HAW HAMBURG

Masterarbeit

Thi Huyen Cao

Video-based Facial Expression Recognition with Deep Learning

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Thi Huyen Cao

Video-based Facial Expression Recognition with Deep Learning

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 1. Juli 2022

Thi Huyen Cao

Thema der Arbeit

Videobasierte Gesichtsausdruckserkennung mit Deep Learning

Stichworte

Gesichtsausdruckserkennung, Deep Learning, diskrete Emotion, Cascade Network, C3D, LSTM, BiLSTM, ConvLSTM, BU4FE, VGGFace

Kurzzusammenfassung

Gesichtsausdruckserkennung ist ein schnell wachsendes Foschungsgebiet im Bereich Computervision. Sie kommt einer Vielzahl von Domänen zugute, darunter Mensch-Computer-Interaktion, Robotik, Bildung, Unterhaltung, Medizin, Sicherheit und weiteren. Mit dem Aufkommen von Deep Learning wurden viele neuronale Netze dafür genutzt, welche vielversprechende Ergebnisse erzielten. Eine Vielzahl von frühen Studien konzentrierten sich jedoch auf statische Bilder und beachteten die zeitliche Charakteristik des Gesichtsausdrucks nicht. Ziel dieser Arbeit ist es, einen umfassenden Überblick über den Stand der Technik der Gesichtsausdruckserkennung und Deep Learning zu geben. Darüber hinaus wird ein Experiment durchgeführt, um weitere Erkenntnisse darüber zu gewinnen, wie verschiedene Arten von neuronalen Netze bei der dynamischen Gesichtsausdruckserkennung abschneiden. Während des Experiments werden Details über das Trainingsverfahren sowie die Bewertung verschiedener Trainingstechniken vorgestellt.

Thi Huyen Cao

Title of Thesis

Video-based Facial Expression Recognition with Deep Learning

Keywords

Facial Expression Recognition, Deep Learning, discrete emotion, Cascade Network, C3D, LSTM, BiLSTM, ConvLSTM, BU4FE, VGGFace

Abstract

Facial Expression Recognition is a rapidly growing field of research in Computer Vision. It benefits a wide variety of domains including human computer interaction, robotics, education, entertainment, medicine, safety, security, and so on. With the rise of Deep Learning, many Neural Networks have been exploited and achieved promising results. Yet, most studies in the early stages focused on static image while ignoring the temporal characteristic of facial expression. This work aims to deliver a comprehensive overview of the state of the art of Facial Expression Recognition and Deep Learning. Furthermore, it conducts an experiment to gain more insights on how different types of Neural Network perform towards dynamic Facial Expression Recognition. Throughout the experiment, details about the training procedure as well as evaluation of different training techniques will be presented.

Contents

List of Figures

List of Tables

Acronyms

CNN Convolutional Neural Network.

ConvLSTM Convolutional Long Short-Term Memory.

DL Deep Learning.

FER Facial Expression Recognition.

LRCN Long-term Recurrent Convolutional Network.

LSTM Long Short-Term Memory.

ML Machine Learning.

NN Neural Network.

RNN Recurrent Neural Network.

SOTA state of the art.

1 Introduction

Emotions are very important to human communication. In fact, emotion is such an essential factor of what makes us who we are. We express emotions on a daily basis regardless of our background, age, gender, culture, etc. We operate partly consciously or unconsciously on our perception of others' emotions while constantly expressing our own. Yet, emotions are very complex. Our desire to understand their core is known for decades in a number of fields including psychology, sociology, medicine, and computer science. One of the biggest research areas is called Affective Computing. It is interdisciplinary and focuses on developing machines/systems with the ability of recognizing, interpreting, processing, and stimulating human emotions, or in other words having emotions. Our dream of having some forms of artificial intelligence as companions is reflected in a lot of books, movies, and articles. With the advanced technology nowadays, we are close to our dream more than ever. In the last decades, there has been growing interest in Affective Computing thanks to its benefits in a variety of domains such as data analysis, human computer interaction, robotics, education, entertainment, autonomous driving, and so on. Concrete examples can be further read in [13].

One of the core studies in Affective Computing is about recognizing emotions. A lot of factors can contribute to Emotion Recognition encompassing facial expression, voice, text, context, gesture, posture, electroencephalography (EGG), along with others. Among them, facial expression is considered as the source of most information when it comes to detecting emotions. Therefore, Emotion Recognition and Facial Expression Recognition (FER) are often mentioned together or implied as one [13].

The very first step of Emotion Recognition is the choice of emotion theories. Emotion approach is a controversial topic, especially in psychology. Two commonly known approaches are discrete emotion and continuous emotion. Discrete emotion refers to a limited number of universal emotions whereas continuous emotion distributes emotions along the axes. FER task prefers discrete emotion rather than continuous emotion due to its simplicity in the mean of algorithm and data.

In the early stages, most studies focused on analyzing static images independently while ignoring the temporal relationships among frames. However, the information of how the face changes throughout the expression is as important as how it looks like at a single moment. Recently, with the rise of Deep Learning (DL), dynamic FER has been tackled in many studies with promising results.

This thesis covers a comprehensive overview of the state of the art of Deep Learning as well as Facial Expression Recognition. Gaps will be identified and a corresponding experiment will be conducted to extend the current knowledge.

The thesis is divided into 4 chapters. This chapter gives a brief introduction to FER and DL. The rest of the thesis is organized as follows. Chapter 2 summarizes related literature around a number of topics including Face Detection, Face Tracking, different types of Neural Network, DL training techniques, DL evaluation techniques, emotion approaches, emotion datasets, challenges of FER, and the algorithms which have been exploited to solve the FER task so far. In the end, the knowledge gap will be identified and the objectives and scope of the thesis will be defined correspondingly. Chapter 3 will follow up with an experiment to serve the defined objectives. It will cover all the details from designing to executing to validating results. The last chapter aims to give a summary of all the work done and future perspectives of FER. Gained knowledge will be highlighted.

2 Literature review

This chapter aims to introduce the general concepts around Deep Learning and the state of the art of Facial Expression Recognition. It covers current methods for preprocessing faces from data sources, a range of different types of up-to-date Neural Network as well as how to train and evaluate them. In addition, it lists a number of approaches, datasets, and challenges related to FER. Many aspects have also been explained in detail in the author's previous works [13] [11].

2.1 General concepts

2.1.1 Face Detection and Tracking

Detection

Face Detection serves as the first essential step for any tasks involving the facial area. It helps reduce redundant information in the surrounding area. Haar-feature-based Cascade Classifier a.k.a Haar Cascade by Paul Viola and Michael Jones [45] [46] is known as the breakthrough in the field of Face Detection and is also one of the most commonly used algorithms till now.

It is a Machine Learning (ML) algorithm, trained with a lot of images with and without faces, known as positive and negative samples. The training process is simplified as follows (detailed explanation in [11]). Firstly, it calculates so-called haar features [1]. Secondly, it applies an efficient boosting algorithm called Adaboost to pick up the most relevant features. It learns a range of weak classifiers and combines them weightedly to a strong classifier. Lastly, it introduces the concept of Cascade of Classifier to increase

[1] A haar feature considers adjacent rectangular regions at a specific location in a detection window, summarizes the pixel intensities in each region and calculates the differences between these sums.

the detection performance and reduce computation at the same time by rejecting non-face regions at the early stages. Cascade of Classifier contains multiple stages that use classifiers with increasing complexity. As that, the early stages use quite simple classifiers and cost consequently less computing power, however, remove efficiently a lot of negative sub-windows. Sub-windows that pass will be further processed in later stages with more complex classifiers. In summary, Haar Cascade is a robust and computationally efficient face detector. Pre-trained networks are available here [2]. Nonetheless, this approach of hand-crafted feature extraction is not powerful enough to handle faces in uncontrolled environments with high precision. [35] provides a comprehensive survey of the state of the art of Face Detection. It summarizes a number of new approaches to Face Detection and their evaluated performance on popular benchmarks including 1) Cascade-CNN Based Model 2) R-CNN and Faster-RCNN Based Model 3) Single Shot Detector Models (SSD) 4) Feature Pyramid Network Based Model (FPN), and so on.

In fact, with the rise of Deep Learning in recent years, many face detectors trained with DL have been exploited and achieved the state of the art of Face Detection in terms of accuracy and speed. Some popular well-known networks are:

- Multi-task Convolutional Neural Network (MTCNN) [49]
- RetinaFace [16]
- TinaFace [51]

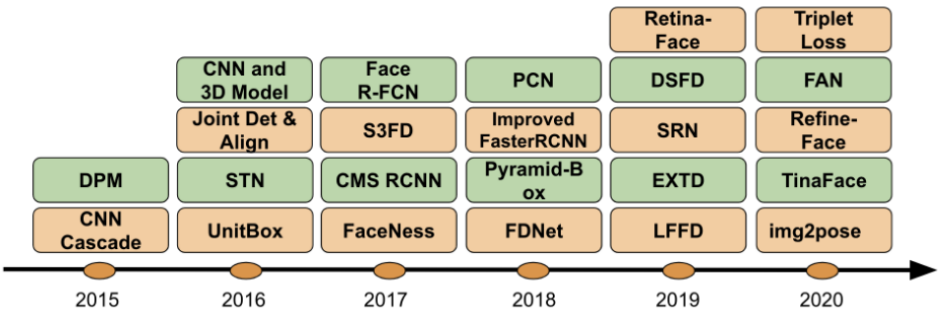


Figure 2.1: The timeline of deep-learning-based face detection algorithms [35]

Many of them do not limit to localizing the faces but they also detect precise facial landmarks. They are very well documented and lots of them come with implementation

[2] https://github.com/opencv/opencv/tree/master/data/haarcascades

and pre-trained parameters as open-source packages (e.g. mtcnn, retina-face). Figure 2.1 illustrates more examples of Face Detection algorithms proposed from 2015 to 2020. To monitor the best models for Face Detection on certain datasets, the author highly recommends a brief glance on this site [3].

Tracking

Detecting faces in videos can be done frame-basedly. However, it tends to ignore the correlation among frames which can bring some advantages in case of occlusion or fast movement. Therefore, tracking is recommended as the extended step while dealing with video data sources. Tracking takes advantage of the face location in previous frame and provides consequently a better result as well as faster processing time. [2] described some of the popular trackers with their advantages and disadvantages such as Boosting, MIL, KCF, TLD, MedianFlow, MOSSE, and CSRT. Ultimately, the decision of a suitable tracker depends on the task requirements itself (accuracy, inference time), as well as the quality of data. CSRT is, for example, a very good candidate if there is no high priority for processing time. MedianFlow, on the other hand, offers high speed and a quite good result if there is no large jump of motion in the video.

Motivated by its huge success in plenty of applied fields, DL has also been introduced to single object tracking, known as deep tracker. [4] gives comprehensive insights into lots of proposed deep trackers in terms of architecture and performance. It explains in detail the components of such deep trackers as shown in figure 2.2 as well as their recorded accuracy on a number of popular visual benchmark datasets such as OTB-2013, OTB-2015, and LaSOT. In each timestamp, the feature extraction module extracts features from target templates in the motion module and the search frame. Then, these two kinds of features are fed through the regression module to generate the bounding box. For more details about each module, please refer to [4].

Some popular successful deep trackers use the family of Siamese networks: SiamFC, SiamRPN, SiamRPN+, SiamRPN++. A detailed explanation of such networks is out of the scope of this work. Please refer to its original papers for more information.

[3] https://paperswithcode.com/task/face-detection

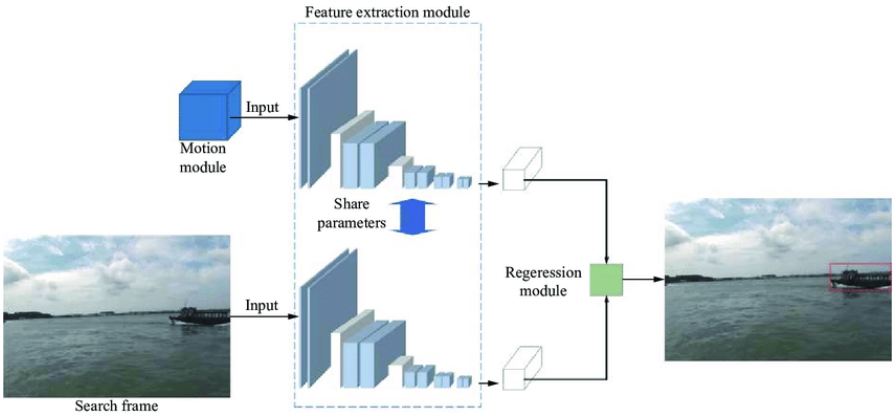


Figure 2.2: General components of deep trackers

2.1.2 Neural Network

DL is a big part of the ML landscape. It uses deep artificial Neural Network (NN) as learning model. Deep Neural Networks have been proven to be capable of solving a lot of challenging tasks including face recognition, voice recognition, image recognition, object detection, video classification, sentiment analysis, along with others. Its power lies in the ability of applying a specific set of algorithms on the learning model to map a number of inputs to a number of output classes, in mathematical term, a complex function approximation. It is done by constructing a number of layers (data representation), each contains a number of neurons. The information is passing through the layers depending on whether the neurons are activated or not. This process is controlled by a system of parameters including number of neurons in each layer, their weights and biases, number of layers, their type, activation functions, cost function, optimization function, learning rate, regularization methods, batch size, and so on. The goal of DL is to find the best set of parameters so that the right information is filtered out. The learned information will then help solve the task. In other words, the art of DL is about learning how to learn. In many cases, it is even able to learn directly from raw data which opens the opportunity of building an end-to-end pipeline.

The process of finding the best parameters is simplified as follows (detailed explanation in [11]). Firstly, the parameters are chosen randomly or due to previous work or developer's personal experience. Secondly, the input values are calculated all the way from input layer through hidden layers to output layer. This step is called forward propagation. The predicted output will then be compared to the correct output, also called label

or ground truth based on a given cost function (e.g. linear, square, or cross entropy). Depending on the difference between the calculated output and the ground truth, the parameters are updated in the corresponding direction based on gradient descent algorithm (backward propagation). This whole process is iterative and should be repeated until the requirements are met. Indeed, training a DL model is empirical and resource-consuming in terms of time, human effort, and computation. In the later section, the author will discuss more about a systematic hyperparameter tuning process.

In the last years, DL have been researched more than ever. At this point, it serves a range of industries from robotics, self-driving car, health care to education, security, marketing, logistics, and a lot more. To fulfill the tasks demanded by such a big spectrum of applied fields as well as to take advantage of the increasing computation and cloud resources, more and more efficient NN are designed. Below is a short introduction of some which are relevant to FER.

Convolutional Neural Network (CNN), C3D, and Residual Network

The traditional network of pure fully connected layers suffers from a lot of challenges when it comes to complex tasks: 1. exploding number of parameters and 2. missing information of pixel in neighborhood or information in the past sentences or segments. CNN was born as the breakthrough in the field of image recognition. The very first successful CNN architecture was proposed back in 1998 by Yann LeCun, one of the pioneers of the Deep Learning revolution. The network was originally designed for handwritten digit recognition, which is later applied to a range of other areas such as document recognition/extraction (e.g. paychecks), image recognition. As core, a CNN contains 3 main types of layer: convolutional layer (CNN), pooling layer (POOL), and fully connected layer (FC). CNN layer applies the mathematical convolution operation (*) by sliding a filter kernel or convolutional kernel (f=nxn) through all possible locations in the input (image in input layer or feature maps in hidden layers). Figure 2.3 illustrates the operation. By doing so, it captures very well the relationship among neighbor pixels which is essential for extracting spatial information. Furthermore, weights are shared across regions which helps reduce the number of parameters. POOL layer is constructed by sliding the kernel through the input and calculating either the average or max value of the region (AVERAGE-POOLING or MAX-POOLING). As that, it helps reduce the spatial size while remaining the depth and as a consequence leads to a significant decrease in

the number of parameters and corresponding computation. FC layer lies mostly at the end of the network and serves as classification layer.

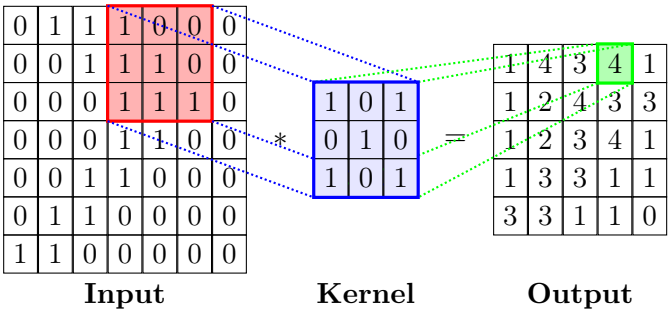


Figure 2.3: Example of convolution operation

Thanks to its power of extracting spatial features, many efficient CNNs were designed and have gained huge success at the annually ImageNet Image Large Scale Visual Recognition Challenge (ILSVRC) e.g. AlexNet, ZF-Net, Inception (GoogLeNet), VGG, ResNet, ResNeXt, SENet, PNASNet-5. These networks were afterward used as transfer learning to many other tasks. This topic will be further discussed in the later section.

Moving to video data sources, the traditional CNN tends to miss the correlation among frames since it only focuses on extracting features in each separated frame. In fact, these temporal features play a very important role when dealing with frame sequences as they encode the motion information across the sequence. As a solution, C3D was introduced in 2010, also known as 3D ConvNet or 3D CNN [30]. It was originally explored to solve the task Human Action Recognition. As the name suggests, C3D uses 3D kernel instead of 2D and as a result, be able to capture both spatial features in each frame as well as temporal features in multiple adjacent frames. Despite that, the learning capacity of C3D when it comes to features in time is limited and sometimes referred as shallow. To deal with deep sequence which requires deep memory inside the network, other network types were subsequently introduced, but more on that later.

Despite the learning ability of CNN, training a deeper CNN is more difficult. The thread of exploding and vanishing gradient descent is one of the well-known struggles. [25] investigated further the problems of training and optimizing deeper CNN and proposed a more proficient CNN architecture called Residual Network. As core, it contains so-called residual block which is built from normal layer with shortcut connection or residual connection. Shortcut connections are those skipping certain layers in between. For instance, see figure 2.4.

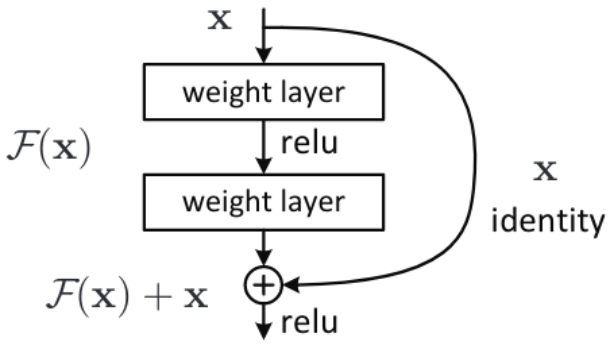


Figure 2.4: Residual learning: a building block [25]

[34] summarized some firm arguments why Residual Network can train more layers.

- Layers with shortcut connection can realize entity mapping and by all means, can not cause training worse. Therefore, it is theoretically possible to add as many residual blocks as desired.
- Residual Network behaves as an ensemble of multiple sub-networks.

Another similar architecture is called DenseNet [27] which contains basically more shortcut connections. In a dense block, there are connections among all inputs and outputs from each layer. DenseNet is proven to be able to train even deeper networks.

Recurrent Neural Network (RNN), LSTM, BiLSTM, and ConvLSTM

Another common data source is time series or sequential data. Its typical characteristics of variational length and the distribution of information through the sequence demand a different type of network. RNN fills this gap. There are a lot of works around the concept of RNN in the 80s. Unlike CNN, RNN is designed to capture the dynamics of sequences by default. It processes the sequence timestep by timestep. In order to remember the information in the previous sequence, it has an internal memory that is created through the feedback connection. A RNN layer of n neurons can be unrolled in time as n feed-forward layers with shared weights. A slightly different aspect of the training process is the backpropagation algorithm called Backpropagation Through Time (BPTT). It is simplified as follows. The loss is accumulated across all timesteps. Weights are then updated in the backward direction. Nevertheless, this way of training can be

slow if the sequence happens to be too long. Therefore, a modification of BPTT was introduced to solve this problem called Truncated BPTT(k1, k2) (TBPTT) with k1 being the number of forward steps until updating and k2 being the number of timesteps used in backpropagation. However, this simple RNN still suffers from other training problems. Concretely, to calculate the derivative of Loss with respect to Weight in the very first layers, one has to repeat the multiplication of derivatives from one step to another until the wanted layer. If the gradient is too big or too small, there is a good chance that the training process ends up with the exploding and vanishing gradients.

Almost a decade later until 1997, [26] proposed a special RNN architecture, marked the breakthrough of RNN in processing sequential data: the Long Short-Term Memory (LSTM) network. The design of LSTM is to memorize long-term dependencies. In order to achieve that, LSTM has an internal memory cell as well as several gate controls GC which manage the information flow. Input-GC/output-GC/forget-GC is composed of a sigmoid layer with a value range of (0,1). Those gate controls determine how much information should be passed through each gate (0=none, 1=all). Furthermore, LSTM resolves the problem of vanishing gradients found in the original RNN [26]. Ever since then, RNN is commonly chosen as baseline architecture for any task involving sequences and have gained a big reputation.

