. For the post-production phase, since there is no existing solution can do such a job, we will have to provide our own solution for it.

For a given movie shooting image, if we want to obtain the facial features, the general approach is that we will need to perform two steps, one is the face detection and the other is facial landmarks (aka. keypoints) detection. Then for facial expression recognition, also two steps are required. One is the same as the previous, face detection, and the other is facial expression recognition.

For our focus of research, we propose our subset solution based on the background research, which is to implement everything except the animation translator within the requested need from CRIFST VFX studio as illustrated in Figure 1.

1.3 Research Problems

In the previous section, we described our focus of research which lies under the general scope of facial analysis and introduce some of the related terminologies. In this section, we will give the scientific definitions of them in this thesis.

1.3.1 Face Detection

Face detection as a specific case of object detection has almost the same definition as object detection. You are given an image I and the task is to detect instances of face within I. For each face i, you need to return a bounding box B to indicate the location of face i.

1.3.2 Facial Landmark Detection

Facial landmark detection is a series of methods that could locate the facial key points in a human face. The number of key points is based on the model we choose, illustrated by Figure 2. So the definition of facial landmark is: you are given one face image I and a facial landmark model M. You need to match or detect facial key points on face image I and return the locations of each key points in image i.

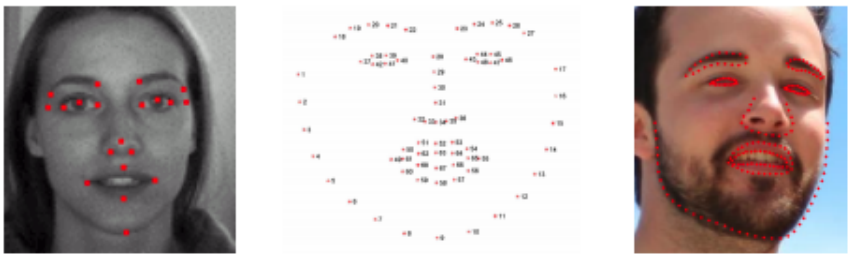

Figure 2: Facial landmark model example [1] with 20, 68 and 194 key points respectively

1.3.3 Facial Expression Recognition

As for facial expression recognition, you are given a face image I and a list of facial expression interpretative categories L. The task is to analyze the facial features and classify this information into one category c which belong to L.

Three facial analysis domains mentioned above cover our facial analysis methods research. The first part for this operation is face detection. Most of time, camera can only fetch video streams, which means there are more than faces information fetched, but also some related background is captured. We need to use a reliable face detection method to find where the faces are to extract related features. On the other hand, facial landmark is a good facial feature as quantifiable feature to define a facial model. In animation modelling technology, the expression of facial model are controlled by adjusting key points on face, which are perfectly matching the characteristics of facial landmark. And with facial landmark and facial expression recognition, we can automatically classify human' s expressions into a specific category.

1.4 Goal

With the background and motivation, we stated the existing solution and pointed out their limitations in Section 1.1.2. Then we describe a proposed idea by an organization, CRIFST, to address two concrete problems in the facial animation movie production procedure.

Inspired by the CRIFST's needs, we would like to provide our concrete solution on top of it. So we propose our goal of this thesis: **we would like to design a facial analysis solution that can realize the proposed idea. Precisely, it should be able to provide the functionality for face detection, facial landmark detection and facial expression recognition. The solution should also be designed in such way that it can integrate more new components easily as well as user-friendly.**

1.5 Scenarios and Requirements

With the definitions defined above, we would like to refine our concrete problems and represent them in more detail usage scenarios. All of them, each in their own, highlight some problems that are not solved by existing solution. And our solution is not only aiming to solve these problems individually, but also to provide a solution, in general, for all these problems.

1.5.1 Real-time Landmark-based Facial Action Mapping

Actors

- Facial animation movie application developer –below referred as A
- Facial animation movie director –below referred as D
- Facial animation movie actor –below referred as C

Effect: Real-time generation of facial animation movie involving human actors on site

Imagine an animation movie production procedure. During the production phase, C will need to act on the scene, and D requires to have a synchronized viewing of the generated animation in order to provide feedback and instruction. In order to allow D to see the corresponding animation on his viewing portal, A decides to take landmark-based facial action mapping technology to develop real-time landmark-based facial action mapping application. More specifically, A integrates our facial analysis solution, which could continuously take images of C as input and output the facial landmarks of C's face in real-time, into his real-time landmark-based facial action mapping application, so D could see the corresponding generated animation on his viewing portal of the application in real-time.

This usage scenario can be abstractly summed-up as the following, which becomes our requirements:

FR1: The solution shall be able to detect the areas of human face inside a given image and provide a corresponding bounding box for each of the detected location.

FR2: The solution shall be able to detect facial landmarks on human face image which is masked by bounding boxes.

FR3: The solution shall be able to connect the required features from **FR1** and **FR2** together seamlessly, ensuring the whole process can be done in real-time.

1.5.2 Human-face-based Facial Animation Expression Matching

Actors

- Facial animation movie application developer –below referred as A
- Facial animation movie director – below referred as D
- Facial animation movie editor – below referred as E
- Facial animation movie character – below referred as C

Effect: Editor gets more appropriate facial animation models by inputting human image

Imagine during an animation movie post-production phase, where the production team will review the entire movie and make adjustments, addition, or elimination of certain frames. For example, the smile that shows on C was not as dramatic as D wanted, so D paused the movie clip and asks E to find a more appropriate animation picture of the smile expression of C to replace the original frame. In a traditional way, E need to browse the whole facial expression database using keyword to find the most likely expression as the one expected by D, then use it to replace the original one. Now, E wants to use a human face image with the required smile, to get a similar animation expression from the database instead of using keyword search.

In order to meet the requirement from E, A decides to build an application based on human-face-based facial animation expression matching technology. More specifically, A uses our facial analysis solution, which could take a static human image as input and output the facial landmarks and expression category of the

face. Then A implements a database matcher which employs facial landmarks and expression category to search a pre-populated facial animation expression database. It is significant to mention that with facial expression category label, we can find all existing expressions in the same category just like keyword search. But with facial landmarks, we can differentiate the expression even within the same category.

Although this matching application greatly helps E to find suitable expressions faster than before, he also wants the result to be as accurate as possible. So in order to improve the accuracy of this application, A analyzes the performance of all the components involved and identifies the bottleneck being a specific component in our facial analysis solution. After doing research, A finds that there is a new deep learning implementation available for this component which could achieve higher accuracy and solve the problem. With a few hours of work, A easily replaces the implementation of the bottleneck component with the new one and thus provides a more accurate facial animation expression matching application to E.

This usage scenario can be abstractly summarized as the following, which becomes our requirements:

FR4: The solution shall be able to recognize the facial expression on human face and assign it to an expression category.

FR5: The solution shall be able to connect the required features from **FR1**, **FR2** and **FR4** together seamlessly ensuring the whole process can be done.

FR6: The solution shall enable the developer to easily achieve adaptability by replacing bottleneck component in our solution to meet their specific expectations.

FR7: The solution shall enable the developer to easily enable deep learning approaches to work with our solution.

1.5.3 Camera Support

Actors

- Facial animation movie application developer –below referred as A
- Facial animation movie production company –below referred as M

- Camera type 1 –below referred as C1
- Camera type 2 –below referred as C2

Effect: Instant integration for new camera types

Imagine A has developed a real-time landmark-based facial action mapping application for M. The whole application is divided into two parts: (1) integrate camera adapter of C1 into application, so this C1 adapter could transfer the data format of C1 into human RGB image stream (2) input transferred human RGB image stream into our landmark-based facial action mapping module to get generated animation video stream. After a few years, M decided to buy a new type of camera C2 with higher resolution and they wanted to add C2 into the shooting flow. After trying to get the new camera in use, M found that this new camera was not supported by their real-time animation shooting application. M asked A to solve the compatibility issue. A did some research and found the root cause being unsupported new data format that C2 uses. This could be solved by adding an adapter for C2 into their real-time animation shooting application. After hours of development, A implemented this new camera adapter and integrated it into their application. Now, M can have C2 available to shoot and have the recorded data successfully processed by the application for later stages.

This usage scenario can be abstractly summed-up as following, which becomes our requirement:

FR8: The solution shall allow developers to add new cameras or switch existing cameras into their current processing platform with minimal effort.

1.5.4 Non-functional Requirements

For a solution to achieve what we have described in previously mentioned usage scenarios, we found that our solution must meet the following non-functional requirements:

NFR1 Real-time performance: *The system should be able to perform the task*

in the scenario mentioned in Section 1.5.1 in a speed of at least 30 frames per second (FPS).

Real-time is the key factor determining the proficiency of the result generation for the scenario mentioned in Section 1.5.1, which enables the director to watch the synchronized viewing of the generated animation and give feedback and instruction in a timely manner. If this process is not real-time, the director's experience will significantly decrease as there are noticeable lags in the generated animation video. Based on the definition provided in Kapoor et al. [18], a real-time facial analysis system needs to have a speed of at least 30 FPS. Thus, we will use 30 FPS as our baseline for real-time evaluation.

NFR2 Accuracy: *The system should be able to perform the scenario tasks mentioned in the Section 1.5.1 and Section 1.5.2 with an accuracy comparable to the state of the art solutions.*

For mapping scenario mentioned in Section 1.5.1, the real-time performance is the first priority, while ensuring that, we still want the whole task to achieve an accuracy as close to state of the art solutions as possible.

For matching scenario mentioned in Section 1.5.2, each matching task is an independent one-time operation, and the purpose of this functionality is to provide a more automated and accurate search result for editors. Therefore, the sole requirement for this scenario is the accuracy performance that ensures the best result possible, which should be comparable to the state of the art solutions.

NFR3 Extensibility: *The ability of a system to be able to add new functionality or modify existing functionality without impairing existing system functions.*

In a fast-developing world with constant hardware and software technology upgrade, professionals in facial analysis industry will need to utilize more and more advanced technologies. It is essential to design a solution with wide range of extensibility that allows us to add, modify or integrate devices and algorithms

into our solution with minimal effort.

NFR4 Usability: *The ability of a software system to effectively provide the expected functionality to the user, in a manner that is as intuitive and the least strenuous or problematic as possible.*

Our solution should provide the required functionalities stated in our goal to ensure the foundations of usability. Secondly, providing simple, meaningful and understandable APIs is the key to make a software user-friendly. This requires our solution to have concise, compact and logical naming conventions throughout the build. Technical documentations and comments should also be provided to ensure other experienced/inexperienced users have complete support for them to quickly adapt to our solution's APIs.

1.6 Contributions

Our contribution is five-fold:

- We offer a way for researchers to perform face detection, facial landmark detection, static-based facial expression recognition, and dynamic-based facial expression recognition on their facial experiment data. Especially, face detection and facial landmark detection ensure real-time performance.
- We offer a way for researchers to integrate their facial analysis implementations and test them in a real-world environment with input being frames directly captured by the camera.
- We offer a way for researchers to combine facial analysis algorithms from various research domains and compare application practicability.
- We offer a platform for researchers to combine deep learning facial analysis implementations written in Python with other facial analysis implementations written in C++.

- We offer a way for researchers to assemble various components to form their applications without worrying about the connection between components or handling intermediate results.

To achieve the above, we did the following works:

- Design and implement a framework solution which consists of the core and facial analysis module that enables face detection, facial landmark detection, static-based facial expression recognition, and dynamic-based facial expression recognition.
- Design and implement a device module which could enable our solution to support different depth cameras or add new camera devices.
- Design and implement a pipeline module that provides a linear execution mechanism, which allows users to execute a series of functionality components in their chosen order, and also automatically handles the actual component connection and transfers intermediate results.
- Design and implement a deep learning support module that enables the integration of Python-based deep learning facial analysis implementations.
- Design a face detection submodule, provide three default face detectors, and compare the accuracy and speed performance between them.
- Design a facial landmark detection submodule, provide three default facial landmark detectors, and compare the accuracy and speed performance between them.
- Design a static-based facial expression recognition submodule, provide three default recognizers, and compare the accuracy and speed performance between them.
- Design a dynamic-based facial expression recognition submodule, provide two default recognizers, and compare the accuracy and speed performance between them.

1.7 Thesis Outline

In this chapter, we introduced the background of facial animation followed by discussing two concrete problems and limitations. With a proposed idea in hand, we pointed out our research of interests. In the end, we refined two phases of FAM production procedure to become more detailed usage scenarios. In the following chapters, the layout of our thesis is listed below:

- In Chapter 2, we will review the major dominated methods in our research area as well as the related libraries we may use in our solution.
- In Chapter 3, we will propose our solution with the reason why we design it in a framework manner followed by the design of core and specialized frameworks.
- In Chapter 4, we will create an instance of the proposed framework stating the implementations of each facial analysis submodule in detail. Also, we describe how do we create two specific applications to address the problems we mentioned in Section 1.1.
- In Chapter 5, we will demonstrate that our framework solution fulfills all the requirements listed in Section 1.5 and report the results obtained from community acknowledged metrics.
- In Chapter 6, we summarize our work with advantages and limitations. Then we propose some potential research directions in the future.

Chapter 2

Related Work

In this chapter, we mainly discuss the related work of our research. There are two parts of related work that we would discuss in this chapter: (1) the representative works of facial analysis domains (2) the related libraries of our work.

2.1 Facial Animation Background

In the traditional animation industry, animators draw images on a transparent paper with a colored pencil, one frame at a time. This is extremely time and budget consuming. Nowadays, animation modeling, also referred as computer-generated imagery (CGI), is made by generating images using computers. Animation modeling works in a completely different way than the traditional animation, which is more like playing with puppets rather than drawing. People animate and pose models, then conform to the frame-by-frame approach with controllable settings to build a complete animation movie. Characters in animation modeling are digitally modeled in the program, then fitted with a skeleton which could be eventually controlled by animators.

The facial expression of characters can also be modeled and controlled by the computers, which known as facial animation. The common way for making facial animation is to build a head model first and cover it with face texture, then control the key points on the face model to change the expression of characters [19]. And there

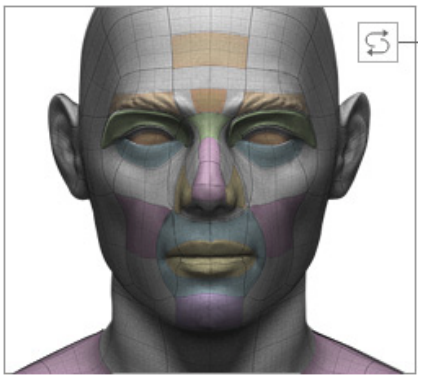

Figure 3: Facial animation model [2]

Figure 4: Facial animation texture [2]

are several animation software providing such convenient method, like REALLUSION [2], Autodesk Maya [17] and etc. Figure 3 and Figure 4 show the facial animation model and texture respectively, and Figure 5 shows an example of facial animation with different facial expressions. The whole process of making an animation video is done by connecting different facial frames in order.

Figure 5: Facial animation example [2]

2.2 Face Detection

In this section, we would like to introduce the related work of face detection domains mentioned in Section 1.3. We select three representative works of face detection domain: haar-like features detector, histograms of oriented gradients (HOG) face detector, and convolution neural network face detector. Haar-like features detector and HOG face detector are more traditional face detectors, whereas convolution neural network (CNN) face detector is recently proposed.

2.2.1 Haar-like Features Detector

Viola and Jones face detector is one of the most impact face detector in 2000s [20]. There are three key points for Viola-Jones face detector: the integral images, classifier learning with AdaBoost, and the attentional cascade structure. The integral image could compute the sum of values in a rectangle subset of a grid quickly and efficiently. In Viola-Jones face detector, integral image is used to calculate haar-like features, which improve the speed of Viola-Jones face detector [21]. Boosting is a kind of linear regression method. The key point of boosting is combining many weak hypotheses with moderate accuracy to a highly accurate hypothesis [22]. AdaBoost corresponds to a single iteration of the back-fitting algorithm, it has better performance in accuracy and does not penalize overconfident predictions. The attentional cascade structure can reject most of the negative sub-windows in early stages and keep almost all the positive examples [23]. A cascade is formed by a series of classifier windows and used as a degenerate decision tree, and this cascade structure has a great improvement in accuracy compared with single classifier.

2.2.2 HOG Face Detector

HOG face Detection is combined with histograms of gradients (HOG) and linear support vector machine (SVM). HOG face detector is proposed by Dalal et al. [3] in 2005. Basically, the HOG is a descriptor for local object appearance and shape. The image is divided into small connected regions called cells, a histogram of gradient directions is calculated within each cell. Figure 6 is an example of combining lots of faces together to represent distribution of intensity gradients or edge directions. Finally trained Linear Support Vector Machine model is used to detect faces in an image.

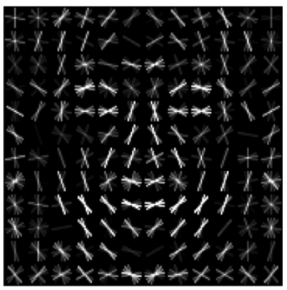

Figure 6: HOG face descriptor from lots of face images [3]

2.2.3 Convolution Neural Network Detector

For convolution neural network (CNN) detector, convolution layer is the core layer. It is able to capture the spatial and temporal dependencies and extract the high-level features from input images. The low-level layers of CNN mainly extract the abstract features from the input image, and high-level layers will reconstruct the extracted features to calculate result.

2.2.3.1 Single Shot MultiBox Detector

Single Shot MultiBox Detector (SSD) is a kind of CNN-based face detector proposed by Liu et al [4]. The main theory of SSD is generating a fixed-size collection of bounding boxes and using CNN to scores for object class instances in those boxes. Finally, using a non-maximum suppression to predict the right bounding box. The models of SSD is shown in Figure 7. In extra feature layers, it produces the key features, multi-scale feature maps, convolutional predictors, default boxes and aspect ratios for prediction. These layers decrease size of feature maps to create multiple-scales feature maps. For a feature layer of size (m, n) with p channels, the basic element is a small kernel with size of $(3, 3, p)$ that produces either a score for a category or a shape offset relative to the default box coordinates. The bounding box offset values are measured relative to the feature map location. For each feature map cell, there are a set of default bounding boxes to detect. The position of each box to its corresponding cell is fixed. For each feature map cell, the offsets relative to the default box shapes and per-class scores in each boxes are predicted.

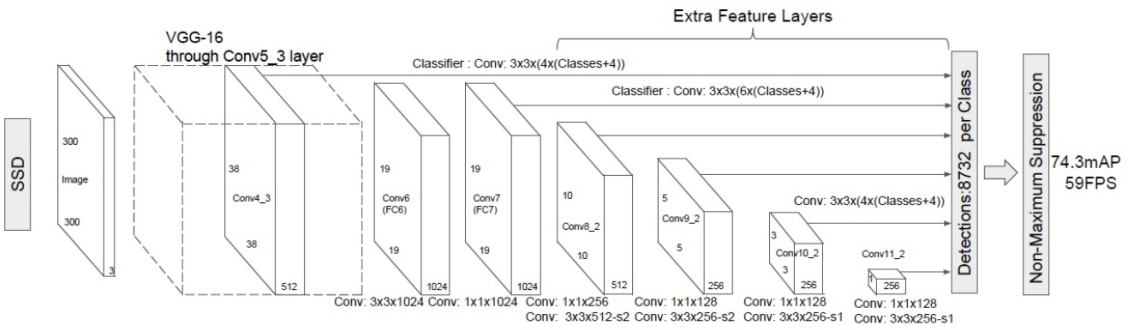

Figure 7: SSD model [4]

2.3 Facial Landmarks Detection

The aim of facial landmark detection is to get key points coordinate on images or videos. The key points consist of two parts, one is the special location of a facial component like profiles of eyes, nose, and mouth, the other part is points connecting facial components and facial contour. In this section, we will discuss three main categories of facial landmark detection: active appearance model methods, constrained local methods and regression-based methods.

2.3.1 Active Appearance Model Methods (AAM)

In order to construct active appearance model, we need to build the global facial shape model and holistic facial appearance model based on principal component analysis. After the construction of model, the model detects landmark locations by fitting the learned appearance and shape models to the input image. There are three steps for AAM to construct model [24]. Firstly, using Procrustes analysis to register training facial shapes, which could remove the affine transformation and normalize training face images. Secondly, using principal component analysis (PCA), to learn mean shape and capture the shape variations. Lastly, using normalized facial images to generate a mean appearance.

2.3.2 Constrained Local Methods

The constrained local methods (CLM) use local appearance model to detect each landmark independently first, then refine the local detection results with global face shape model variations [25]. Minimizing the misalignment error subject to the shape patterns could find the landmarks for local appearance model.

$$x = arg(minQ(x) + \sum D_d(x_d, I)) \tag{1}$$

x_d represents the position of different landmarks in x. $D_d(x_d, I)$ represents the local confidence score around x_d. $Q(x)$ represents a term to penalize the infeasible or anti-anthropology face shapes in a global sense. Through the calculation, we could find the best set of landmark locations that have strong independent local support for each landmark and satisfy the global shape constraint.

2.3.2.1 Constrained Local Neural Fields (CLNF) Facial Landmark Detector

CLNF facial landmark detector is based on CLM facial landmark detector and proposed by Tadas et al. [5]. They try to detect facial landmarks with poor lighting conditions and extreme pose or occlusions. Compared with the traditional CLM method, CLNF detector uses a Local Neural Field (LNF) patch expert to learn non-linear and spatial relationships between the input pixels and probability of a landmark being aligned. On the other hand, CLNF detector uses a Non-uniform Regularised Landmark Mean-Shift (NU-RLMS) optimization technique to optimise the model.

The overview of CLNF model is shown in Figure 8. In CLNF model, LNF patch expert is used to calculate more reliable response maps. NU-RLMS method is used to optimize the patch responses and takes the reliability of each patch expert into account to get more accurate fitting.

LNF patch expert is shown in Figure 9. Solid lines represent vertex features f_k, dashed lines represent edge features g_k or l_k. The input vector x_i is connected to the relevant output scalar y_i through the vertex features, the neural layer, and the

vertex weights. The outputs of model are connected with edge features g_k or l_k. The neighboring and longer distance between pixels is learned by LNF patch expert. LNF can also capture complex non-linear relationships between pixels and output.

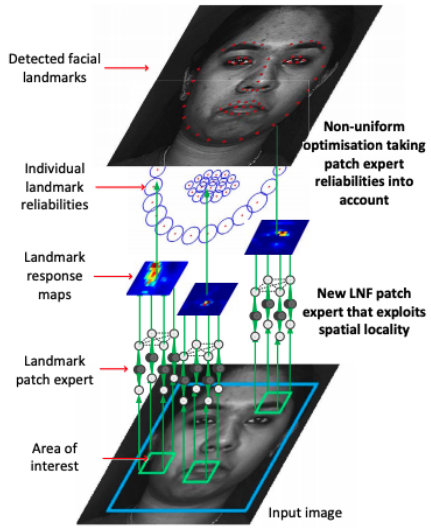

Figure 8: CLNF model [5]

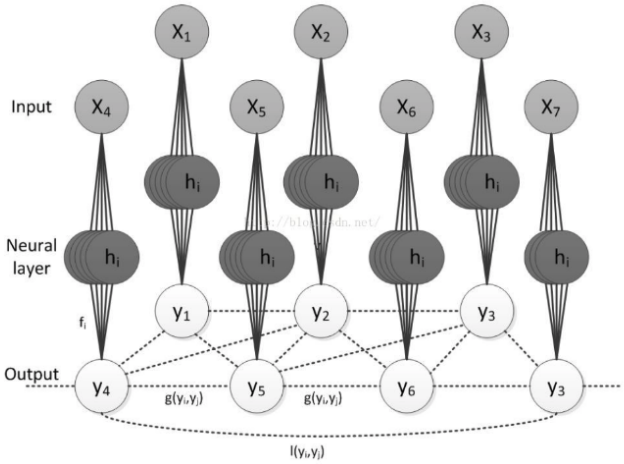

Figure 9: LNF model [5]

2.3.2.2 Convolutional Experts Constrained Local Model (CECLM) Facial Landmark Detector

CECLM detector is proposed by Zadeh et al. [6]. Compared with tranditional CLM facial landmark detectors, CECLM detector uses a kind of new structure, Convolutional Experts Network (CEN), which combines the advantages of neural architectures and mixtures of experts in end-to-end frameworks. There are two main parts of CECLM: shape parameter update and CEN-based response map computation. For parameter updating, the position of aligned landmarks are updated jointly, then penalized for misaligned landmarks and irregular shapes using a point distribution model.

The CEN model is shown in Figure 10. There is a nxn pixel region of interest (ROI) as input and a response map as outputs. This model aims to compute a response map to localize individual landmarks. According to the model, ROI as the first layer is a contrast normalizing convolutional layer and using shape 500 x 11 x 11 which performs Z-sore normalization. Mixture of Expert Layer (ME-layer) as the most important layer can model different landmark appearance prototypes and is a convolutional layer of 100 x 1 x 1 using sigmoid activation and output individual experts vote on alignment probability. According to CEN theory, the landmark i in position x_i follows the equation:

$$y^i{}_{x_i} = p(l_i = 1, L = L_{x_i}) \tag{2}$$

l_i is an indicator for aligned landmark number i. L is the image ROI at location x_i. y_i represents the response maps of size n x n.