Followed the success of LSTM, multiple options have been explored. Among them, one has been investigated and proven to be even more effective than the above described LSTM in many cases: the bidirectional LSTM a.k.a BiLSTM. The core idea of a bidirectional RNN was first introduced back in 1997 by Mike Schuster and Kuldip K. Paliwal [40]. Instead of only remembering information from the past as unidirectional LSTM, BiLSTM offers an additional training capacity of learning simultaneously from the past and from the future or, in other words, in positive and negative time directions. In fact, in many scenarios, the information in the future is just as important as the one in the past. The simple example below demonstrates this point. Without the information in the future, it is very difficult to predict the word after Robin.

> *Robin [Scherbatsky] is my favorite character in the series "how I met your mother".*
>
> *Robin [Hood] is a folk hero who steals from the rich to help the poor.*

In the last years, there are a number of works which confirmed the effectiveness of BiLSTM as well as provided detailed performance comparison among unidirectional LSTM

and BiLSTM e.g. for traffic prediction [39], forecasting time series [42], human activity recognition [6].

Nonetheless, [41] pointed out that LSTM seems to perform poorly on image sequence since it does not take spatial correlation into consideration. They then came up with a new architecture called ConvLSTM. ConvLSTM resolves the learning problem by taking advantage of the convolution operation in both input-to-state and state-to-state transitions to capture underlying local spatial features. Figure 2.5 visualizes the inner structure of ConvLSTM.

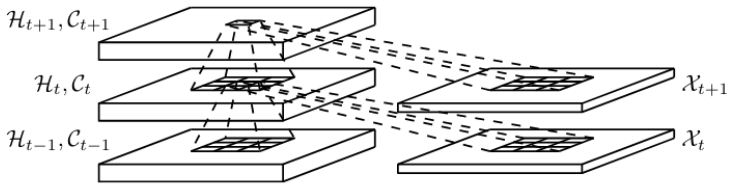


Figure 2.5: Inner structure of ConvLSTM [41]

LSTM	ConvLSTM
$i_t = \sigma(W_i.x_t + W_i.h_{t-1} + b_i)$	$i_t = \sigma(W_{xi} * X_t + W_{hi}.H_{t-1} + W_{ci} \odot C_{t-1} + b_i)$
$f_t = \sigma(W_f.x_t + W_f.h_{t-1} + b_f)$	$f_t = \sigma(W_{xf}.X_t + W_{hf}.H_{t-1} + W_{cf} \odot C_{t-1} + b_f)$
$\widetilde{C} = tanh(W_c.x_t + W_c.h_{t-1} + b_c)$	$\widetilde{C} = tanh(W_{xc}.X_t + W_{hc}.H_{t-1} + b_c)$
$C_t = f_t \star C_{t-1} + i_t \odot \widetilde{C}$	$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}$
$o_t = \sigma(W_o.x_t + W_o.h_{t-1} + b_o)$	$o_t = \sigma(W_{xo}.X_t + W_{ho}.H_{t-1} + W_{co} \odot C_t + b_o)$
$h_t = o_t \odot tanh(C_t)$	$h_t = o_t \odot tanh(C_t)$

Table 2.1: Equations of LSTM and ConvLSTM

As a matter of fact, all the inputs X_n, cell outputs C_n, hidden states H_n, and gates i_t, f_t, o_t of the ConvLSTM are 3D tensors whose last two dimensions are spatial dimensions. The key equations of LSTM and ConvLSTM are shown in table 2.1 with W_{ab} as the weight matrix from a to b relationship; $\odot$ denotes the Hadamard product and * the convolution operator. Also, it is to notice that the ConvLSTM does not use the simple LSTM but a modification with peephole connections which was proposed in 2000 [24]. The design of peepholes was made to fuse the information of the cell into the gates by direct multiplicative connections between cell and gates. [24] stated that this LSTM

variant can learn the fine distinction between sequences of spikes separated by either 50 or 49 discrete time steps, without the help of any short training exemplars.

Other networks

Besides ConvLSTM, there are many other ways to capture spatial-temporal features. One commonly used candidate is the hybrid network which combines multiple types of networks together to take advantage of them all.

In addition, there are a number of special networks for certain purposes which can not be all discussed here. AutoEncoder is a very strong network when it comes to the task of representation learning or, in simple words, learning the core knowledge embedded in the data.

Ultimately, the goal of DL is to find the best model. Yet sometimes one model can not capture all the necessary aspects. [23] covers an extensive review of ensemble learning and determines that by combining several individual models, the generalization performance improves. It also discussed a number of fusion strategies that are broadly used.

2.1.3 Deep Learning techniques

Set up datasets

No matter which network topology is applied, training a DL model requires in general a big amount of data. To avoid the learn-by-heart phenomenon where the model happens to create a huge mapping of input to output without really capturing the features, data is normally split into different sets. A common one is train and validation set (sometimes refers as dev or development set): one for the training and one for performance validation during the hyperparameter tuning process. Thus, the parameters are changed during training so that the performance on both train and validation set are not too bad which leaves a certain bias on the statement about network quality provided by validation set. Therefore, in many cases, another set is wanted for a more reliable result called the test set. The idea of test set is to apply the best-trained model on a number of completely new samples to provide an unbiased estimation of the final model before moving to the real application. A classic ratio of train/validation/test set is 60%/20%/20%. In the era of Big Data where the amount of samples is counted in millions or even billions, the

percentage of validation and test set can be reduced so that 1. There is still enough data for training and 2. The evaluation of algorithm takes less time which enables testing more algorithms and as a result speeds up the process of finding the best algorithm among all. The number can go to less than 10% or occasionally even less than 1%.

Some noticeable disadvantages of this 3 datasets approach are that the number of samples for learning reduces drastically and the result relies on a random choice of validation set. A solution for this problem is so-called k-fold Cross Validation. The procedure is described as follows. Firstly, the train set is divided into k folds: 1 for validation and k-1 for training. Secondly, the training is executed as usual, thus k times. In each iteration, another fold is chosen as validation set. After the loop is done, the final result is calculated as average of the result after each iteration. Figure 2.6 illustrates an example of 5 folds.

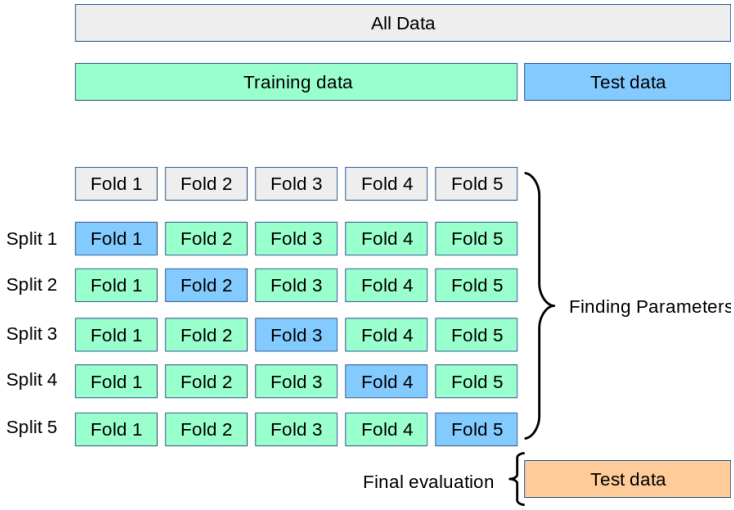


Figure 2.6: 5-fold Cross Validation [3]

This approach is indeed expensive in terms of computation, yet valuable especially in case of small dataset. The decision of k depends on a lot of factors. Common questions that should be concerned about before making this decision are:

- Are the computation resources available?
- Is the train set big enough for learning?
- Is the validation set representative enough to measure the quality of algorithms?

Hyperparameter tuning

The difficulty of training a DL model lies in the huge number of hyperparameters.

- number of layers, number of neurons in each layer
- type of network, type of layer
- weight initialization method
- learning rate
- optimization algorithm
- cost function
- and so on

Training a network can be simplified, in technical terms, as finding the best set of hyperparameters so that the model has the best generalization capacity. This process is unfortunately complex and resource-consuming because of the search around the huge combination of hyperparameters. A key aspect here is to construct a systematic way of finding and monitoring each modification of the parameters.

One of the simplest possible ways to get good hyperparameter is to brute force all possible combinations a.k.a Grid Search. This approach is unfortunately extremely computationally expensive and considered as inefficient when the data used for Grid Search is big. Random Search is another option that uses randomly candidates in the parameter's distribution. Yet those 2 options suffer hardly from the high computation cost. Bayesian Optimization algorithm is a more efficient way for hyperparameter tuning as it redeems the drawbacks of Grid Search and Random Search. It builds a probability model of the objective function to propose a better set of hyperparameters to evaluate and update the model correspondingly. As a result, it tends to find the best combination in fewer iterations.

$$F(score|hyperparameters)$$

Thus, there is no guarantee that Bayesian Optimization will always perform best among those search algorithms. If the resources are abundant, it does not seem such a bad idea to try Grid Search.

Another way to optimize the training process is to understand the current performance of the model. By monitoring the loss and accuracy course, the weights and feature maps, one can simply determine the capacity of the model, whether it is currently in Underfitting or Overfitting area, whether it is learning well or experiencing some problems. With that precious knowledge, bad parameters can be fast eliminated and the model can be pushed in fewer steps in the right direction and can achieve its best performance in a more efficient manner.

One of the biggest problems most DL practitioners face sooner or later is the phenomenon called Overfitting. Overfitting refers to the scenarios where a model performs quite well on train set but poorly on validation set, in other words, a poor generalization. It indicates that the model tends to learn by heart the samples and not the features embedded in the data. To fight against Overfitting, there are plenty of regularization techniques to try. Thus, be aware that there is no guarantee that certain methods will work 100% in all cases.

[32] emphasizes the crucial role of regularization in DL and presents an overview of regularization methods based on five main elements of NN training (data, architecture, error term, regularization term, optimization procedure). In addition, they summarized a number of works in different regularization categories and gave concrete recommendations for users of existing regularization methods. Below is a brief description of some worth-a-try techniques.

- L2 a.k.a weight decay belongs to the category of regularization via regularization term. [32] states that regularization can be achieved by adding a regularizer R into the loss function where λ is a weighting term controlling the level of regularization wanted.

  $$R = \frac{\lambda}{2m} \|w\|^2$$

  A simple explanation of the effectiveness of weight decay is that by adding a term based on weight to the loss function, the weight is automatically kept small as the loss is optimized. This leaves a lot of neurons with near-zero weights or, in common

terms, deactivated. Smaller network corresponds normally with less Overfitting. Weight decay has gained big popularity and is still used successfully until now.

- Dropout, Normalization, and Data Augmentation are the three most popular methods in the category of regularization via data. They seem to distinguish at first sight but are very similar in the way they use some sort of transformations with (stochastic) parameters to either 1. perform feature extraction, modify feature space or the distribution of the data to some representation simplifying the learning task or 2. generate more data.
  - Data Augmentation applies the transformation on the input data in order to invent more data from the limited dataset. It is crucial for certain applications where it is hardly possible to get more data such as health care, military, and security. By augmenting available data, Data Augmentation also generates a variety of contexts and backgrounds which sometimes enhance the quality of the data itself. More data corresponds commonly with better generalization.
  - Normalization applies the transformation on the input and hidden layers (Input or Batch Normalization) to re-balance the distribution of data. An unwanted imbalance of data range can have a negative effect on the training process e.g. sample with input x1 in range [0,1] and x2 in range [10000-1000000]. Ultimately, data with higher value (x2) will have more impact on the training while others tend to be ignored. This will slow down the learning process if not even prevent the model from learning enough features. Normalization solves this problem. A balanced and stable distribution of data help accelerate the training. [29] also pointed out the effectiveness of Batch Normalization against Overfitting and determined that by normalizing layer inputs, Batch Normalization allows us to use much higher learning rates and be less careful about weight initialization.
  - Dropout applies the transformation on the input and hidden layers. The key of Dropout is to randomly drop units and all their connections during training. At each time, a thinned network with fewer neurons is trained. A network of n units can then be seen as 2^n possible thinned networks [43]. Training with Dropout is like training a collection of 2^n networks with extensive weight sharing where each network gets trained rarely or at all. [43] stated that Dropout is a technique that prevents Overfitting and provides a way of

approximately combining exponentially many different NN architectures efficiently. Dropping units results in a smaller network and as a result less Overfitting. By combining different models, Dropout demonstrates a strong improvement in performance. Dropout is nowadays one of the most powerful regularization techniques. [22] investigated more about the application of Dropout in RNN and came to the conclusion that the variational inference based Dropout technique, also referred as recurrent Dropout outperforms the naive Dropout. The fundamental difference between these two is the connections dropped. While naive Dropout masks only connections from inputs to outputs, recurrent Dropout masks additionally the recurrent connections. Their proposed technique also suggests using the same mask at each timestep.

- Regularization via optimization covers 3 types of method corresponding with 3 phases of the training: weight initialization, weight update, and training termination. The general recommendation in [32] is to try several methods before picking the best one:
  - Initialization: Random weight initialization (Xavier, He) etc.
  - Update: Adam, Momentum, RMSProp, learning rate schedules etc.
  - Termination: Early stopping, fixed number of iterations etc.

Transfer learning

It is no need to "reinvent the wheel". Learned knowledge should be applied if and where it's possible. Transfer learning addresses exactly this issue by extracting useful information from data in a related domain and transferring them to a new task. On one hand, it helps avoid redundant resources needed to re-learn similar features. On the other hand, it is believed that prior knowledge assists the new learning process. That means that learning for new tasks based on previously gained knowledge might be faster and more efficient. [48] studied the transferability of deep NN and concluded that the transferability of features decreases as the distance between the base task and the target task increases. And even though the transfer effect is low, transferring features from distant tasks can still be better than using random features. [52] conducted an insightful survey about over 40 representative transfer learning approaches and their applied fields. They emphasized the potential of transfer learning in the modern ML landscape, yet

pointed out some areas which need to be further researched such as how to measure the transferability across domains and avoid negative transfer, interpretability of transfer learning, effectiveness and applicability, along with others.

In fact, there are a lot of pre-trained successful networks which are accessible for community such as the winners of ImageNet Image Large Scale Visual Recognition Challenge (ILSVRC) including AlexNet, ZF-Net, Inception (GoogLeNet), VGG, ResNet, ResNeXt, SENet, PNASNet-5. One well-known approach is called fine-tuning. Fine-tuning a network means to keep a number of base layers frozen (mostly CONV layers), replace some layers at the end with some new ones and a suitable output layer and then re-train the network with new data. By doing so, the new network has the ability of capturing not only certain basic information from pre-trained network such as edge, basic pattern, curve but also new features needed for the new task. One example mentioned in [52] is about fine-tuning AlexNet for the detection of Alzheimer's disease. The process is described as follows: 1. Use pre-trained AlexNet as a starting point 2. Replace the last 3 fully connected layers with one softmax layer, one fully connected layer, and one output layer 3. Fine-tune the new network with Alzheimer's dataset. A detailed comparison of fine-tuning different popular pre-trained models for the task of detecting plant disease can be further read in [15] where DenseNet is reported to achieve the best performance.

Despite the various advantages of decrease in training time, data, resources, and enhancement of performance, transfer learning faces like almost every other technique certain limits. One of its biggest limitations is so-called negative transfer. Negative transfer refers to the case where transfer learning undesirably hurts the performance of the target task instead of improving. Transfer learning works well under the assumption that the new task is similar to the old task. If it is not satisfied, negative transfer may occur. Furthermore, regardless of how similar developers determine the relationship between those tasks, the algorithm may not always be of the same opinion. After all, it is difficult to define the factors based on which the algorithm make decision about the transferability. [50] presents a systematic survey on negative transfer with review of fifty representative approaches for overcoming negative transfer according to four categories: secure transfer, domain similarity estimation, distant transfer, and negative transfer mitigation.

Evaluation

It is to distinguish between evaluation during training and model final evaluation. It is essential to provide fast evaluation during training such as Accuracy or F1 score. For final evaluation, it is recommended to apply techniques that provide more insights such as Confusion Matrix or detailed Error Analysis. Below is a brief description of those methods mentioned above.

- Accuracy is a commonly used evaluation metric. It is defined by the percentage of number of correctly predicted samples to the total number of samples. Yet one drawback of this simple metric is that it is not reliable in case of imbalanced data since the result hardly relies on the performance of dominated classes.

- F1 score is a function of Precision and Recall as illustrated in the equations below with FP=False Positive, TP=True Positive, and FN=False Negative. Precision measures how many of the predicted positives is actually positive. A high False Positive is easily to determine with Precision which is important for e.g. spam email detection. Detecting a spam as non-spam (FN) can be more tolerated than a non-spam as spam (FP) as important emails might be lost. A high Precision is a decisive factor for such models. Recall measures, on the other hand, how many of the positive samples are correctly predicted. Using Recall metric, a high FN is easy to detect. In case of fraud detection, anomalies detection, or disease prediction, detecting a positive case as negative (FN) is very critical. For such models, Recall is a more reliable metric to go with. F1 score combines those 2 metrics. Either a bad Recall or a bad Precision will both results in a bad F1 score.

Precision	Recall	F1 score
$P = \frac{TP}{TP + FP}$ $= \frac{TP}{Total\ Predicted\ Positive}$	$R = \frac{TP}{TP + FN}$ $= \frac{TP}{Total\ Actual\ Positive}$	$F1 = 2 * \frac{P * R}{P + R}$

- Confusion Matrix, also known as error matrix, is a detailed presentation of the predicted result in matrix form. It consists of 2 dimensions: actual and predicted. All the correctly predicted samples are located on the diagonal of the matrix. Colors are commonly used to highlight the result statistics. In case of imbalanced data,

it's better to use percentage or range [0,1] to display the result instead of absolute number. Figure 2.7 shows an example of Confusion Matrix where 75 samples of class A are predicted as class A (75%), 5 as class B (5%) and 20 as class C (20%).

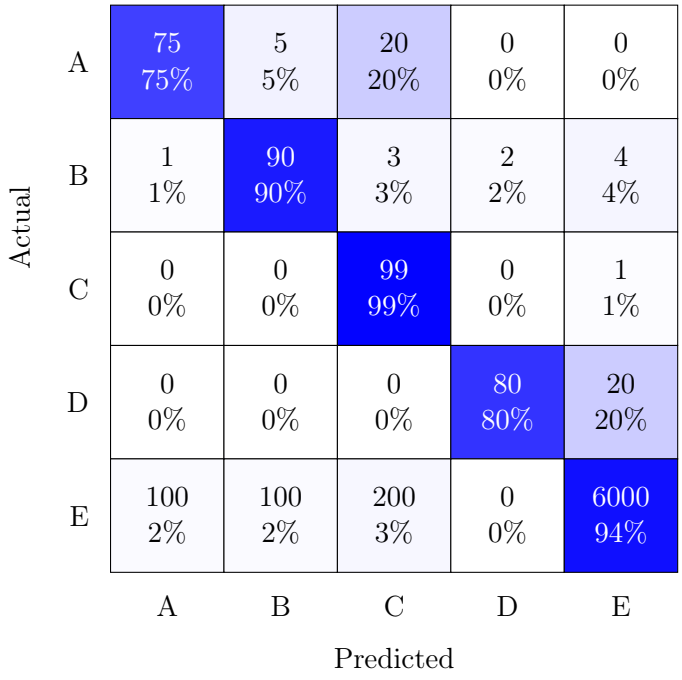


Figure 2.7: Example of Confusion Matrix

- Error Analysis is a collection of any method which helps analyze the result, especially the wrong predicted samples, for example, a table of wrong predicted samples with detailed note of the sample's characteristic as well as the predicted score.

Nonetheless, it depends on you to apply certain metrics for certain phases. Yet it is very important to set up the evaluation metrics so that performance can be tracked efficiently and the network is evaluated in a reliable manner. In fact, if the training is stuck at some points, an Error Analysis might shed some light on the matter e.g. data mismatch, different data distribution, low data quality, and so on.

2.2 Sota FER

2.2.1 Emotion approaches

There are 2 popular approaches when it comes to FER: discrete (categorical) and continuous (dimensional) approach.

Discrete emotion model refers to a limited number of universal basic emotions. One of the well-known models is the set of 6 emotions {anger, disgust, fear, happiness, sadness, surprise} proposed by Ekman and Friesen. In [19], they investigated the question whether any facial expression is indeed universal. They came up with an experiment, in which they demonstrated the fact that the members of a preliterate culture who had minimal exposure to literate cultures would associate the same emotion with the same facial expression as do members of literate cultures. The experiment was done by telling 342 Ss[4] a story, showing them 3 faces, and asking them to select the one which showed the emotion in the story. The result gave strong evidence of their hypothesis of universal emotions. The approach was followed by a lot of researchers with different number of emotions e.g. {fear, anger, joy, sadness} {anger, fear, sadness, disgust, surprise, anticipation, trust, joy}. Still, there is no consensus about the precise number of basic emotions after all.

On the other hand, continuous approach characterizes emotions on a dimensional basis. After their hypothesis, there is no distinguished emotion. Instead, there are certain factors that define emotions. These factors can be seen as axes in a coordinate system and each emotion can be shown as a point in this n-factor/n-dimension system. Some well-known models [18] are:

- two-dimensional model of valence (a pleasure-displeasure continuum) and arousal (activation-deactivation) a.k.a circumplex model of affect [37]
- three-dimensional model of tension arousal, energetic arousal, and valence
- three-dimensional model of **P**leasure (**V**alence), **A**rousal, and **D**ominance a.k.a PAD or VAD model [38]. Figure 2.8 visualizes the 6 basic emotions by Ekman in the VAD model.