2.3.3 Regression-based Methods

Compared with methods mentioned above, regression-based methods do not build any face shape model but learn the mapping from image appearance to the landmark locations directly [26].

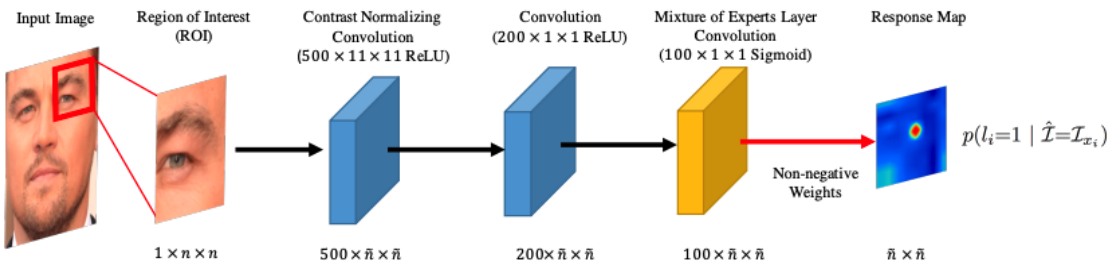

Figure 10: CECLM model [6]

2.3.3.1 Local Binary Features (LBF) Facial Landmark Detector

The LBF facial landmark detector we mentioned here, is based on paper titled "Face Alignment at 3000 FPS via Regressing Local Binary Features" by Shaoqing at el. [7]. LBF facial landmark detector is based on the regression approach. The basic training process of LBF facial landmark detector is shown in Figure 11.

There are two steps in the training phase. They begin by learning a feature mapping function $\phi^t(I_i, S_i^{t-1})$ to generate local binary features. Then based on given features and target shape increments $\{\Delta S_i^t = S_i - S_i^{t-1}\}$, they learn a linear projection W^t by linear regression. In the testing phase, the shape increment is directly predicted and applied to update the currently estimated shape [7].

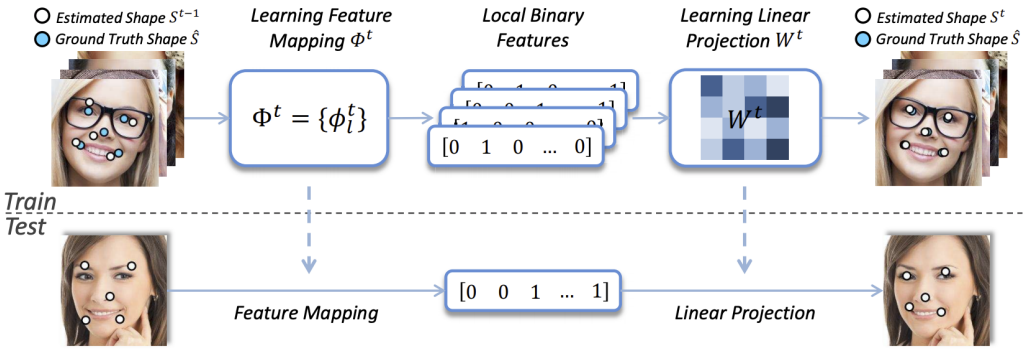

Figure 11: Overview of LBF training process [7]

2.4 Facial Expression Recognition

Facial expressions as an important signal for human beings, which could convey their emotional states and intentions, has always been a cutting-edge research

field. Facial expression recognition could combine with lots of other fields to create specialized applications. For example, sociable robotics, human figure recognition, and human-computer interaction systems. According to a cross-culture study [27] in the twentieth century, there are six basic expressions: anger, disgust, fear, happiness, sadness, and surprise. Most of facial expression recognition studies focus on recognizing these six expressions plus neutral expression. There are two main categories for facial expression recognition: static image facial expression recognition and dynamic sequence facial expression recognition. Static-based methods define human's expression based on the current single image, and dynamic-based methods recognize expression by analyzing a set of contiguous frames.

Handcrafted features and regression learning are the most popular methods before machine learning, just like local binary patterns and non-negative matrix factorization. Since 2010, deep learning has achieved great accuracy and speed due to dramatically increased calculation ability of computer and better training datasets. In this paper, we do not introduce traditional facial expression recognition and instead, pay more attention on deep learning facial expression recognition methods. We first introduce the common facial expression recognition datasets that are used to judge the performance of algorithms, and show the regular steps for deep learning facial expression recognition methods. Then we show details about representative studies for static-based methods and dynamic-based methods.

2.4.1 Facial Expression Recognition Datasets

For facial expression recognition, a dataset with sufficient labeled training data and many variations of the face and environments is very helpful. The datasets that we introduce are public available which contain basic expressions and are used widely in many researches.

CK+ The Extended CohnKanade (CK+) dataset proposed by Cohn et al. [8] is the most popular facial dataset for evaluation with a laboratory-controlled environment. There are 593 video sequences that vary in duration from 10 to 60 frames and contain

123 subjects, and all of these sequences show the shifting process from the neutral expression to the peak expression. Figure 12 shows the example images in CK+ dataset.

Figure 12: CK+ image sequences example [8]

FER2013 FER2013 database [9] is an unconstrained dataset which contains 28709 training images, 3589 validation images, and 3589 test images. The images are collected by Google image search. The facial region is cropped from each image and resized to 48 x 48 pixels. All images are labeled with seven expressions (sadness, neutral, surprise, happiness, anger, disgust, and fear). Figure 13 shows the example images.

Figure 13: FER2013 image example [9]

AFEW The Acted Facial Expressions in the Wild (AFEW) dataset [10] was established and used in annual Emotion Recognition In The Wild Challenge (EmotiW) as evaluation dataset. The data in AFEW was collected by selecting different movies clips with various head poses, occlusions, illuminations, and seven basic expressions: anger, fear, disgust, surprise, neutral, sadness, and happiness. After several years of updating, AFEW contains 773 training samples, 383 validation samples, and 653 testing samples. Figure 14 shows the examples of AFEW dataset.

Figure 14: AFEW image sequences example [10]

SFEW The images in the Static Facial Expressions in the Wild (SFEW) [28] were selected from the AFEW dataset, whose selection algorithm is based on facial point clustering. The SFEW dataset was used as evaluation dataset for the SReco sub-challenge in EmotiW 2015. There are 958 training samples, 436 validation samples and 372 testing samples in SFEW dataset. The expression classes of SFEW is same as the AFEW dataset.

MMI The MMI dataset proposed by Pantic et al. [11] is a laboratory-controlled facial dataset. There are 326 sequences from 32 subjects and 213 sequences labeled with six basic expressions without contempt. The expression sequences follow the rules of neutral-peak-neutral process, which means that the sequence starts with neutral expression and reaches peak, then returns to neutral again. Compared with most public facial datasets, MMI provides more challenging factors for facial expression recognition with glasses, mustache and various occlusion. Figure 15 shows the examples of the MMI dataset.

Figure 15: MMI sequences image example [11]

Multi-PIE The Multi-PIE database was built by Gross et al. [29] and contains 755,370 images from 337 subjects. The differences between Multi-PIE database and other facial datasets are that the Multi-PIE dataset provides multi-viewpoints and various illumination conditions. There are six expressions used to label the images: smile, squint, surprise, neutral, scream and disgust.

EmotioNet EmotioNet database [30] contains millions of facial expression images which are collected automatically from the internet. Especially, EmotioNet provides not only expressions labels but also the action unit (AU) labels, there are 95000 images labeled with AU model.

VGGFace VGGFace dataset [12] is a large face dataset containing hundreds of example images for thousands of unique identities. There are over 2.6 million images and 2,6000 people. This dataset provides a good face collection for facial research, especially in facial re-identification. Lots of researches train the face-related network based on this dataset. Figure 16 shows the example images from VGGFace dataset for six identities.

Figure 16: VGGFace image examples [12]

2.4.2 Deep Learning Based Facial Expression Recognition

In this section, we introduce three main steps for deep-learning based facial expression recognition methods, i.e., pre-processing, feature learning and feature classification. we describe briefly about popular methods for each step.

2.4.2.1 Pre-processing

Before training deep learning networks, we need to pre-processing datasets to normalize irrelevant variations. Backgrounds, illuminations head poses and so on are common influence for training networks, so some pre-processing methods, just like face alignment, data augmentation and face normalization are used to deal with input data.

2.4.2.2 Deep FER Networks

Deep learning networks capture high-level abstractions through neural networks. For facial expression recognition, there are several kinds of neural networks architectures to be used.

Convolutional Neural Networks: For convolutional neural network, there are two types of important layers: convolutional layers and pooling layers. The convolutional layer does most of the computational heavy lifting that is the core block of CNN. With convolution operation, the network learns correlations among neighboring pixels, reduces parameters quantity by sharing weight in the same feature map, and becomes shift-invariance to input images. The pooling layer reduces calculation cost of the network by reducing size of feature maps. Besides convolutional layer and pooling layer, the fully connected layer could convert 2D feature maps to 1D feature maps for classification at the end of networks. There are kinds of CNN architecture to be proposed to solve facial expression recognition problem: Faster region-based CNN, 3D CNN, and so on. Each architecture shows different advantages and disadvantages for facial expression recognition, but most of CNN-based architecture have better performance compared with other deep learning architectures.

Dropout Layer: Dropout layers is a kind of structure which could randomly ignore units during the training phase. More technically, each node are chosen to be dropped or kept with specific probability at each training stage, and reduced network is left

to continue calculating. The dropout layer is proposed by Srivastava et al. [13] to prevent over-fitting.

Dropout layer helps to reduce interdependent learning amongst the neurons by regularizing neural networks. Regularization is a kind of approach to prevent over-fitting in machine learning by adding a penalty to the loss function. Figure 17 compares the difference between standard neural network and dropout neural network. In the training phase, for each training sample, each hidden layer ignores a random fraction of nodes for every iteration.

Based on the structure of dropout layer, networks learn more robust features that are useful in various random subsets. As for learning calculation, the number of iterations is doubled, but training time for each epoch is shorter.

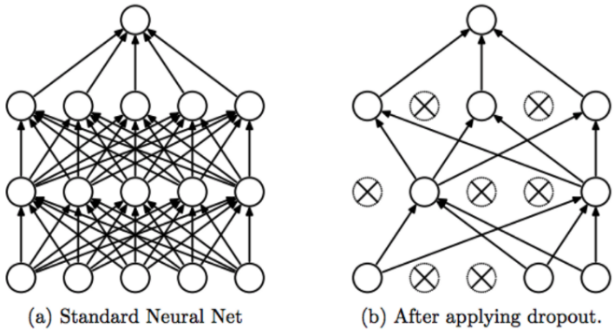

Figure 17: Dropout model [13]

Batch Normalization Layer: Firstly, we explain what internal convariate shift is. In most of neural networks training, we need to do data pre-processing to normalize input. One of data normalization is resembling data to normal distribution (zero mean and unitary variance). This kind of normalization could prevent the early saturation of non-linear activation functions, assuring that all input data is in the same range of values. But in the intermediate layers, the problem appears because the distribution of the activation keeps changing during training. The whole training process is slow because each layer needs to learn new distribution during every training step, and this problem is known as internal convariate shift.

Ioffe et al. [31] proposed Batch Normalization (BN) layers to solve internal

convariate shift. The basic theory of BN layers is normalizing the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. Batch Normalization provides two trainable parameters to each layer, so the normalized output is multiplied by a standard deviation parameter and add a mean parameter. This operation lets stochastic gradient descent do the denormalization by changing only these two weights rather than all the weights, and keep the whole network stable. Besides, we can use higher learning rates since BN layers keep the output of activation stable and reduce over-fitting. According to Ioffer's article, their ImageNet-based model achieves the same accuracy with fewer training steps and reaches higher accuracy with unlimited steps.

ResNet: The ResNet was proposed by Kaiming et al. [14]. Before ResNet, deeper neural networks are difficult to train. When networks go deeper, a degradation problem will be exposed, accuracy get saturated and degrades rapidly. As we can see, in deeper network, the gradient back-propagated to earlier layers is very small and these layers are hard to change. Figure 18 shows the experiment of comparison of two different depth networks. It shows that with the same number of iterations, the network with 56 layers gets worse performance compared with the network with 20 layers.

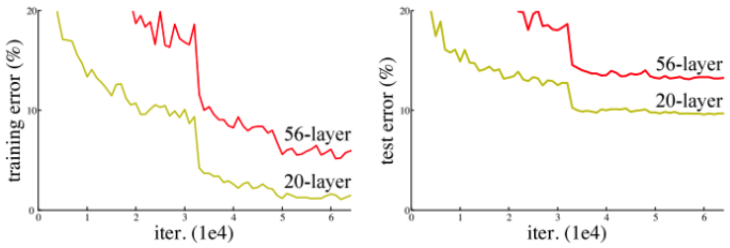

Figure 18: Deeper network with worse performance [14]

In the ResNet, shortcut connection is used to solve the problem of vanishing/exploding gradients. Figure 19 shows the structure of shortcut connection. The output of this block is the following:

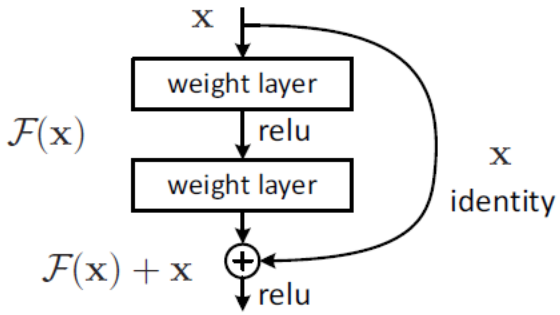

Figure 19: Block of ResNet network [14]

Equation 3 [14] represents the output of the block during training where $F(x)$ and x represent the stacked non-linear layers and the identity function respectively. Equation 4 [14] represents the actually learnt output for weight layers. This structure makes sure the earlier layers can still have the identity x to gradient, even if there is no gradient for the layers. If the identity mapping is optimal, the residuals can easily be pushed to zero $F(x) = 0$ than to fit an identity mapping (x) by a stack of non-linear layers.

$$H(x) = F(x) + x \tag{3}$$

$$F(x) = H(x) - x \tag{4}$$

Figure 20 shows the three structures (VGG-19, VGG-34, ResNet-34) that Kaiming has tested. The one on the left is VGG-19 which is a state-of-the-art method in ILSVRC 2014, and the one in the middle is VGG-34 which is the deeper network version of VGG network. The one on the right is 34-layer residual network (ResNet). The Figure 21 shows the comparison result. When plain network is used, VGG-19 has a better validation error than VGG-34 because of the vanishing gradient problem. On the other hand, ResNet-34 is better than ResNet-19, which means the vanishing gradient problem has been solved by shortcut connections. Comparing the left result with the right result, ResNet-19 has almost the same performance as VGG-19 since there is rarely any vanishing gradient problem.

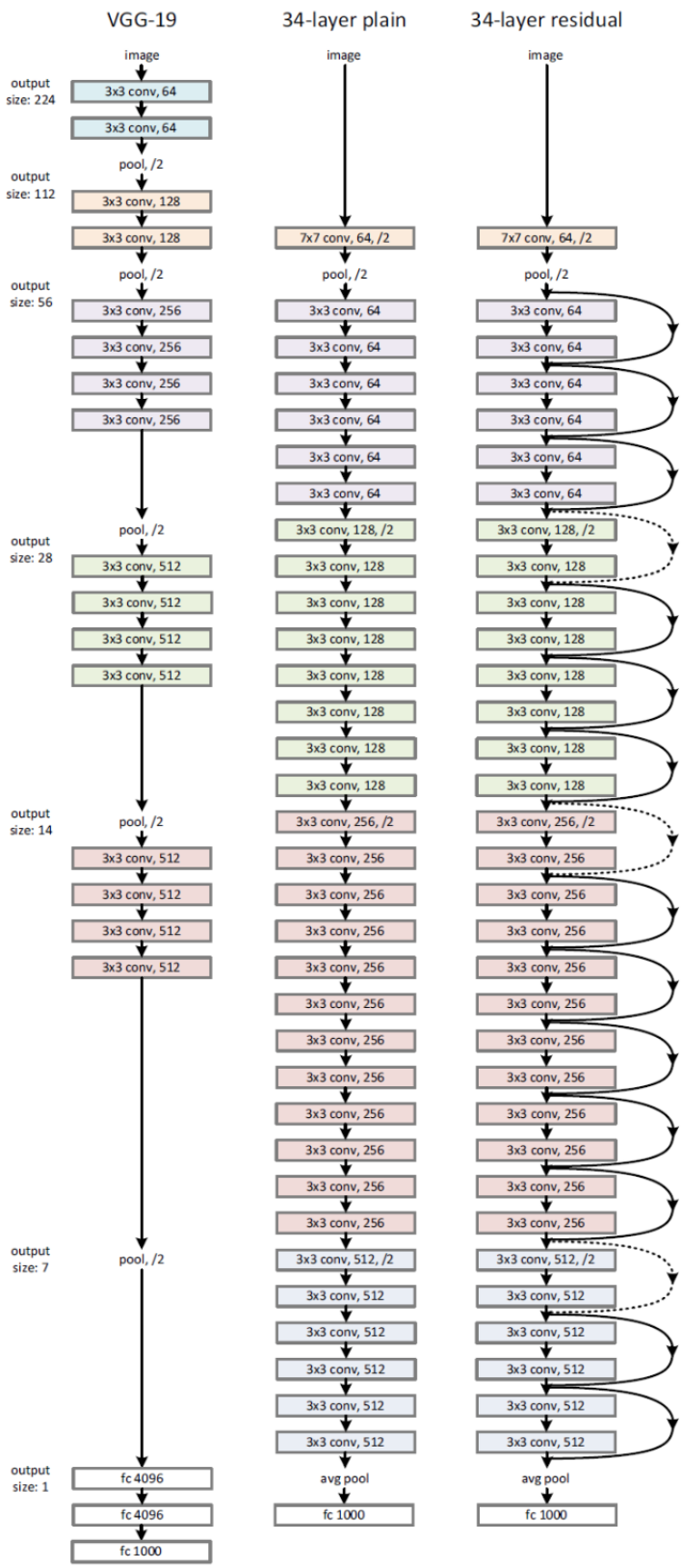

Figure 20: Plain VGG and ResNet structures [14]

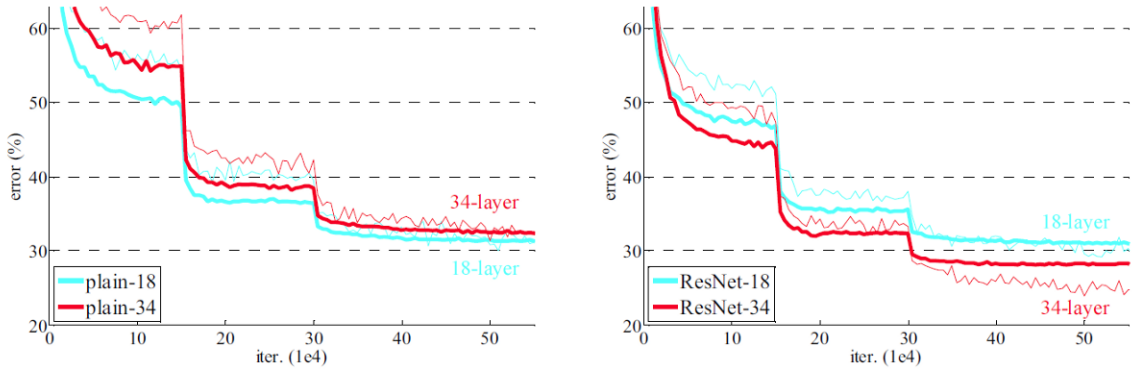

Figure 21: The validation error comparison between VGG-19, VGG-34 and ResNet-34 [14]

2.4.2.3 Facial Expression Classification

For facial expression classification, it tries to classify the feature maps from networks to expression categories. Most of the time we could directly use fully connected layer to calculate results, and it is based on the back-propagation theory of deep learning. In CNN, the most common loss to be used is softmax loss. Besides softmax loss, support vector machine and neural forests could be used to classify expression categories. Support vector machine minimizes a margin-based loss instead of the cross-entropy, whereas deep neural forests use softmax loss with neural forests structure to achieve multiple results.

2.5 Static-Based Deep FER Networks

To recognize facial expressions on static images through deep learning networks, people tried with lots of strategies and network structures to improve speed and accuracy. In this section, we will introduce some good performance studies.

2.5.1 Fine-Tuning

To do facial expression recognition, normal shallow networks can not provide higher accuracy, even when we provide better quality images or larger datasets. However, when we take complex deep neural networks like AlexNet, GoogleNet, and VGG,

which have a huge number of parameters, the result overfits easily if we train them with a small dataset. Therefore, one would use fine-tuning method to train complex networks to solve the over-fitting problem.

For facial expression recognition, there are many existing deep networks training on large face dataset. VGG-Face is used to perform object recognition and is trained on ImageNet [32]. Based on the study of Knyazev et al. [33], fine-tuning VGG-Face networks with additional datasets can have good performance on facial expression recognition. Besides fine-tuning deep networks directly, some studies do multistage fine-tuning to achieve better performance. According to the work of Ng et al. [34], they used two stage fine-tuning strategy with three different datasets and achieved better performance than directly training.

2.5.2 Diverse Network Input

Most of the time, input images are only pre-processed by face alignment and face normalization, which makes input data lack some information, such as textures or invariance of scaling, rotation, occlusion and illumination. Thus, some studies try to diverse input images by withdrawing multiple features as input.

According to G Levi [35] study, Local Binary Patterns (LBP) feature could provide illumination-invariant for facial expression recognition, whereas SIFT [36] could provide scale-invariant. Zeng et al. [37] tried to combine different kinds of feature descriptors together and improved performance. Besides extracting features from global images, splitting faces into regions of interest is a kind of approach to diverse input. Chen et al. [38] tried to split face images into eyebrows, eyes and mouth regions, and obtained 89.31% accuracy in CK+ dataset.

2.5.3 Network Ensemble

Cirecsan et al. [39] achieved near-human performance for image classification for the first time by assembling multiple networks. According to previous research, the diversity of networks and effective ensemble method are two important factors

for network ensemble. In order to have diverse networks, using different training data or architectures is a good way. Besides, pre-processing methods described in Section 2.4.2.1 can generate different kinds of input to train networks. Size of filters, size of neurons and number of layers also enhance network diversity. For network ensemble, finding an optimal set of weights is an important part, SE Kahou[40] proposed a method called random search which was able to exploit the complementary information within the predictions of all models and yielded highest performing technique on the test set.

2.6 Dynamic-Based Deep FER Networks

In this section, we introduce all kinds of dynamic-based deep networks and methods for facial expression recognition. We can classify these methods into three categories: frame aggregation, expression intensity network and deep spatio-temporal network.

2.6.1 Frame Aggregation

Frame aggregation techniques combine feature maps learned from sequential images to predict facial expressions. The quality of each frame decides the accuracy of facial expression recognition. There are kinds of studies proposed to aggregate the output of network in each sequence to improve the performance.

According to the study of Kahou et al. [40], for each convolutional network, 7-class probability vectors are aggregated to a fixed-length representation for each video-clip and order their representations based on time. The result of frame aggregation is represented by concatenating the averaged probabilities of a vector of 70 features. And three years later, Kahou et al. [41] developed another way to do frame aggregation by repeating frames uniformly to get 10 frames in total, which is able to solve the insufficient frames with detected faces.

2.6.2 Expression Intensity Network

Most of the time, the videos in databases for facial expression recognition use fixed high-intensity to get better quality, but the intensity of videos keep changing in the real world. Some of the studies tried to develop intensity-invariant facial expression recognition methods.

Yu et al. [42] proposed Deeper Cascaded Peak-piloted Network (DCPN) for weak expression recognition. There are three main techniques for DCPN: supervise non-peak expression of the same type and subjects with peak expression; design a special back-propagation algorithm that makes intermediate-layer feature maps close to corresponding peak expression; use cascaded fine-tune to prevent the network from over-fitting. DCPN hits 99.6% accuracy on CK+ database.

2.6.3 Deep Spatio-Temporal Network

The differences between normal convolutional neural networks and deep spatio-temporal networks is that the later one takes a range of frames in a temporal window as a single input, so deep spatio-temporal networks do prediction not only by spatial relation feature, but also the temporal relation. There are lots of studies about deep spatio-temporal network, such as RNN, C3D, and network ensemble. We introduce the representation studies of each method and discuss their features.

2.6.3.1 RNN and C3D

The RNN is a kind of artificial neural network where connections between nodes form a directed graph to extract temporal features in sequential frames. The decision reached in a recurrent network at time step $t-1$ affects the decision at time step t. Based on this architecture, recurrent networks have two sources of input, the present and the recent past, that is how RNN extract temporal features. After the development of RNN, a variation of recurrent networks called long short-term memory (LSTM) was proposed by Sepp et al. [43]. One of the most important improvements for LSTMs is preservation of the error that can be back-propagated through time

and layers. LSTMs allow recurrent neural units to continue learning over many time steps, which opens a channel to link causes and effects.