.

[4]Ss were members of the Fore linguistic-cultural group, which up until 12 years before the experiment was an isolated, Neolithic, material culture.

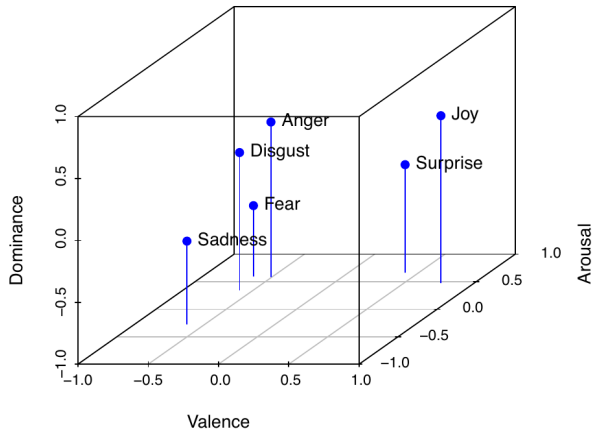


Figure 2.8: Ekman's basic emotions within the emotional space spanned by the Valence, Arousal, and Dominance axis of the VAD model. Ratings are taken from [38] [10]

There are continuous debates around these two approaches. The main argument for dimensional over categorical approach lies in its ability of capturing all possible human emotions. [18], for instance, works with emotion perceived by music and did a detailed comparison of multiple models from both approaches. [9] also shed some light on the matter from another perspective: neuroimaging studies. They indicate that there is still inefficient evidence whether certain brain circuits define certain basic emotions. Yet, they emphasized that the reason could also lie in the stimuli method, the measurement tool, or the research design itself. More research will undoubtedly clarify this concern.

For FER, both approaches have been adapted. So far, categorical approach is more often chosen because of its simplicity in terms of model and dataset.

2.2.2 Datasets

There are a lot of datasets serving FER. Thus, they differ from a number of characteristics [13]:

- Environment: laboratory-controlled, in the wild (web, movies, and real world application)
- Emotion approach: categorical, dimensional

- Ground truth: emotion, action units (AUs)
- Size: total size, number of objects, number of emotions, resolution, frame rate
- Type: static images (2D), frame sequences, video (3D)
- Model of face: 2D, 3D
- Color: gray, RGB
- Intensity: only peak emotion, neutral-peak-neutral a.k.a onset-apex-offset
- Capture: posed, spontaneous
- Bias: gender, ethics, background

Table 2.2 shows the summary of some popular FER benchmark databases. A further read about each dataset can be found in [33] [20].

Database	Sub.	Samples	Condit.	Elicit.	E.
CK	97	486 image sequences	Lab	P, S	7
CK+	123	593 image sequences	Lab	P, S	8
MMI	25	740 images + 2900 videos	Lab	P	7
JAFFE	10	213 images	Lab	P	7
TFD	N/A	112,234 images	Lab	P	7
BU-3DFE	100	2500 3D images	Lab	P	7
BU-4DFE	101	606 3D sequences	Lab	P	7
Oulu-CASIA	80	2880 image sequences	Lab	P	7
4DFAB	180	1,800,000 3D faces	Lab	P, S	7
RaFD	67	1608 images	Lab	P	8
KDEF	70	4900 images	Lab	P	7
Bosphorus	105	4888 images	Lab	P	7
AFEW	N/A	1809 videos	In the wild (Movie)	P, S	7
SFEW	N/A	1766 images	In the wild (Movie)	P, S	7
FER-2013	N/A	35,887 images	In the wild (Internet)	P, S	7
RAF-DB	N/A	29672 images	In the wild (Internet)	P, S	7+
AffectNet	N/A	450,000 images	In the wild (Internet)	P, S	7
ExpW	N/A	91,793 images	In the wild (Internet)	P, S	7
EmotionNet	N/A	1,000,000 images	In the wild (Internet)	P, S	23+

Table 2.2: An overview of FER datasets. Sub.=Subject; P=posed; S=spontaneous; Condit.=Collection condition; Elicit.=Elicitation method; E.=Expressions: 6 = anger, disgust, fear, happiness, sadness, and surprise; 7 = {6 expressions} ∪ {neutral}; 8 = {7 expressions} ∪ {contempt}; + = with compound expressions

Each dataset has its own strengths and weaknesses. While datasets in laboratory have in general good light condition with faces that are recorded in frontal position, it fails to cover the complex environment in the real world. On the other hand, while datasets collected from the internet can expose a variety of different setups, they commonly suffer from low quality. In the early days, FER works mostly on images which results in a big number of static datasets. Later on, there are more and more works focusing on the dynamic characteristic of FER. Recently, there are some datasets that even capture the 3D model of face. It opens a bright chance to solve problems like occlusion.

DL requires in general a big amount of data. Indeed, it is possible to combine different datasets. If this is the case, please be aware of the data distribution and quality before mixing them together.

2.2.3 Challenges

One of the biggest problems FER faces is data. The process of creating ground truth for training data might suffer from certain biases from the annotator (person or algorithm). It is especially difficult to create reliable labels for dimensional emotion approach as it requires expertise in the area. Collecting or creating more high-quality data can also be a problem in terms of time, human effort, and budget. Combing datasets has to be done under careful consideration of data imbalance. In case it is hardly possible to manually get more data, Data Augmentation is highly recommended to generate more data from the available data.

[33] emphasized that there are many other factors which can have an impact on emotion detection besides facial expression:

- context
- electroencephalography (EEG)
- audio
- mimics

They affirmed the effectiveness of using facial expression alone, yet pointed out the potential of a multimodal system that is capable of capturing and processing data from multiple channels.

Ultimately, a FER system should be applied in the real world which does not have the ideal environment as in the experiment. Some limitation that it might encounter is the limited computation and memory such as in mobile devices, edge devices, the imperfect light and contrast, the occlusion by objects in the surrounding or even by glasses, beard, etc. that might degrade the performance of the model. Thankfully, there is continuous research in this area e.g. hardware development to fulfill the high requirement of such tasks, network optimization methods to reduce size while avoiding accuracy degradation - more on that later.

2.2.4 Proposed methods

From the perspective of Facial Analysis, there are 2 levels of expression descriptor: global and local expression. It results in 2 research areas called Facial Expression Recognition (FER) and Facial Action Unit Recognition (FAUR), one tries to decode the global facial expression and one only certain regions.

Action Unit	Movement
AU1	Inner Brow Raiser
AU2	Outer Brow Raiser
AU4	Brow Lowerer
AU5	Upper Lid Raiser
AU6	Cheek Raiser
AU7	Lid Tightener
AU9	Nose Wrinkler
AU15	Lip Corner Depressor
AU17	Chin Raiser
AU20	Lip stretcher
AU26	Jaw Drop

Table 2.3: Action Unit example

Emotion	Action Units
Happiness	6,12
Sadness	1,4,15
Surprise	1,2,5B
Fear	1, 2, 4, 5, 7, 20, 26
Anger	4, 5, 7, 23
Disgust	9, 15, 16

Table 2.4: EMFACS for 6 basic emotions

Action Unit is defined as the movement of specific facial regions such as eyes, mouth, cheeks, and so on. Table 2.3 shows an example of some Action Units. For more, see here [5]. While FAUR serves multiple purposes, it is also used to enhance FER. Concretely, a set of

[5] https://www.cs.cmu.edu/ face/facs.htm

Action Units can define a specific emotion. Table 2.4 [6] presents a simple mapping based on the Emotional Facial Action Coding System (EMFACS) by Ekman and Friesen.

With the rise of ML and DL, many researchers have adopted the new approach of working directly with raw data instead of using Action Units for FER. Facial features are directly either handcrafted or deep-learned from raw data.

In the previous work [13], the author has summarized a number of handcrafted features which are evaluated in many works. Sometimes, a combination of different handcrafted features can further improve the performance.

feature category	description	examples
appearance	encodes pixel intensity information	Local Binary Pattern (LBP), Gabor, Histogram of Oriented Gradients (HOG), Three Orthogonal Planes (TOP), Gaussian Laguerre (GL), Weber Local Descriptor (WLD), Discrete Contourlet Transform (DCT)
geometric	encodes feature embedded on landmarks location	distance among landmarks, angle between neutral face and input face, angle formed by the segments joining three landmarks , Local Curvelet Transform (LCT)
motion	captures changes among frames	Motion History Images

Table 2.5: Examples of handcrafted feature

With the features available, a ML classifier will be trained to recognize the emotion underneath. Some popular choices are K-nearest neighbor, Support Vector Machine (SVM), Hidden Markov Model (HMM), Decision Tree (DT). Yet the performance of such classifiers is very limited and they are often categorized as "shallow learning".

Recently, DL has become the state of the art of FER. With its ability to learn directly from raw data, DL opens the opportunity of training an end-to-end FER pipeline. Figure 2.9 summarizes once again the FER framework with the current ML methods.

[6] https://www.tu-chemnitz.de/informatik/KI/projects/facedetection/index.php

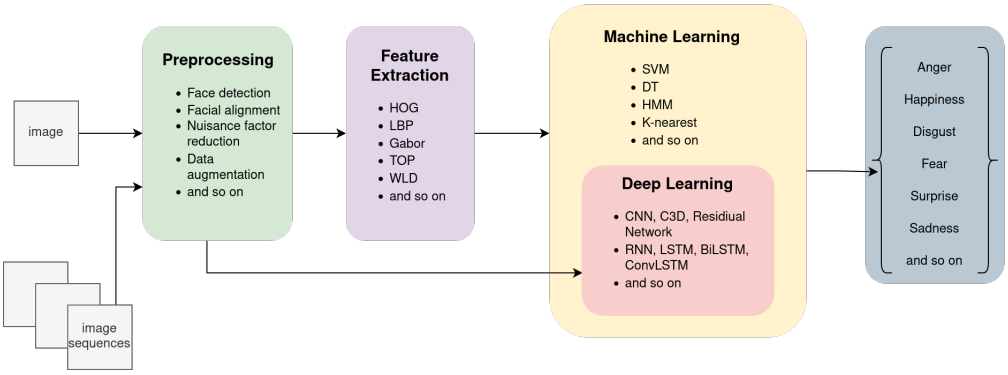


Figure 2.9: FER framework

[33] provides a comprehensive survey about deep FER. It introduces a range of neural networks and training techniques used for static and dynamic FER including:

- Transfer learning (fine-tuning pre-trained networks)
- Design special auxiliaries, layers, and blocks for FER
- Ensemble networks a.k.a fusion
- Design special network architectures

Table 2.6 shows some networks and techniques used for sequences and their performance in comparison [33].

Network type		data	spatial	temporal	frame length	accuracy	efficiency
Frame aggregation		low	good	no	depends	fair	high
Expression intensity		fair	good	low	fixed	fair	varies
Spatio-temporal network	RNN	low	low	good	variable	low	fair
	C3D	high	good	fair	fixed	low	fair
	FLT	fair	fair	fair	fixed	low	high
	CN	high	good	good	variable	good	fair
	NE	low	good	good	fixed	good	low

Table 2.6: Comparison of different types of methods for dynamic image sequences in terms of data size requirement, representability of spatial and temporal informa- tion, requirement on frame length, performance, and computational efficiency. FLT = Facial Landmark Trajectory; CN = Cascaded Network; NE = Network Ensemble

As visualized in table 2.6, the 2 best candidates for dynamic FER are Cascade Network and Network Ensemble since they are capable of learning both temporal and spatial features well. This ability is essential for FER on sequences.

Cascaded Network is about combining the strength of different types of networks. One well-known example of Cascaded Networks is the hybrid network Long-term Recurrent Convolutional Networks (LRCN) which combines CNN as front-end layers and LSTM as back-end layers so that the network can take advantage of the excellent ability of decoding spatial features from convolutional network as well as of extracting temporal features from recurrent network. As a consequence, LRCN is known to be "doubly deep", deep in space and deep in time. It has been proven to be a very effective architecture while working with image sequences or video such as Human Action Recognition, Lip Reading, and without exception FER.

In addition to CNN, other network frameworks can also be used for learning spatial features such as C3D as introduced in the previous section. [33] emphasized the power and high potential of transfer learning from a range of successful object detection models and face recognition models to FER. Because those networks have been trained to learn the basic spatial features embedded in almost every visual data such as curve, line, edges or even high-level features such as a nose, an eye, a face, etc. [33] also pointed out that transfer learning is the mainstream in deep FER to solve the problem of limited data and bad generalization. [21] and [44] demonstrated some examples of transfer learning for FER: finetuing DCN, GoogLeNet, CaffeNet, VGG16, VGGFace with the database FER2013 of 35889 images. In fact, fine-tuning network that is pre-trained with face data (VGGFace) happens to achieve the best result.

Similarly, there are also alternatives for learning temporal features. As mentioned in the previous section, there are a number of new recurrent network architectures that can capture long-term dependencies even better than LSTM e.g. BiLSTM and ConvLSTM. However, [44] compared unidirectional and bidirectional LSTM for FER and observed that BiLSTM was prone to overfitting on their training set and therefore does not perform well on their validation set even with increasing number of layers.

Network Ensemble is about fusing multiple separate trained networks together. Those networks can be trained on the same or different data sources. [21] showed a fusion of 3 models (CNN-RNN and C3D trained with image sequences and SVM trained with audio data) that gave impressive results. Additionally, there are many works that investigate

different fusion strategies including score average fusion, SVM-based fusion, and neutral-network-based fusion [33].

2.3 Research resume

In the previous sections, the author has 1. walked through the general concepts of DL including a short description of some well-known network architectures and training techniques and 2. given an introduction and an overview of the state of the art of FER.

In conclusion, Deep Learning is undoubtedly the way for FER. A number of works have demonstrated the effectiveness of different DL models for FER and achieved unsurprisingly the state of the art. Most works focus on evaluating their proposed algorithms on classic databases such as CK+, AFEW, JAFFE, FER2013, etc.

The author deeply believed in the spatio-temporal characteristic of FER and the high potential of such networks which can capture both those features. However, there are just a few works on dynamic FER which experiment on modern datasets such as MMI or BU4FE. Additionally, there are even fewer works that investigate in detail the training procedure and effectiveness of training techniques. This thesis should fill the gap.

2.3.1 Objective

The objective of this work is to reach the following goals through practical experiment on modern datasets:

- To find out the performance of different types of NNs on the task FER. Does a particular network perform especially well on FER? Or does a combination of different networks (a.k.a fusion) have any advantages overall? Can transferred knowledge from pre-trained networks enhance FER? (a.k.a transfer learning techniques)
- To compare different training techniques and evaluate their effectiveness through visualization and diagnosing the network performance during the learning procedure
- To re-evaluate the effectiveness of DL on FER. Is the best model sufficient for FER in real-life applications?

- Giving detailed Error Analysis to find out which emotion happens to be recognized easily and which laboriously. Are there any noticeable relations among those emotions and are there any failure patterns e.g. emotion A is often predicted as emotion B.

In addition, there will also be a brief research on the real-time FER including tools to optimize NNs for edge devices such as OpenVino, as well as the development of hardware for real-time inference.

2.3.2 Scope

The experiment will be executed on the BU4FE dataset. The access to MMI dataset was unfortunately not possible in the timeline of this work. Categorical approach is chosen, with 6 basic emotions {happiness, sadness, surprise, fear, anger, disgust}. Recognition of other types of emotion e.g. neutral, anxiety, depression, etc. is out of the scope of this work.

3 Experiment

This chapter is about designing and conducting an experiment that serves the objectives mentioned above. It provides a detailed description of the dataset, the preprocessing step as well as the training procedure and the evaluation of the results.

3.1 Design

This experiment is designed to classify 6 basic emotions {happiness, sadness, surprise, fear, anger, disgust} from the dataset BU4FE from University Binghamton. Below are some of its characteristics based on the categories listed in subsection Datasets in the previous chapter.

- Environment: laboratory-controlled with setup and instruction from professional psychologist
- Emotion approach: categorical
- Ground truth: emotion
- Size
  - Number of objects: 101 (Unfortunately, there is a missing video of emotion "happiness" for object F016. For this experiment, a request of missing data is not done. Instead, all videos from object F016 is simply left out.)
  - Number of emotions: 6
  - Resolution height x width: 480 x 640
  - Frame rate: 25 frames per second

- Type: 3D video sequence with the resolution of approximately 35,000 vertices; Model of face: 3D (However, the experiment simply uses 2D frontal view.)
- Color: RGB
- Intensity: neutral-peak-neutral a.k.a onset-apex-offset
- Capture: posed
- Bias: objects of both genders with a variety of ethnic/racial ancestries, including Asian, Black, Hispanic/Latino, and White

In the author's previous work [12], it is pointed out that neutral faces at the beginning and end of each video could mess up the training. The author is aware that there are also works that try to address this issue by providing on-the-fly prediction based on partial expression sequences [7]. Nonetheless, for the purpose of this work, the author will focus on recognizing the emotion by cutting off the neural emotion in the video. As human emotion is expressed continuously from neutral to peak emotion and back, it is necessary to determine when the actual emotion takes place. Building such a detector of neutral vs non-neutral emotion is unfortunately out of the scope of this thesis. In addition, to increase the data volume, the author proposes several augmentation methods. The detailed process of preparing data is covered in the next subsection.

As the dataset consists of dynamic sequences, the author proposes networks that are capable of encoding both spatial as well as temporal features. Since each network has its own learning capacity, Cascade Network is known as one of the best candidates while dealing with mixed features.

Network	Special characteristic	Pre-trained with
VGG16	consistent convolution blocks	ImageNet
ResNet50	residual blocks	ImageNet
DenseNet201	dense blocks	ImageNet
VGGFace	pre-trained with faces	Faces

Table 3.1: Pre-trained networks

To encode features in space or, in other words, pattern, structure, information among pixels, the author will use Convolutional Neural Network. Instead of training some CNN blocks from scratch, the author will take advantage of pre-trained networks and fine-tune them so that their learned knowledge can be transferred to this task. The pre-trained

networks are chosen because of their popularity, performance, and their characteristics of innovative architecture (see table 3.1).

The chosen networks will be fine-tuned on another dataset (FER2013) instead of BU4FE to avoid bias.

To encode features in time a.k.a temporal relationship among frames, how the face is moving, the author proposes the following networks:

- LSTM (bidirectional and unidirectional)
- 1D-CNN

Alternatives for Cascade Network are, for example, C3D and ConvLSTM. Although the origin network of CNN and LSTM is popular for extracting either features in space or features in time, these 2 networks are altered to capture the other type of feature as well. While C3D tries to remember spatial features through its convolution operation with 3D kernel, ConvLSTM attempts to perform the convolution of CNN as part of LSTM. Therefore, they are just as suitable for spatio-temporal data as Cascade Network.

Finally, there might be different networks that work fine. They are then fused together to see whether Network Ensemble results in better performance.

The whole experiment is summarized once again in figure 3.1.

The experiment was done in the renderfarms provided by CSTI [1]. Below are some of their hardware information:

- OS: CentOS Linux release 7.7.1908 (Core)
- GPU: 4 NVIDIA Quadro P6000 each with $\approx$ 24 GiB
- CPU: Intel Core Processor (Broadwell, no TSX) 2.3 GHz 18 cores 18 threads
- RAM: $\approx$ 163 GiB

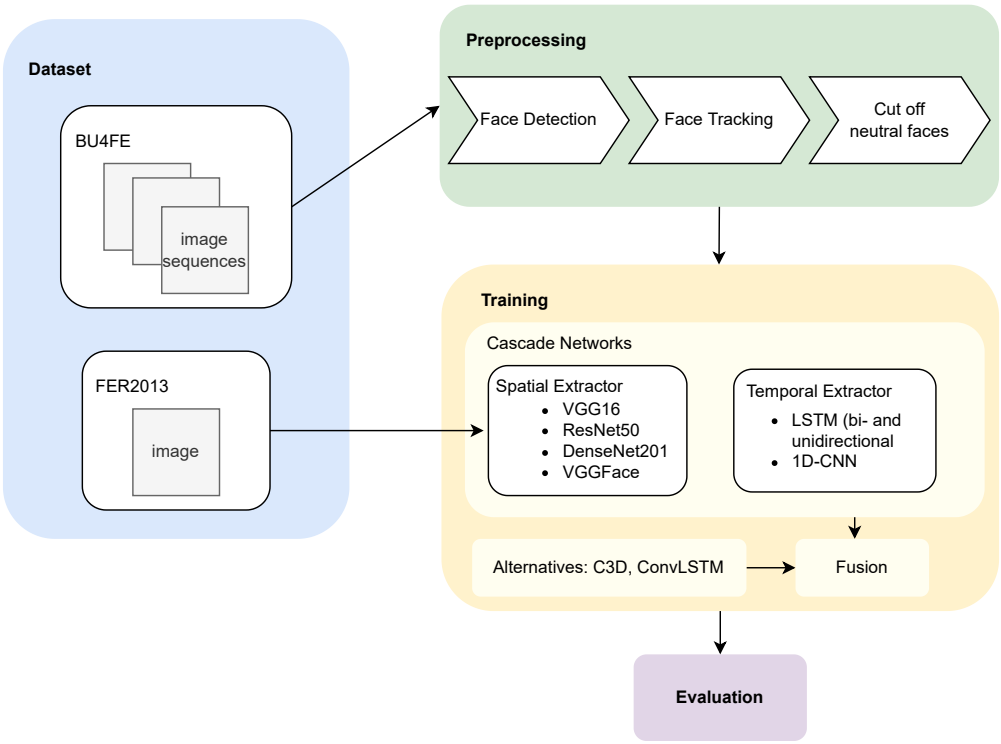


Figure 3.1: Experiment summary

3.2 Implementation details

3.2.1 Preprocessing

Certain steps in the preprocessing (Face Detection + Face Tracking) were already done in the author's previous work. Below is the summary of the result. For more details, follow [12].