It is always difficult for the recurrent neural network to train because of vanishing and exploding gradients. Quoc et al.[44] proposed a method to use the identity matrix or scaled version to initialize the recurrent weight matrix to solve hard-learning problem in RNN. Some works tried to adjust LSTM structure to improve performance. Zhenbo et al. [45] proposed an end-to-end architecture called Spatio-Temporal Convolutional features with Nested LSTM (STC-NLSTM). This architecture used 3DCNN to extract spatio-temporal features from video and modeled the dynamics of expressions with two sub-LSTMs, T-LSTM and C-LSTM. T-LSTM modeled the temporal dynamics of the spatio-temporal features in each convolutional layer, and C-LSTM integrated the outputs of all T-LSTMs to encode the multi-level features. Overall, STC-NLSTM achieved 99.8% on CK+ database.

Actually, the advantages of convolutional neural networks in computer vision is remarkable compared with recurrent neural networks. In 2015, Du et al. [46] proposed an effective approach for convolutional neural networks to learn spatio-temporal feature, called 3D ConvNets. The key points in 3D ConvNets are the structure of the 3D convolution layer and pooling. Unlike 2D kernels, 3D convolutional layer and pooling layer use 3D kernels with shared weights, which is able to capture the temporal features. Besides the kind of convolution kernel whose weight is sharing along the time axis, there is proposed deep temporal appearance network (DTAN) whose 3D filters vary in importance over time proposed by Jung et al. [47]. By using this kind of kernels, each filter plays a different role depending on the time.

2.6.3.2 Network Ensemble

Network Ensemble in facial expression recognition is almost the same as ensembling all kinds of networks. The outputs of the whole architecture are based on combination of various networks. The earliest network ensemble method in facial expression recognition was proposed by Simonyan et al. [48]. Two CNN are trained to recognize human action in sequential frames. One CNN is provided to extract dense optical

flow on the multi-frame, while the other stream of CNN extracts appearance features.

In 2019, There is a multi-channel deep neural network that learns and fuses the spatial and temporal features for facial expression recognition in image stream proposed by Sun et al. [49]. The network extracts optical flow from the changes between the peak expression face image and the neutral face image as the temporal information, and uses emotion images as the spatial information. Then they fuse the information to predict the expression result.

2.7 Related Libraries

In this section, we introduce the related softwares and libraries used to implement our solution. They are Freenect, OpenNI2, NITE2, RealSense SDK, and CPython.

2.7.1 Freenect

Microsoft Kinect is one of the most popular RGB-D cameras presently, which is developed by Microsoft Company. It is widely used to develop motion-related games and applications. Freenect is an open-source driver for Kinect, which could help Kinect to work on Linux, macOS and Windows. The features that Freenect library provides are color image processing, IR and depth image processing, registration of color and depth images, and multiple GPU and hardware acceleration implementations for image processing.

2.7.2 OpenNI2

The main usage of OpenNI2 is to access compatible depth sensors of PrimeSense Inc. The application could use it to initialize a sensor and receive depth, RGB, and IR video streams from the device. OpenNI2 not only provides a single unified interface to sensors, but also an interface that developers could use for implementing third party middleware. The main three functions that OpenNI2 provides are: voice and voice command recognition, hand gestures, and body motion tracking.

2.7.3 NITE2

Like we mentioned in Section 2.7.2, there are many developed middlewares that could provide powerful functions for users and NiTE2 is one of them. NiTE2 was developed by PrimeSense Inc. The features provided are human detection, posture estimation, hand tracking, and gesture detection. Although NiTE2 has been closed source for many years, there are still many useful functions we can use, such as gesture detection and skeleton detection.

2.7.4 RealSense SDK

RealSense SDK is a cross-platform library for RealSense cameras, which is also used by OpenISS. This SDK has many functions that users could use to develop applications. There are five main categories in SDK: viewer, depth quality tool, debug tools, code samples, and wrappers.

In the viewer part, users could view the depth stream from multiple cameras, record and playback camera depth stream data files, and control the camera settings. As for depth quality tool, the z-accuracy, spatial noise, fill rate and distance to target are already provided by this SDK, which saves a lot of extra work for users. RealSense SDK also provides debug tools and allows users to access enumerate devices, fireware logger, data collection and basic terminal. All of them cover almost everything to tackle the possible issues. Finally, RealSense SDK supports multiple programming languages and wrappers, such as C, C++, Python, Node.js and etc.. In OpenISS, this SDK is integrated with C++.

2.7.5 CPython

CPython is implemented by Python and C, which could be defined as an interpreter and a compiler since the Python code part is compiled into binary code before being interpreted. The foreign function interface supports several other programming languages. In OpenISS framework, CPython is used to pass the Python data structure to C++ data structure and connect two different programming languages.

2.8 Summary

In this chapter, we gave a brief review on facial analysis domains, especially on face detection, facial landmark detection, and facial expression recognition. In face detection part 2.2, we gave an introduction about the most common face detection algorithms, which are based on haar-like detectors, HOG detectors, and convolution neural network. And for facial landmark detection part 2.3, we explained the three main categories, active appearance model methods, constrained local methods and regression-based methods. As for facial expression recognition part 2.4, we mainly researched for the deep learning based facial expression recognition works in this area. We gave some details about datasets used in facial expression recognition part, and discussed the main steps for developing a deep learning based facial expression recognition solution. Besides that, there are two main categories for facial expression recognition, which are static-based facial expression recognition and dynamic-based facial expression recognition. We showed the basic network structures used in these two areas in 2.5 and 2.6 separately. Lastly, we introduced the libraries that we used in our work. In the next chapter, we will explain how we create our solution based on these concepts.

Chapter 3

Framework Design

To achieve our goals mentioned in Section 1.4, we would like to propose an original design for our solution. In this chapter, we will first explain why we decided to use a framework solution. After, we introduce the architecture design of our proposed framework, and explain how it addresses our requirements. Then, we present in more details the structure of the modular components for the core framework. Lastly, we list and explain all module components within the facial analysis specialized framework.

3.1 Why Framework Solution?

To achieve our goal, we decided to implement our solution using a framework design approach. Below, we will explain how a framework solution can enable us to meet each of the functional requirement listed in Section 1.4.

A software framework, with inherent generic architecture and development environment, allows us to take it at our ease for its provided generic functionality, or to implement customized code within its structure and make our application-specific software. It is a commonly utilized tool in the software engineering and computer programming industry with wide options of customization while providing a guided structure to enforce the application flow [50] [51] [52]. According to the work of Pree et al. [53], a software framework consists of two components:

- Frozen spot: Within a framework, frozen spots define the overall architecture

of a software system, that is to say its basic components and the relationships between them. These remain unchanged (frozen) in any instantiation of the application framework.

- Hot spot: Within a framework, hot spots represents those parts where the programmers using the framework add their own code to add the functionality specific to their own project.

Based on analyzing our goal and usage scenarios mentioned in Chapter 1, we find that the features of the framework design solution perfectly matches our requirements. From the information we gathered, the three main features of the framework design solution are as follows:

Inversion of control: In a library or application solution, the control flow of the program is dictated by the caller, whereas it is dictated by the framework's frozen spots in a framework solution [54]. In order to realize the landmark-based facial action mapping scenario mentioned in Section 1.5.1, we need to satisfy the following concrete requirements: face detection in *FR1* and facial landmark detection in *FR2*. And in *FR3*, we determined to provide a solution to connect the processes of face detection and facial landmark detection seamlessly, which means there exists a data flow from the former output to the latter's input. But from the user's perspective, they only want to obtain a valid result to continue with their proceeding work rather than to take extra time to process intermediate results between the processing phases. In this case, the framework solution could help them achieve a clean and simple program flow by setting the process flow in advance. It is like boarding on a non-stop flight rather than a connecting flight. Under such consideration, having a predictable and controllable program flow that is important. Using a framework design approach, the framework's frozen spots architecture can allow the user to defined such a control flow, which is then enacted automatically by the framework's inversion of control.

Extensibility: A user can extend the framework, usually by selective overriding or by adding specialized code to provide specific functionality [54]. In our case, we have mentioned in Section 1.5.4 that one of our non-functional requirements

is extensibility, which should allow developers to add new functionality or modify existing functionality without impairing the existing system functions. We need an extensible solution which can support various kinds of cameras (**FR8**) as well as facial analysis algorithms (**FR6** and **FR7**). For example, one of the requirements is to allow the utilization of deep learning algorithm to increase the accuracy in expression matching, as we move forward, the demand for more and more accurate results will lead the incorporation of new algorithms in the system, which should not impact other pre-existing solutions. Framework extensibility provides us with such a convenience that enables us to easily change or add specialized implementations and cameras to provide the required functionality.

Non-modifiable framework code: The framework code, in general, is not supposed to be modified, while accepting user-implemented extensions. In other words, users can extend the framework, but cannot modify its existing code [54]. A framework solution has hot spots for addition and modification to allow developers to implement their own application, and it also has frozen spots that are fixed unchangeable code. The frozen spots controls the flow of the program by acting as a skeleton. This structure allows developers to add their own extensions (i.e. hot spot instances) to the existing skeleton to make a complete solution. This increases the reliability of our solution and reduces the programming and testing effort.

Based on the above features, we can conclude that a framework solution addresses our needs and requirements. Our abstract structure of the framework solution can be defined with two major components: core framework formulated by frozen spots and specialized implementations as hot spot instantiation. Within the core framework, frozen spot helps define infrastructures like cameras, common data structure, viewer portal and etc. Then we add specialized implementations such as specialized algorithms on top of the core framework to build the complete framework solution that achieves desired functionality. The framework is designed so that it can eventually be used for the implementation of a wide array of image processing needs for the computer animation and augmented reality applications. In our work, we only focus on facial analysis.

3.2 OpenISS Framework Design Overview

Figure 22 shows the overall design of our framework solution named the OpenISS framework. It is designed to implement three kinds of computer analysis: facial analysis, body analysis and gesture analysis. The OpenISS framework has two major components. The first is the core framework, which consists of frozen spots. The second is a group of three analysis modules hot spots.

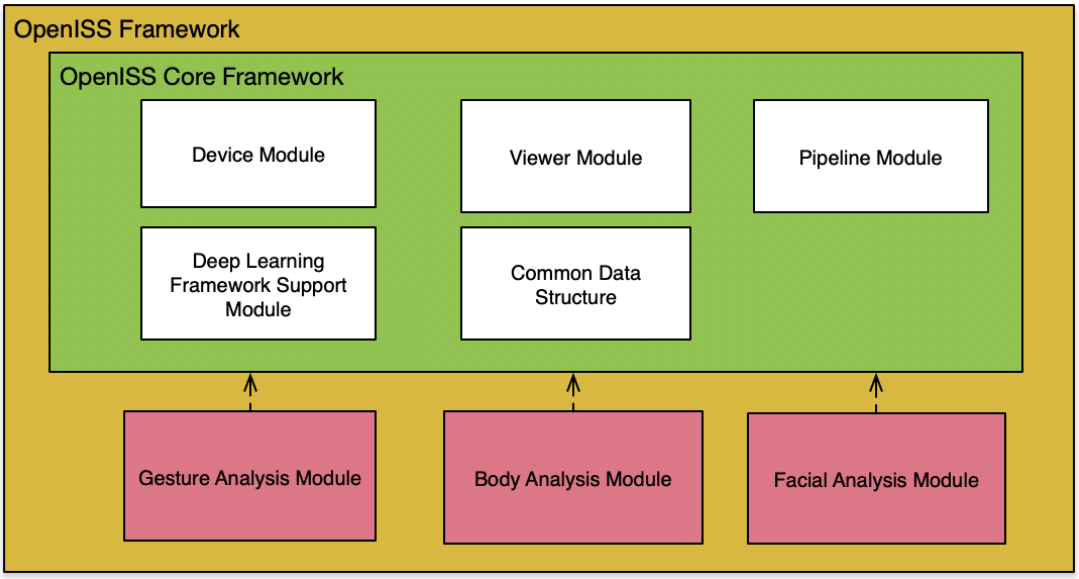

Figure 22: Framework structure diagram

The core framework provides fundamental service modules for the entire framework, here are their basic definitions:

- Device Module: Provides the device extensibility of our framework, which could support different camera devices or add new cameras to be used as OpenISS camera input.
- Common Data Structures Module: Provides the framework-wise data structures for our own algorithms which were adapted from existing lower level libraries and software.
- Viewer Module: Provides the framework's visualization abstraction, which in our case can be used to help visualize our system's result, and ultimately allows

the user to check their output (i.e. images, videos).

- Deep Learning Support Module: Provides support for users to implement or invoke deep learning based solutions.

- Pipeline Module: Provides an infrastructure to define and control the task flow. It provides control of flow from the entry for the program until the end of the entire process, while being responsible for constructing different functional pipelines.

As for the other three analysis modules in OpenISS framework, facial analysis module, gesture analysis module and body analysis module, each of them provides different analysis functionalities. For our facial analysis focus, the facial analysis module provides face detection, facial landmark detection, and expression recognition, which are all kinds of facial analysis functionalities that satisfy our research need.

Note that the facial analysis module combined with the framework core forms the **OpenISS Facial Analysis Specialized Framework**. This naming convention will be used for our custom designed framework from now on. As shown in the Figure 23 below.

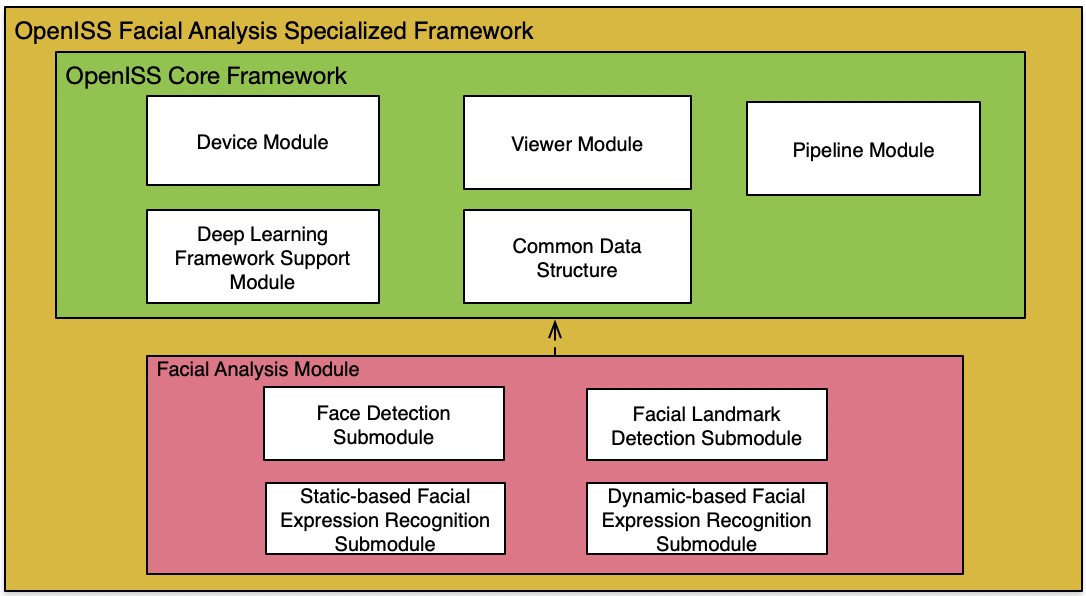

Figure 23: OpenISS facial analysis specialized framework

3.3 OpenISS Core Framework

In this section, we explain the details of the five modules in OpenISS core framework: device module, viewer module, pipeline module, deep learning support module and common data structure module.

3.3.1 Device Module

The device module provides functionality for adapting different data structures of various camera devices to provide common APIs for data processing. For example, the data streams provided by different camera devices are various and need to be analyzed and processed by different SDKs and libraries. The library for transferring Kinect v1 data stream is Freenect, the library for Kinect v2 data stream is Freenect2, and web cameras can directly get image stream without any SDK or library. This requires us to provide a specialized adapter to analyze different data streams for each type of camera device we use. Figure 24 shows the structure of device module.

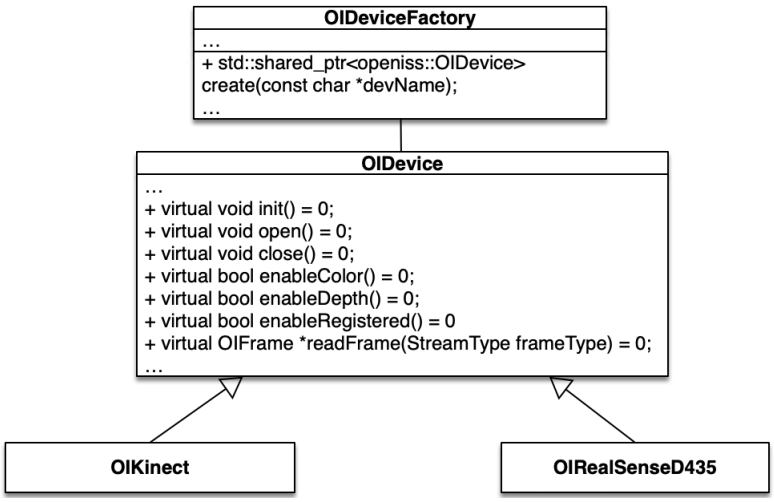

Figure 24: Device module structure

Class *OIDevice* is an abstract class, which contains the following functions: Function *init()* is used to initialize different camera devices before using it. Functions *open()* and *close()* are used to control the opening and closing of the camera device. Functions *enableColor()*, *enableDepth()*, and *enableRegistered()* enable

device to capture different types of frame stream, like RGB stream, depth stream and IR stream. Function *readFrame(StreamType frameType)* allows users to obtain different frames by passing different enumerate values of the stream type, which represents the type of the stream's frame elements. For example, if users pass value *openiss::COLOR_STREAM* of *StreamType* to *readFrame*, then they can get a *OIFrame* instance with captured RGB frame information inside.

This module incorporates the factory design pattern, which removes the instantiation of actual implementation classes from user-implemented code. In this way, we can manage different concrete subclasses more easily. For example, now, our framework can support three types of camera devices: Kinect v1, Kinect v2, and RealSenseD435. Assuming we want to use RealSenseD435 in our system as the camera device, there are only two steps to implement: create a *OIDeviceFactory* instance, get the *OIDevice* pointer that points to *OIRealSenseD435* instance by inputting "RealSenseD435" into the *create(const char *devName)* function of *OIDeviceFactory* instance. After these two steps, we can get the *OIDevice* pointer and use it to transfer RealSenseD435's data structure into our common data structure.

On the other hand, there are only two steps for device module to provide support for new device: (1) create a subclass inheriting *OIDevice* and implement the virtual methods (2) add the logic to instantiate the new device in *OIDeviceFactory* class.

3.3.2 Common Data Structures Module

The OpenISS framework modules need to communicate in a uniform and consistent way, which leads to the requirement for having common data structures. These common data structures are accepted by all modules within the framework. They enable a uniform representation of various data captured by input device and ensure the standardization of the intermediate result between modules from the capture until the final output.

The core class is *OIFrame*, whose structure is shown in Figure 25. Each *OIDevice* will transfer data stream into this common data structure class. And the data structure inside *OIFrame* is saved as *cv::Mat* which belongs to the OpenCV library.

Besides, RGB image, depth image and IR image (image captured by infrared imaging technique) are also saved as different *OIFrame* instances.

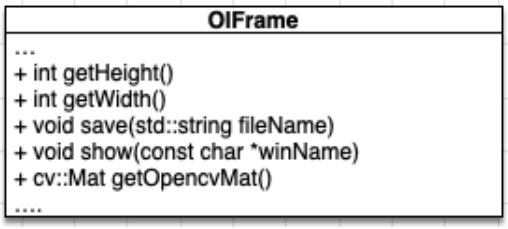

Figure 25: OIFrame class

3.3.3 Deep Learning Support Module

Based on our functional requirements **FR7**, we want to support deep learning implementations(**FR7**) in our framework. According to our research, Python has better support for deep learning libraries such as TensorFlow, Keras and etc. compared with C++. Thus, it is common for people to train or to execute deep learning algorithms under Python. Since our framework is implemented with C++ for better execution speed, we need this module to realize data transfer from our framework input in C++ to Python based deep learning implementations, and allows the computation result from Python deep learning script to transfer back to our framework.

In order to bridge between C++ and Python, we integrate CPython into our framework. The introduction of CPython are mentioned in Section 2.7.5. Figure 26 shows how an adapter written with C++ could communicate with *Python implementation* written with Python. And in Section 4.1.1.3 and 4.1.1.4, we will give more concrete examples for this module.

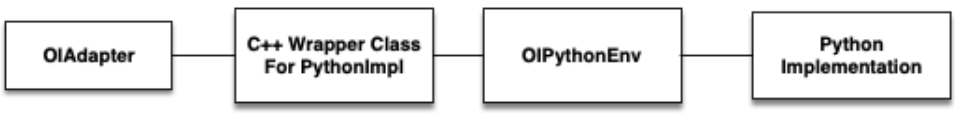

Figure 26: Data flow between C++ and Python

In this module, *OIPythonEnv* is the core class. Figure 27 shows the main functions

of *OIPythonEnv*. With this class, you can instantiate an object of a Python class, load a Python function, invoke a Python method, etc. In order to integrate Python-based deep learning implementations into the framework, you can create a C++ wrapper, which contains a *OIPythonEnv* instance. Then set up the Python script file name and target C++ function name in *OIPythonEnv*, and so *OIPythonEnv* will know which python file it can access and which function it could invoke. Finally, you can access your Python-based implementations in our C++ framework. More details will be discussed in Section 4.1.1.3 and Section 4.1.1.4.

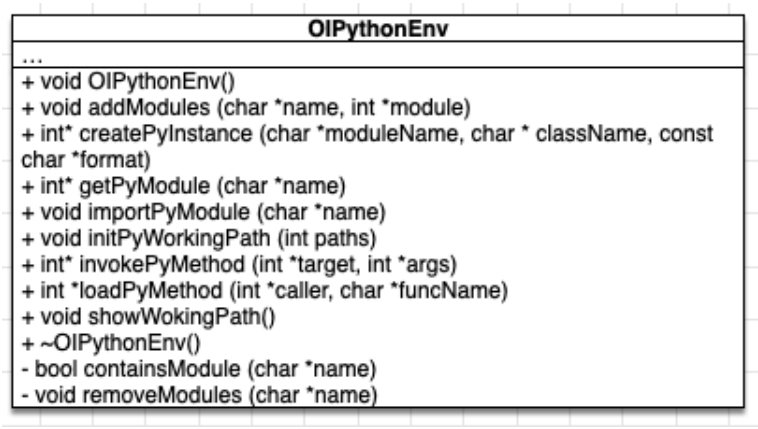

Figure 27: OIPythonEnv class

3.3.4 Viewer Module

The viewer module is responsible for displaying the resulting data for visualization purpose. Figure 28 shows the structure of viewer module.

OIViewer is an abstract class which provides functionalities around data structure visualization. Via *OIViewer*, we are able to look at RGB image, depth image and other image data types that exist in our framework on the same screen. *OIViewerFactory* is used to create concrete subclass instances for *OIViewer* by passing different strings. Currently, we only provide one implementation, *OIOpenCVViewer*, based on the OpenCV library.

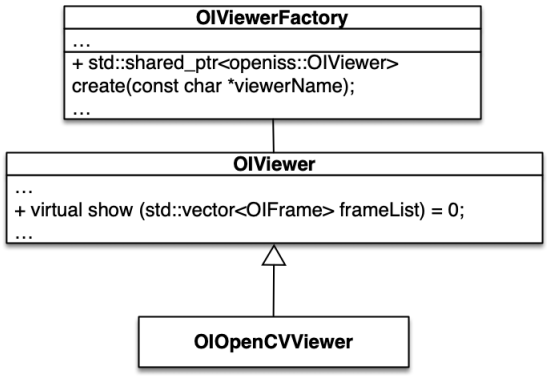

Figure 28: Viewer module structure

3.3.5 Pipeline Module

The pipeline module is the framework's pipeline definition/execution module. It allows developers to set up an execution sequence within the framework. It determines and then controls the whole process execution from the entry point to the final output of our framework solution. Figure 29 shows the structure of pipeline module. *OIPipeline* is an abstract class used to execute an image processing process in the OpenISS framework. The *execute()* function starts the execution of framework solution and the *close()* function is used to terminate the execution.

OIPipelineFactory is used to create the concrete subclass of *OIPipeline*. In the function of *createPipeline(const char *functionName, std::vector<string> paramStrings)*, parameter *functionName* determines the type of *OIPipeline*'s subclass, and paramter *paramStrings* determines the functionalities and execution order inside concrete instance. We are going to discuss more details with specific examples in Section 4.1.2 and Section 4.2.

3.4 Facial Analysis Specialized Framework

In this section, we introduce the design of our facial analysis specialized framework. Like we mentioned at the end of Section 3.2, the facial analysis specialized framework consists of two parts: OpenISS core framework and facial analysis module. The

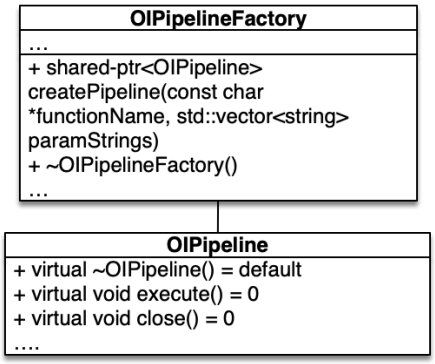

Figure 29: Pipeline module structure

combination of these two constitutes the Facial Analysis Specialized Framework. We have given details of the OpenISS core framework in Section 3.3, so we introduce the facial analysis module here. We shall explain how we instantiate this specialized framework and create concrete applications through it in Chapter 4. According to Figure 22, the facial analysis module consists of four submodules (face detection submodule, facial landmark detection submodule, static-based facial expression recognition submodule, dynamic-based facial expression recognition submodule).