Haar Cascade face detector was chosen over other deep face detectors e.g. MTCNN thanks to its efficiency at frontal faces. The algorithm implementation as well as pre-trained weights are provided by the open-source library OpenCV. After finding the face in the first frames, trackers were applied to track through the video based on their advantages over frame-to-frame detection as discussed in the previous chapter. The detected bounding box is extended 10 pixels in the vertical direction before the tracking process to ensure that big movement of mouth (for instance while expressing emotion "surprise") will not be missed. As the face tends to move very minimally, the author decided to go

(a) Detected face with dimension 159x159 (width x height), extended by 10 pixel in the height dimension for tracking

(b) Resized face

Figure 3.2: Detected, tracked, and resized face

with simple trackers provided by OpenCV instead of deep trackers. Different trackers including CSRT, KCF, MedianFlow were compared after some tests on a part of the dataset (24 videos). The results are as expected: All trackers performed well enough, no failed example was recorded. MedianFlow was chosen to proceed with the rest of the dataset due to its fastest inference time. Faces were resized to 160x160 and saved with lossless compression for further process. Figure 3.2 visualizes an example of object 42 displaying emotion "happy".

This work continues with the final step of preparing data for training: cutting off neutral faces at the beginning and end of videos. Based on the fact that neutral faces in the dataset hardly display a move, the author proposed a simple method based on Action Unit score to separate them from the rest of the frames. OpenFace [5], an open-source tool indicated for facial behavior analysis, was used to generate Action Unit (AU) score for each frame. OpenFace is able to recognize a subset of AUs, specifically: 1, 2, 4, 5, 6, 7, 9, 10, 12, 14, 15, 17, 20, 23, 25, 26, 28, and 45. AU 1 displays, for example, when the inner brow is raising. A full description with visual examples of all AUs can be further seen here[1]. OpenFace provides 2 scores for each AU: one indicates the presence of AU (e.g. AU01_c) and the other the intensity (e.g. AU01_r). Presence is encoded as 0 for

[1]https://www.cs.cmu.edu/ face/facs.htm

absent and 1 for present. Intensity can range from 0 to 5 with 5 being the maximum intensity. For this work, only the intensity score was used. For each frame, the average intensity scores of all AUs were calculated. AU 45 (blink) tends under observation to add a little noise to the result and was, therefore, left out.

Each video contains 3 phases: onset, peak, and offset where the face displays a neutral state, moves to a peak emotion and then goes back to neutral state. The AU score calculated is in fact synchronized with the movement. Concretely, at onset and offset states, AU scores are quite small while they are at peak especially high. The score distance among phases is different for each particular emotion but the trend is indeed clear to see. Below is an example of AU scores of video M042_happy_tex.avi

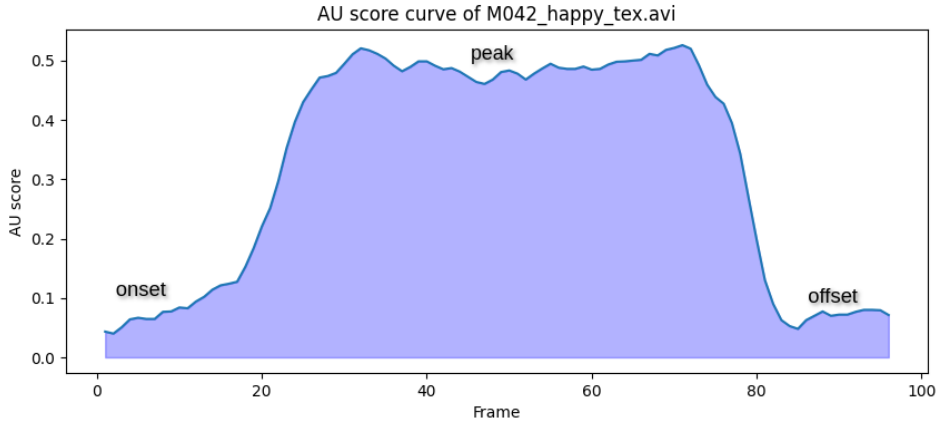


Figure 3.3: AU score curve

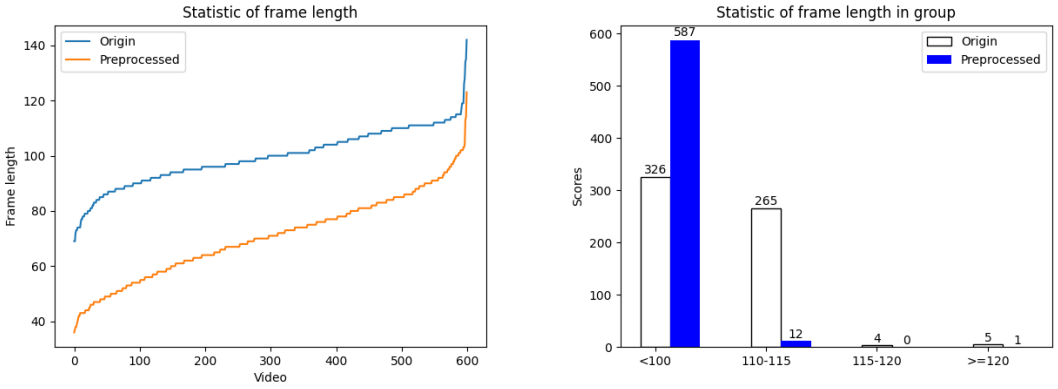
With the average AU scores available, the neutral faces were cut off after the following algorithm (see algo 1). Basically, all frames from the beginning or end of the video which have an AU score less than half of the maximum AU score were cut off. The number of cut frames should not exceed 50% of the total number of frames. Under careful observation, the offset phase seems to be quite long and therefore 10 last frames will always be cut off. Another noticeable point is that many objects end the offset phase with a smile, blink, or slightly move of the head which causes the AU scores to be very high despite the neutral emotion. Therefore, in case the maximum AU scores lie in the last 20 frames, the sample will be marked as "anomaly" and will not be processed with the proposed algorithm but will be manually reworked instead.

Algorithm 1 Cutting off neutral faces algorithm

```
min_frames_end ← 10                          ▷ minimum number of frames to cut
                                               from the end of the video
min_intensity_allowed ← 50% of maximum AU score
max_frames ← video length / 2                ▷ maximum number of frames allowed to cut
i ← 0
while True do
    frame_begin ← frame from the beginning of the video by i
    if AU score(frame_begin) < min_intensity_allowed then
        Cut off frame_begin
    else
        Stop cutting from the beginning
    end if
    frame_end ← frame from the end of the video by i
    if AU score(frame_end) < min_intensity_allowed OR i < min_frames_end then
        Cut off frame_end
    else
        Stop cutting from end
    end if
    if stop cutting from both beginning and end
       OR number of cut frames ≥ max_frames  then
        Exit program
    end if
    i ← i + 1
end while
```

The author is aware and confident that there are more precise ways to do so. Yet building such a neutral face detector or a complex algorithm to detect the rise and fall of AU score curve can be time-consuming and will not be done within this work.

In summary, the data were preprocessed as follows: Faces were detected, tracked, resized, and stored; Neutral faces from the onset and offset phases were cut off. Figure 3.5 visualizes an example of the final result. Some statistics from the original dataset and the preprocessed dataset are drawn in 3.4. It appears obvious that after preprocessing most videos have a length of fewer than 100 frames.



(a) Frame length sorted

(b) Frame length in group

Figure 3.4: Dataset statistic

(a) Detected, tracked, and resized faces

(b) Cut off neutral faces

Figure 3.5: Preprocessed final results

3.2.2 Dataset

Cross Validation

As briefly introduced above, the final dataset contains 600 videos from 100 objects. It is indeed a small dataset for the FER task. A traditional split of a fixed train, validation, and test set raises a lot of bias since the quality of trained models relies hardly on the chosen validation set. Therefore, Cross Validation is adopted. The procedure divides the train data into k non-overlapping folds with k-1 folds for training and 1 validation fold. K models are trained and evaluated separately and the final performance is the average performance reported in each fold. By using a different validation set each iteration, the performance of trained models can be estimated in a more precise and reliable way. As a result, it is more likely to find the most robust algorithm. The configuration of the Cross Validation is considered carefully under the given data and infrastructure. The following parameters were applied:

- The optimal number of folds recommended in many papers are 3, 5, and 10. These numbers are determined through a number of practical studies. The main argument is based on the trade-off between the bias of the estimator and the computation cost. A higher k is claimed to have low bias but comes with a high cost in computation. On the opposite, a very low k tends to bring a lot of noise to the estimation despite its advantage in computation. A special case, known as Leave One Out Cross Validation (LOOCV), describes an extreme version of Cross Validation, where each sample has the opportunity to represent the entire validation set. The benefit of LOOCV is its robustness in estimating. Yet its most disadvantage is the maximum computation cost. If an accurate estimate of model performance is critical and the dataset is small, LOOCV might be a good candidate to go with. In this experiment, a good estimation has indeed the highest priority. The data volume is small in terms of number of samples. However, given its characteristic of image sequences with 3 color channels, it might require a lot of memory and computation time. In addition, the validation set has to contain enough data to represent the model performance. Hence, k=5 is chosen as the best balance between the given requirement and condition.
- The purpose of this training is to find models which can give good predictions on completely new people. As that, it should be simulated in the process as well. Each fold, therefore, contains videos from different objects. For each object, there are 6

videos corresponding 6 emotions. Since the class distribution is balanced, each fold consists of all videos from a number of objects.

Figure 3.6 visualizes how the dataset is split up for Cross Validation. A separate test set is left out for final evaluation.

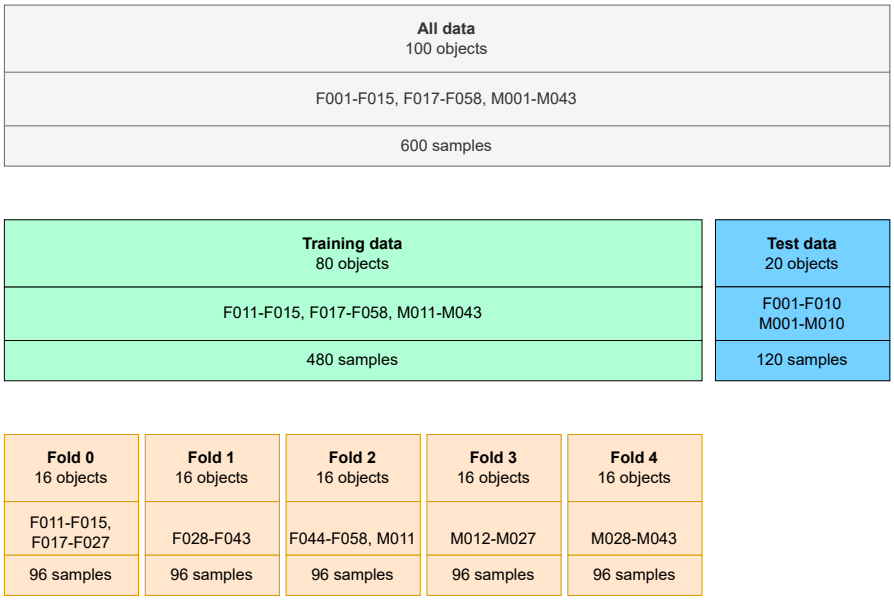


Figure 3.6: Cross Validation summary

Data Augmentation

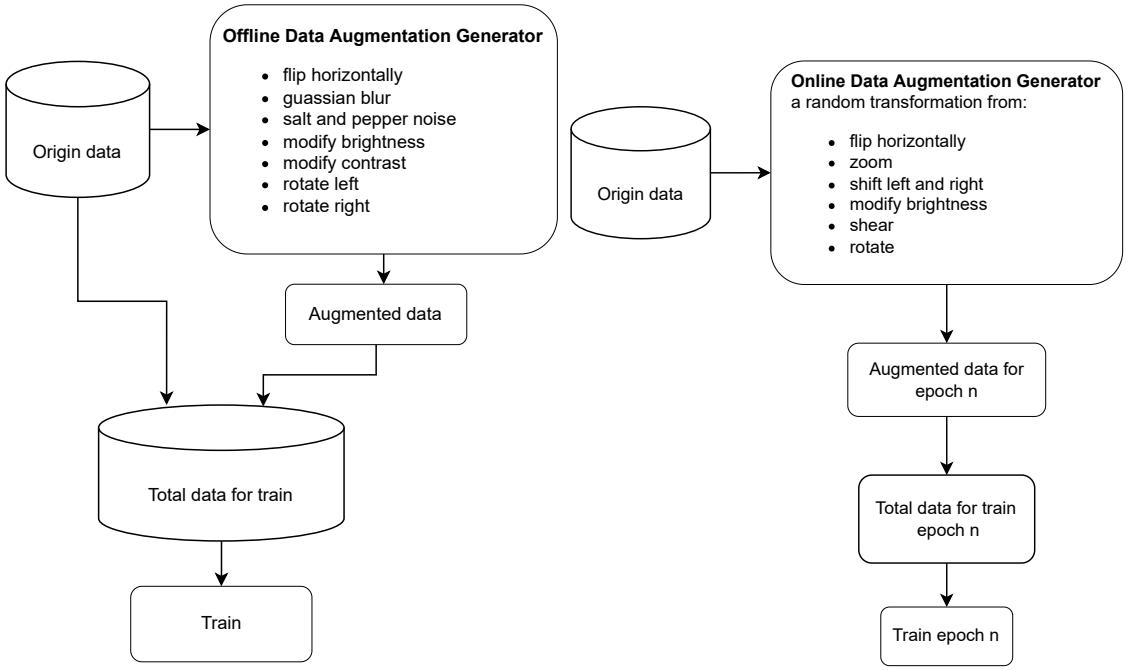
Be aware of the small dataset, Data Augmentation was applied to increase the data volume. The effectiveness of Data Augmentation is well-known. By letting the model see more data, the network is able to learn more robust features, hence a better generalization. There are 2 main types of Data Augmentation known as online and offline Data Augmentation. Offline means expanding the existing dataset through a number of transformation methods. This type is yet simple but less commonly used. Ultimately, the model is trained with more data but it is still seeing the very same data every epoch. Especially for large datasets, increasing the data volume requires more memory and might not always be a possible option. The second and more common type is online Data Augmentation. This is also the method that is implemented by Keras ImageDataGenerator class. Online or sometimes also called in-place, on-the-fly means generating for each epoch new data based on a set of transformation methods. This ensures that the model

will see new variations of the data each and every epoch. As a consequence, it is more effective and helps achieve better generalization. Another important point when it comes to Data Augmentation is the decision of transformation methods. "Invented" data from Data Augmentation has to reserve the main characteristics of the original data. Shifting the face too far to the left or right might be a poor choice as part of the face, part of the features set might be cut off. In addition, the same transformation should be applied to all frames in a sequence to avoid confusing the trained model.

For this experiment, both methods were indeed explored. Online Data Augmentation is the first obvious choice as the generalization capacity is the most important target. However, while training Cascade Network with a pre-trained spatial feature extractor, extracting spatial features every epoch takes unfortunately a lot of time, which slows down the training process significantly. A rough estimation as demonstrated below shows a clear concern. It might take up to over 4 months to execute only 20 runs with the given hardware (see section Design). For this reason, the author decided to also exploit offline Data Augmentation. By doing so, there is a good opportunity to evaluate the effectiveness of these 2 methods as well.

feature extracting time / epoch	1 hour
number of epochs	30
number of folds	5
1 experiment	1*30*5=150 hours ≈ 6.25 days
number of experiments	20
total training time	125 days ≈ 4.2 months

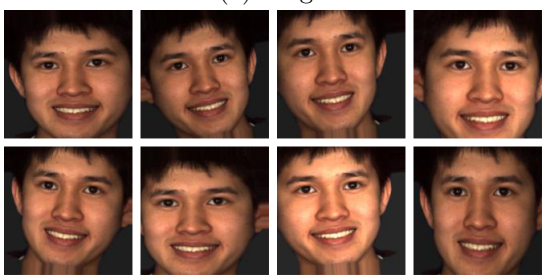
Figure 3.7 summarizes the process of Data Augmentation and the chosen transformation methods. Some examples are drawn in figure 3.8. All the transformation methods are carefully chosen so that the main area of face is reserved and the facial movement is clear to identify. As visualized below, for offline Data Augmentation each transformation method was applied separately while online Data Augmentation chose a random mixture from all given transformation methods.



(a) Offline (b) Online

Figure 3.7: Data Augmentation summary

(a) Origin



(b) Online augmented (randomly transformed)

(c) Offline augmented (left to right: flip horizontally, gaussian blur, salt and pepper, modify brightness, modify contrast, rotate left, rotate right)

Figure 3.8: Online and offline Data Augmentation examples

3.2.3 Training procedure

The preprocessed, as well as augmented data, were then trained after the standard Cross Validation procedure. For each run, a fold was chosen as validation set. If Data Augmentation was used at any point, it was only used to the train data and not to the chosen validation set. The final result was averaged epoch-wise from all folds. Once again, the following networks were chosen for training. The detailed training process is presented in the next sections.

- Cascade Networks
- Alternatvies: C3D and ConvLSTM

Strategy

Training NN is an imperative process. Finding the best model is, in other words, finding the optimal hyperparameter set. The process is therefore often referred as "hyperparameter tuning". In this work, tuning hyperparameter is relied on the practical recommendation from Prof. Dr. Andrew Ng in his courses at [2]Coursera and university Stanford. In summary, if the model tends to underfit, it is recommended to use a bigger and more complex network or train longer. On the contrary, if it is overfitted, then regularization such as Dropout, L1, L2 or Data Augmentation or Early Stopping should be applied. Alternatives could be using a different network architecture.

All models were built with Keras Functional API to have more advanced control over the process than the simple commonly used Sequence API. According to the data statistic after preprocessing, almost 98% videos (587 out of 600) have a length of less or equal to 100. Therefore, the dimension of input is set to 100x160x160x3. Longer videos were truncated and shorter videos were zero-padded, which were later masked out during training if possible. Output layer used softmax as activation function and contained 6 neurons corresponding to 6 emotions which indicates that the output will show the probability of each class. The class with the highest confidence is taken as the predicted label. If multiple classes have the same confidence, the first-mentioned one is simply taken. Accuracy was chosen as the main metric during training. All models used Adam as optimizer and a small learning rate of 0.0001. A bigger learning rate prevented the model from learning properly. Data was shuffled during training. Random seed was set to

[2]https://de.coursera.org/

have minimum consistency among experiments. The following callbacks were configured to monitor the train performance.

- Tensorboard for tracking and visualizing model accuracy, loss as well as graph
- ModelCheckPoint for saving model and weights at best performance. The saved weights/model can be loaded for further training.
- CSVLogger for streaming the results to a CSV file. The results from all folds were then collected, averaged, and visualized.

As the training was proceeded with Cross Validation, it is not recommended to early stop the training since the purpose is to evaluate the algorithm across all folds and not to find the best model, particularly for each fold. A better approach would be to treat the number of epochs as a hyperparameter and tune it along with others. With a fixed number of epochs for each run, it is very simple to average the results across folds which helps give a more reliable estimation of the model performance.

Finalizing model

Once the training is done or, in other words, the best set of hyperparameters is found, it is time to choose the final model. In fact, there are multiple ways of doing so. One is to take the best models from each fold and evaluate them on the test set, the average score is then the final performance of the experimented algorithm. Another more commonly used procedure is to train a final model with all available data for training. Without sacrificing an amount of data for validation, it is more likely that the final model performs even better. The process is proceeding as follows: All folds are executed lastly with Early Stopping. The number at which training was stopped is recorded and an average is calculated. This number of epoch is then used to train the final model.

The final model will be evaluated on the test set. The evaluation metrics and further details will be covered in the later section. As the test set is fixed, there is a degree of bias that is unavoidable. A possible solution for it is so-called nested Cross Validation in which the same procedure is done with the test set (see algorithm 2).

Algorithm 2 Nested Cross Validation

```
Split all data in K1 folds
for each k1 ∈ [1...K1] do                                   ▷ go through outer folds
    test ← data of fold k1
    train ← data of all other folds
    Split train in K2 folds
    for each k2 ∈ [1...K2] do                               ▷ go through inner folds
        validation ← data of fold k2
        train_in ← data of all other folds
        m[k2] ← model trained on train_in and evaluated on validation
    end for
    M ← final model for fold k1 based on m[1],...m[K2]
    score[k1] ← evaluated of M on test
end for
final_score ← average of score[1], score[2]...score[K1]
```

However, due to the limitation of time and computation, the author decided not to pursue this approach.

In the following, the detailed training process for each chosen network will be unfolded.

3.2.4 Cascade Network

Cascade Network is the first candidate for FER. By cascading multiple types of networks, it is able to learn multiple types of features especially well. For FER, it is crucial to recognize both spatial and temporal patterns. The training for Cascade Network is split into 2 phases as described in figure 3.1:

1. Train spatial feature extractor
2. Train the Cascade Network of spatial feature extractor in phase 1 and temporal feature extractor

1. Spatial feature extractor

In order to extract visual features from frames, the author proposed using transfer learning instead of training from scratch. Transfer learning is widely adopted and proved to be effective, especially while training with limited data. The author chose 4 pre-trained models as basis and fine-tune them with new data. The pre-trained models are chosen based on their popularity, performance, innovative architecture, and especially the similarity in purpose to FER as summarized in 3.1. VGG16, ResNet50, and DenseNet201 are very well-known participators of the ImageNet challenge with remarkable results which indicates their capacity of extracting visual information.

Network	Top-1 Accuracy	Top-5 Accuracy	Parameters
VGG16	0.713	0.901	138,357,544
ResNet50	0.749	0.921	25,636,712
DenseNet201	0.773	0.936	20,242,984

Table 3.2: Pre-trained networks performance. The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

VGGFace, on the other hand, is originally developed for face recognition tasks such as face identification and verification. It is, therefore, trained and evaluated with/on large benchmark face recognition datasets, demonstrating that the model is effective at generating basic features from faces [36]. The data for fine-tuning was chosen to be related to FER, yet not the BU4FE to avoid bias in the later training phase. FER2013 was used. FER2013 is a publicly available dataset. It consists of 35889 images: 28709 for training,

Emotion	Train	Public test	Private test
angry	3995	467	491
sad	4830	653	594
disgust	436	56	55
surprise	3171	415	416
fear	4097	496	528
happy	7215	895	879
neutral	4965	607	626

Table 3.3: FER2013

3589 for public test, and 3589 for private test. The image is in grayscale, has a dimension of 48x48 and displays one of the following 7 emotions {angry, sad, disgust, surprise, fear, happy, neutral}. One noticeable point is that the data distribution of FER2013 is in fact

imbalanced. Table 3.3 shows that "happy" has the highest number of samples, about 25% the train set while "disgust" accounts for only 1.5%. Data Augmentation was used to increase the volume of data. Data was loaded and augmented during training a.k.a online augmentation with the Keras ImageDataGenerator class. It is to notice that while augmentation, especially offline augmentation refers to a combination of original and augmented samples, ImageDataGenerator works slightly differently. ImageDataGenerator will generate a different batch each time, replace the original with the new batch and feed it to the training. Given that the model is constantly seeing new data, it is able to learn more robust features and as a result, enhances the generalization ability.

```
train_aug = ImageDataGenerator(
            rotation_range=10, # rotation
            zoom_range=0.1, # zoom
            width_shift_range=0.1, # horizontal shift
            height_shift_range=0.1, # vertical shift
            shear_range=0.15, # shear intensity
            horizontal_flip=True, # horizontal flip
            brightness_range=[0.4, 1.5], # brightness
            fill_mode='nearest', # fill with same neighbour pixel
            preprocessing_function=func_preprocessing)
```

The degree of changes were kept minimal so that the face and the emotion are still well presented. Vertically flipping the face is definitely not a meaningful thing to do. Concretely, the following transformations are applied to the data at each training epoch. Figure 3.9 visualizes some examples of Data Augmentation.

- Rotate randomly between $-10°$ and $10°$
- Zoom randomly between 0.9 and 1.1. Any value smaller than 1 will zoom on the image and larger will zoom out.
- Shift randomly in the horizontal direction between 10% the image width to the left and 10% the image width to the right
- Shift randomly in the vertical direction between 10% the image height up and 10% the image height down
- Apply Shear transformation [3] with a *shear_range=0.15* which represents a Shear angle in a counter-clockwise direction in degrees. Shearing is sort of stretching the

[3] https://en.wikipedia.org/wiki/Shear_mapping

image at a certain angle with one axis fixed. In a simple term, it generates the object from different points of views.

- Flip horizontally
- Change the brightness randomly between 0.4 and 1.5. Any value smaller than 1.0 darkens the image, whereas values above 1.0 brighten the image.
- If the image is transformed in a way that some pixels are moved outside the image and leaves some empty area, then the *fill_ mode='nearest'* indicates that this area will be simply replaced with the nearest pixel values.

(a) Origin

(b) Augmented

Figure 3.9: Data Augmentation examples

The networks and pre-trained weights were loaded using Keras Application [4]. VGGFace was loaded using the python package keras-vggface [5].

The training process proceeds as follows: Firstly, the pre-trained network and its weights are loaded without its original classifier. A new classifier for the new task is then added on top of the loaded base. For the network to learn properly, it is important not to

[4]https://keras.io/api/applications/
[5]https://github.com/rcmalli/keras-vggface

plug randomly initialized fully connected layers on top of the pre-trained base because the large gradient updates triggered by the random weights could wreck the learned weights in the base. Therefore, all layers from the base will be firstly frozen and only the classifier will be trained. Next, a couple of last layers or last convolution blocks will be unfrozen and jointly trained with the newly-added classifier. There is no need and also not recommended to unfreeze all layers from the base. As it is believed that the early layers are capable of extracting basic features and as a consequence don't need to be re-trained while the last layers are able to encode high-level features for the original task and have to be adjusted to the new task. Another reason is that the more layers we re-train, the more capacity the network has, the bigger the risk of Overfitting is. Finally, the network can be further refined if needed. For example, in case of a large Overfitting gap, it might be a good solution to re-train it with more regularization. Another essential point is to keep the learning rate small and adaptive in order to make sure that the updates are stable and not affect negatively the previously learned ability.

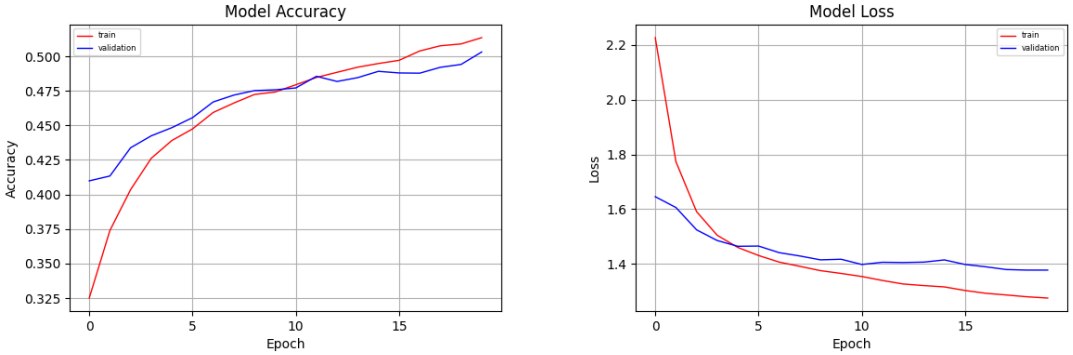
Some basic setups for the training are listed below.

- All models were built with Keras Functional API.
- The hyperparameter tuning was done with KerasTuner [6], a library with strong integration with Keras workflows that offers tuning for model architecture, training process as well as data preprocessing. Bayesian Optimization was used to search for the best set of hyperparameters because of its efficiency and low resource consumption in comparison to Grid Search. The search matrix covers for instance the number of fully connected layers for the classifier, the number of neurons in each layer, the degree of regularization, the optimizer, the number of convolution layers to unfreeze, along with others.
- For training, accuracy of the validation set was chosen as the optimizing metric. The training process was monitored by Tensorboard, a visualization extension created by Tensorflow team that provides result graphs such as accuracy and loss, network insights such as architecture, hyperparameters information such as their distribution, histogram or trends and many other functionalities. Early Stopping was used to stop the iteration in case the validation accuracy increases continuously - a sign of inefficient learning. Learning rate was slowly reduced if there was no improvement after a defined number of epochs.

[6] https://keras.io/guides/keras_tuner/getting_started/

- For testing, multiple metrics were used to have better insights including Accuracy, Precision, Recall, F1, Confusion Matrix, and an Analysis.

An example of the process of fine-tuning VGG16 is given in detail in the following. A similar process was done for ResNet50, DenseNet201, and VGGFace. As mentioned above, the training consists of 3 phases: training classifier, fine-tuning and refining. In phase 1, the following search was initialized for the classifier: 1-2 fully connected layers, each with either 256, 512, or 1024 neurons, follows by a Dropout layer with a drop rate in the range from 0.2 to 0.9. The model was built with all layers from the base frozen, SGD as optimizer, categorical_crossentropy as loss function, and accuracy as metric. 20 trials were executed and Bayesian Optimization algorithm was used to find the best set of parameters. The top 3 best classifiers recorded very promising results with no sign of Overfitting. Figure 3.10 visualizes the performance of the best classifier.



(a) Accuracy (b) Loss

Figure 3.10: Best classifier loss and accuracy

The best classifier was chosen to proceed with phase 2, the fine-tuning. Experiments of unfreezing 1-2 last convolution blocks showed that the network with 2 last blocks unfrozen gained a slightly better result (see figure 3.11). Nonetheless, the training was early stopped where the accuracy of train set reached only 80%. Figure 3.11 visualizes the result of fine-tuning. A degree of Overfitting is to acknowledge. In phase 3, the refinement, some attempts against Overfitting such as an increase of drop rate reported unfortunately no clear improvement.

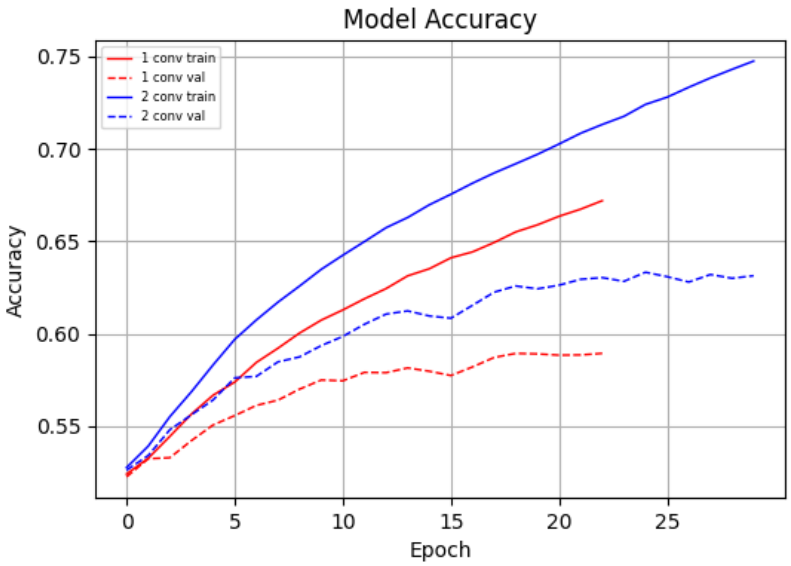


Figure 3.11: Result of unfreezing 1-2 last convolution blocks

The overall result of fine-tuning chosen pre-trained networks is displayed in table 3.4

Network	Unfrozen convolution blocks	Test Accuracy	#params	Note
VGG16	2	65%	38,677,511	-
ResNet50	2	63%	24,189,959	strongly overfitted
VGGFace	2	**69%**	39,727,111	-

Table 3.4: Fine-tuning result

The fact that VGGFace achieved the best result was as expected since it was pre-trained with faces. However, it was surprising that ResNet50 and DenseNet201 scored lower than VGG16 even though they record better results on the ImageNet dataset. These 2 networks showed an extremely high Overfitting gap, especially DenseNet201. Many attempts (different architecture, different amount of regularization, etc.) have been made, yet no reasonable result was recorded with DenseNet201. A detailed investigation in this matter could unfortunately not be done in the timeline of this work. Thus, it raises the attention when it comes to choosing a pre-trained network for fine-tuning: Certain architecture might suit better certain domains. It could be difficult for certain networks to transfer their knowledge and adapt to new tasks, hence difficult to be fine-tuned.

Table 3.5 shows the detailed performance of fine-tuned VGGFace for each emotion. A Confusion Matrix and a normalized Confusion Matrix are also presented in 3.12

	precision	recall	f1-score	#samples
angry	0.63	0.61	0.62	491
disgust	0.68	0.69	0.68	55
fear	0.53	0.51	0.52	528
happy	0.83	0.91	0.87	879
neutral	0.69	0.66	0.67	626
sad	0.54	0.51	0.53	594
surprise	0.79	0.81	0.80	416

Table 3.5: Classification result

(a) Origin (b) Normalized

Figure 3.12: Confusion Matrix

It is clear to see that "happy" achieved the best performance with an accuracy of 91%. The values of Precision, Recall, and F1 also align with this statement. The result raised no surprise as "happy" tend to have a very clear facial expression. Also, the high number of train samples of "happy" could play a role in its success. Emotion "sad" and "fear" have high proportions as well, yet their results are very poor, slightly over 50%. It is important to keep this validation in mind while training this spatial extractor with the temporal extractor to see whether this bias has any noticeable effect on the final result.

2. Temporal feature extractor

The best spatial extractor architecture is visualized in figure 3.13. The network contains 5 CNN blocks. The result from the convolution operations is then flattened. The classifier contains 2 fully connected layers with a degree of neurons dropped and a softmax layer of 7 neurons corresponding to 7 emotions.

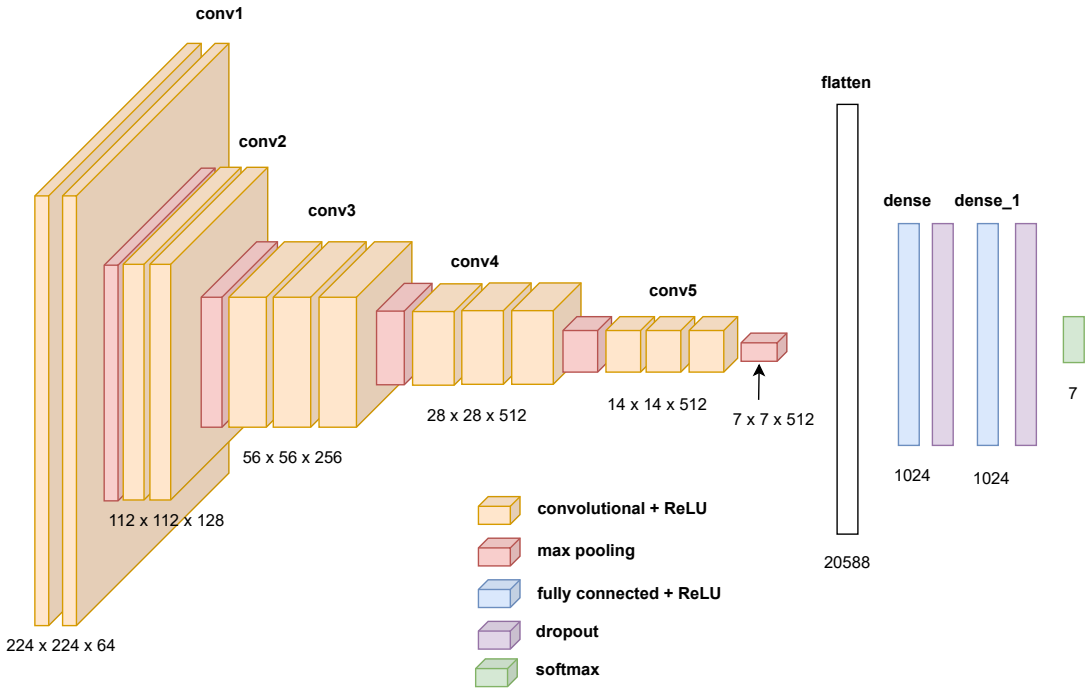
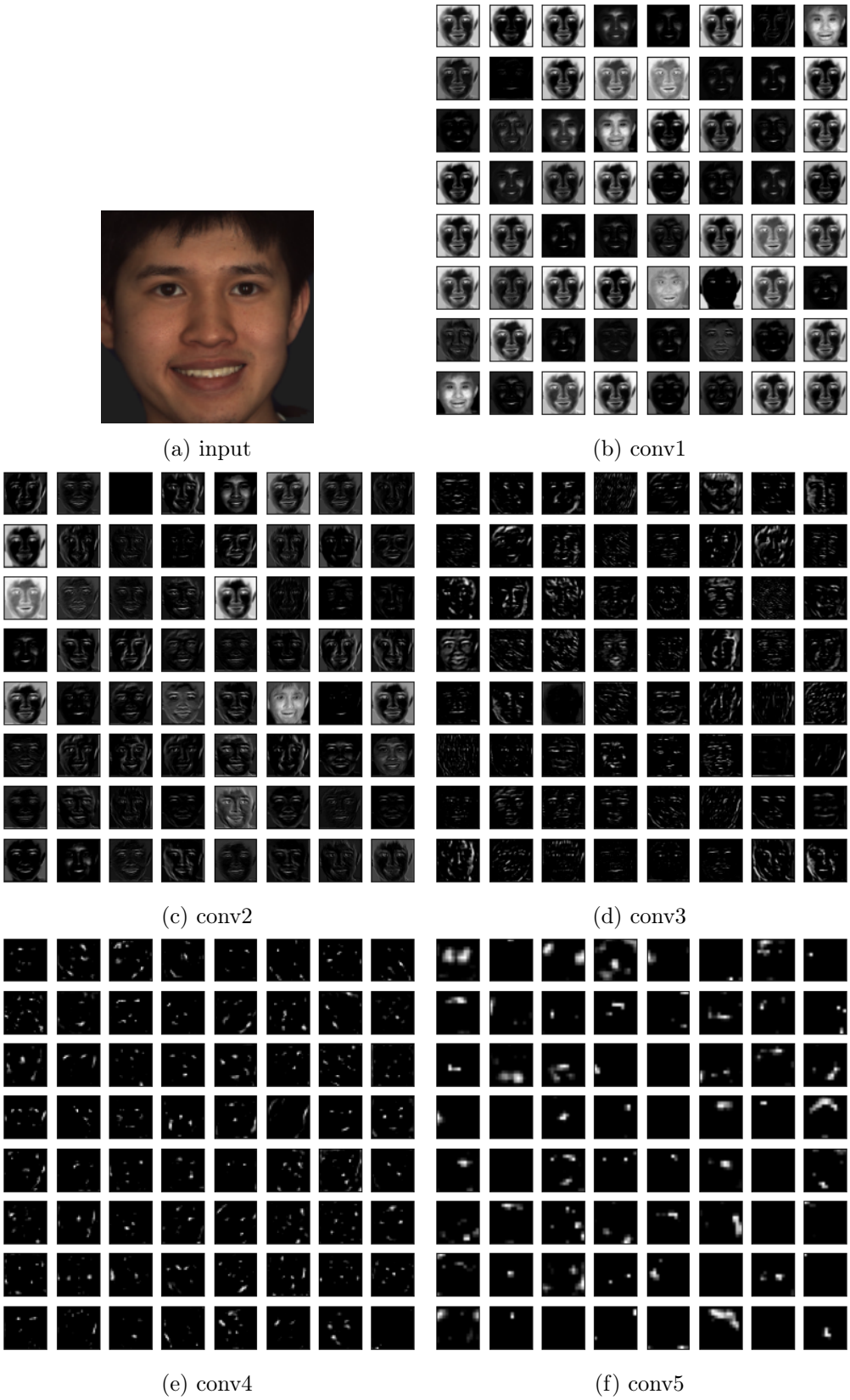


Figure 3.13: Spatial extractor architecture

The features can basically be extracted from any layer. However, it is believed that the layers close to input can only detect simple patterns whereas the ones close to output are likely to be able to extract complex patterns. The evidence for that could be observed in the feature maps. A feature map or also called an activation map is the result of the convolution operation or, in other words, the result from applying filters to the input. Figure 3.14 visualizes the feature maps from each convolution block which could provide more insights into the internal representation that the model has for the input at different points in the model. Although there are more filters in the later layers (64-128-256-512), 64 feature maps were visualized consistently for all layers.



(a) input (b) conv1

(c) conv2 (d) conv3

(e) conv4 (f) conv5

Figure 3.14: Feature maps

It is obvious to see that in the early stages the model sees a lot of fine details of the input without much interpreting. As the network goes deeper, more abstracts are identified which is clear evidence that the model is forming more and more general concepts that can be used to make a classification. Yet how the model recognizes emotion "happy" based on those abstraction, those concepts is unclear. We unfortunately lack the ability to understand those deeper and deeper feature maps.

Based on the assumption that deep layers hold more important concepts and patterns related to the prediction as demonstrated above, the author decided to take the output after all convolution operations as the feature extraction. The last max pooling layer has dimension of 7x7x512 which results in 25088 features after flattening. Furthermore, as the extractor was pre-trained with FER dataset, there is a possibility that even the dense layer in the classification part can hold certain information which can be useful for making classification. As that, the author chose to exploit features from layer "dense" (see figure 3.13) with 1024 neurons as well. Extracting features from all samples took a lot of time and was done in advance. They were then stored in Hierarchical Data Format 5 (HDF5) under the following naming structure *fold-<i>-features-<number of features>.h5*. HDF5 is a widely used library to store extremely large and complex data collection. The saved collections in each .h5 file are:

- 'features': extracted features, each with dimension of 1x1024 or 1x25088
- 'masks': numpy array with the same dimension as 'features' with value of either 1 for not masked frame or 0 for masked frame. Both 'masks' and 'features' will be used as inputs to the train model so that the model know where the real sequence ends.
- 'labels': one hot encoded label. For example, given the labels list of [angry, disgust, fear, happy, sad, surprise], their one hot representations are in order: $\begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\1\\0\\0\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\0\\0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\0\\0\\0\\1\end{bmatrix}$
- 'names': full name of each video. For instance M042_happy_tex.avi

The extracted features were then ready for further training.