Figure 30 shows the general relationship between our framework common data structure, *OIFrame*, and the specialized data structures for each analysis module. Generally, after initial processing, *OIFrame* passes information into each specialized data structure of each analysis module, so each module can use this frame information to do corresponding analysis. Our focus is the facial analysis module which has *OIFace* as its common data structure. As data arrives in facial analysis module, *OIFace* copies the frame information from *OIFrame*. After that, *OIFace* obtains the produced facial analysis data from each submodule, such as bounding box location of detected face, facial landmarks, and expression category probabilities. We will explain the details about *OIFace* and the functional submodules in this section.

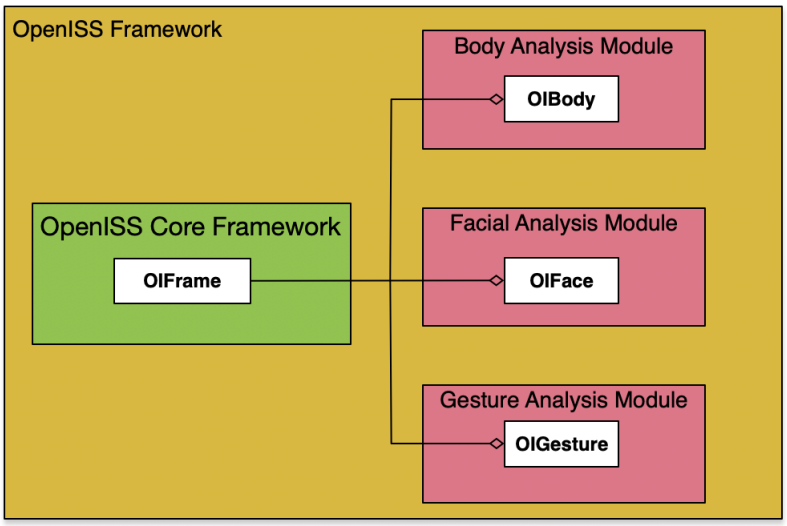

Figure 30: Data relationship between core framework and analysis modules

3.4.1 OIFace

OIFace can store different types of images (RGB image, depth image, IR image) based on the input data of camera devices. For example, web camera only provides RGB image, whereas Kinect v1 could provide RGB image, depth image and IR image at the same time for one frame. On the other hand, there could be many faces that exist in one frame.

OIFace
- std::vector<std::vector<std::Point2f>> faceVector - std::vector<std::vector<cv::Point2f>> facialLandmarkVector - std::vector<std::vector<double>> expressionVector - cv::Mat* rgbImg - cv::Mat* depthImg - cv::Mat* irImg ...
+ cv::Mat& getRGBImg() + cv::Mat& getDepthImg() + cv::Mat& getIRImg() + void setRGBImg(const OIFrame& oiFrame) + void setDepthImg(const OIFrame& oiFrame) + void setIRImg(const OIFrame& oiFrame) + std::vector<std::vector<std::Point2f>> getFaceVector + std::vector<std::vector<cv::Point2f>> getFacialLandmarkVector + std::vector<std::vector<double>> getExpressionVector ...

Figure 31: OIFace class

The variable *faceVector* represents a vector which is used to store the bounding

box locations for all faces in a captured frame. The nested vector is used to save the four location points of one bounding box. The outer vector stores all these nested vectors which belong to each of the different faces in the frame. Variable *facialLandmarkVector* has the same structure as variable *faceVector*. The nested layer vector is used to store facial landmark location points for one face and the outer vector is used to store all these nested vectors which belong to different faces. It is the same for variable *expressionVector*, but the nested vector is used to store the expression categories probability for each face. For example, if the nested vector stores five numbers, (0.1, 0.2, 0.2, 0.4, 0.1), it represents the corresponding probability value calculated for each expression category. For our example, it means: *expression 0* is 0.1, *expression 1* is 0.2, *expression 2* is 0.2, *expression 3* is 0.4, and *expression 4* is 0.1.

3.4.2 Functional Submodules of Facial Analysis Module

In this section, we will discuss the common structure design of the four submodules of the facial analysis module. Then we shall explain why we landed on this structure design. The details of instantiating each functional submodule will be discussed in Chapter 4.

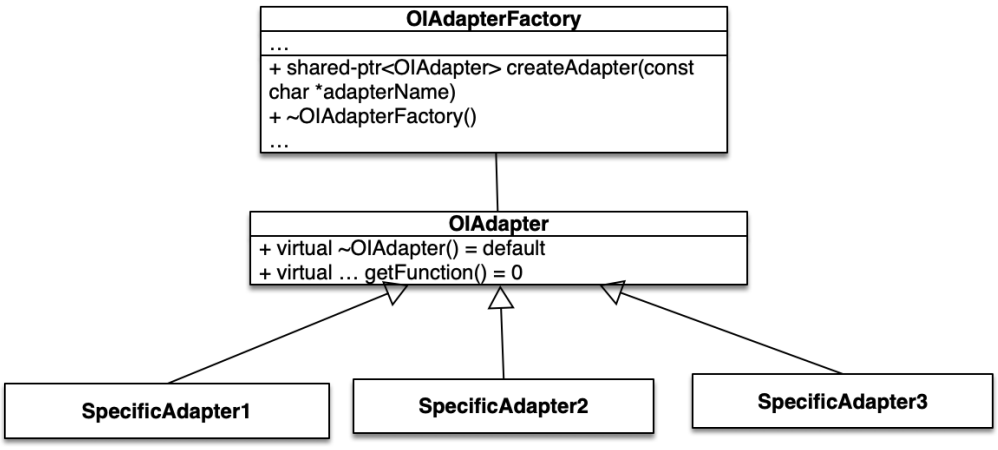

Figure 32: Common structure of functional submodule

Figure 32 shows the common structure of facial analysis submodules. As we can see from this figure, all our facial analysis submodules follow *factory* design pattern. For each submodule, there is one *AdapterFactory* and one *Adapter* abstract superclass. In this way, we could switch among different adapters and add new adapters with different implementations easily. For example, assuming that some developers want to integrate their facial analysis implementation into our framework. Since we employed the factory design pattern, you still need to add the logic to instantiate the appended device in *OIDeviceFactory* class as well as the device-related memory deallocation when it is useless. There are two steps for integration:

- create a new adapter inheriting *OIAdapter* and implement the virtual methods.
- add the logic to instantiate the new adapter in *OIDeviceFactory* class.

As for the switching between existing implementations, the user could easily create different instantiation of *OIAdapter* through *AdapterFactory*. More details will be discussed in Chapter 4. Here are the structure of four functional submodules.

3.4.2.1 Face Detection Submodule

This submodule is aiming to get the bounding box location points of all the faces in one frame. Figure 33 shows the structure of face detection submodule.

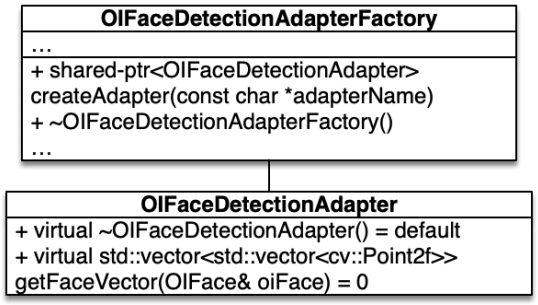

Figure 33: Face detection submodule structure

Class *OIFaceDetectionAdaptorFactory* is used to create concrete subclass of *OIFaceDetectionAdater* and *OIFaceDetectionAdapterFactory* will create different

subclass objects according to different input parameters of the function *createAdapter(const char *adapterName)*.

OIFaceDetectionAdapter is an abstraction class which contains a main function: *getFaceVector()*, which is used to get bounding box for all faces in one frame. The input for this function is *OIFace*, which contains the necessary information, like RGB image, depth image or IR image for face detection algorithm. And the detected bounding box will be stored inside the variable *faceVector* of *OIFace* object, and the copied result be returned.

3.4.2.2 Facial Landmark Detection Submodule

This submodule is aiming to get the facial landmark location points which are the key points on a face. Figure 34 shows the structure of facial landmark detection submodule.

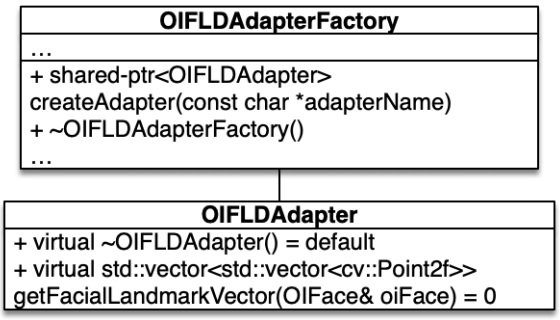

Figure 34: Facial landmark detection submodule structure

The class *OIFLDAdapterFactory* is used to create concrete subclass of *OIFLDAdapter*, where "FLD" represents facial landmark detection. The class *OIFLDAdapterFactory* will create different subclass objects according to different input parameters of the function *createAdapter(const char *adapterName)*.

OIFLDAdapter is an abstraction class. The main function for this class is *getFacialLandmarkVector*. The input for this function is *OIFace* which contains related facial information for facial landmark detection. In order to get the facial landmark location points, the algorithm needs the detected face images, which could be obtained from the *OIFace* object. And after getting the result of the

facial landmark detection algorithm, these landmarks will be stored in the variable *faceFacialLandmarkVector* of the *OIFace* object and also be returned

3.4.2.3 Static-based Facial Expression Recognition Submodule

This submodule is aiming to get the probabilities of facial expression categories for each face in one frame. Figure 35 shows the structure of this submodule.

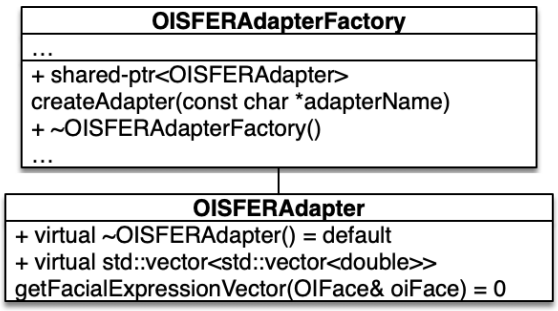

Figure 35: Static-based facial expression recognition submodule structure

The class *OISFERAdapterFactory* provides the function for creating concrete subclass of *OISFERAdapter*. It can create different subclass objects according to different input parameters of the function *createAdapter(const char *adapterName)*.

OISFERAdapter is an abstract class. The main function for this class is *getFacialExpressionVector()*. As we discussed in related work Section 2.5, static-based facial expression recognition could recognize the expression categories based on single human face image.

By passing an *OIFace* instance to the *getFacialExpressionVector()* function, the static-based facial expression recognition algorithm can receive the detected face image from variable *faceVector* of *OIFace* and calculate the probability for each expression category. The result will be stored in variable *expressionVector* of *OIFace*.

3.4.2.4 Dynamic-based Facial Expression Recognition Submodule

Unlike static-based facial expression recognition (FER) that only classifies the single frame expression to a category, dynamic-based FER focuses on the underlying

expression progression, further analyze the facial features change across the frames to determine the trend in expression change, and assign the trend demonstrated over the frames to a certain expression category.

Based on the characteristics of these two FER approaches, static-based FER takes a single frame as input whereas dynamic-based FER takes frame sequences as input. Therefore, they belong to two submodules with two different abstract classes.

This submodule is aiming to generate calculated probabilities for facial expression categories for the detected face images according to their frame sequence, while supporting multi-face concurrent processing. The definition of dynamic-based facial expression recognition can be found in Section 2.6. The number of input frames depends on the algorithm. Figure 36 shows the structure of this submodule.

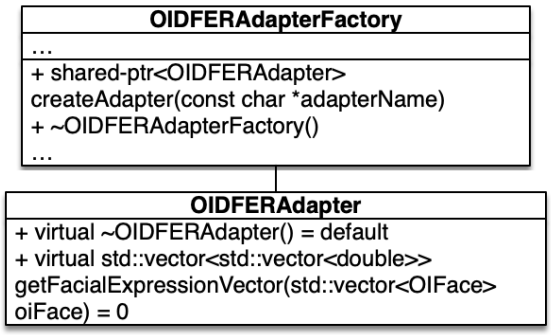

Figure 36: Dynamic-based facial expression recognition submodule structure

Class *OIDFERAdapterFactory* provides the function for creating concrete subclass of *OIDFERAdapter*. It can create different subclasses according to different input parameters of function *createAdapter(const char *adapterName)*.

OIDFERAdapter is an abstract class. The main function for this class is *getFacialExpressionVector*. As we discussed in related work Section 2.6, dynamic-based facial expression recognition could recognize the expression categories in continuous facial expression frame sequences. So the input for this function is vector of *OIFace*. The algorithm can receive the stream of detected facial expression images from the vector of *OIFace* and calculate the probability for each expression category. The result will be stored in variable *expressionVector* of each *OIFace*.

3.5 Summary

In this chapter, we introduced the design of our framework solution. At the beginning, we introduced how a framework solution could fulfill our goals and requirements in Section 3.1. Then we presented the structure of our framework solution, which could be split into two parts: core framework design in Section 3.2 and facial analysis specialized framework design in Section 3.4. In each part, we gave details about the structure, the functions and general usage of each component and how we composed each component together. In the next chapter, we will introduce how we instantiate our framework and how to build concrete applications with framework instance.

Chapter 4

Framework Instantiation and Application

In Chapter 3, we have discussed the design of our framework. In this chapter, we are going to introduce how we create an instance of the framework to implement our facial analysis solutions. We will introduce how we instantiate different functional submodules inside the facial analysis module, which includes the implementation details of each facial analysis methods. Besides that, we will also propose an instance of the pipeline module, *OIFacePipeline*, which is used to provide the key feature known as inversion of control of our framework solution. It can help the users to create their customized applications easily. At the end of this chapter, we give concrete examples of how we build applications using our framework. Figure 37 shows the relationship between framework instance and applications.

4.1 Framework Instantiation

4.1.1 Facial Analysis Module Instantiation

In this section, we explain the methodology used in our facial analysis module instantiation process. From Figure 37, we can see that our facial analysis module consists of four submodules, and for each of them there are 2 or 3 kinds of

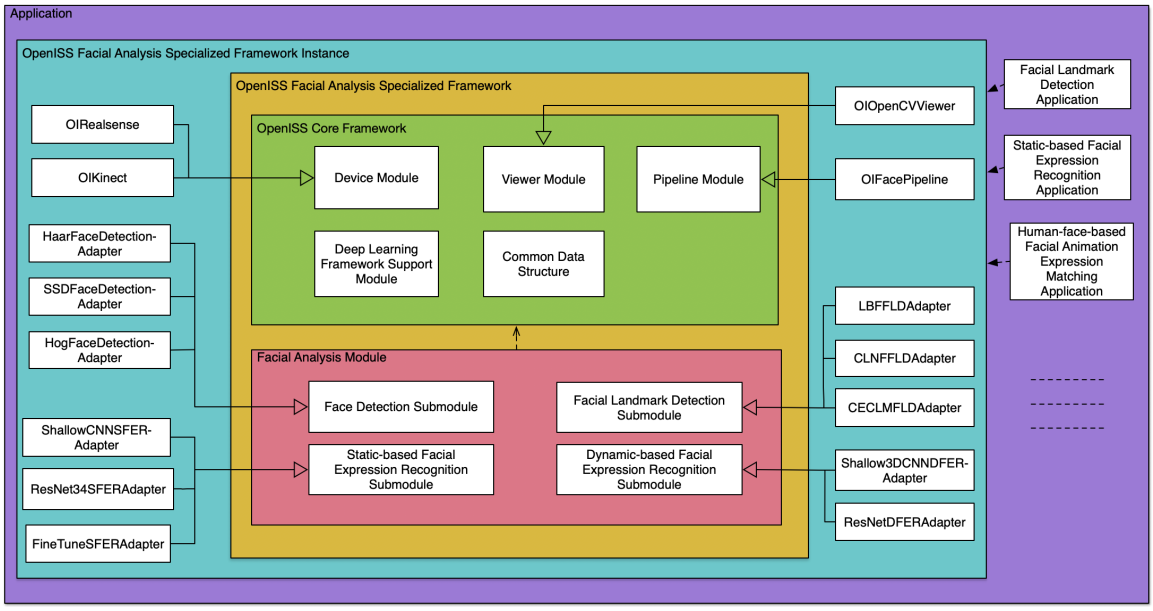

Figure 37: Framework instance

implementations. The details are listed below:

Submodule Name	Number of Implementation
Face Detection	3
Facial Landmark Detection	3
Static-based Facial Expression Recognition	3
Dynamic-based Facial Expression Recognition	2

In the following subsection, we will describe them in detail one by one.

Our framework design is based on the assumption that some users might prefer to add their implemented algorithm into our framework, including using ready-to-use algorithms provided by other libraries. Therefore, we named our concrete subclass as 'adapter' meaning that the user can choose either to write their own code inside our framework, or to simply adapt existing functions of other libraries into our framework.

4.1.1.1 Face Detection Submodule Instantiation

According to our research on face detection area in Section 2.2, we select three face detectors and integrate them into our framework. They are:

- Haar-cascade face detector

- HOG face detector
- Single shot multiBox face detector (SSD)

The reason why we choose them is that these detectors are all widely used in real-world applications. And more importantly, they can satisfy the real-time requirement and achieve good accuracy according to our survey. Figure 33 shows the structure of face detection submodule.

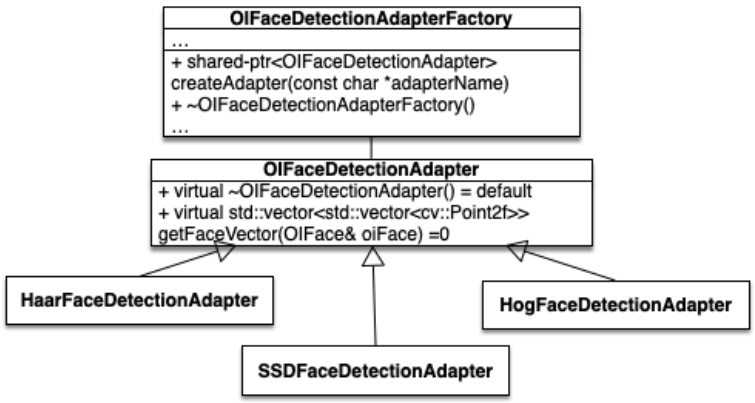

Figure 38: Face detection submodule instantiation

From the implementation point of view, these three face detectors are implemented as three concrete subclasses. They are *HaarFaceDetectionAdapter*, *HogFaceDetectionAdapter* and *SSDFaceDetectionAdapter*. "HaarAdapter", "HOGAdapter", and "SSDAdapter" are three key words which could be fitted into function *createAdapter(const char *adapterName)* of *OIFaceDetectionAdapterFactory* to determine which concrete subclasses instance will be created. In the following paragraph, we will describe their implementation in detail.

(1) Haar-cascade Face Detection

Haar-cascade face detector (HFD) is a technique for face detection depending on handcrafted feature.

In our solution, we are not going to implement the HFD algorithm from scratch but import the OpenCV's implementation into our framework. More specifically, that

can be done by the class named *CascadeClassifier*. The following 3 steps are required to enable this functionality:

1. Create an instance of *cv::CascadeClassifier*
2. Load a Haar-cascade pretrained model into *cv::CascadeClassifier* by *cv::CascadeClassifier::load* method
3. Process the input image with *cv::CascadeClassifier::detectMultiScale* method, and the boundary rectangles for the detected faces will be returned with type *std::vector<cv::Rect>*

(2) Histograms of Oriented Gradients Face Detection

Histograms of Oriented Gradients face Detection is a combination of Histograms of Gradients (HOG) with LinearSVM, which has been mentioned in Section 2.2.2.

In our solution, we adapted the implementation of HOG face detector from the Dlib library [55]. The model they produced was trained on 2825 images from LFW dataset [56]. The adaptation process can be described as following:

1. Get an instance of class *dlib::frontal_face_detector* by using method *dlib::get_frontal_face_detector()*
2. Convert the format of the input image to type *cv_image<bgr_pixel>*
3. Fit the transferred input image into face detector and get the boundary box of the detected face with type *std::vector<dlib::rectangle>*

(3) SSD Face Detection

The face detection method we chose here is Single Shot MultiBox Detector (SSD)[4], which is a CNN-based face detection. More details have been given in Section 2.2.3.1.

The way we implement the SSD face detector is based on the ResNet-10 architecture of the OpenCV library [57]. This architecture provides the OpenCV

library with the ability to load serialized networks models from different deep learning frameworks and calculate a model result. The SSD model that we provide are trained on the WIDER-FACE dataset [58] with 14000 epochs under Caffe deep learning framework. The basic framework integration steps are the following:

1. Load trained SSD model into class *cv::dnn::Net* of OpenCV dnn architecture
2. Preprocess our input image by calling function *cv::dnn::blobFromImage()*
3. After calculation of SSD model, save the detected bounding box of each face into *OIFace*

4.1.1.2 Facial Landmark Detection Submodule Instantiation

According to our research on facial landmark detection area in Section 2.3, we select three face detectors and integrate them into our framework. They are:

- Local Binary Features (LBF) facial landmark detector
- Constrained Local Neural Fields (CLNF) facial landmark detector
- Convolutional Experts Constrained Local Model (CECLM) facial landmark detector

According to our survey, these detectors satisfy our real-time functional requirements (**FR3**) mentioned in Section 1.5.1. Meanwhile, these detectors are all implemented based on 68-points facial landmark model, which is consistent in terms of performance speed and accuracy, and is easy for us to compare the performance result. Figure 39 shows the instantiations of this submodule.

From an implementation point of view, these three facial landmark detectors are implemented as three concrete subclasses. They are *LBFFLDAdapter*, *CLNFFLDAdapter* and *CECLMFLDAdapter*. "LBFAdapter", "CLNFAdapter", and "CECLMAdapter" are three keywords which could be fitted into function *createAdapter(const char *adapterName)* of *OIFLDAdapterFactory* to determine

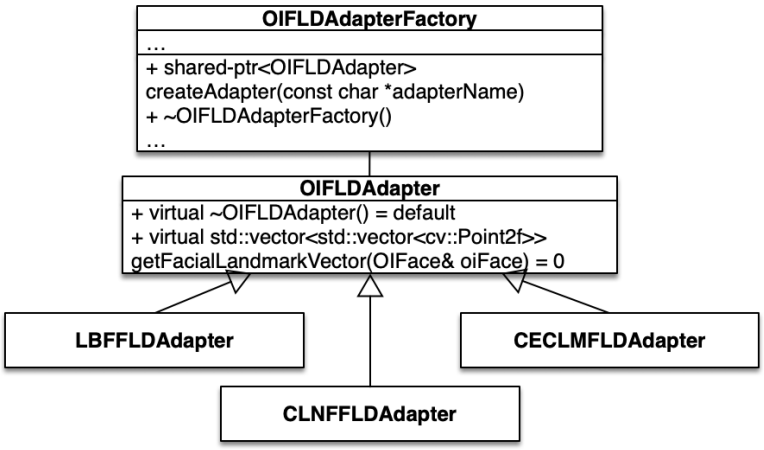

Figure 39: Facial landmark detection submodule instantiation

which concrete subclass instance will be created. In the following paragraphs, we will describe their implementations in detail.

(1) LBF Facial Landmark Detector

The LBF facial landmark detector is based on the paper titled "Face Alignment at 3000 FPS via Regressing Local Binary Features" by Shaoqing at el. [7] and it is also a regression approach. The details of LBF facial landmark detector have been mentioned in Section 2.3.3.1.

The way we implement this detector is based on OpenCV library [57]. In order to implement LBF facial landmark detector, we load the LBF detector model into the *Facemark* class. The implementation steps to integrate this algorithm in our framework are the following:

1. Use *cv::FacemarkLBF::create()* function to create a facial landmark detector instance of class *Facemark*

2. Load a pretrained model of LBF into *Facemark* instance

3. Pass the captured frame and detected face rectangles to the created facial landmark detector, the output of the facial landmark detector will be 68 facial landmarks on each face of the frame

(2) CLNF Facial Landmark Detector

CLNF facial landmark detector is based on constrained local methods (CLM) facial landmark detection algorithm and proposed by Tadas et al. [5]. The details of CLNF facial landmark detector have been mentioned in Section 2.3.2.1.

The way we implement this detector is based on OpenFace library [59]. The implementation steps to integrate this algorithm in our framework are the following:

1. Create a facial landmark detector parameter setter instance of class *LandmarkDetector::FaceModelParameters*
2. Pass the path of a pretrained CLNF model to *LandmarkDetector::FaceModelParameters* instance
3. Create a class *LandmarkDetector::CLNF* instance as facial landmark detector by passing *LandmarkDetector::FaceModelParameters* instance as a variable to constructor function
4. Fit class *LandmarkDetector::CLNF* instance and a grayscale face image into function *LandmarkDetector::DetectLandmarks()*, the output of the facial landmark detector will be the 68 facial landmarks on each face of the frame

(3) CECLM Facial Landmark Detector

The CECLM facial landmark detector is proposed by Zadeh et al. [6]. The details of the CECLM facial landmark detector have been mentioned in Section 2.3.2.2.

The way we implement the CECLM facial landmark detection algorithm is based on OpenFace library [59]. The implementation steps to integrate this algorithm in our framework are the following:

1. Create a facial landmark detector parameter setter instance of class *LandmarkDetector::FaceModelParameters*
2. Pass the path of a pretrained CECLM model to *LandmarkDetector::FaceModelParameters* instance

3. Create a class *LandmarkDetector::CECLM* instance as a facial landmark detector by passing *LandmarkDetector::FaceModelParameters* instance as a variable to the constructor function

4. Fit class *LandmarkDetector::CECLM* instance and a grayscale face image into function *LandmarkDetector::DetectLandmarks()*, the output of the facial landmark detector will be the 68 facial landmarks on each face of the frame

4.1.1.3 Static-based Facial Expression Recognition Submodule Instantiation

According to our research on static-based facial expression recognition area in Section 2.5, we pretrained three deep learning models and adapted them into our framework, which could help us implement our static-based expression recognition functionality. These models are:

- Shallow 2D-CNN model
- ResNet34 model
- Fine-tuning ResNet50 model

The reason why we choose them is that according to our survey, deep learning methods are predominant methods with the highest accuracy in static-based facial expression recognition area. Most methods that are not implemented with deep learning have rather poor performance from the accuracy point of view. Therefore, due to the solid requirement in expression recognition accuracy (**FR6** and **FR7**), we chose to incorporate deep learning methods for our implementations.

On the other hand, with the research time and budget constraints, it is unrealistic to do deep learning training on large human face expression database (multimillion images). In this case, there might be issues like over-fitting if we insist on training the state-of-the-art complex deep learning models on a small facial expression database. To avoid these issues, we chose the shallow 2D-CNN and the ResNet network architecture for model training. The former, as a shallow model with a small number

of parameters, would be less likely to cause over-fitting on small facial expression database. The latter has a special architecture designed for eliminating the over-fitting issue, and will be discussed in detail later in this section. Figure 40 shows the instantiations of this submodule.

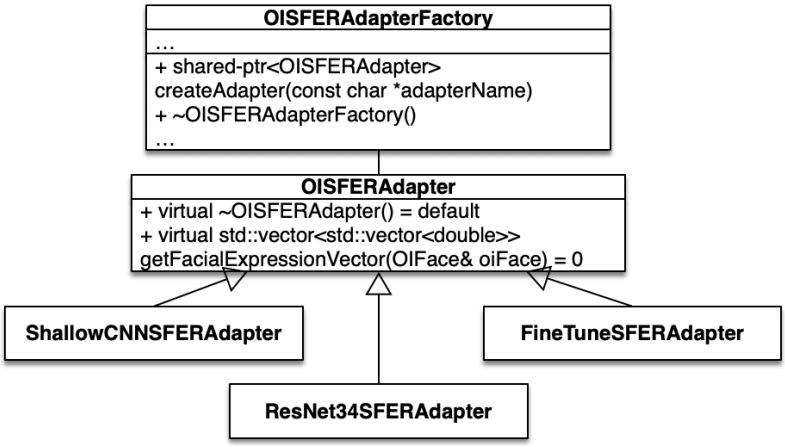

Figure 40: Static-based facial expression recognition submodule instantiation

From an implementation point of view, these three face detectors are implemented as three concrete subclasses. They are *ShallowCNNSFERAdapter*, *ResNet34FERAdapter* and *FineTuneSFERAdapter*. "ShallowCNNAdapter", "ResNet34Adapter", and "FineTuneAdapter" are three keywords which could be fitted into function *createAdapter(const char *adapterName)* of *OISFERAdapterFactory* to determine which concrete subclasses instance will be created. In the following paragraphs, we will describe their implementation in detail.

(1) 2D-CNN

2D-CNN shallow network learns the complex and abstract features to recognize facial expression. The structure of our shallow network is a 14-layer convolutional neural network, which includes convolutional layers, pooling layers, dropout layers and batch-normalization layers.

The whole structure of the network is shown in Table 1. The input of the network is 48 x 48 pixels face image. In our framework, we take the RGB image of the detected face as input of our network. And the output of the network

are probabilities of 7 expression categories . Besides regular convolutional blocks (convolutional layer, pooling layer), we plug in additional dropout layers and batch-normalization layers. Dropout layers reduce over-fitting and improve generalization error remarkably by randomly ignoring nodes in each iteration during training phase. Batch normalization layers solve the internal covariate shift problem by normalizing the output of activation layers. The details of dropout layers and BN layers are discussed in Section 2.4.2.2.

Table 1: 2D-CNN Shallow Network Model

Layer	Type	Output shape
1	Conv2D	(None, 48, 48, 32)
2	Conv2D	(None, 48, 48, 64)
3	Conv2D	(None, 48, 48, 64)
4	MaxPool	(None, 9, 9, 64)
5	Dropout	(None, 9, 9, 64)
6	Conv2D	(None, 7, 7, 128)
7	Conv2D	(None, 5, 5, 128)
8	MaxPool	(None, 1, 1, 128)
9	Dropout	(None, 1, 1, 128)
10	BN	(None, 1, 1, 128)
11	Flatten	(None, 128)
12	Dense	(None, 500)
13	Dropout	(None, 500)
14	Dense	(None, 7)

Implementation Details There are two main steps for the implementation: train a suitable 2D-CNN FER model and integrate the model into our framework.

In the model training phase, we choose the FER2013 database mentioned in Section 2.4.1 as our training database. Based on the features of the FER2013 dataset, we take data augmentation and normalization as our pre-processing methods. In the data pre-processing phase, we use Python to read 32,298 face images and split them into a 28,709 images training set and a 3,589 images testing set. After encoding the training labels with one-hot encoding, we take flipping, rotating from -15 degree to 15 degree, noising and cropping operations on training images as our data augmentation methods, which extend our dataset and improve our network robustness. In order to normalize our training images, each pixel of all images are normalized to float

number and ranged among (-1, 1). After resizing each image to 48 x 48 pixels, we input these processed images to train our model. According to our experiment, the data augmentation improves the accuracy by 10% and the image normalization improves the accuracy by 4%.

After we get the trained model, we integrate the model into our framework. We have discussed the deep learning support module in Section 3.3.3. Figure 41 shows how one static-based facial expression recognition adapter calls the Python-based deep learning model to get the predicted facial expression probabilities result. *OISFERAdapter* represents the concrete three adapter subclasses. *FER2D.cpp* is a C++ wrapper class mentioned in Section 3.3.3, which is used to transfer data structures betweem C++ and Python. Class *OIPythonEnv* only accepts or returns the *PyObject* data structure, which needs to be transferred inside C++ wrapper class. *OIPythonEnv* invokes the function we defined in *fer_2d.py* and the Python script will call our trained neural network model to calculate the facial expression probabilities result of our input image. Reversing the flow, *OISFERAdapter* can get the facial expression probabilities result calculated by deep learning model. Figure 42 shows the main function of our C++ wrapper class *FER2D*. Function *predict()* takes an image *img* and an integer *model* as input. The *img* represents the current face image and integer *model* represents the model that we choose to calculate our result. According to our design, we save all the deep learning models in file *fer_2d.py*. For static-based facial expression recognition, there are three deep learning networks saved in *fer_2d.py*, "Shallow 2D-CNN", "ResNet34" and "Fine-tuning ResNet50". Once function *predict()* of *fer_2d.py* gets parameter *model*. It can choose different trained models to calculate results.

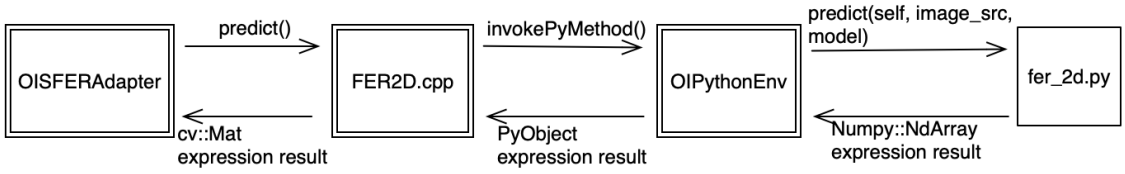

Figure 41: Process for static-based facial expression recognition adapter calling Python network model

For shallow 2D-CNN, the *OISFERAdapter* is *ShallowCNNSFERAdapter*. Once we fit *model* with "1", we select "Shallow 2D-CNN" network model as our deep learning network model for static-based facial expression recognition. In order to satisfy the requirements of network input, each image needs to be pre-processed by resizing and normalization. And after being calculated by our trained 2D-CNN model, the one-hot encoding expression vector is returned as the recognition result.

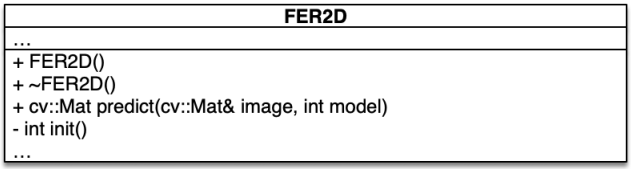

Figure 42: Main function of FER2D class

(2) ResNet34:

The structure of ResNet34 that we take is based on Kaiming et al. [14]. We have discussed the details of ResNet in Section 2.4.2.2. The reason for choosing the ResNet34 neural network structure is for solving the vanishing gradient problem in deeper networks. Unlike the original structure of ResNet34, we add dropout layers, BN layers, and we do not initial CP blocking, which increase the accuracy by 6% compared with the accuracy of the original ResNet34 model. At the flatten layer, we provide a more narrow structure with 256 feature maps rather than the original 512 feature maps. The input of our ResNet is 3 x 48 x 48, which is a 48 x 48 pixels image with three channels. And our output are probabilities of seven expressions.

Implementation Details We first discuss the details of how to train the ResNet34, then propose the integration of network model and our framework. Based on the input and output of our network, we choose FER2013 dataset mentioned in Section 2.4.1 as our training dataset. After obtaining face images from FER2013, we divide these images into three parts: 25,839 training images, 2,870 validation images and 3,589 testing images. The input for the network is 3 x 48 x 48, so we resize the images into 48 x 48 pixels with three channels. Besides resizing, we provide rotation, shearing, zooming and flipping as data augmentation methods to extend our training

set. In order to have a good accuracy for our network, we use Adam as our optimizer with 0.001 learning rate. The loss function that we choose is categorical cross-entropy.

The deep learning support module provides same integration method for "ResNet34" network model as "Shallow 2D-CNN" network model. The data flow between *ResNet34SFERAdapter* and *fer_2d.py* is the same as Figure 41. The slight difference between the two models is that ResNet takes 3 x 48 x 48 as input and 2D-CNN takes 1 x 48 x 48 as input, which means we need to use different pre-processing methods to process our input image.

(3) Fine-tuning ResNet50

This network is the most accurate method that we provide in static-based FER submodule. The network structure is 50-layer ResNet and the input is 3 x 197 x 197. 3 represents the number of channels of each image and 197 x 197 represents the size of each image.

Implementation Details In the training phase, we use the fine-tuning methods to train our ResNet50 model. Fine-tuning is a kind of method for neural network training, which tries to solve the over-fitting problem by multi-stage training. The standard process for fine-tuning is to first use a large-scale dataset to train the network, then use a more specific dataset to help the network focus on the requirements of our research. The details of the fine-tuning method is mentioned in Section 2.5.1. In our fine-tuning method, we use VGGFace [12] dataset to pre-train our neural network. VGGFace dataset is a large face dataset with 2,6000 people and 2.6 million images. This dataset provides a thousand images per identity, which is a good collection for face. The details of VGGFace are mentioned in 2.4.1. So the first step of training is using VGGFace dataset to train ResNet50 network. After obtaining the proper network model with the ability to recognize face, we start to train this network model to recognize facial expressions. In this stage, we choose FER2013 as our fine-tuning dataset. The details of FER2013 are mentioned in 2.4.1.

The input of our network model is 3 x 197 x 197, so we need to resize each image of FER2013 into 197 x 197 pixels. In order to keep the quality of the training images,

we choose bilinear interpolation as our scale-up algorithm, which means each value of the interpolated pixel is a weighted average of the value of four surrounding pixels. Figure 43 shows the performance of bilinear interpolation. After resizing, we provide multiple data augmentation methods (rotation, shearing, zooming, and flipping) to extend our dataset and keep the network robust. Then we choose centering and meaning normalization to deal with our input images. In order to keep the face recognition features that the earlier layers learn, we divided the training process into two phases: train top-3 layers only by freezing later layers, and then train all the layers together. In the earlier layers training, we use Adam optimizer as our optimizer with $lr = 0.001$ and $epsilon = 1e - 08$. The loss function is categorical cross-entropy. For this training phase, we have trained seven epochs, which could not only extract face feature from FER2013 but also keep the face features learned from the VGGFace dataset. In the second phase, we keep the pre-processing methods for input images, but change the hyper-parametric and defreeze all later layers. The optimizer that we take is stochastic gradient descent (SGD) with lr = 0.0001, momentum = 0.9, and decay = 0. And we setup the scheduler for the learning rate of the network that it halves the learning rate every 3 epochs.

Figure 43: Bilinear interpolation example

As for the framework integration for Fine-tuning ResNet50 and facial analysis framework, it is basically like the integration of the other two static-based FER networks mentioned above. The data flow diagram can be found in Figure 41. In order to fit the captured ROI face images into the network, we need to resize the

image to 3 x 197 x 197. On the other hand, the input image needs normalization to fit into the network and needs to be the same as training images.

4.1.1.4 Dynamic-based Facial Expression Recognition Submodule Instantiation

According to our research on dynamic-based facial expression recognition area in Section 2.6, we pre-trained five deep learning models and adapted them into our framework, which could help us implement our dynamic-based expression recognition functionality. These models are:

- 8F-7E Shallow 3D-CNN model
- 4F-7E Shallow 3D-CNN model
- 8F-4E Shallow 3D-CNN model
- 4F-4E Shallow 3D-CNN model
- ResNet18 3D model

For the naming conventions followed by the first four models, "F" represents the number of input frames and "E" represents the number of output for facial expression categories. They are the same type of model with different input and output configurations for different accuracy needs, but they all belong to *Shallow3DCNN* subclass. Therefore, we have two subclasses in total. Figure 44 shows the instantiations of this submodule.

The reason why we chose them is similar to what we stated in Section 4.1.1.3, which is to avoid over-fitting issues when training on small facial expression databases. The detailed analysis will be given below. From an implementation point of view, they are implemented in two concrete subclasses: "8F7EShallow-3DCNNAdapter", "4F7EShallow3DCNNAdapter", "8F4EShallow3DCNNAdapter", "4F4EShallow3DCNNAdapter" and "ResNet3DAdapter" are five keywords which could be fitted into the *createAdapter(const char *adapterName)* function of the

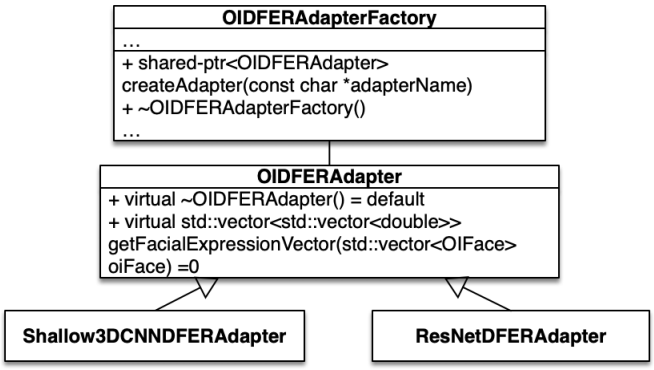

Figure 44: Dynamic-based facial expression recognition submodule instantiation

OIDFERAdapterFactory to determine which concrete subclass instance will be created. The words "8F7E", "4F7E", "8F4E", "4F4E" represent the different modes for *Shallow3DCNNDFERAdapter*, which tell the class *Shallow3DCNNDFERAdapter* which specific shallow 3D CNN model to use. In the following paragraphs, we will describe their implementations in detail.

(1) Shallow 3D-CNN

Shallow 3D-CNN is a kind of structure with shallow depth and 3D convolutional block. 3D convolutional block consists of 3D convolutional layer and 3D pooling layer. The difference between a normal convolutional layer and a 3D convolutional layer is the structure of the kernel. A normal convolutional kernel is a two-dimensional filter that extracts static features from an image, whereas a 3D convolutional kernel is a three-dimensional filter which could not only calculate the static features from one image, but also combine multiple continuous frames together to extract temporal features. The details and basic structure of 3D convolutional layers are discussed in Section 2.6.3.1.

The structure of our shallow 3D-CNN is a 9-layer 3D-CNN network. The structure of this network is shown in Table 2. This network model could predict the probabilities of seven expression categories with four continuous face frames as input. Actually, we provide four kinds of structures of shallow 3D-CNN which could do different dynamic-based facial expression recognition according to different conditions. These network

structures are 8F-7E shallow 3D-CNN, 4F-7E shallow 3D-CNN, 8F-4E shallow 3D-CNN, and 4F-4E shallow 3D-CNN. The notations of "8F" and "4F" represent that the network could recognize expressions through eight continuous face frames or four continuous face frames. The notations of "7E" and "4E" represent the network could recognize seven expression categories or four expression categories. The reason for training these different structures is to study the influence of number of input frames and number of expression categories available on accuracy and speed performance.

"4E" includes the selected four expressions, which are anger, happiness, sadness, and surprise, out of the main seven categories. The reason for excluding fear and disgust is that they are more difficult to distinguish as they share similar facial expression traits. Figure 45 shows the comparison between happiness, fear, disgust, and sadness. Neutral expression, in dynamic-based facial expression recognition, is less meaningful for detecting since they will all be faces with no expression(blank expression). Overall, we exclude fear, disgust, and neutral which leaves four stronger expression categories for testing.

Figure 45: Comparison between happiness, fear, disgust, and sadness

Table 2 shows the structure of "4F 7E shallow 3D-CNN", the input of the network is 4 x 64 x 64 x 3, which means that we need fit four frames with size of 64 x 64 x 3 images into our network. 64 x 64 represents the height and width of each image, and 3 represents the number of channels of each image. The reason for only taking 9 layers of neural network structure is the following:

In order to train our network to recognize facial expressions from continuous sequence, we choose CK+ [8] as our training dataset. CK+ provides 593 video sequences with 123 subjects. However, we only have hundreds of training sequences

Figure 46: CK+ frame image[8]

Figure 47: Detected face image[8]

for each expression to fit the network, which means the network is prone to over-fitting once the network gets too deep.

Table 2: 4F-7E 3D-CNN Shallow Network Model

Layer	Type	Output shape
1	Conv3D	(2, 64, 64, 3)
2	MaxPool	(1, 30, 30, 3)
3	Conv2D	(1, 26, 26, 29)
4	MaxPool	(1, 13, 13, 29)
5	BN	(1, 13, 13, 29)
6	Flatten	(None, 4901)
7	Dense	(None, 512)
8	Dropout	(None, 512)
9	Dense	(None, 7)

Implementation Details In order to train the shallow 3D-CNN network, we need to perform face detection for each frame. Figure 46 shows the original frame. As we can see, each image is captured from video, which can not be used as a network input directly as there are some unrelated information inside the image like the sign of time and background. The face detection method we chose for pre-processing is CNN face detection. The details of this face detection method are mentioned in Section 4.1.1.1. Figure 47 shows the output of the face detection method.

On the other hand, for each video sample, it shows the whole process from the neutral expression to peak of an expression, just like Figure 48. To train the network model to extract static and dynamic features from frame sequences, we choose the frames which could represent the changing process of facial expression. Figure 49

shows an example of a changing expression sequence. In order to extend our training dataset to prevent the over-fitting issue during the training phase, we take data augmentation as our solution, which increased the model accuracy on test dataset by about 18%.

Figure 48: Frame sequence example[8]

Figure 49: Selected frame sequence example[8]

Like we mentioned in Section 3.3.3 and Section 4.1.1.3, after obtaining the pre-trained network, we integrate the network into our framework to implement different instantiations. Figure 50 shows how a dynamic-based facial expression recognition adapter calls the Python network model to get the expression categories result. *OIDFERAdapter* represents the two concrete adapter subclasses (*Shallow3DCNNDFERAdapter* and *ResNetDFERAdapter*). Figure 51 shows the main function of *FER3D.cpp*, which is the C++ wrapper class mentioned in Section 3.3.3. Unlike the *predict()* function in *fer_2d.py* which takes one image as an input parameter, the *predict()* function in *fer_3d.py* takes a vector of images, *imageVector*, as the input argument. Variable *imageVector* saves a frame sequence for one expression, which could be different according to different network model. For example, *imageVector* fits four image frames into the model saved in *fer_3d.py* for "4F Shallow 3D-CNN" networks, where eight image frames are fit into "8F Shallow

3D-CNN" networks. Additionally, parameter *model* is used to choose different pre-processing methods and different models to calculate the results, which is the same as static-based facial expression recognition flow.

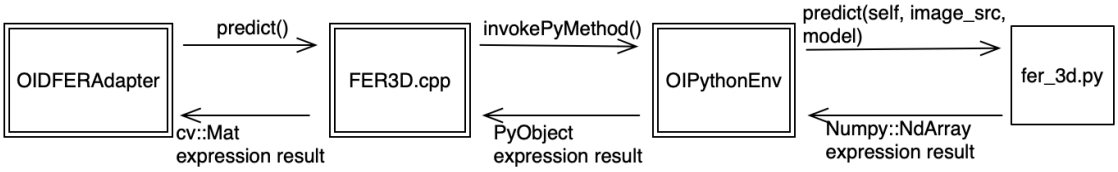

Figure 50: Process for dynamic-based facial expression recognition adapter calling Python network model

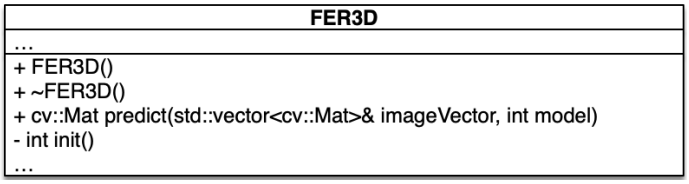

Figure 51: Main function of FER3D class.

(2) 3D-ResNet18

3D-ResNet18 is an 18-layers-ResNet with 3D kernel for each neuron. The ResNet structure can avoid the over-fitting problem when the network gets deeper. The 3D kernel extracts the spatial and temporal features from the input. The details of ResNet and 3D-CNN can be found in Section 2.4.2.2 and Section 2.6.3.1.

Implementation Details This network is used for four expression recognition with four frames. The input of network is 4 x 64 x 64 x 3, which denotes four frames with 64 x 64 pixels and 3 channels per image. During the pre-processing of the training phase, we provide almost the same pre-processing method as shallow 3D-CNN. Expanding the dataset by data augmentation and avoiding exploding/vanishing gradient by data normalization. The details of pre-processing can refer to the data pre-processing of Shallow 3D-CNN 4.1.1.4. The training dataset chosen for 3D-ResNet18 is CK+, and the optimizer for training is Adam.

The integration between the "3D-ResNet18" network and facial analysis framework is basically like the "4F4E Shallow 3D-CNN" model since they have the

same input and output structure. Once we choose the parameter *model* to be "5", the pre-processing and model will be set up for the "3D-ResNet18" model.

4.1.2 Pipeline Module Instantiation

In order to control the data flow of the facial analysis specialized framework, we create a subclass, *OIFacePipeline*, inheriting *OIPipeline*. Figure 52 shows the structure of *OIFacePipeline*.

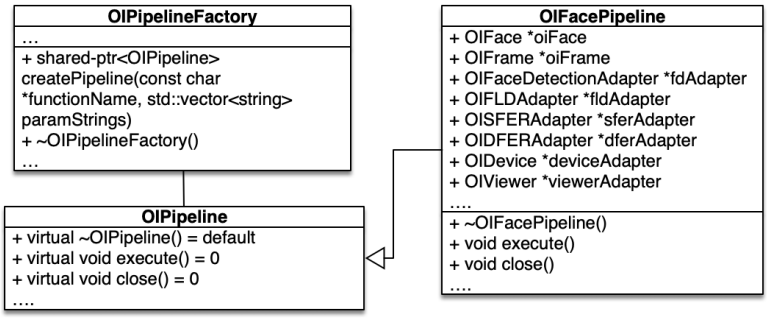

Figure 52: OIFacePipeline structure

In *OIFacePipeline*, we declared an abstract type pointer variable for each module inside the facial analysis specialized framework. Based on different applications, we make these pointers referring to different concrete instances, which could help the developer to integrate different implementations into their applications. For example, we can attach a *SSDFaceDetectionAdapter* instance to the member variable with type *OIFaceDetecionAdapter*, so the face detection algorithm used in the application will be the CNN-based face detection method. On the other hand, if we choose to not attach any instance for specific member variable, then the functionality of this member will not be used in this customized *OIFacePipeline* instance. For example, if we choose to not attach instances to *OISFERAdapter* and *OIDFERAdapter*, then the created *OIFacePipeline* instance could not realize the functionality of facial expression recognition. As for the member method *execute()* and *close()* of *OIFacePipeline*, they are used to control the execution and termination of program's flow.