The following networks were chosen to explore temporal features. They were trained on the extracted spatial features from dataset BU4FE with the configured Cross Validation procedure.

- LSTM is de facto standard when it comes to extracting features in time. Its default characteristic of holding long-term dependencies is essential for FER. Once the features in space or, in other words, information in each frame are extracted, they are then fed to the LSTM in a sequence. LSTM will then hopefully be able to figure out the change in motion across frames.
- BiLSTM or bidirectional LSTM is a variation of the traditional LSTM. By examining the sequence in both forward and backward directions, BiLSTM might be able to unfold even more information.
- 1D-CNN is shown in many recent studies to have comparative results as RNN on sequence modeling. [8] conducted an empirical evaluation and highlighted that CNN even outperforms RNN across a diverse range of tasks and datasets. They raised the attention that the common association between sequence modeling and RNN might need to be reconsidered. With that said, 1D-CNN is also a very potential candidate.

In the following, a detailed process of hyperparameter tuning for LSTM will be presented. The same procedure was applied for BiLSTM and 1D-CNN and therefore would be skipped.

A very simple baseline of only 1 LSTM layer with 128 neurons was chosen to begin with. Figure 3.15 summarizes the architecture with 1024 features as well as 25088 features as input. Some parameters were set after experimental runs:

- Inputs: features and masks of either 100x1024 or 100x25088
- Optimizer: Adam with a fixed learning rate of 0.0001 for training; and with learning rate decay of factor 0.5 if accuracy does not improve after 5 epochs for finalizing model
- Batch size: 32
- Epoch: 40

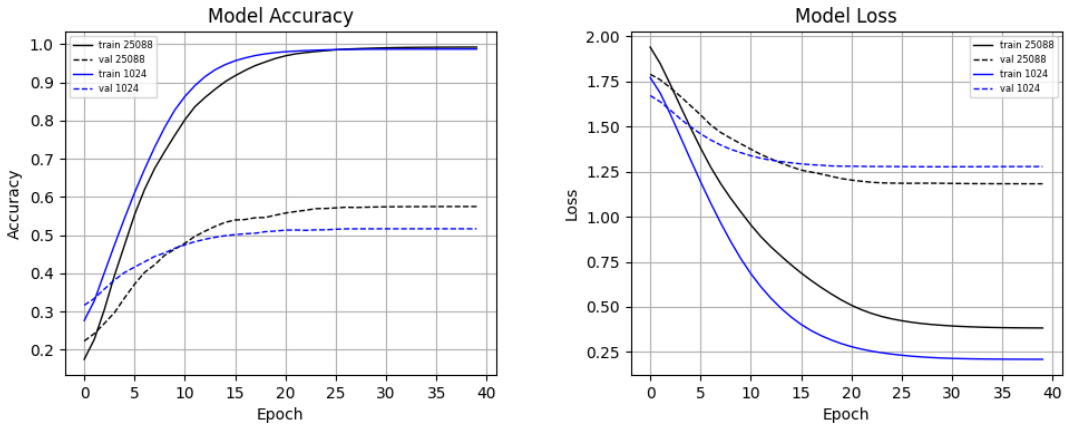
- Shuffle data during training: True
- Early Stopping: not for training but for finalizing model with patience of 10 epochs
- ModelCheckPoint: saving only best weights based on accuracy of validation set
- CSVLogger: saving each iteration result to a csv file for further analysis

```
Model: "model-features-1024"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, 100, 1024)]       0           []

 input_2 (InputLayer)        [(None, 100)]             0           []

 lstm (LSTM)                 (None, 128)               590336      ['input_1[0][0]',
                                                                    'input_2[0][0]']

 dense (Dense)               (None, 6)                 774         ['lstm[0][0]']

=================================================================================================
Total params: 591,110
Trainable params: 591,110
Non-trainable params: 0
_________________________________________________________________________________________________
Model: "model-features-25088"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, 100, 25088)       0           []
                             ]

 input_2 (InputLayer)        [(None, 100)]             0           []

 lstm (LSTM)                 (None, 128)               12911104    ['input_1[0][0]',
                                                                    'input_2[0][0]']

 dense (Dense)               (None, 6)                 774         ['lstm[0][0]']

=================================================================================================
Total params: 12,911,878
Trainable params: 12,911,878
Non-trainable params: 0
_________________________________________________________________________________________________
```

Figure 3.15: LSTM baseline

Baseline

As seen in figure 3.16, no matter how many features were used as input, the model experienced a very high Overfitting. The train accuracy increased very fast and reached 100% after about 20 epochs. The train loss decreased correspondingly as expected. Yet the validation accuracy increased until slightly over 50% for 1024 features and almost 58% for 20588 features and then stayed stable. The validation loss reduced alignedly.



(a) Accuracy (b) Loss

Figure 3.16: Baseline performance

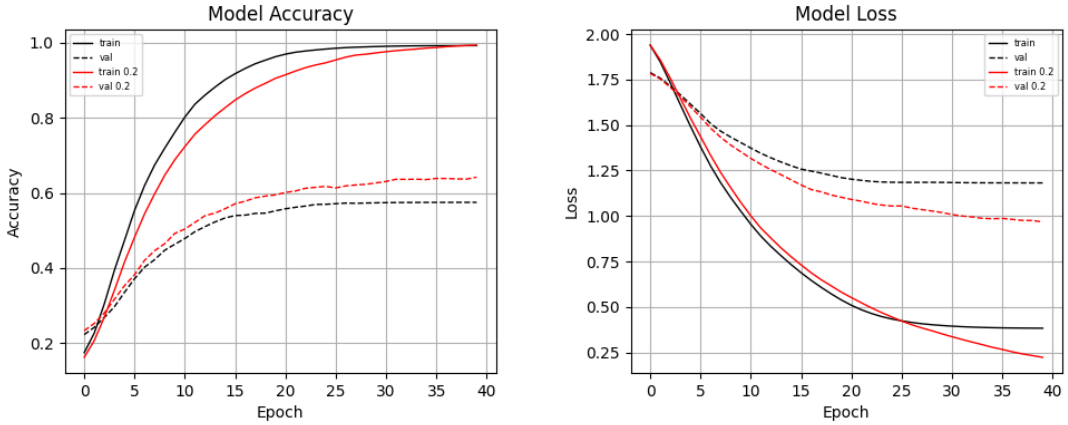
Given a classification task for 6 classes, a random baseline is 100/6=16.67%. Both show obviously better performance than a random baseline. Baseline model with 25088 features as input achieved even comparative result in comparison to the author's last experiment on Long-term Recurrent Convolutional Network (LRCN) in [12] where the spatial features were extracted during training from some convolution layers instead of from pre-trained extractor. This showed the first evidence of the effectiveness of using a pre-trained spatial extractor as well as careful data preprocessing. In addition, a noticeable point is that validation accuracy among folds differed clearly [59%, 61%, 63%, 53%, 51%]. While fold 0, 1, and 2 achieved reasonable and quite similar results, fold 3 and 4 had much worse scores. A random fixed validation set in this case would, as a consequence, give a less reliable result than averaging performance among folds according to Cross Validation procedure. For example, if fold 2 were chosen as validation, the model would slightly be overrated. Deploying such a model in real applications might expose surprise and unexpected results.

The high accuracy of the train set as well as the course of loss demonstrated that the network has enough learning capacity. However, the ability for generalization seemed to be limited. As next steps, a number of regularization methods were tried to boost it.

Dropout

Dropout is considered as one of the most powerful regularization techniques nowadays. In each iteration, a number of random neurons and their connections are removed which leads to a different version of the network, a smaller one. Training with Dropout is like

training a number of different networks and ensemble their power together. Dropout was experimented with drop rate in a range from 0.0 to 1.0 where 0.0 means keeping all neurons and 1.0 means dropping all neurons. Furthermore, 2 types of Dropout were exploited: input or also called naive Dropout and recurrent Dropout. Input Dropout refers to dropping neurons on the input/output at each step while recurrent Dropout drops neurons on the recurrent connection from timestep to timestep. Both could of course be used together. In fact, applying a small input dropout of 0.2 to the model with 25088 features as input showed a clear improvement. The gap of Overfitting was reduced by about 7%. Validation loss decreased continuously and faster accordingly as visualized in 3.17. The evidence showed that a bigger drop rate might uplift the validation accuracy further, which means a better generalization.



(a) Accuracy (b) Loss

Figure 3.17: Dropout effect

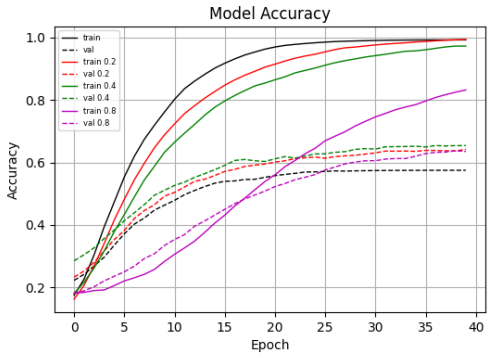


Figure 3.18: Comparison among different drop rates

Figure 3.18 features the result of different drop rates. As bigger drop rates were applied, the training was slowed down. This was after observation minimal and totally acceptable. Increasing Dropout probability to 0.4 showed further slight improvement. A brutal drop rate of 0.8 slowed down training significantly without further enhancement. Training the model 20 epochs more showed no difference. Applying recurrent Dropout as well as a mixture of both types presented the same result as naive Dropout. In conclusion, a naive Dropout of 0.4 gained the best result.

25088 vs 1024 features

The same Dropout effect was observed with 1024 features as input. The best model so far for 1024 features used a brutal dropout of 0.7 whereas for 25088 only 0.4. It is clear to see that 25088 features achieved a better result. It was indeed under expectation. Extracting features from the classifier part is rarely exploited as the features are filtered out for the particular task and don't contain general patterns anymore. Even though the spatial extractor was trained for the same task, extracting features from layer "dense" (1024 features) might scan out the particular features that the temporal extractor needs later on. That could be one possible explanation why 25088 features performed better than 1024. For further experiments, the author decided to use 25088 features as input only.

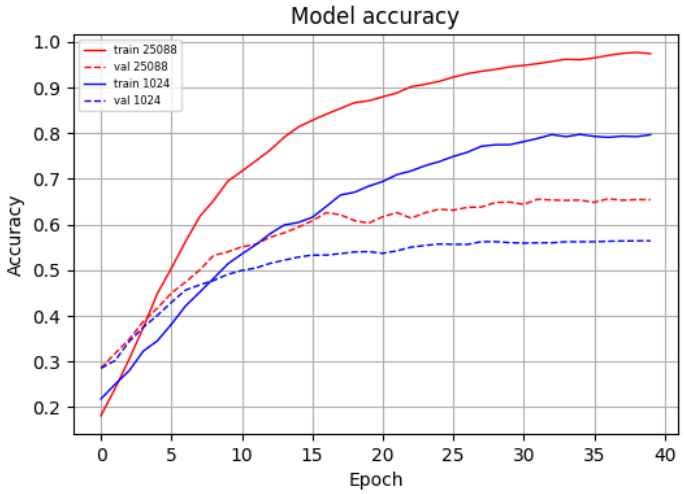


Figure 3.19: Comparison among different extracted features

Data Augmentation

Many studies have pointed out the effectiveness of Data Augmentation in fighting against Overfitting. By letting the model process more samples, it will be able to generalize

better. As argued before, online Data Augmentation would require extracting spatial features on the fly and cost a lot of time. Therefore, offline Data Augmentation was used. After the validation fold was chosen in each run, augmented data for the train data was loaded. The following transformation methods were implemented.

```
1  flipped_horizontally = cv2.flip(image, 1)
2  guassian_blur = cv2.GaussianBlur(image, (3, 3), 0)
3  brightness_higher = cv2.convertScaleAbs(image, alpha=1, beta=50)
4  brightness_lower = cv2.convertScaleAbs(image,   alpha=1,  beta=20)
5  contrast_higher = cv2.convertScaleAbs(image, alpha=2, beta=0)
6  contrast_lower = cv2.convertScaleAbs(image, alpha=1.7,  beta=50)
7  rotate_left = imutils.rotate(image, 8) // PyImageSearch
8  rotate_right = imutils.rotate(image, -8) //PyImageSearch
9  salt_and_pepper = salt_and_pepper(image, 0.01) // add 1% noise
10
11 def salt_and_pepper(image, amount):
12         b, g, r = cv2.split(image)
13         b = salt_and_pepper_one_channel(b, anount=amount)
14         g = salt_and_pepper_one_channel(g, anount=amount)
15         r = salt_and_pepper_one_channel(r, anount=amount)
16         return  cv2.merge((b, g, r))
17 def salt_and_pepper_one_channel(image, amount):
18         s_vs_p = 0.5
19         out = image.copy()
20         # Salt mode
21         num_salt = np.ceil(amount * image.size * s_vs_p)
22         coords = [np.random.randint(0, i - 1, int(num_salt))
23                   for i in image.shape]
24         out[coords] = 255
25         # Pepper mode
26         num_pepper = np.ceil(amount * image.size * (1. - s_vs_p))
27         coords = [np.random.randint(0, i - 1, int(num_pepper))
28                   for i in image.shape]
29         out[coords] = 0
30         return out
```

In total, for each video, there are 9 augmented one corresponding 9 methods as shown from line 1-9. Features were extracted from all augmented videos and stored for training. The train data was increased slowly with augmented data. Applying Data Augmentation indicated a noticeable improvement both in train and validation set. The best result was recorded when applying all augmented data, which means the train data was increased 10 times. As visualized in figure 3.20, the train accuracy curve suggests a huge speedup

in training where it reached 100% after only 20 epochs. In addition, validation accuracy increased clearly and achieved 70%, shrank the Overfitting gap to 30%.

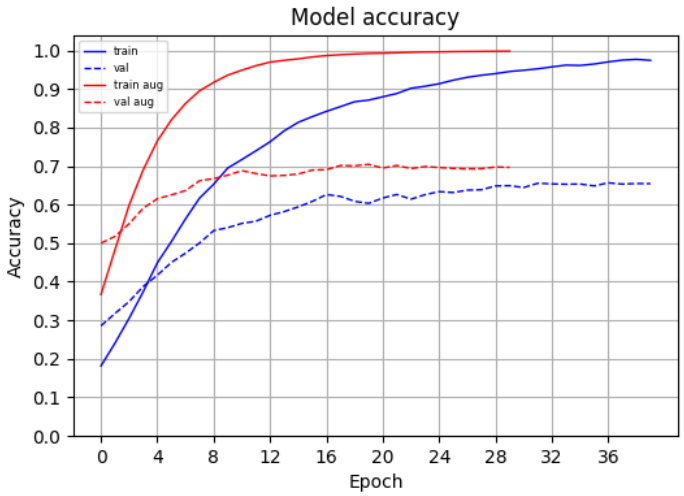


Figure 3.20: Data Augmentation effect

In summary, both Dropout and Data Augmentation demonstrated once again their reputation for fighting high variance a.k.a Overfitting. The Overfitting gap was reduced from 43% to 30%. The best-recorded validation accuracy was 70% which was averaged from all folds [72.9% 71.9% 71.9% 69.8% 64.6%]. Further attempts to close the Overfitting gap would require deep analysis and insights from the network as well as data and were unfortunately not done within this work.

Final model

Once the best set of hyperparameters was chosen, a final run was executed for each fold with Early Stopping. Train was stopped if the validation accuracy did not improve after 10 epochs. Lastly, a final model was trained with all train data available. As there was no validation set, the train was stopped at epoch 11 which were calculated by the number of epochs at which each fold was (early) stopped.

fold	max validation accuracy	epoch
0	75%	9
1	77%	22
2	77%	4
3	63.54%	4
4	66.67%	16

3.2.5 Alternatives

C3D and ConvLSTM were chosen as alternatives for Cascade Network. Both those architectures have the potential to extract spatio-temporal features. Concretely, C3D uses 3D kernel to scan through the input and therefore is able to detect changes among neighbor frames while ConvLSTM performs the convolution operation as part of the LSTM network. The raw videos were fed directly to the train model. In order to apply online Data Augmentation as well as to satisfy the memory need, a custom generator was implemented. Videos were loaded batch-wise and transformed if configured before feeding to the train model. The following ImageDataGenerator object was optionally passed to the generator which contains the transformation methods.

```
transformer = ImageDataGenerator(
            rotation_range=10,
            zoom_range=0.1,
            width_shift_range=0.1,
            height_shift_range=0.1,
            shear_range=0.15,
            horizontal_flip=True,
            brightness_range=[0.4, 1.5],
            fill_mode='nearest')
```

For each iteration, if Data Augmentation was configured, a new batch was generated by applying a random mixture from flipping horizontally, rotation between $-10°$ and $10°$, zooming in and out, shifting right and left, applying shear transformation, modifying brightness on the origin videos. A brief description of the training process for C3D is demonstrated below. ConvLSTM was trained in a similar manner and would be skipped.

As a starting point, the following parameters were configured.

- Inputs dimension: 100x160x160x3
- Optimizer: Adam with a fixed learning rate of 0.0001 for training; and with learning rate decay of factor 0.5 if accuracy does not improve after 5 epochs for finalizing model
- Batch size: 16 (A bigger batchsize exceeds the available memory.)
- Epoch: 20

- Shuffle data during training: True
- Early Stopping: not for training but for finalizing model with patience of 10 epochs
- ModelCheckPoint: saving only best weights based on accuracy of validation set
- CSVLogger: saving each iteration result to a csv file for further analysis

Baseline

It was very hard to find a reasonable baseline for C3D. Different network architectures were experimented. Figure 3.21 shows the result of a simple C3D with 1,2, or 3 convolution layers, each with 16 or 32 neurons. As observed, a deeper network performed clearly better. Concretely, model with 2 convolution layers of 16 neurons achieved about 7% higher than model with 1 convolution layer no matter with 16 or 32 neurons (see the red and dark blue dash lines). Increasing the number of convolution layers to 3, the accuracy reached slightly higher as displayed by the green dashed line. Yet it is clear to see that all models learned very fast and reached 100% on training set at very early epochs 8-10. A big gap of Overfitting remained no matter how long the model was trained.

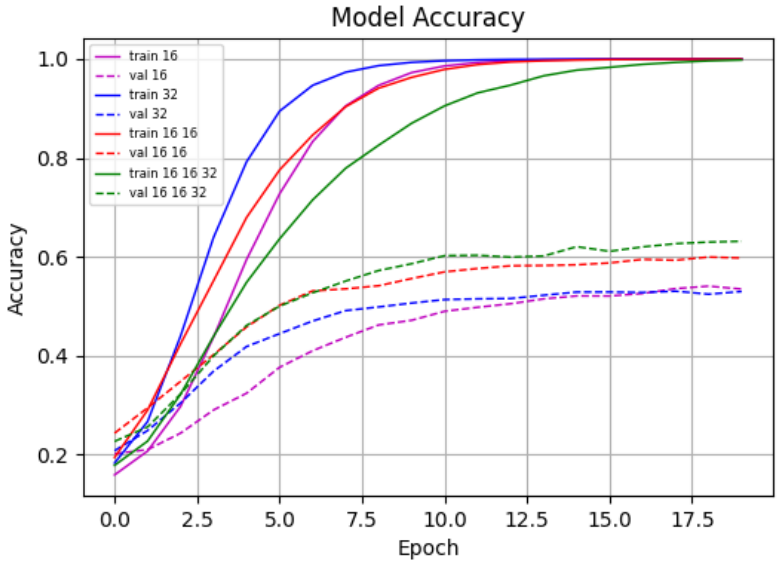


Figure 3.21: Comparison among different C3D networks

The final chosen baseline contained 3 convolution layers of 16 16 and 32 neurons, followed by a Flatten layer and 2 dense layers of 512 neurons which scored about 65% on validation set after 30 epochs. Below is its summary.

```
Model: "model-c3d"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 100, 160, 160, 3  0
                             )]

 conv3d (Conv3D)             (None, 98, 158, 158, 16)  1312

 max_pooling3d (MaxPooling3D  (None, 49, 79, 79, 16)   0
 )

 conv3d_1 (Conv3D)           (None, 47, 77, 77, 16)    6928

 max_pooling3d_1 (MaxPooling  (None, 23, 38, 38, 16)   0
 3D)

 conv3d_2 (Conv3D)           (None, 21, 36, 36, 32)    13856

 max_pooling3d_2 (MaxPooling  (None, 10, 18, 18, 32)   0
 3D)

 flatten (Flatten)           (None, 103680)            0

 dense (Dense)               (None, 512)               53084672

 dense_1 (Dense)             (None, 512)               262656

 dense_2 (Dense)             (None, 6)                 3078

=================================================================
Total params: 53,372,502
Trainable params: 53,372,502
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.22: C3D baseline summary

Against Overfitting

The same large Overfitting was observed as the baseline of Cascade Networks. Bigger kernel size was exploited k=5, 7, 9 with the hope that it might be able to capture motion among more frames. Yet it showed no effect. Other methods such as using BatchNormalization, using leaky relu as activation function, different initialization methods ([7]he_normal/[8]he_uniform), average pooling brought no further improvement either. Dropout was applied with different rates on both convolution layers as well as dense layers. No significant increase of validation accuracy was recorded.

Most surprised was the result observed while applying Data Augmentation. Online Data Augmentation was the first choice. However, instead of improving generalization, the

[7]https://www.tensorflow.org/api_docs/python/tf/keras/initializers/HeNormal
[8]https://www.tensorflow.org/api_docs/python/tf/keras/initializers/HeUniform

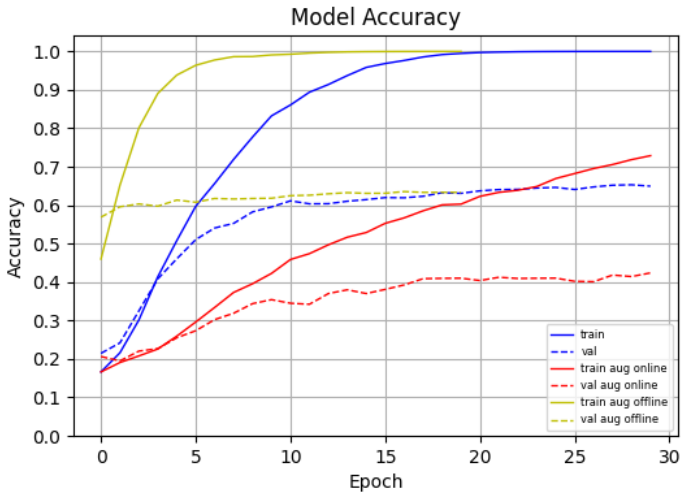


Figure 3.23: Data Augmentation online vs offline

training process was slowed down for no result. By looking at the red line and dashed line, it is to recognize that despite the slow training, the Overfitting gap did not decrease. With further training till 70 epochs, the train accuracy slowly increased while the validation accuracy remained around 40%. The author hardly found a reasonable explanation for this phenomenon. Attempting to understand whether the transformation methods were poor choices, the author decided to run the same model with a more simple ImageDataGenerator. Data was only augmented by slightly modifying brightness and shifting left/right.