The *OIFacePipeline* class is used to realize inversion of control in the framework, so users could directly focus on input, output of *OIFacePipeline* instance, and simply

use *execute()* and *close()*. And the details about how to connect different modules and how to check correctness of customized instance will be implemented inside the *OIPipelineFactory* class. Figure 53 shows the *OIFacePipeline* usage diagram.

Figure 53: OIFacePipeline usage diagram

In order to create our *OIFacePipeline* instance, we fit "Face" into *functionName*. And in order to control the data flow, we fit different string parameters into *paramStrings*. For example, assuming a developer wants to create a *OIFacePipeline* which could detect face and facial landmark. More specifically, the developer want to use Haar-cascade face detector and LBF facial landmark detector to implement detection parts. Here are steps that are required:

1. Create a *OIPipelineFactory* instance *oiPipelineFactory*
2. Fit function *createPipeline(const char *functionName, std::vector<string> paramStrings)* of *oiPipelineFactory* with "Face" and {"OIFaceDetection-Adapter:HaarFaceDetectionAdapter", "OIFLDAdapter:LBFAdapter"} to get *oiFLDPipeline* instance of *OIPipeline*.

4.2 Applications Using the Framework

In this section, we introduce the applications that we built based on our framework. They are real-time facial landmark detection application, human-face-based facial animation expression matching application, and static-based facial expression recognition application.

4.2.1 Real-time Facial Landmark Detection Application

In the first scenario we described in Section 1.5.1, we need to provide the movie director with a synchronized view with the character's face when shooting a human performer's acting. To demonstrate our realization of this scenario, we created a real-time facial landmark detection application which depends on two components of our framework. One is the face detection submodule and other is the facial landmark detection submodule. In this application, for each incoming frame, we perform the steps described below which can be visualized by Figure 54.

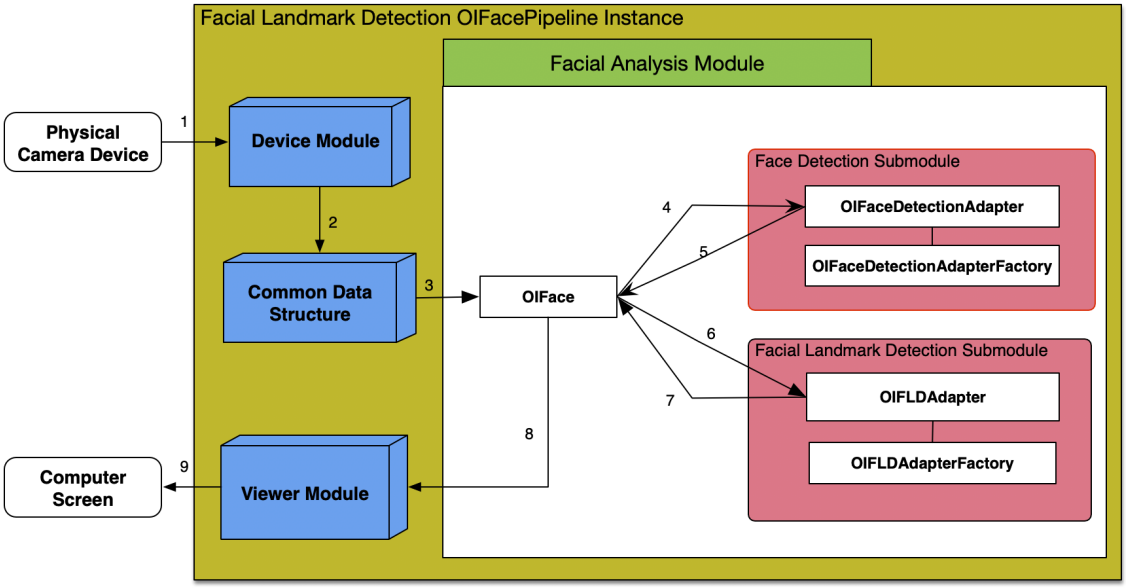

Figure 54: Facial landmark detection application

1. The data from the physical camera device will firstly flow into the device module in the core framework then is encapsulated as our framework's common data structure, precisely, *OIFrame*.

2. *OIFace* copies the frame data from *OIFrame*, so our frame data is now transferred to and stored as facial-specific data structure.

3. The frame data of *OIFace* is sent to one of the face detectors resided in the face detection submodule of our face analysis module. All the faces within the input image will be detected and their location will be populated back into *OIFace*.

4. The detected faces are masked out from the original image using their location and feed into a selected facial landmark detector within the corresponding submodule. Finally, the landmarks will be found and stored back to the corresponding data holder in *OIFace*.

5. For demonstration purpose, the original input image with all detected landmarks drawn will be displayed by the viewer module.

As mentioned in Section 4.1.1.1 and Section 4.1.1.2, there are three kinds of implementations for the face detection submodule in our framework as well as facial landmark detection submodule. In this application, we selected the SSD method for face detection and CLNF method for facial landmark detection. With the support from the pipeline framework module, the application developer don't need to worry about handling the intermediate results but just inform the framework which implementation they want to employ then the framework will take care of the rest including pipeline construction, selective filters instantiation and data flow control. More precisely, the procedure of this specific application can be described as Algorithm 1.

With the real-time facial landmark detection application, for each streaming frame, we can obtain an annotated image with detected facial landmarks, regardless of one more multiple faces presented, which can be shown by Figure 55 and 56. The discussion of real-time response will be presented in Section 5.1.1.

Algorithm 1: The procedure of real-time facial landmark detection application

Input: An image captured by a supported depth camera.
Output: Facial landmarks for all detected faces.

```
plFactory ← create an instance of OIPipelineFactory
noEcsPressed = True
params ← create an instance of vector
params.push_back("OIDevice:KinectV1")
params.push_back("OIFaceDetectionAdapter:SSDAdapter")
params.push_back("OIFLDAdapter:CLNFAdapter")
params.push_back("OIViewer:OIOpenCVViewer")

fldApp ← plFactory.createPipeline("Face", params)

while noEsrPressed do
    // process one frame each execution
    fldApp.execute();
    if Ecs Pressed then
        noEcsPressed = False;
    end
end
```

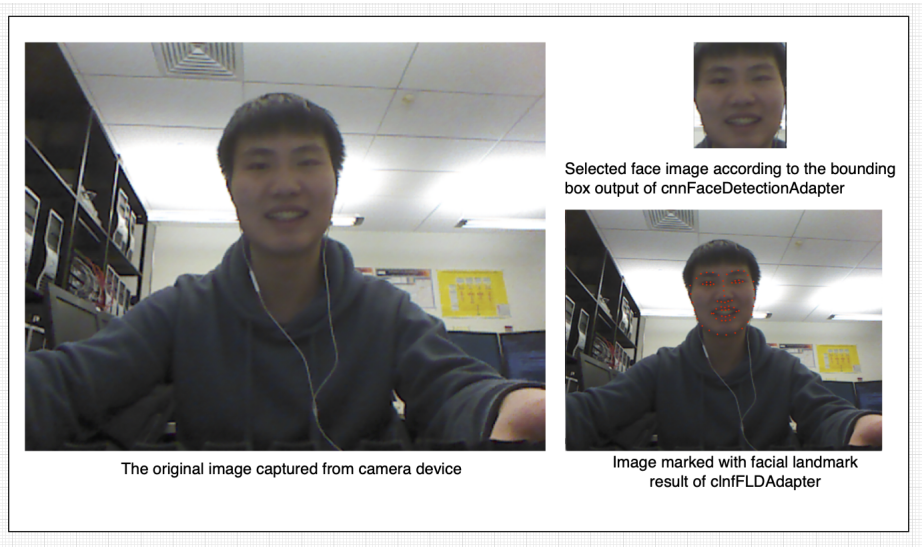

Figure 55: Facial landmark detection example (single face)

Figure 56: Facial landmark detection example (multiple faces)

4.2.2 Human-face-based Facial Animation Expression Matching Application

In the second scenario we described in Section 1.5.2, we need to make a facial expression matching application that can take a frame of expected human facial expression image as input, output the expression category with landmarks detected, then pass them to the database matcher to find the appropriate expression image. To demonstrate the realization of this scenario, we created a human-face-based facial animation expression matching application. This application depends on three components of our framework: face detection submodule, facial landmark detection submodule, and static-based facial expression recognition submodule. In this application, for each incoming frame, we perform the steps described below which can be visualized by Figure 57.

1. The expected expression image is firstly be encapsulated as our framework's common data structure, precisely, *OIFrame*.

2. *OIFace* copies the frame data from *OIFrame*, so our frame data is now transferred to and stored as facial-specific data structure.

3. The frame data of *OIFace* is sent to one of the face detectors residing in the face detection submodule of our facial analysis module. All the faces within the input image are detected and their location are populated back into *OIFace*.

4. The detected faces are masked out from the original image using their location and feed into a selective facial landmark detector within the corresponding submodule. Finally, the landmarks are found and stored back to the corresponding data holder in *OIFace*.

5. The detected faces are masked out from the original image using their location and feed into a selected static-based facial expression recognizer within the corresponding submodule. Finally, the facial expression category probabilities are calculated and stored back to the corresponding data holder in *OIFace*.

6. Facial expression category probabilities and landmarks stored in *OIFace* are passed to the database matcher, then the matcher will output the image with highest similarity score.

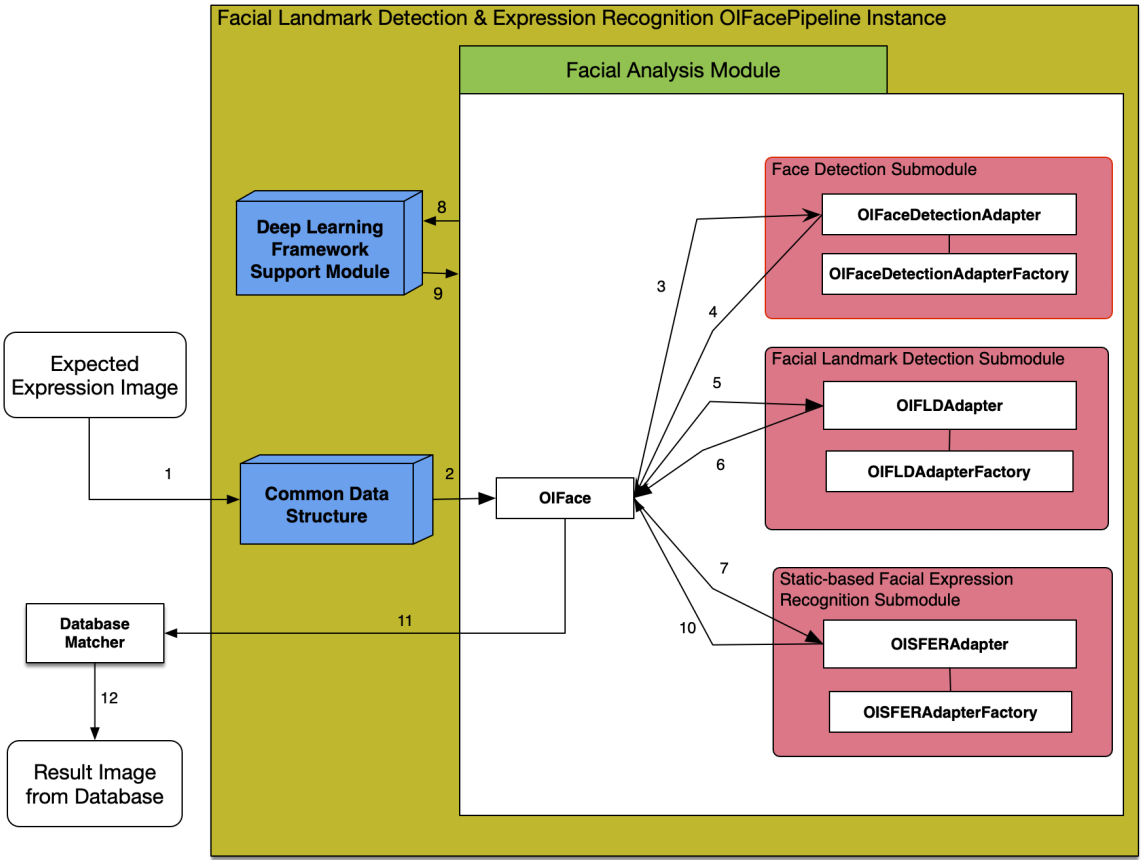

Figure 57: Human-based-face facial animation expression matching application

In this application, we selected the SSD method for face detection, LBF method for facial landmark and Fine-tune ResNet50 method for static-based facial expression recognition. The procedure of this specific application can be described as Algorithm 2.

Algorithm 2: The procedure of human-face-based facial animation expression matching application

Input: Expected expression image: inputImg
Output: Similar image from database

```
plFactory ← create an instance of OIPipelineFactory
params ← create an instance of vector
matcher ← create an instance of Matcher
params.push_back("OIFaceDetectionAdapter:SSDAdapter")
params.push_back("OIFLDAdapter:LBFAdapter")
params.push_back("OISFERAdapter:FineTuneAdapter")

matchApp ← plFactory.createPipeline("Face", params)

matchApp.oiFrame ← new instance of OIFrame(inputImg)
matchApp.execute()

landmarks = matchApp.oiFace->getFacialLandmarkVector()
expressions = matchApp.oiFace->getExpressionVector()

result ← matcher(landmarks, expressions, Database)
```

The database mentioned here is an animation expression database. But according to our research, there are no existing suitable animation expression databases that allows us to do the experiment. In order to solve this problem, we decided to build a simple human expression database, which could also provide the expression category and landmarks for each expression image. Realizing the matching process on this database could also prove the integrity of our application. We built an expression database including seven expression category subsets: anger, disgust, fear, happiness, sadness, surprise, and neutral. There are five images for each subset with different types of that same expression category. Figure 58 shows sample images of expression category subset of happiness. Furthermore, each image in the database has its own text file that stores its landmarks and expression category.

Figure 58: Happiness subset of database

With the database content implementation explained above, we need to further discuss how we implemented our database matcher. Since the matcher is not our research focus, here we only give a brief explanation. There are two steps to implement our matcher:

1. Based on expression category probabilities to downscale our search range.

2. Based on landmarks to calculate the similarity score between our input expression image and the target expression image in database.

For step 1, our expression recognizer can perform a preliminary test on the input image regarding seven expression categories, anger, disgust, fear, happiness, sadness, surprise, neutral. For each expression category, the probabilities for resemblance will be calculated and stored in the tuple ($P_{anger}, P_{disgust}, P_{fear}$,- $P_{happiness}, P_{sadness}, P_{surprise}, P_{neutral}$), and the sum of all elements will add to 1. If any subset has a probability value greater than 0.5, we will move on to search only within that expression category subset. For example, if the probability value $P_{happiness}$ is greater than 0.5, then we will only search for expressions that belong to the happiness subset. On the other hand, if no probability value exceeds 0.5, we then search for expressions that belong to subsets with probability value greater than 0.2. For example, if the expression preliminary test result is ($P_{anger} = 0.1, P_{disgust} = 0.3, P_{fear} = 0.3, P_{happiness} = 0.1, P_{sadness} = 0.05, P_{surprise} = 0.05, P_{neutral} = 0.1$), which has no expression category exceeding probability value of 0.5, we then downscale our search range and focus only on disgust and fear subsets.

For step 2, after obtaining the downscaled search range, there is a pre-processing step before comparing which is eye-corner-based face alignment. This alignment will

be performed on both input image and the image in the database so that the eyes of two source images will be vertically and horizontally aligned with the same eye distance. This method will help us minimize inaccurate calculations due to rotated face or faces of different scales. After alignment, we will generate an overlapped composite image that contains aligned landmarks from two source images. This intermediate result will be used for calculating similarity score, which is performed using the following formula:

$$similarityScore = \frac{1}{\frac{1}{50}\sum_{i=18}^{68} ||d_i - g_i||^2} \tag{5}$$

Note that we are using a 68-point facial landmark model. In order to avoid the impact on the result from different facial profiles, we will not incorporate landmarks for facial profile (point 0-16) in our calculation. Figure 59 shows an example of the 68-points facial landmarks model.

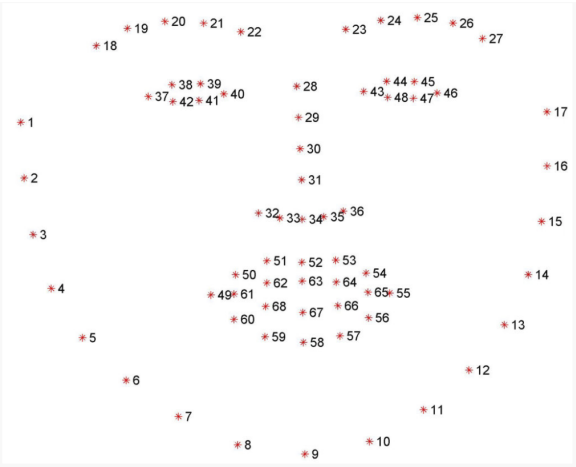

Figure 59: Example of 68-points facial landmark model [15]

This formula will output the similarity score between the two expression images. Eventually, the expression image from the database with the highest similarity score will be chosen as the matching result. Figure 60 shows two sample results generated through human-face-based expression recognition matching process.

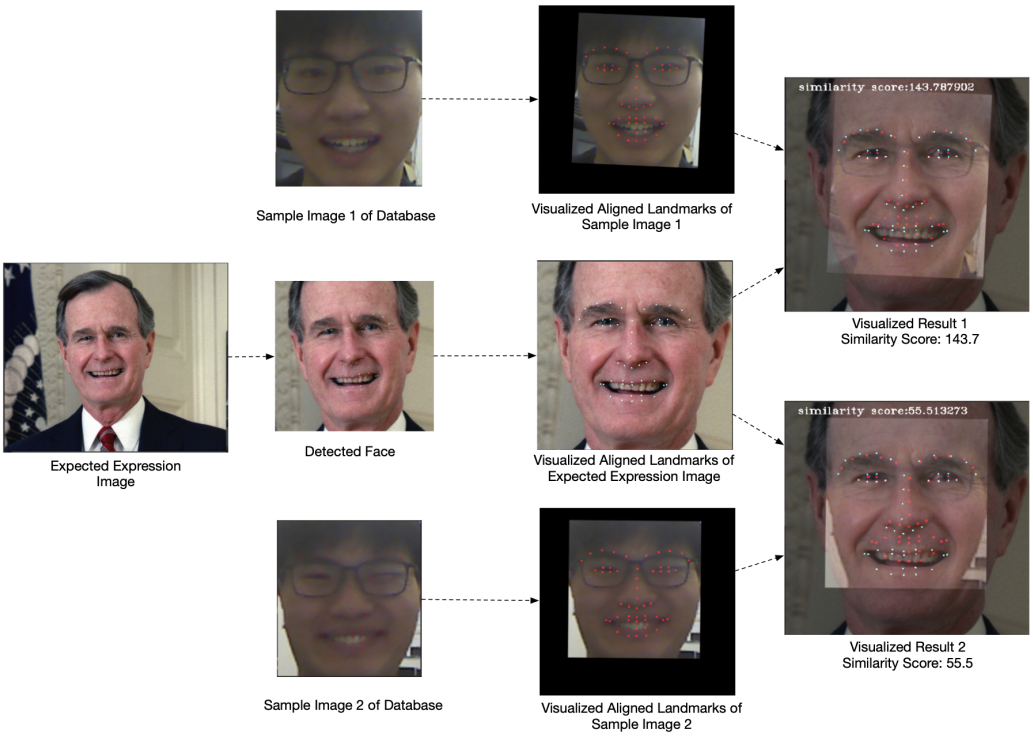

Figure 60: Two sample matching results

4.2.3 Static-based Facial Expression Recognition Application

Following the last framework application on human-face-based facial animation expression matching, we aim to test the performance of the expression recognition functionality. We created a static-based facial expression recognition application which takes input from a camera device rather than a single expression image and output the visualization result on our screen directly. This application depends on two components of our framework. One is the face detection submodule and other is the static-based facial expression recognition submodule. For each incoming frame, we perform the steps described below which can be visualized by Figure 61.

1. The data from the physical camera device firstly flows into the device module in the core framework then is encapsulated as our framework's common data structure, precisely, *OIFrame*.

2. *OIFace* copies the frame data from *OIFrame*, so our frame data is now transferred to and stored as facial-specific data structure.

3. The frame data of *OIFace* is sent to one of the face detectors resided in the face detection submodule of our facial analysis module. All the faces within the input image are detected and their locations are populated back into *OIFace*.

4. The detected face is masked out from the original image using their location and fed into a selected static-based facial expression recognizer within the corresponding submodule. Finally, the facial expression category probabilities are calculated and stored back to the corresponding data holder in *OIFace*.

5. For demonstration purpose, the original input image and calculated probabilities for each expression category are displayed by the viewer module.

As mentioned in Section 4.1.1.1 and Section 4.1.1.3 there are three kinds of implementations for the face detection submodule in our framework as well as the static-based facial expression recognition submodule. In this application, we selected the the HOG method for face detection and the ResNet50 method for static-based facial expression recognition. The procedure of this specific application can be described as Algorithm 3.

Figures 62 and 63 show two sample frames generated by the static-based facial expression recognition application. The output includes the original image, the detected face area and the calculated probabilities for each expression category. According to "Probability distribution histogram of happiness face image recognition result" in Figure 62, the recognized probability of "happiness" is over 70%, which is higher than any other expression. As for sad face example, Figure 63 shows that the probability of "sadness" is over 80%.

Algorithm 3: The procedure of static-based facial expression recognition application

Input: An image captured by a supported depth camera.
Output: Expression category probabilities for all detected faces.

```
plFactory ← create an instance of OIPipelineFactory
noEcsPressed = True
params ← create an instance of vector
params.push_back("OIDevice:RealSenseD435")
params.push_back("OIFaceDetectionAdapter:HOGAdapter")
params.push_back("OISFERAdapter:FineTuneAdapter")
params.push_back("OIViewer:OIOpenCVViewer")

sferApp ← plFactory.createPipeline("Face", params)

while noEsrPressed do
    // process one frame each execution
    sferApp.execute();
    if Ecs Pressed then
        noEcsPressed = False
    end
end
```

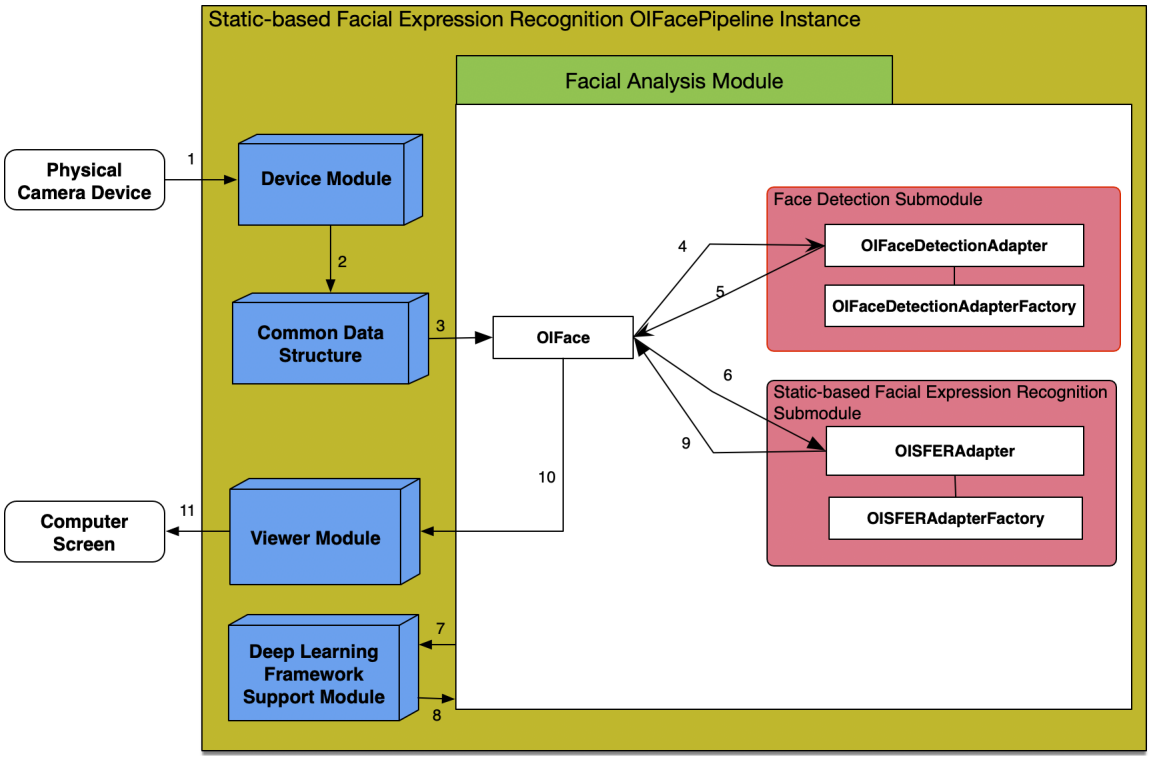

Figure 61: Static-based facial expression recognition application

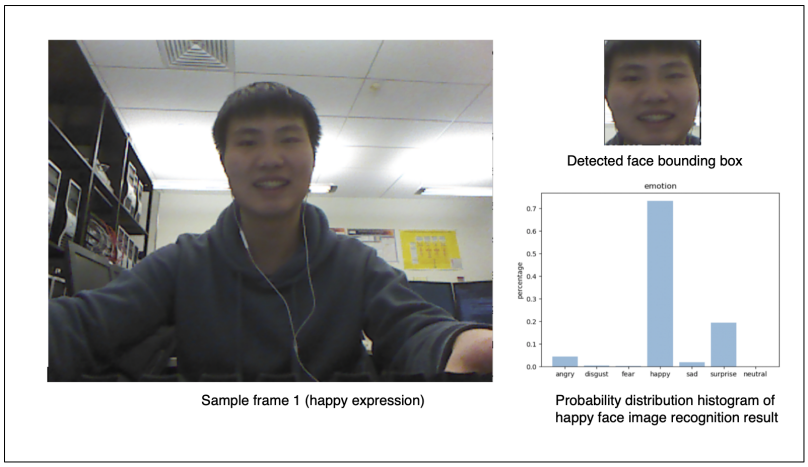

Figure 62: Happy face recognition example of static-based facial expression recognition

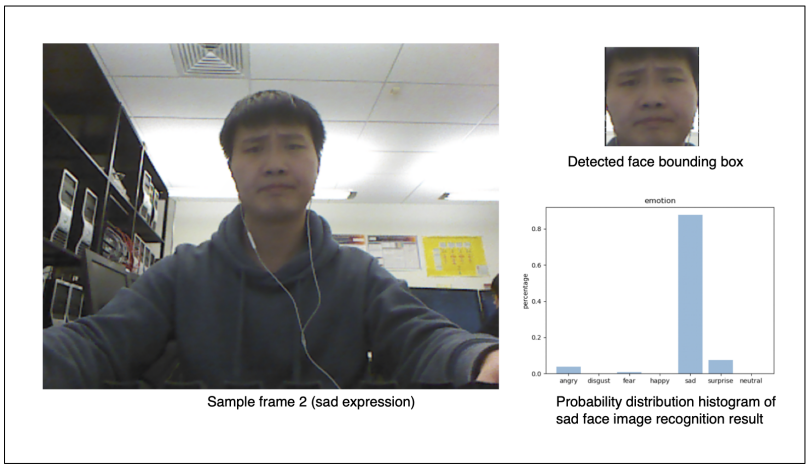

Figure 63: Sad face recognition example of static-based facial expression recognition

4.3 Summary

In this chapter, we describe the instantiation of our specialized framework, which includes the instantiation of different facial analysis submodules as well as the details of their implementations. We then gave three concrete examples of how to build facial analysis applications based on our framework solution. In the next chapter, we will describe the tests we designed and demonstrate that we have met the objectives and requirements that we stated in Chapter 1.

Chapter 5

Evaluation

The purpose of this chapter is to give an overview of our research results and evaluation. We begin by a discussion to evaluate the quality of our framework in Section 5.1. Then we do a series of experimental evaluation in Section 5.2 to assess the quality of the applications we have developed using the framework. In both of these sections, we do our assessments versus the goals we had identified in Section 1.4 and the usage scenarios and requirements identified in Section 1.5. For the experimental evaluation, we provide supporting experimental data for common metrics that are acknowledged by the research community. Moreover, we follow the same approach for algorithms or models evaluation, and finally, report our result based on the metrics.

5.1 Framework Evaluation

In this section, we evaluate our solution by checking whether or not goals mentioned in Section 1.4 are attained. At the same time, we check whether the motivation scenarios mentioned in Section 1.5 can be addressed by using our framework solution.

5.1.1 Real-time Facial Landmark Detection

In Section 1.5.1, we stated the scenario of real-time landmark-based facial expression mapping. We extracted some functional requirements from this scenario. Here are

our evaluations about these requirements.

In Section 4.1.1.1 and Section 4.1.1.2, we described the design of our framework solution, which shows the ability to implement the face detection and facial landmark detection. In Section 4.2.1 we described how to use our framework solution to build an application which could detect the bounding box of human face from human image and detect the landmark locations from a human face image. This application exactly fulfills **FR1**, **FR2** and has the ability to run processes of **FR1** and **FR2** as a whole process, which is part of **FR3**.

According to real-time definition of Kapoor et al. [18], if a system could achieve 30 FPS, then we could say it is a real-time system. In order to prove that our solution has real-time performance, we designed experiments to test our implementations in face detection and facial landmark detection in Section 5.2.1 and Section 5.2.2 respectively. According to the results in Table 4 and Table 5, we could say that, with the combination of LBF facial landmark detector and any other implemented face detectors presented in our solution, the whole analysis process eventually achieves real-time performance. For real-time performance evaluation, details will be further elaborated in Section 5.1.6. So according our analysis above, we can say that **FR3** is clearly fulfilled by our solution.

5.1.2 Facial Animation Expression Matching

In Section 1.5.2, we stated the scenario of facial animation expression matching and we extracted some functional requirements from this scenario. Here are our evaluations about these requirements.

In Section 4.1.1.3 and Section 4.1.1.4, we described the design of our framework solution, which shows the ability to implement the static-based facial expression recognition and dynamic-based facial expression recognition. In Section 4.2.3 we described how to use our framework solution to build an application which could detect the bounding box of human face from a human image and recognize the expression category from a human face image. This application exactly fulfills **FR4** proposed in this scenario. As for **FR5**, we have built a facial animation expression

matching application in Section 4.2.2, which proves that our solution fulfills the functional aspect of the requirement **FR5**.

In order to prove our solution could satisfy **FR6**, we give a specific scenario as the following: Assuming a developer has used *FaceDetectionAdapter1* and *FacialLandmarkDetectorAdapter1* to implement a facial landmark detection application by following the steps mentioned in Section 4.2.1. Now this developer finds out that the accuracy of *FacialLandmarkDetectorAdapter1* is too low. And there exists another implementation in *FacialLandmarkDetectorAdapter2* which could achieve higher accuracy. The developer now wants to use the better implementation for facial landmark detector to replace the bottleneck, *FacialLandmarkDetectorAdapter1*, of the existing application. The developer only need to change one line in the existing code: replace the input parameters *paramStrings* of *createPipeline(const char *functionName, std::vector<string> paramStrings)* from {"OIFaceDetectionAdapter:FaceDetectionAdapter1", "OIFLD-Adapter:FacialLandmarkDetectorAdapter1", ...} to {"OIFaceDetectionAdapter:-FaceDetectionAdapter1", "OIFLDAdapter:FacialLandmarkDetectorAdapter2", ...}. After saving the modified code, the developer could get a better performing application consisted of *FaceDetectionAdapter1* and *FacialLandmarKDetectorAdapter2* ready to work immediately. So according to our analysis above, we can say that **FR6** is clearly fulfilled by our solution.

In addition to process optimization by replacing a bottleneck, our framework also allows users to add new implementations including deep learning approaches to our solution, as stated in **FR7**. In Section 3.3.3 we have explained how we provide Python-based deep learning support module, and how we integrate this module into our facial analysis framework. Specifically, the user could create a specific *OIAdapter* first, which is used to implement deep learning based facial analysis algorithm. Then users could create a Python invocation flow shown in Figure 26 to support Python invocation flow for the above *OIAdapter*. With Python invocation flow, the developer can then invoke Python-based deep learning implementations within *OIAdapter*. This is how we provide deep learning support for our solution.

Section 4.1.1.4 provides a successful example for realization and fulfillment of **FR7**. In the implementation of "Shallow 3D-CNN" method, we integrate deep learning support module with *Shallow3DCNNDFERAdapter* by creating invocation flow shown in Figure 50. After all the details mentioned in Section 4.1.1.4, we implement "Shallow 3D-CNN" deep-learning based facial expression recognition method in *Shallow3DCNNDFERAdapter*. So according to our analysis above, we can say that **FR7** is clearly fulfilled by our solution.

5.1.3 Camera Support

In Section 1.5.3, we stated the scenario of camera support and we extracted a functional requirement from this scenario. Here are our evaluations about this requirement.

The steps of how to create a new camera device adapter or switch to a different camera device adapter can be found in Section 3.3.1. To prove that our solution fulfills this requirement, we can first look at the two applications mentioned in Section 4.2.1 and Section 4.2.3. As you can see, we used two different cameras for the applications. There's only one line change in the code that allows us to switch input camera device: replace the input parameters *paramStrings* of *createPipeline(const char *functionName, std::vector<string> paramStrings)* from {"OIDevice:KinectV1", ...} to {"OIDevice:RealSenseD435", ...}. In addition to camera switch, we can also easily add new camera by: (1) create a subclass inheriting class *OIDevice* and implement the virtual methods (2) add the logic to instantiate the new device in *OIDeviceFactory* class. Therefore, we can say that **FR8** is clearly fulfilled by our solution.

5.1.4 Extensibility

In Section 1.5.4, we noticed that our system needs to have extensibility as a non-functional requirement by analyzing the scenarios mentioned in Section 1.5. In Section 5.1.3, we have proved that our solution could add a new camera or switch between existing cameras without impairing existing system functions.

In terms of extensibility of facial analysis components, we take static-based facial expression recognition as our example. As we explained in Section 4.1.1.3, we follow the design of "Adapter + AdapterFactory". When we want to develop and integrate a new adapter, we can follow the steps below:

1. Create a concrete adapter subclass inheriting the *OISFERAdapter* abstract class, so the pure virtual methods of *OISFERAdapter* will also be inherited from the subclass, and then implement the static-based facial expression recognition algorithm inside the inherited pure virtual method.

2. Add a new string as name to indicate the appended adapter and create an entry logic of it in the *OISFERAdapterFactory* class.

According to this example, we confirmed that we can add new implementation to the specific facial analysis module. In Section 5.1.2, based on proof of **FR6**, we have proved that our solution can switch different facial analysis implementations without impairing existing system.

5.1.5 Usability

In Section 1.5.4, we noticed that our system needs to have usability as a non-functional requirement by analyzing the scenarios mentioned in Section 1.5.

To prove the usability of our framework solution, we would like to demonstrate how new users could start using our solution quickly with a short learning curve. Below is a list of tested sample tasks that our framework application can do:

- perform human face detection from RGB image
- perform real-time facial landmark detection on the previously detected human face image
- perform facial expression recognition on single human face image
- support multiple types of input from different camera devices

- allow users to easily add new implementations

Our framework provides a universal entry point through the pipeline module, which consists of function names that all follow a logical and concise naming convention. To perform certain tasks provided by our framework, users only need to first create an *OIPipelineFactory* instance, then enter their desired functional module and implementation names as a string into the *createPipeline()* function within the factory instance. Take the example from Section 4.2.2. There are only 16 lines of code needed to obtain functionalities such as camera support, face detection, facial landmark detection, and static-based facial expression recognition. On the other hand, as mentioned in Section 3.3.1 and Section 3.4.2, users only need to perform two steps for integrating new implementations into our framework. Plus, our framework shall be very user-friendly with superior learnability given the provided comments in source code and technical documentations that will be provided upon the completion of the entire solution. Users only need to consult to API documentations instead of diving into each functional module to get insights of framework utilization. However, the user-friendliness is not yet proven as our solution is still under development and have no actual user that can provide feedback. This is an assumption based on the user support resources we planned, which will be verified later when our finished OpenISS framework is used as a tool in FAM production.

5.1.6 Real-time Performance

We want to evaluate the real-time performance since it is one of our non-functional requirements mentioned in Section 1.5.4. To ensure the facial landmark detection application to be real-time, which should be faster than 30 FPS according to Kapoor et al. [18], we ran a test on the speed of the application described in Section 4.2.1.

The experiment we designed is about recording the time t spent to process 1000 frames captured from the camera. Then the speed of our application is calculated as $\frac{1000}{t}$. The combination of our fastest face detector, HOG, and fastest facial landmark detector, LBF, achieved 16.94 FPS, which is slower than 30 FPS. However, we found

that our camera devices, Kinect V1, Kinect V2, and RealSenseD435, have their own performance limit and can only capture frames up to 30 FPS. To eliminate limitations from our camera devices, we decided to read the human face image directly from 300 Faces taken from the in-the-Wild Challenge (300-W) [15] database. In order to calculate the speed performance of our application, we calculated how much time the application needs to process 1000 images and derived the FPS value of our application. The experiment result demonstrated that even with the slowest face detector and facial landmark detector combination, "SSD" and "LBF", which is implemented in *SSDFaceDetectionAdapter* and *LBFFLDAdapter*, we still achieved 31.81 FPS according to our experiment, which is above the real-time standard of 30 FPS. Additionally, our fastest combination, "HOG" and "LBF", which is implemented in *HOGFaceDetectionAdapter* and *LBFFLDAdapter*, achieved 50.58 FPS. This proves that our implementation meets the real-time performance when we have high-frame-rate cameras.

5.1.7 Accuracy

We want to evaluate the accuracy of our solution since it is one of our non-functional requirements mentioned in Section 1.5.4. As we mentioned in Section 1.5.2, the human-face-based facial animation expression matching should achieve an accurate result in general. Since there is no state-of-the-art method or evaluation database available to test the accuracy of the whole animation expression matching pipeline, we can only segment the pipeline and test the accuracy for each functionality to conclude that our entire process indeed achieves a certain level of accuracy comparable to what is found in the literature.

Our segmented pipeline accuracy test includes individual tests on three functionalities: face detection, facial landmark detection, and static-based facial expression recognition. For face detection accuracy test, SSD face detector had the highest accuracy result among all our available detectors, which ranked 19th over 60 published face detectors according to the evaluation of Face Detection Data Set and Benchmark (FDDB) [60].

For facial landmark detection accuracy test, we chose 300 Faces in-the-Wild Challenge database [15] as our test database and we chose the survey of Wu et al. [1] as our reference. We found that the CECLM facial landmark detector had the highest accuracy result among all our available detectors, which ranked 3rd over 11 detectors that were tested from the experiment in the survey.

For the static-based facial expression recognition accuracy test, we chose FER2013 [9] as our evaluation database and we chose the survey of Shan et al. [61] as our reference. Based on the six state-of-the-art tested methods in the article, we can see that the overall accuracy range is 71.2% - 75.2%. The accuracy of our trained model, ResNet50, achieved an accuracy of 71.02%. Thus, we consider that our facial expression recognizer has a rather satisfying performance on accuracy. More details about these tests will be discussed in Section 5.2.1, Section 5.2.2, and Section 5.2.3.

5.2 Experimental Evaluation

In this section, we discuss the experimental evaluation of each phase of our facial analysis implementation: face detection, facial landmark detection, static-based facial expression recognition, and dynamic-based facial expression recognition. Each implementation is tested using multiple facial image databases, which is a standard way to evaluate an algorithm in the facial analysis research domain. Our evaluation focuses on the accuracy on specific image database and the average speed under specific large image database. Especially, we need to point out that our speed test is using facial analysis algorithm to process large number of images, and we are going to record the average number of images the algorithm can process per second. The unit of our speed parameter is FPS. We will discuss the results of experiments and compare algorithms at the end of each subsection. The experimental environment of an algorithm has great influence on speed. We show our experiment environment in Table 3. All experiments stated below are tested under this environment.

Table 3: Experimental Environment

Hardware	Type
Operating System	Ubuntu 18.04.2 LTS
Processor	Intel i7 920
RAM	12GB
GPU	NVIDIA GTX 1080 Ti with 11 GB RAM

5.2.1 Face Detection

For the face detectors in our framework, they are used in two scenarios mentioned in Section 1.5.1 and Section 1.5.2. For landmark mapping scenario, the face detector needs to satisfy real-time performance as the first priority, then try to achieve higher accuracy. For human-face-based facial animation matching scenario, the face detector only needs to try and achieve higher accuracy according to the sole requirement for the scenario.

In order to test the accuracy and speed of each face detection algorithm, we chose mAP (mean Average Precision) as our accuracy unit and FPS (Frames Per Second) as our speed unit. Before introducing our experiment details, we would like to give an overview of the evaluation concepts used in face detection domain.

Confusion Matrix: Confusion matrix is a specific table layout that allows performance visualization of a classification algorithm. Each row represents the instances in a predicted class and each column represents the instances in an actual class. Figure 64 shows the example of a confusion matrix.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Figure 64: Confusion matrix example

- "True Positive(TP)" represents the model correctly predict the positive class.
- "True Negative(TN)" represents the model correctly predict the negative class.
- "False Positive(FP)" represents the model incorrectly predict the positive class.
- "False Negative(FN)" represents the model incorrectly predict the negative class.

Precision and Recall: Precision measures how accurate are your predictions, i.e. the percentage of how your predictions match with the actual classification. Recall measures how good you can find all the positives, i.e. the percentage of the possible positive cases in top K predictions. Here are mathematical definitions for precision and recall:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

IoU (Intersection over union): IoU measures the overlap between 2 boundaries. In face detection domain, we use IoU to measure how much our predicted face boundary overlaps with the real face boundary (ground truth). Figure 65 gives an example of IoU. $IoU = \frac{area\_of\_overlap}{area\_of\_union}$.

Figure 65: IoU example

Precision-Recall (P-R) Curve: The P-R curve can be used to show the relationship between precision and recall with various prediction confidence. For example, in face detection domain, we can set a threshold for IoU to calculate precision and recall. We consider the following definitions to describe a detection:

- TP, when calcualted IoU >= threshold
- FP, when calcualted IoU < threshold
- TN, does not apply
- FN, a ground truth not detected

We collect all the predictions made for faces in all the test images and rank predictions in descending order according to the predicted confidence level. Here we give a simple example to demonstrate the calculation of the average precision. Assuming that there is a dataset that contains five faces, we collect all the predictions made for faces in all the images and rank them in descending order according to the predicted confidence level. The second column indicates whether the prediction is correct or not. We chose IoU >= 0.5 as correct prediction. Figure 66 shows the precision-recall table ordered by predicted confidence level.

For the row with rank 3, $Precision = \frac{TP}{TP+FP} = \frac{2}{2+1} = 0.67$, $Recall = \frac{2}{2+3} = 2/5 = 0.4$. According to the relation between precision, recall and rank, we could draw the Precision-Recall (P-R) curve. Figure 67 shows the matched P-R curve under $IoU >= 0.5$.

Rank	Correct	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

Figure 66: Example of precision-recall table ordered by predicted confidence level

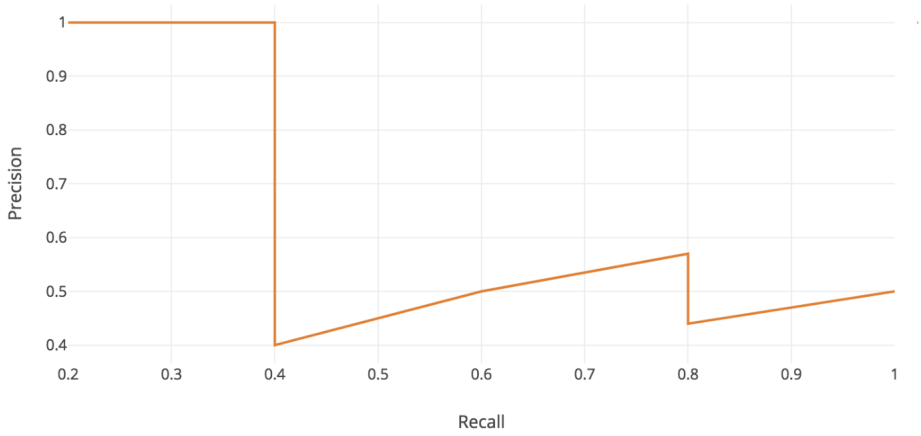

Figure 67: Precision-recall curve example

AP (Average precision): AP is defined as the area under the precision-recall curve. The mathematical definition of AP is Equation 8.

$$AP = \int_0^1 p(r)dr \tag{8}$$

The definition of mAP is averaging the value of AP under different IoU. Equation 9 shows the calculation equation of mAP.

$$mAP = \frac{\sum_{i=0}^{9} AP^{IoU=(0.5+0.05i)}}{10} \tag{9}$$

5.2.1.1 Experiment Design

In the evaluation part of face detection, we took the most common evaluation database, Face Detection Data Set and Benchmark (FDDB) [60], which could help us to compare our implementations with the state-of-the-art algorithms. It is a facial image dataset with 2845 labeled images and 5171 faces in total. With the default evaluation method provided by the FDDB dataset, we directly got the mAP for each method. As for the speed test, we calculated the average time for each algorithm to process all 2845 images and calculated their respective FPS.

5.2.1.2 Experiment Result

Table 4 shows the evaluation result for each face detection algorithm we implemented in Section 4.1.1.1.

Table 4: Accuracy and speed evaluation result of face detection

Face Detector	mAP	Speed(FPS)
Haar-Cascade	0.639	51.24
HOG	0.538	71.03
SSD	0.891	36.84

From the FDDB dataset [60], we checked the published state-of-the-art algorithms for face detection and we found that the highest mAP ranking is the HR face detection algorithm proposed by Hu et al[62], which achieved 0.969 mAP. According to Table 4, we can see that our implementation of SSD gets the highest mAP, which means it is the most accurate face detection algorithms among all implemented methods. On the other hand, in terms of speed, HOG face detection algorithm is a better choice than SSD since HOG face detector has almost double the speed of SSD face detector.

Based on test results, we conclude that, for landmark mapping scenario, SSD face

detector is more suitable since it achieves higher accuracy with real-time performance; for facial animation expression matching scenario, SSD face detector is also more suitable since it produced the highest accuracy result.

5.2.2 Facial Landmark Detection

For the facial landmark detectors in our framework, they are used in two scenarios mentioned in Section 1.5.1 and Section 1.5.2. For landmark mapping scenario, the facial landmark detector needs to satisfy real-time performance as the first priority, then try to achieve higher accuracy. For human-face-based facial animation matching scenario, the facial landmark detector only needs to try and achieve higher accuracy according to the sole requirement for the scenario.

In order to test the accuracy and speed of our implemented facial landmark detection algorithms, we choose normalized error as our accuracy standard and FPS as our speed unit. Before introducing our experiment details, we would like to give an overview of evaluation concepts used in facial landmark detection domain.

Normalized Error: In order to evaluate the accuracy of facial landmark detection algorithm, we use normalized error mentioned in Wu et al. [1] as the measurement criteria, which focuses on comparing the detected landmark locations to the groundtruth facial landmark locations. Assuming a location i, the detected landmarks locations as $d_i = (d_{xi}, d_{yi})$ and ground truth landmark locations $g_i = (g_{xi}, g_{yi})$. Regular landmark detection error is:

$$error_i = ||d_i - g_i||^2 \tag{10}$$

The problem for this equation is that the error could change because of different size of faces. So in our experiments, we normalize our error by calculating the distance of left and right pupil centers. Assuming left pupil center location and right pupil

center location as g_{le} and g_{re}. The normalized error for location i is :

$$normError_i = \frac{||d_i - g_i||^2}{||g_{le} - g_{re}||^2} \tag{11}$$

The average error for one image is:

$$normErrorImage = \frac{1}{N}\sum_{i}^{N} \frac{||d_i - g_i||^2}{||g_{le} - g_{re}||^2} \tag{12}$$

5.2.2.1 Experiment Design

We chose 300 Faces in-the-Wild Challenge (300-W) [15] as the validation dataset, which comprises of five sub-datasets: Annotated Faces in the Wild(AFW) [25], AFLW[63], LFPW[64], Helen[65] and IBUG [66]. Especially, we chose IBUG, the testing sets of LFPW, and the testing sets of Helen as our testing dataset since they use 68-points facial landmark model, which has 689 images in total. Since the 300-W dataset provides the bounding boxes for initialization, we directly performed affine transformation of bounding box to match the bounding box around the 68 labelled facial landmarks. We used inter-ocular distance (IOD) normalized error mentioned above to measure accuracy. For profile faces where one of the eyes is visible, we used the average of width and height of the face, and we then tested the speed for each algorithm to process 500 selected images and record the FPS as speed. In this way, we could compare our implementations with other baseline and state-of-the-art facial landmark detection algorithms according to the survey of Wu et al. [1]. Our evaluation method followed the same way in the work of Wu et al. [1].

5.2.2.2 Experiment Result

Table 5 lists the normalized error and speed for each implemented algorithm.

Table 5: Facial Landmark Normalized Error and Speed Evaluation

Facial Landmark Detector	Normalized Error	Speed (FPS)
LBF	6.34	256.29
CLNF	6.77	24.64
CECLM	5.89	19.38

According to the survey work of Wu et al. [1], we chose the TCDCN facial landmark detection algorithm proposed in paper, "Facial Landmark Detection by Deep Multi-task Learning", of Zhang et al. [67] and the 3DDFA algorithm proposed in paper, "Face alignment across large poses: A 3d solution", of Zhu et al. [68] as our baseline algorithms. These two algorithms achieved the best results listed in the survey paper of Wu et al. [1] on 300-W datasets, the normalized error of TCDCN achieved 5.54 and 3DDFA achieved 6.31.

According to Table 5, we can see that LBF-based facial landmark detector has much faster speed than the other two facial landmark detectors. On the other hand, in terms of accuracy, CECLM facial landmark detector has better performance than the other two detectors.

Based on test results, we conclude that, for landmark mapping scenario, LBF facial landmark detector is more suitable since it is the only one achieving real-time performance. For facial animation expression matching scenario, CECLM facial landmark detector is more suitable since it produced the highest accuracy result.

5.2.3 Static-based FER Evaluation

For the static-based facial expression recognizer in our framework, they are used in one scenario mentioned in Section 1.5.2. The expression recognizer only needs to try and achieve higher accuracy according to the sole requirement for the scenario, and we also tested the speed performance as an extra research interest.