```
simple_transformer =ImageDataGenerator(
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        brightness_range=[0.4, 1.5],
        fill_mode="nearest")
```

Yet the result shed no light on this matter. Ultimately, Data Augmentation is no magic pill. If the network does not have the capability to understand certain complex patterns, Data Augmentation might confuse the network instead of helping it. One last experiment was executed with offline Data Augmentation. The result is visualized as the yellow lines in figure 3.23. By increasing 9 times the train data, the network seemed to map the input to the output very fast. Evidence is that train accuracy reached 100% in only 7 epochs and validation accuracy scored low and stayed stable very soon which is more or less a

symbol of the so-called learn-by-heart phenomena. That means the model tried to map the input exactly to the output instead of learning the underneath features, patterns, and concepts.

3.2.6 Fusion

There are indeed different fusion techniques: early fusion, middle fusion, and late fusion. Early fusion, also known as feature level fusion, refers to merging input from multiple modalities into an input vector before feeding to the model. The merging inputs could be raw or extracted features. The merging techniques are for example concatenation, pooling or applying gate unit. Middle fusion or intermediate fusion, joint fusion is, as the name suggests, the process of joining representations from intermediate layers. Lastly, late fusion or decision-level fusion consists of unimodal decision values and leveraging them to make a final decision. That means that all models are trained separately in the first step. Some examples of aggregation methods are averaging, majority voting, weighted voting, using a meta-classifier such as Decision Tree or Support Vector Machine, using the maximal value among all, along with others. More about their key characteristic can be further read in [28]. In this work, the author decided to exploit only late fusion. One of its advantages is that it allows easy training and more flexibility at predictions. Training each modality is independent and the number of modalities to fuse can be decided at the very end. Furthermore, the experiment covers training different network architectures with very different styles of learning, it is better to let them learn their best skills before combining their power.

Concretely, once the best models were finalized for each type of network, they were then fused together to see whether the result could be further enhanced. The output of softmax layer from each model was taken as input for the fusing process. Moreover, 3 merging methods were tested. F(x_i) denotes the vector score of softmax layer and W_i the performance of model i.

Average:

$$f(x_1, x_2, ..., x_n) = \frac{f(x_1) + ... + f(x_n)}{n} \tag{3.1}$$

Weighted average:

$$f(x_1, x_2, ..., x_n) = \frac{W_1 * f(x_1) + ... + W_n * f(x_n)}{n} \tag{3.2}$$

Max:

$$f(x_1, x_2, ..., x_n) = max(f(x_1), ..., f(x_n)) \tag{3.3}$$

3.3 Evaluation

3.3.1 Human ground truth

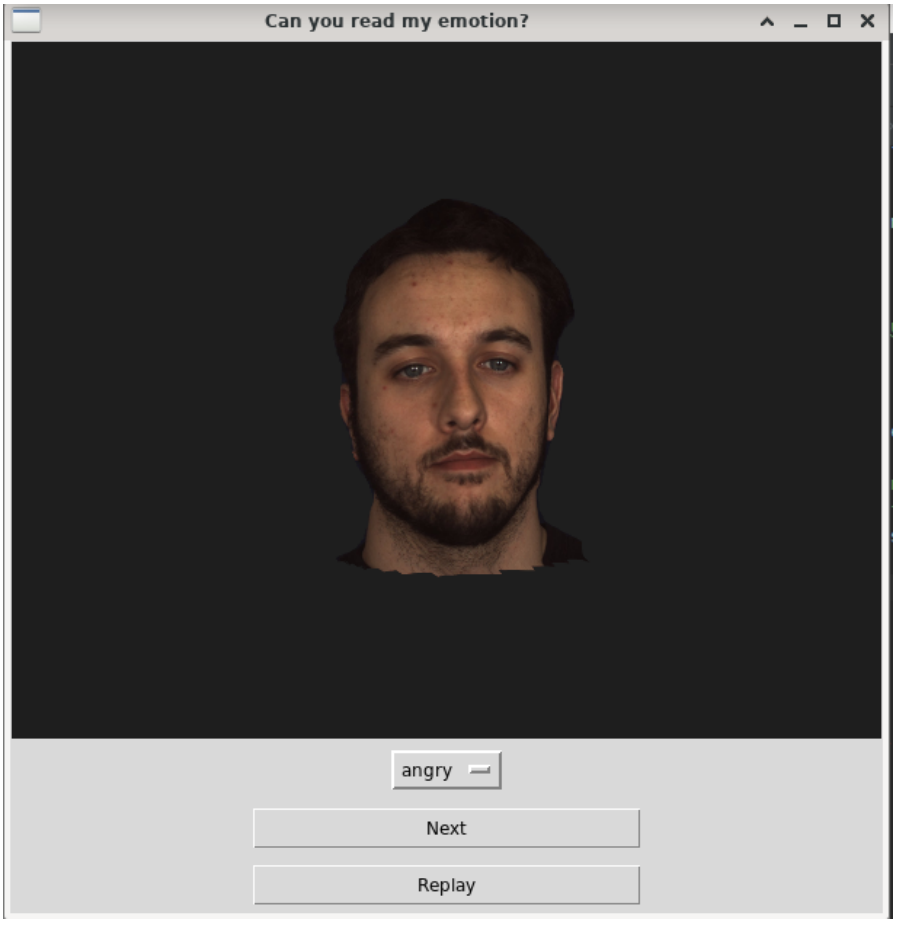


Figure 3.24: Experiment setup for human ground truth

This work followed the categorical emotion approach which states that there are indeed universal basic emotions regardless of gender, background, ethics, and so on. Therefore, they should be easily recognized. However, as the dataset was created in laboratory environment and the objects were provoked to express those emotions, the author decided to set up a small experiment to evaluate the human ground truth on the chosen test set

before further evaluation. Again, the chosen test set contains 120 videos from 20 people. All videos were randomly mixed. A group of people was asked to identify the emotion in each video. The choices were a fixed set of [angry, disgust, fear, happy, sad, surprise]. The environment was set up so that each object can perform the annotation uninterruptedly. Furthermore, the object was asked to give the best judgment instead of guessing. As that, they can replay each video as many times as necessary. Figure 3.24 shows the experiment setup.

The box and whisker plot summarizes the result, which brings up a lot of unexpected points. Each box visualizes one emotion. The lines extending along the box are called whiskers and denote the following key values: maximum, upper quartile, median (the yellow line), lower quartile, and minimum. There is only one outlier for emotion "happy" which is plotted as an individual dot.

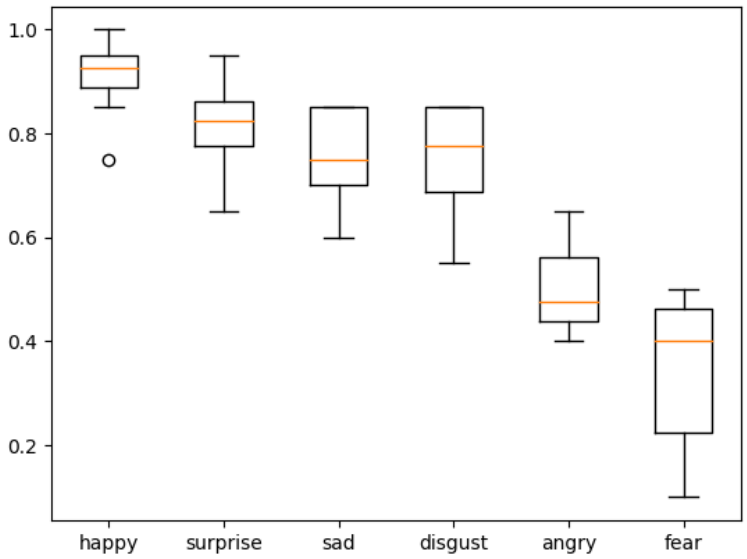


Figure 3.25: Human ground truth

It is clear to see that the overall result is not as high as expected. Instead of reaching roughly 100% for all emotions, the result is far worse. Median values recorded are happy: 90.63%, surprise: 81.25%, sad: 75.6%, disgust: 75%, angry: 50%, fear: 34% which yields an average of only 67.81% in total. Furthermore, the result differs significantly among classes. Emotion "happy" has the highest score of over 90%. "Disgust", "sad",

and "surprise" have lower results of around 80%. Especially bad results were recorded at emotion "angry" and "fear" with not more than 50%. In addition, the long box at emotion fear or, in other words, the broad data distribution demonstrates that the opinion among annotators for samples of this emotion distinguished a lot. Further analysis of all samples where more than 50% of annotators mistook showed that "angry" was mostly mislabeled as "disgust". The rest was mislabeled as "sad" or "fear". More details are visualized in figure 3.26. A review of some mislabeled samples across all classes showed in fact unclear states of emotion (e.g. a happy face with no smile) which explains more or less the unexpected evaluated ground truth.

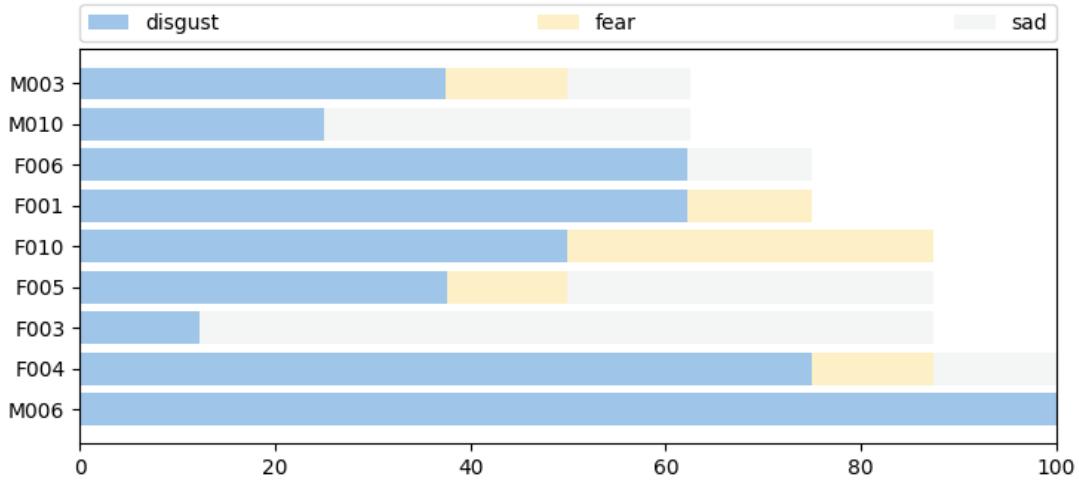


Figure 3.26: Mislabeled "angry" samples

Evaluation of trained models in the next subsection will be based on and respectively compared with this evaluated ground truth.

3.3.2 Best models

From each chosen network architecture, the best one was picked for final evaluation.

Architecture

For Cascade Networks, the spatial and temporal extractors were trained separately. Concretely, after fine-tuning a number of pre-trained CNN networks on the FER2013 dataset,

VGGFace was chosen as the spatial extractor. Output from Flatten layer of 25088 neurons was used as input to feed the temporal extractors which are of the following variants. All models ended up with a softmax layer as output layer.

- one LSTM layer with 128 embedding outputs. The final model used a drop rate of 0.4 and was trained with 9 times offline augmented data.
- one BiLSTM layer with 128 neurons. The final model used a brutal drop rate of 0.7 and was trained with 3 times offline augmented data.
- the temporal extractor contains 4 1D convolution layers, 2 max pooling layers, and 1 fully connected layer as visualized below:

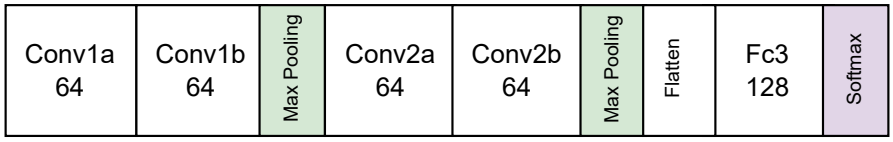


Figure 3.27: 1D-CNN as temporal extractor

The final ConvLSTM network has 2 2D ConvLSTM layers of 8 neurons, followed by a Flatten layer. The C3D network consists of 3 3D convolution layers of 16 16 and 32 neurons, all with a kernel size 3x3, each followed by a max pooling layer.

Accuracy

These models were evaluated on the chosen test set. Table 3.6 summarizes the result.

		Test accuracy
Cascade Networks	VGGFace + LSTM	62.5%
	VGGFace + BiLSTM	62.5%
	VGGFace + 1D-CNN	**65%**
C3D		50%
ConvLSTM		44.2%
Human ground truth		**67.8%**

Table 3.6: Best models

It is obvious that all best models exceed the random baseline of 100/6=16.67%. Among all chosen experimented network architectures, a combination of fine-tuned VGGFace and 1D-CNN achieved the highest accuracy, only 2.8% less than the human ground

truth. Furthermore, when the samples were correctly predicted, the confidence scores were pretty high. The overall confidence scores were after observation a lot better than from other candidates.

angry: ['0.53', '0.96', '0.98', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00']
disgust: ['0.54', '0.62', '0.77', '0.80', '0.97', '0.99', '0.99', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00']
fear:['0.55', '0.60', '0.75', '0.84', '0.98', '0.98', '1.00', '1.00', '1.00']
happy:['0.94', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00']
sad:['0.60', '0.60', '0.64', '0.74', '0.75', '0.95', '0.97', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00']
surprise:['0.74', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00']

This strengthens the result from recent studies about the effectiveness of 1D-CNN when working with sequences. In this experiment, 1D-CNN did indeed outperform LSTM and BiLSTM. Besides, all Cascade Networks demonstrated strong results of over 60%. Compared to the simple LRCN trained in [12], this shows evidence that transfer learning does have a positive impact after all. Alternative networks without separate temporal and spatial extractors such as C3D, ConvLSTM gained also quite low scores, both not more than 50%. There are a lot of studies that demonstrated the effectiveness of C3D, yet only on short sequences. This work showed its limitation in handling long ones.

Fusion

Different networks were merged together by the 3 chosen merging methods: average, weighted average, and maximum. The results are shown in table 3.7. Average and weighted average fusion of 2 types of LSTM networks showed slight improvement from 62.5% to 63.33%. Another little enhancement was found when merging the best Cascade Network VGGFace+1D-CNN with the best alternative C3D.

	average	weighted average	maximum
VGGFace+LSTM VGGFace+BiLSTM	63.33%	63.33%	62.5%
VGGFace+LSTM VGGFace+BiLSTM VGGFace+1D-CNN	63.33%	63.33%	62.5%
C3D ConvLSTM	47.5%	46.67%	48.33%
VGGFace+1D-CNN C3D	62.5%	65.83%	63.33%
All	63.33%	64.17%	62.5%

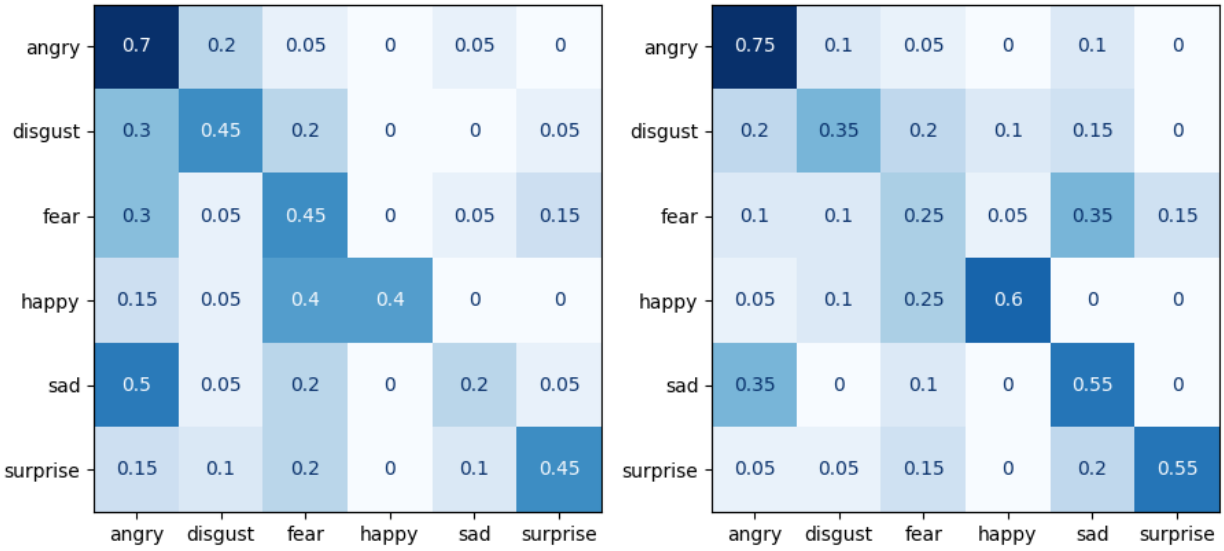
Table 3.7: Fusion

Emotion

Further details on the performance of particular emotions are shown in the normalized Confusion Matrix below (see figure 3.28). The first insight is that the accuracy differs a lot among classes regardless of network architecture. This is indeed aligned with the recorded human ground truth. It seems that even though they are universal emotions, some of them happens to be recognized easier while other are more difficult.

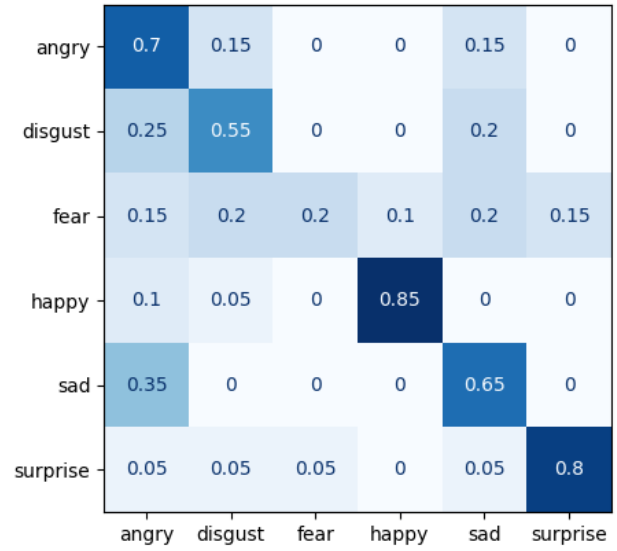
Concretely, for Cascade Networks, irrespective of the combination, emotion "happy" scored consistently highest (85%) while emotion "fear" landed lowest (20-45%). This actually lines up with the evaluation of the fine-tuned spatial extractor on the FER2013 dataset. In fact, it draws more evidence that the transferred knowledge from fine-tuning process seems to play an important role in the final prediction.

In addition, in comparison to the human ground truth, results from emotion "surprise" are also similar which strongly suggests that "happy" and "surprise" are the two emotions that can be identified easiest. The high confidence scores of almost 100% for correctly labeled samples listed above backed up this proposition as well. A simple explanation could be the consistent and clear motion while expressing those emotions: a smile for happiness or a big mouth with raised eyebrows for surprise. Especially, the result from VGGFace+1D-CNN seems to align even closer to the ground truth where "happy" scores best, "disgust", "sad", and "surprise" have reasonable scores and the lowest belongs to "angry" and "fear". It shows undeniable evidence that Cascade Networks are capable of capturing the necessary features to make such correct decisions as human, as a result, in possession of corresponding generalization ability.

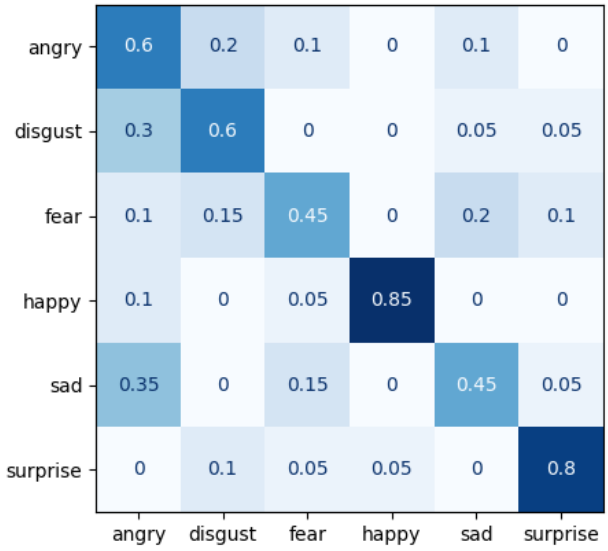


(a) ConvLSTM

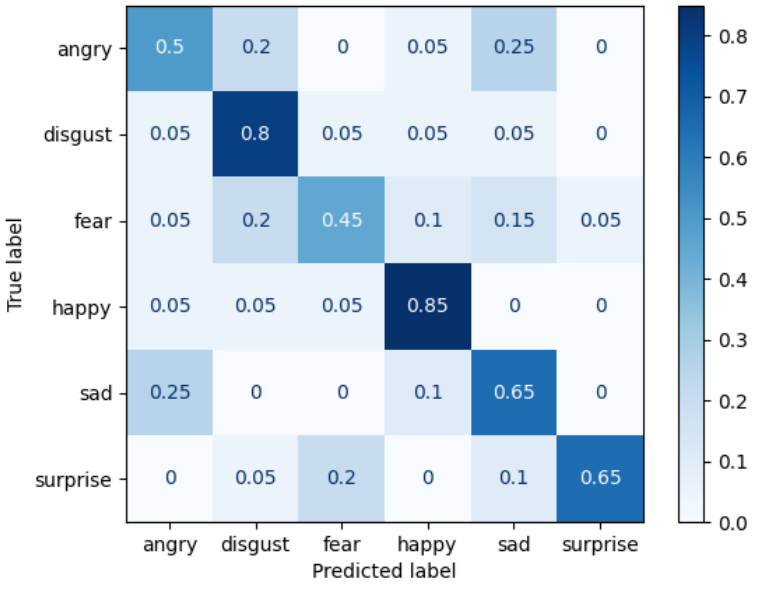
(b) C3D



(c) VGGFace+LSTM



(d) VGGFace+BiLSTM



(e) VGGFace+1D-CNN

Figure 3.28: Normalized Confusion Matrix

Training techniques

The following hyperparameters were tuned during training:

- Number of layers, number of neurons in each layer
- Activation function: relu or leaky relu
- Initialization methods: glorot_uniform, he_uniform, he_normal
- Dropout types, drop rate
- Data Augmentation: online or offline
- For CNN: kernel size, pooling layers, BatchNormalization

Overall, the author found it extensively hard to fight against Overfitting. Regardless of the chosen network architecture, all models overfitted very early. Especially, C3D mapped the input to the output very fast without really extracting the needed features which is known as learn-by-heart phenomenon. Similar to C3D, it was incredibly difficult to regularize ConvLSTM network. No matter which regularization techniques were applied, no significant improvement was acknowledged. As the network did not react to Data Augmentation clearly, it was impossible to compare the effectiveness of online vs offline Data Augmentation in a reliable manner. Fortunately, Dropout and Data Augmentation were helpful while applying to Cascade Networks. The Overfitting gap sank by 13%.

Cross Validation was conclusively a good choice. The author acknowledged the difference among folds. The accuracy among folds varied considerably, mostly around 5-10% and sometimes up to 40%. Without Cross Validation, the result would rely hardly on the randomly chosen validation set which is not a reliable performance measure for testing algorithm. By applying Cross Validation procedure, a much better algorithm could be picked based on accuracy averaged from all folds.

Error Analysis

Despite the fact that the best model achieved a comparative result towards human, the author decided to examine the mislabeled samples further. Figure 3.29 showed all mislabeled samples from VGGFace+1D-CNN with detailed output of softmax layer which

indicates the probability of each class being the right label. The same analysis could be done for other models and would be skipped.

The table consists of 8 columns of index and name of the mislabeled sample, correct label, predicted label, and predicted scores for all classes C1-C6 (angry, disgust, fear, happy, sad, surprise). For each row, 2 numbers are marked green which are at the correct label and the wrongly predicted label. The red number indicates all other classes which have a probability of bigger than 0.01. For example, row 2 displays the sample of emotion "angry" from object M002 which was mislabeled as "sad". The confidence score given to C1 - angry and C5 - sad of 0.22 and 0.76 are marked green. Besides, the model also predicts this sample as C3 - fear with a probability of 0.02 which is marked red.

Overall, there are 42 mislabeled samples including angry (10), disgust (4), fear (11), happy (3), sad (7), and surprise (7). There are very less numbers marked red which suggests that when the model did mislabel, it indeed only confused the correct label with another class. Further observation indicates that the model gave the wrong class in most cases a rather high confidence score. 2 explanations are either the model lacks certain abilities to learn the difference among certain emotions or that these emotions themselves have so much in correlation that is hard to separate cleanly. Digging further into the analysis, the author found some shreds of evidence for the second proposition. The model confused "angry" quite often with "disgust" and "sad" similar as observing at human. In fact, by rechecking a number of mislabeled samples, similar motions around the eyes region can be seen at the expressions of emotion "angry", "sad", and "disgust". It can be a possible reason why it is hard for both human and machine to distinguish those emotions. Furthermore, the author found some videos with very poor quality. Even though the object got professional instructions to provoke these emotions, some still looked very unnatural. For example, emotion "happy" can be identified quite easily with a smile and raised cheek. Yet some videos failed to express so. Object M002 expressed happiness with one facial expression throughout the whole sequence and it is difficult to say for sure whether a smile is displayed. Similarly, object M001 expressed emotion "happy" confusingly with a lower brow and no smile.

Index	Real	Predict	C1	C2	C3	C4	C5	C6
107.M008	1. angry	5. sad	**0.45**	0.00	0.00	0.00	**0.55**	0.00
110.M002	1. angry	5. sad	**0.22**	0.00	**0.02**	0.00	**0.76**	0.00
17.M007	1. angry	4. happy	**0.00**	**0.17**	0.00	**0.83**	0.00	0.00