In order to test the accuracy and speed of each static-based facial expression recognition model, we chose the top-1 accuracy as our accuracy measure and FPS as our speed measure.

5.2.3.1 Experiment Design

For static-based FER evaluation, we chose FER2013 and SFEW 2.0 as our validation datasets. The details of FER2013 and SFEW 2.0 have been mentioned in Section 2.4.1. With these two datasets, we could compare our implementations with state-of-the-art methods according to the survey of Shan et al. [61]. Since the training dataset we used for training networks is the training subset of FER2013, we can only use the other 3589 test images left of FER2013 and 372 test images of SFEW2.0 to evaluate our models.

5.2.3.2 Experiment Result

According to the survey of Shan et al. [61], the highest accuracy performance result on FER2013 is 0.752, which was achieved by Pramerdorfer et al. [69]. As for SFEW 2.0 dataset, the highest accuracy achieved 0.6129, which was achieved by Kim et al. [70].

Table 6 shows the result of static-based FER evaluation. According to this table, we can see that Fine-tuning ResNet50 is slower than 34 layers ResNet and is much slower than shallow 2D CNN model. On the other hand, ResNet50 achieved the highest accuracy on FER2013 and SFEW2.0 datasets. Based on the six tested state-of-the-art methods in the survey for FER2013 dataset, the overall accuracy range is 71.2% - 75.2%; and for SFEW2.0 dataset, the overall accuracy range is 54.51% - 61.29%. The accuracy of our trained model, ResNet50, achieved an accuracy of 71.02% on FER2013 dataset and 54.17% on the SFEW2.0 dataset.

Based on test results, we conclude that, for facial animation expression matching scenario, fine-tuning ResNet50 recognizer is more suitable since it produced the highest accuracy result.

Table 6: Static-based Facial Expression Recognition Evaluation

Model	Accuracy on FER2013	Accuracy on SFEW2.0	Speed (FPS)
Shallow 2D CNN	0.6712	0.5127	15.26
ResNet34	0.6917	0.5315	11.18
Fine-tuning ResNet50	0.7102	0.5417	10.04

5.2.4 Dynamic-based FER Evaluation

For the dynamic-based facial expression recognizers in our framework, they can not be used in any scenario mentioned in our thesis. For our facial animation expression matching scenario, the user only inputs one human expression image, but dynamic-based facial expression recognizer takes a frame sequence as input to predict expression category. However, we provide this functionality as an extra research interest, which could help us understand the influences of network structure, the number of input frames and the number of expression categories on the accuracy and speed of models.

In order to test the accuracy and speed of each of the dynamic-based facial expression recognition models that we have implemented, we chose the top-1 accuracy as our accuracy measure and FPS (Frames Per Second) as our speed measure. Especially, we would like to emphasize that for dynamic-based facial expression recognition, they each take several continuous frames as input to predict the corresponding probabilities for each expression category.

The speed of dynamic-based facial expression recognition is defined as the number of frames that can be processed per second, not the number of frame sequences that can be processed per second. For example, there are 250 frame sequences with four images per sequence group, which means there are 1000 frames in total. Assuming there is a dynamic-based facial expression recognition model which could process these frames in one second, then the speed of this model is 1000 FPS, not 250 FPS.

5.2.4.1 Experiment Design

For the dynamic-based FER evaluation, we chose CK+ and AFEW7.0 as our validation datasets. With these two datasets, we could compare our implementations with the state-of-the-art implementations according to the survey of Shan et al. [61].

CK+ is a facial image dataset with six labeled expression categories plus neutral face. Since it is our training dataset, we only used the test subset of CK+ as our testing dataset. The details of CK+ can be found in Section 2.4.1. In addition, we

used the test subset of AFEW7.0 as the testing dataset. The details of AFEW7.0 can be found in Section 2.4.1. In total, there are 100 frame sequences and 653 frame sequences chosen from CK+ and AFEW7.0 database, respectively.

5.2.4.2 Experiment Result

According to the survey of Shan et al. [61], the highest accuracy performance result on CK+ is 0.993, which was achieved by Zhao et al. [71]. The highest accuracy performance result on AFEW7.0 is 0.486, which was achieved by Vielzeuf et al. [72].

Table 7 shows the evaluation result for accuracy on different datasets and speed with the different models' implementation. The sign of "F" at the beginning of model implementations denotes the number of frames used to classify expressions and "E" denotes the number of the total expressions that need to be recognized. For example, "8F 7E" indicates that this model uses 8 frames as input to recognize 7 expressions.

Table 7: Dynamic-based Facial Expression Recognition Evaluation

Model	Accuracy on CK+	Accuracy on AFEW7.0	Speed (FPS)
8F 7E Shallow 3D CNN	0.9621	0.3732	13.83
4F 7E Shallow 3D CNN	0.9520	0.3587	19.68
8F 4E Shallow 3D CNN	0.9775	0.3915	14.59
4F 4E Shallow 3D CNN	0.9689	0.3868	20.14
4F 4E ResNet18 3D CNN	0.9829	0.4118	18.20

According to Table 7, we can see some potential patterns of these models. Firstly, while keeping number of expression category consistent, the more frames processed for recognition, the higher accuracy the network can achieve. For example, the "8F 7E Shallow 3D CNN" has higher accuracy in CK+ and AFEW7.0 compared with "4F 7E Shallow 3D CNN", and it is the same for "8F 4E Shallow 3D CNN" and "4F 4E Shallow 3D CNN". Besides the pattern on the number of input frames, if we keep the number of frame input consistent, the networks with fewer expressions to classify has better accuracy performance. For example, "8F 7E Shallow 3D CNN" and "4F 7E Shallow 3D CNN" have higher accuracy compared with "8F 4E Shallow 3D CNN" and "4F 4E Shallow 3D CNN" separately.

In terms of accuracy performance, "4F 4E ResNet18 3D CNN" has higher accuracy compared to other shallow 3D CNN models; "4F 7E Shallow 3D CNN" has the lowest accuracy which follows the pattern we discussed above.

As for the speed performance, the solutions with 8 frames as input are slower than the solution that has 4 input frames, which could be seen from speed result comparison between "8F 7E Shallow 3D" and "4F 7E Shallow 3D", or comparison between "8F 4E Shallow 3D" and "4F 4E Shallow 3D". Overall, "4F 4E Shallow 3D" solution has the highest speed compared with other solutions.

Note, the patterns mentioned above are tested under a small dataset with limited experiment. We need to gather more data and conduct more experiments to prove the legitimacy of these potential patterns in the future.

5.3 Summary

In Section 5.1, we described how our framework solution meets its goal and requirements. We evaluated our facial analysis methods implemented in our solution in Section 5.2. The evaluation approaches we used are based on the literature and are used to evaluate some state-of-the-art algorithms in specific facial analysis domains. On the other hand, the results in Section 5.2 are also used to prove the realization of our goals in stated in Section 5.1. In the next chapter, we will give a summary of our work. Then we will discuss the limitations of our work and the future work we are planning to do.

Chapter 6

Conclusion and Future Work

In this chapter, we restate the major ideas of this thesis and summarize the results obtained in Chapter 5 briefly. We also discuss the known limitations we have currently and analyze some potential research focuses for other researchers who would like to work on top of this thesis.

6.1 Summary of Results

In this thesis, we developed a developer-friendly and extensible facial analysis framework solution that realizes the functionalities of face detection, facial landmark detection, facial expression recognition, and such solution accomplishes our goal in Section 1.4.

By analyzing the requirements mentioned in Section 1.5, we implemented our solution in a software framework manner. More specifically, our solution is built upon a core framework which provides infrastructure services, and a facial analysis module consists of four submodules: face detection submodule, facial landmark detection submodule, static-based facial expression recognition submodule, and dynamic-based facial expression recognition submodule.

Within the core framework, we provided a device module, a framework-level common data structure module, a deep learning support module, a viewer module, and a pipeline module. The device module provides functionality for adapting

different data structures of various camera devices to provide common APIs. The common data structure module ensures communication between different modules uniformly and consistently. The deep learning support module realizes data transfer between Python and C++, hence allowing the integration of Python-based deep learning implementations. The viewer module displays the resulting data for visualization purpose. The pipeline module allows the developers to define and control the whole process execution by setting parameters without worrying about the connection between the pipeline filters or handling intermediate results.

In order to realize the three facial analysis functionalities proposed in our goal (Section 1.4), we instantiated three face detectors, three facial landmark detectors, three static-based facial expression recognizers, and two dynamic-based facial expression recognizers. With different instantiations on the same functionality, we found the implementation that better address the actual requirements, and at the same time made our framework more versatile.

For the face detector, we implemented three kinds that are widely-practical and are advanced in speed performance: The Haar-cascade face detector, the HOG face detector, and the SSD face detector. In Section 5.2.1, we designed more calibrated experiments to verify the accuracy and speed performance of the three face detectors, and we landed on the statement that the three detectors all satisfy our real-time performance requirement. Furthermore, the SSD face detector had the highest accuracy result among all our implemented detectors, which ranked 19th over 60 published face detectors according to the evaluation of Face Detection Data Set and Benchmark (FDDB) [60].

For facial landmark detectors, we chose to implement three different types of fast facial landmark detectors, LBF, CLNF, and CECLM, based on the same 68-points facial landmark model, which implies that they all detect and output the same number of landmarks. This enables us to compare the performance results intuitively. In Section 5.2.2, we designed more calibrated experiments to verify the accuracy and speed performance of the three detectors, and we found that only the LBF detector can achieve real-time performance with the benchmark of real-time being 30 FPS,

and its accuracy performance ranked 5th over 11 detectors that were tested from the experiment in the survey. The CECLM facial landmark detector had the highest accuracy result among all our available detectors, which ranked 3rd over 11 detectors that were tested from the experiment in the survey.

For facial expression recognition (FER), we provided two approaches: The first approach is static-based FER, and the second is dynamic-based FER. We trained three different deep learning models for static-based FER to compare the performance on accuracy and speed for networks with different structures. The ResNet50 model achieved 71% accuracy, which is only 4% lower than the best state-of-the-art method and is very close to the accuracy range 71.2% - 75.2% for representative work in the survey we used for comparisons [70].

We trained five different models for dynamic-based FER to demonstrate how the number of input frames, output expression categories, and different network structures might affect the accuracy and speed performance. All models tested in the CK+ database achieved accuracy of 95% and above. Especially, the ResNet model reached 98.29% accuracy, which is only 1% lower than the accuracy of 99.3% obtained by the state-of-the-art method in survey [70].

To prove that our solution satisfies the proposed scenarios listed in Section 1.5, we created concrete applications employing our framework instance in Section 4.2. For the two scenarios mentioned in Section 4.2.1 and Section 4.2.2, we will next provide more specific recommendations based on the scenarios and experimental performance analysis conducted on each instantiation. For real-time landmark-based facial action mapping to keep a relatively high accuracy while ensuring real-time performance, we recommend the combination of SSD face detector and LBF facial landmark detector. For human-face-based facial animation expression matching, we recommend the combination that achieved the highest accuracy performance, which consists of SSD face detector, CECLM facial landmark detector, and fine-tuning ResNet50 facial expression recognition.

6.2 Limitations

Objectively speaking, although we claim that our proposed solution in this thesis accomplished the goals we set up in Section 1.4 and fulfilled the requirements stated in Section 1.5, there are still some limitations with our current solution:

- Our current framework solution could only support the depth camera. Other types of camera devices like motion capture camera and 3D camera would not be supported, which hinders our solution from achieving a broader definition of extensibility.

- Although we proved the mechanism integrity of our current human-face-based facial animation expression matching application, we cannot verify the accuracy of the system without an actual animation expression database.

- Our currently implemented algorithms for facial expression recognition are based on deep learning network, so these default facial expression recognition algorithms could be slower in speed performance on a CPU-based running environment.

- Our current facial expression recognition models are trained on single dataset which is minimally compliant for the validity of our scientific experiments. If we trained the models under multiple datasets, the models could learn more generic features, hence making them more accurate in practice.

- Our facial landmark detection algorithms can only detect 68 landmarks on human face and do not allow users to add more landmarks for finer facial detection. This may limit the area of application for our solution, as it might not suffice for more granulated facial feature detection.

- Our current framework solution is only a subset of CRIFTS VFX Studio's requested need and has not yet been tested in the context of movie production.

6.3 Future Work

The work in this thesis is a foundation stone of the OpenISS framework and facial analysis specialized framework. We believe our solution can further evolve and be leveraged and segmented into more powerful specialized applications in the future, such as dysmorphic facial signs detection for medical diagnosis, biometrics/face recognition for security systems, driver fatigue monitoring that spans numerous areas of application. Below, we would like to list some research topics that we are continuously working on or planning to do.

More devices support: Just as we mentioned in the limitations, our framework can only support three kinds of RGB-D camera devices at the current stage. Recently, Azure Kinect has been released which could provide better performance in speed and resolution, and at the same time, RealSense released two new cameras named D435i and T265. We would like to provide timely support for new released camera devices of different types in the future, which could enhance the usability and practicality of our framework.

More facial analysis functions: At this moment, our facial analysis specialized framework provides facial analysis functionalities in face detection, facial landmark detection, and facial expression recognition. Nevertheless, there are lots of works in facial analysis domains, such as eye gaze detection, head pose detection, facial action unit detection and etc, which could be integrated into our solution. We would like to provide more facial analysis functions for our solutions and build it towards a comprehensive facial analysis solution.

Java API wrapper: Currently, our solution is written in C/C++, in order to use the OpenISS framework as the back-end, we would like to support a Java API wrapper in the future.

Visual Effect: Besides the analysis on face, we could also combine the visual effect with detected facial features to propose more creative applications, for instance, live communication with animation character or creation of an animated human face.

Full-platform support: Up until now, our solution can only work with Linux

(Ubuntu distribution) and macOS. In the future, we would like to provide support for more operating systems.

Integration to production pipeline: Integrate the OpenISS instance with CRIFST VFX Studio production pipeline to deploy our complete solution in the context of movie production.

Bibliography

[1] Y. Wu and Q. Ji, "Facial landmark detection: A literature survey," *International Journal of Computer Vision*, vol. 127, no. 2, pp. 115–142, 2019.

[2] Wikipedia contributors, "Reallusion — Wikipedia, the free encyclopedia," 2018. [Online; accessed 22-June-2019].

[3] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *international Conference on computer vision & Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893, IEEE Computer Society, 2005.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[5] T. Baltrusaitis, P. Robinson, and L.-P. Morency, "Constrained local neural fields for robust facial landmark detection in the wild," in *The IEEE International Conference on Computer Vision (ICCV) Workshops*, June 2013.

[6] A. Zadeh, T. Baltrusaitis, and L.-P. Morency, "Convolutional experts network for facial landmark detection," in *Proceedings of the International Conference on Computer Vision & Pattern Recognition (CVPRW), Faces-in-the-wild Workshop/Challenge*, vol. 3, p. 6, 2017.

[7] S. Ren, X. Cao, Y. Wei, and J. Sun, "Face alignment at 3000 fps via regressing local binary features," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1685–1692, 2014.

[8] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops*, pp. 94–101, IEEE, 2010.

[9] I. J. Goodfellow, D. Erhan, P. L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, *et al.*, "Challenges in representation learning: A report on three machine learning contests," in *International Conference on Neural Information Processing*, pp. 117–124, Springer, 2013.

[10] A. Dhall, R. Goecke, S. Lucey, T. Gedeon, *et al.*, "Collecting large, richly annotated facial-expression databases from movies," *IEEE multimedia*, vol. 19, no. 3, pp. 34–41, 2012.

[11] M. Pantic, M. Valstar, R. Rademaker, and L. Maat, "Web-based database for facial expression analysis," in *2005 IEEE international conference on multimedia and Expo*, pp. 5–pp, IEEE, 2005.

[12] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *British Machine Vision Conference*, 2015.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[15] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic, "300 faces in-the-wild challenge: The first facial landmark localization challenge," in *The IEEE International Conference on Computer Vision (ICCV) Workshops*, June 2013.

[16] H. Huang, J. Chai, X. Tong, and H.-T. Wu, "Leveraging motion capture and 3d scanning for high-fidelity facial performance acquisition," in *ACM Transactions on Graphics (TOG)*, vol. 30, p. 74, ACM, 2011.

[17] T. Palamar, *Mastering Autodesk Maya 2016: Autodesk Official Press*. John Wiley & Sons, 2015.

[18] A. Kapoor and R. W. Picard, "Real-time, fully automatic upper facial feature tracking," in *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*, pp. 10–15, IEEE, 2002.

[19] R. O'Neill, *Digital character development: Theory and practice*. AK Peters/CRC Press, 2015.

[20] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM computing surveys (CSUR)*, vol. 35, no. 4, pp. 399–458, 2003.

[21] M. K. Dabhi and B. K. Pancholi, "Face detection system based on viola-jones algorithm," *International Journal of Science and Research (IJSR)*, vol. 5, no. 4, pp. 62–64, 2016.

[22] J. Zhu and Z. Chen, "Real time face detection system using adaboost and haar-like features," in *2015 2nd International Conference on Information Science and Control Engineering*, pp. 404–407, IEEE, 2015.

[23] B. K. Savaş, S. İlkin, and Y. Becerikli, "The realization of face detection and fullness detection in medium by using haar cascade classifiers," in *2016 24th Signal Processing and Communication Application Conference (SIU)*, pp. 2217–2220, IEEE, 2016.

[24] G. J. Edwards, T. F. Cootes, and C. J. Taylor, "Face recognition using active appearance models," in *European conference on computer vision*, pp. 581–595, Springer, 1998.

[25] D. Ramanan and X. Zhu, "Face detection, pose estimation, and landmark localization in the wild," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2879–2886, Citeseer, 2012.

[26] Y. Wu and Q. Ji, "Constrained joint cascade regression framework for simultaneous facial action unit recognition and facial landmark detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3400–3408, 2016.

[27] P. Ekman, "Strong evidence for universals in facial expressions: a reply to russell's mistaken critique.," 1994.

[28] A. Dhall, R. Goecke, S. Lucey, and T. Gedeon, "Static facial expression analysis in tough conditions: Data, evaluation protocol and benchmark," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 2106–2112, IEEE, 2011.

[29] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker, "Multi-pie," *Image and Vision Computing*, vol. 28, no. 5, pp. 807–813, 2010.

[30] C. Fabian Benitez-Quiroz, R. Srinivasan, and A. M. Martinez, "Emotionet: An accurate, real-time algorithm for the automatic annotation of a million facial expressions in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5562–5570, 2016.

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[33] B. Knyazev, R. Shvetsov, N. Efremova, and A. Kuharenko, "Convolutional neural networks pretrained on large face recognition datasets for emotion classification from video," *arXiv preprint arXiv:1711.04598*, 2017.

[34] H.-W. Ng, V. D. Nguyen, V. Vonikakis, and S. Winkler, "Deep learning for emotion recognition on small datasets using transfer learning," in *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pp. 443–449, ACM, 2015.

[35] G. Levi and T. Hassner, "Emotion recognition in the wild via convolutional neural networks and mapped binary patterns," in *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pp. 503–510, ACM, 2015.

[36] D. G. Lowe *et al.*, "Object recognition from local scale-invariant features.," in *iccv*, vol. 99, pp. 1150–1157, 1999.

[37] N. Zeng, H. Zhang, B. Song, W. Liu, Y. Li, and A. M. Dobaie, "Facial expression recognition via learning deep sparse autoencoders," *Neurocomputing*, vol. 273, pp. 643–649, 2018.

[38] L. Chen, M. Zhou, W. Su, M. Wu, J. She, and K. Hirota, "Softmax regression based deep sparse autoencoder network for facial emotion recognition in human-robot interaction," *Information Sciences*, vol. 428, pp. 49–61, 2018.

[39] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *arXiv preprint arXiv:1202.2745*, 2012.

[40] S. E. Kahou, C. Pal, X. Bouthillier, P. Froumenty, c. Gülçehre, R. Memisevic, P. Vincent, A. Courville, Y. Bengio, R. C. Ferrari, M. Mirza, S. Jean, P.-L. Carrier, Y. Dauphin, N. Boulanger-Lewandowski, A. Aggarwal, J. Zumer, P. Lamblin, J.-P. Raymond, G. Desjardins, R. Pascanu, D. Warde-Farley, A. Torabi, A. Sharma, E. Bengio, M. Côté, K. R. Konda, and Z. Wu, "Combining modality specific deep neural networks for emotion recognition in video,"

in *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, ICMI'13, (New York, NY, USA), pp. 543–550, ACM, 2013.

[41] S. E. Kahou, X. Bouthillier, P. Lamblin, C. Gulcehre, V. Michalski, K. Konda, S. Jean, P. Froumenty, Y. Dauphin, N. Boulanger-Lewandowski, *et al.*, "Emonets: Multimodal deep learning approaches for emotion recognition in video," *Journal on Multimodal User Interfaces*, vol. 10, no. 2, pp. 99–111, 2016.

[42] Z. Yu, Q. Liu, and G. Liu, "Deeper cascaded peak-piloted network for weak expression recognition," *The Visual Computer*, vol. 34, no. 12, pp. 1691–1699, 2018.

[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[44] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *CoRR*, vol. abs/1504.00941, 2015.

[45] Z. Yu, G. Liu, Q. Liu, and J. Deng, "Spatio-temporal convolutional features with nested LSTM for facial expression recognition," *Neurocomputing*, vol. 317, pp. 50–57, Nov. 2018.

[46] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[47] H. Jung, S. Lee, J. Yim, S. Park, and J. Kim, "Joint fine-tuning in deep neural networks for facial expression recognition," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[48] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 568–576, Curran Associates, Inc., 2014.

[49] N. Sun, Q. Li, R. Huan, J. Liu, and G. Han, "Deep spatial-temporal feature fusion for facial expression recognition in static images," *Pattern Recognition Letters*, vol. 119, pp. 49–61, Mar. 2019.

[50] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, *Building application frameworks: object-oriented foundations of framework design*. John Wiley & Sons, Inc., 1999.

[51] I. Jacobson, M. Griss, and P. Jonsson, *Software reuse: architecture process and organization for business success*, vol. 285. acm Press New York, 1997.

[52] D. Roberts and R. Johnson, "Patterns for evolving frameworks," in *Pattern languages of program design 3*, pp. 471–486, Addison-Wesley Longman Publishing Co., Inc., 1997.

[53] W. Pree, "Meta patterns a means for capturing the essentials of reusable object-oriented design," in *European Conference on Object-Oriented Programming*, pp. 150–162, Springer, 1994.

[54] Wikipedia contributors, "Software framework — Wikipedia, the free encyclopedia." [onlne], accessed: 2019-06-23, 2019. https://en.wikipedia.org/wiki/Software_framework.

[55] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1755–1758, 2009.

[56] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," Tech. Rep. 07-49, University of Massachusetts, Amherst, October 2007.

[57] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[58] S. Yang, P. Luo, C. C. Loy, and X. Tang, "Wider face: A face detection benchmark," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[59] T. Baltrusaitis, A. Zadeh, Y. C. Lim, and L.-P. Morency, "Openface 2.0: Facial behavior analysis toolkit," in *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pp. 59–66, IEEE, 2018.

[60] V. Jain and E. Learned-Miller, "Fddb: A benchmark for face detection in unconstrained settings," Tech. Rep. UM-CS-2010-009, University of Massachusetts, Amherst, 2010.

[61] S. Li and W. Deng, "Deep facial expression recognition: A survey," *arXiv preprint arXiv:1804.08348*, 2018.

[62] P. Hu and D. Ramanan, "Finding tiny faces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 951–959, 2017.

[63] M. Koestinger, P. Wohlhart, P. M. Roth, and H. Bischof, "Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization," in *2011 IEEE international conference on computer vision workshops (ICCV workshops)*, pp. 2144–2151, IEEE, 2011.

[64] P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar, "Localizing parts of faces using a consensus of exemplars," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 12, pp. 2930–2940, 2013.

[65] V. Le, J. Brandt, Z. Lin, L. Bourdev, and T. S. Huang, "Interactive facial feature localization," in *European conference on computer vision*, pp. 679–692, Springer, 2012.

[66] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic, "300 faces in-the-wild challenge: The first facial landmark localization challenge," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 397–403, 2013.

[67] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "Facial landmark detection by deep multi-task learning," in *European conference on computer vision*, pp. 94–108, Springer, 2014.

[68] X. Zhu, Z. Lei, X. Liu, H. Shi, and S. Z. Li, "Face alignment across large poses: A 3d solution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 146–155, 2016.

[69] C. Pramerdorfer and M. Kampel, "Facial expression recognition using convolutional neural networks: state of the art," *arXiv preprint arXiv:1612.02903*, 2016.

[70] B.-K. Kim, H. Lee, J. Roh, and S.-Y. Lee, "Hierarchical committee of deep cnns with exponentially-weighted decision fusion for static facial expression recognition," in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pp. 427–434, ACM, 2015.

[71] X. Zhao, X. Liang, L. Liu, T. Li, Y. Han, N. Vasconcelos, and S. Yan, "Peak-piloted deep network for facial expression recognition," in *European conference on computer vision*, pp. 425–442, Springer, 2016.

[72] V. Vielzeuf, S. Pateux, and F. Jurie, "Temporal multimodal fusion for video emotion classification in the wild," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pp. 569–576, ACM, 2017.