27.M010	1. angry	5. sad	**0.18**	**0.13**	**0.04**	0.00	**0.65**	0.00
3.F005	1. angry	5. sad	**0.00**	0.00	0.00	0.00	**1.00**	0.00
35.F006	1. angry	2. disgust	**0.02**	**0.92**	**0.06**	0.00	0.00	0.00
62.F010	1. angry	2. disgust	**0.00**	**1.00**	0.00	0.00	0.00	0.00
70.F003	1. angry	5. sad	**0.20**	0.00	0.00	0.00	**0.80**	0.00
77.M006	1. angry	2. disgust	**0.00**	**1.00**	0.00	0.00	0.00	0.00
84.F004	1. angry	2. disgust	**0.00**	**1.00**	0.00	0.00	0.00	0.00
1.F005	2. disgust	3. fear	**0.02**	**0.00**	**0.98**	0.00	0.00	0.00
105.M008	2. disgust	4. happy	0.00	**0.36**	0.00	**0.64**	0.00	0.00
112.M002	2. disgust	1. angry	**0.66**	**0.12**	**0.22**	0.00	0.00	0.00
8.F008	2. disgust	5. sad	**0.07**	**0.00**	0.00	0.00	**0.93**	0.00
108.M002	3. fear	1. angry	**0.86**	0.01	**0.02**	0.00	**0.11**	0.00
118.F007	3. fear	5. sad	0.01	0.00	**0.13**	0.00	**0.86**	0.00
13.M007	3. fear	4. happy	0.00	**0.03**	**0.16**	**0.81**	0.00	0.00
25.M010	3. fear	2. disgust	**0.21**	**0.70**	**0.08**	**0.01**	0.00	0.00
47.M009	3. fear	5. sad	0.00	0.00	**0.00**	0.00	**1.00**	0.00
5.F005	3. fear	5. sad	0.00	0.00	**0.25**	0.00	**0.75**	0.00
56.M004	3. fear	2. disgust	0.00	**1.00**	**0.00**	0.00	0.00	0.00
75.M006	3. fear	2. disgust	0.00	**0.86**	**0.00**	**0.13**	0.00	0.00
80.F009	3. fear	6. surprise	0.00	0.00	**0.01**	0.00	0.00	**0.99**
91.M001	3. fear	2. disgust	0.00	**1.00**	**0.00**	0.00	0.00	0.00
98.M005	3. fear	4. happy	0.00	0.00	**0.00**	**1.00**	0.00	0.00
111.M002	4. happy	3. fear	0.00	0.00	**1.00**	**0.00**	0.00	0.00
83.F009	4. happy	1. angry	**0.96**	**0.03**	0.00	**0.01**	0.00	0.00
93.M001	4. happy	2. disgust	0.00	**1.00**	0.00	**0.00**	0.00	0.00
100.M005	5. sad	1. angry	**1.00**	0.00	0.00	0.00	**0.00**	0.00
16.M007	5. sad	4. happy	0.00	**0.01**	0.00	**0.98**	**0.01**	0.00
28.M010	5. sad	4. happy	0.00	0.00	0.00	**0.99**	**0.00**	0.00
53.F001	5. sad	1. angry	**0.82**	0.00	0.00	0.00	**0.18**	0.00
73.M006	5. sad	1. angry	**0.77**	**0.02**	**0.13**	**0.06**	**0.01**	0.00
79.F009	5. sad	1. angry	**0.97**	**0.01**	0.00	0.00	**0.02**	0.00
92.M001	5. sad	1. angry	**1.00**	0.00	0.00	0.00	**0.00**	0.00
10.F008	6. surprise	3. fear	0.00	0.00	**1.00**	0.00	0.00	**0.00**
115.F007	6. surprise	5. sad	0.00	0.00	**0.06**	0.00	**0.94**	**0.00**
42.M009	6. surprise	5. sad	0.00	0.00	0.00	0.00	**1.00**	**0.00**
50.F001	6. surprise	3. fear	0.00	0.00	**1.00**	0.00	0.00	**0.00**
81.F009	6. surprise	3. fear	**0.25**	**0.08**	**0.67**	0.00	0.00	**0.00**
86.F004	6. surprise	3. fear	0.00	0.00	**1.00**	0.00	0.00	**0.00**
95.M001	6. surprise	2. disgust	0.00	**1.00**	0.00	0.00	0.00	**0.00**

Figure 3.29: Error Analysis for VGGFace+1D-CNN

4 Summary

4.1 Conclusions & future work

Based on the literature review, this thesis conducted an experiment of classifying 6 basic emotions {anger, sadness, disgust, surprise, fear, happiness} from the dataset BU4FE using different Neural Networks and DL techniques.

All 5 chosen architectures achieved better accuracy than a random baseline of 16.67%. The best results are observed at Cascade Networks with fine-tuned spatial feature extractor. A combination of fine-tuned VGGFace and a simple 1D-CNN gained the highest score of 65% on the test set. Furthermore, re-evaluating the ground truth of the test set showed a surprising bad result of only 67.8%. As that, it is safe to say that the best algorithm performed nearly as well as human. Different late fusion techniques were exploited. Yet no significant enhancement was found. A slight improvement was shown while fusing the best Cascade Network with C3D using weighted average method. Ultimately, the result of the trained model can only be as good as the data it sees. When it has already reached its capacity, it is extremely hard to improve beyond. Even though the error rate is still quite high, the author deeply believed that FER algorithm is ready for real-life applications. In the end, it is undeniable that even for human, emotion is very complex, especially in the way it is interpreted. While building applications, it is important to bear this fact in mind instead of expecting unrealistic results from the built models.

The fact that VGGFace+1D-CNN gained the best performance raises 2 important points. Firstly, it validated once again the effectiveness of transfer learning, especially when knowledge is transferred from a similar task. Among all chosen pre-trained networks for fine-tuning, VGGFace is indeed the only network that was pre-trained with faces. Secondly, the result shows that 1D-CNN outperformed LSTM and BiLSTM which are the standard choices when it comes to sequences. It confirmed once again the results in

recent studies about the effectiveness of 1D-CNN when dealing with temporal data. The common association between sequence modeling and RNN needs to be reconsidered and 1D-CNN should be conceded as a potential candidate for such tasks.

Among all emotions, "happiness" and "surprise" were found to have consistently good results whereas "fear" mostly worst result. Emotion "anger" was often mistaken as "disgust" and "sadness". A reasonable explanation could be the clear presence of certain elements such as a smile and raised cheek (happy) or a big opened mouth (surprised). On the other hand, lower brow was found often in all 3 emotions "anger" "disgust" and "sadness" which might confuse the trained models. Without further factors, it is even difficult for human to distinguish those emotions.

Training NN is hard, especially due to the limited dataset. Regardless of the chosen architectures, a big gap of Overfitting was to see. The author found extensively dense to fight against Overfitting. A number of regularization methods were exploited. Yet only some seemed to show effect. The author acknowledged the effectiveness of Dropout and offline Data Augmentation while training Cascade Neworks. However, no improvement was found while applying them to C3D and ConvLSTM. Most unexpected was that even online Data Augmentation had no effect on C3D. The training process was slowed down noticeably for no result.

The author endorsed the observable difference among folds. A fixed chosen validation set would have brought a lot of bias to the performance estimation of the tested algorithms. An average score of folds provided a much more reliable result. Therefore, Cross Validation is ultimately the right procedure.

In conclusion, FER is ready for real-life application and Deep Learning is the right choice for this task. However, more works need to be done in order to continuously 1. expand the dataset both in volume and quality and 2. re-evaluate and improve the current successful Neural Networks as well as encourage new architectures to come on the scene.

Nonetheless, many of the applied fields of FER require the task to be handled in real-time. [31] listed some examples including:

- intelligent monitoring: In a classroom or especially online classroom, the emotional states of participants can be detected and analyzed. Based on their concentration and mental state, corresponding actions might be executed such as a break, easier or more difficult exercises or certain actions towards the unfocused individuals to regain their attention.

- criminal interrogation: During questioning, real-time detection of the psychological state of the investigated person can help determine whether she/he is lying or concealing the truth.
- telemedicine: FER can assist doctors in understanding the patients precisely and as a result giving better treatment.
- fatigue-driving detection: It can be dangerous in terms of traffic safety. Identifying the state of the driver can help the system make judgments and give warnings in time.
- intelligent robots such as hospital/hotel lobbies, education assistants, and so on: It is important for such systems to react in a human manner towards users. In order to do so, it firstly needs the ability to understand the expressed emotion.

Real-time FER faces a lot of challenges. Inference time is for instance critical. Yet the outcome is futuristic and worth further researching. Therefore, the author dedicates the last section to summarize the current development of real-time FER.

4.2 FER in real-time

In order to facilitate real-time applications, FER needs to be fast and be able to handle unideal environments such as occlusion, imperfect brightness, and so on. Furthermore, it has to deal with the memory and computation limitation on the devices the system is occupied such as mobile and edge devices. In the following, the author will discuss 2 main aspects pertaining to real-time DL application in general and FER in particular.

Hardware

Hardware is more or less a general topic when it comes to not only FER but all DL applications. It is important to continuously improve the quality of current hardware both for training and production. For running in real-time, it is essential to have as low inference time as possible. There are [1]FPGAs and [2]ASICs developed explicit for DL, for example [3]TPU which was originally designed by Google can accelerate NN on Tensorflow software significantly. Besides cost and power, another essential key when

[1]Field programmable gate array
[2]Application-specific integrated circuit
[3]Tensor processing unit

it comes to designing processor for edge, mobile, and embedded applications is thermal dissipation. [4]VPU by Intel is exactly a designed chip that provides ultra-low power capabilities without compromising performance. The author deeply believes in the potential of Edge Computing and Artificial Intelligence or also known as Edge AI. Without doubt, the author expects many more applications in the context of smart cities, [5]IoT networks in the near future.

Another less discussed, yet very important point regarding to hardware is how to choose the right hardware, especially for inference at the edge. It depends on a lot of factors such as the system requirements in terms of performance and power consumption, the available budget, along with others. Therefore, it is essential to analyze the problem carefully before moving to the implementation. FPGA can be easily reprogrammed and could be a very good choice if the system has to fulfill multiple purposes. Even though it is possible to inform about the characteristic of each type of hardware as well as their advantages and disadvantages, the best way to figure out the most suitable hardware is indeed through testing. In addition, it helps save a lot of money investing in the hardware upfront. One of the few platforms which enables hardware testing is Intel DevCloud. It allows developers to easily test a range of Intel processors.

Neural Network

There are a number of things about NN for real-time DL applications and FER which have been investigated in recent studies.

1. developing the right algorithm to handle data with limitation
2. designing and training the right network for running on specific types of devices
3. optimizing the trained model for specific types of devices

Limitations mentioned in 1 which can hinder FER are e.g. occlusion by glasses, masks, jewelry or non-frontal face. In fact, there are different ways to overcome those issues. The first one is by training FER models on datasets with occluded objects. Another way is by completing occluded regions before further process. Generative adversarial net, abbreviated as GAN and its variants have been shown to be very helpful in this matter. It can be used to regenerate the occluded areas, to construct frontal faces based on the angles as well as to generate data with specific characteristics e.g. masks,

[4]Vision processing unit
[5]Internet of Thing

bear, glasses, skin winkles, make-up, nose ring which can then be used to train certain models. [17] summarized the related work around face frontalization and face synthesis and found out that most works proposed to solve the pose and occlusion sep arately. As a consequence, the effect of synthesis is worse clearly if both occlusions and pose variations exist simultaneously. Then, they proposed BoostGAN to achieve both frontalization and deocclusion. Finally, they validated their method on a number of benchmarks and took it under comparison with other state of the art (SOTA) GAN.

The second trend is about designing and training device-oriented models. There is no surprise that a lot of applications are nowadays designed for mobile and edge devices. However, these devices usually have limited resources in terms of memory and computation. Yet, most effective NNs tend to have large parameter sizes and require massive computational resources. Therefore, it is obviously difficult to apply such models on these devices. In order to solve this problem, many researchers have introduced a number of DL models designed for specific types of devices. [14] mentioned some mobile DL models such as BlazeFace, MobileFaceNet, MnasNet, and EfficientNet and proposed in addition their own for FER with less than 0.5 million parameters without sacrificing the accuracy. Another possibility to handle the limitation of mobile devices is to store the model and run the heavy work in the cloud. Nevertheless, [31] investigated this approach and pointed out its disadvantage of communication delay where data has to be uploaded to the cloud and result has to be sent back to end devices. It further proposed a lightweight edge computing-based distributed system using Rasberry Pi.

Building models for specific devices can be expensive in case of cross-device applications. Furthermore, it is inflexible as new devices are coming in the future. The last trend tries to optimize the trained model for specific devices like edge devices by e.g. reducing the complexity of the model so that it will take up fewer resources to store and run. There are 2 typical ways to do so including reducing the model size and reducing the number of operations or layers which are introduced in detail [6]here. Reducing the size of the model means removing unimportant or redundant parameters. Reducing the number of operations means reducing the number of operations or calculations needed to execute the network. While reducing the size of the model helps reduce the time it takes to load and compile the model, reducing the number of operations will reduce the time needed to perform inference. Both will help decrease the execution time which is crucial for real-time applications. Furthermore, the lower the size of the model is, the less space

[6]https://www.udacity.com/course/intel-edge-ai-for-iot-developers-nanodegree–nd131

will be required for storing the model. There are a number of techniques to reduce the model size including:

- Quantization is about converting high precision weights to low precision weights. For example: converting 32-bit floating-point FP32 to 16-bit floating-point FP16 or 8-bit fixed-point INT8
- Model compressing refers to reducing the model size while storing by finding clever ways to represent weights in memory.
- Knowledge distillation is to train a small network to imitate the knowledge of the large network trained in advance [47]. This is often compared to a "teacher-student" relationship where the large network as a teacher transfers its generalization ability by different methods to the student - the small network.

Reducing the number of operations can be done either by using more efficient layers or by removing connections in between neurons a.k.a pruning. Convolutional layers in CNN model often require a lot of computational power and can be easily fixed by replacing them with less compute-intensive layers such as depthwise separable convolution. Another technique is to use downsampling layers to reduce the number of parameters fed to the next layers. Pruning aims to reduce redundant weights and networks by selecting and deleting trivial parameters that have a small impact on the model's accuracy and then retrain the model to recover the performance [47].

A further read on all techniques for model optimization can be found [7]here and [47]. One of the toolkits for this purpose is the open-source library OpenVINO.

Besides optimizing the model, optimizing the pipeline such as the preprocessing step, the network, etc. can also speed up the run time.

[7]https://www.udacity.com/course/intel-edge-ai-for-iot-developers-nanodegree–nd131

Bibliography

[1] Homepage Creative Space for Technical Innovation (CSTI). URL: https://csti.haw-hamburg.de/.

[2] Object Tracking using OpenCV (C++/Python). URL: https://learnopencv.com/object-tracking-using-opencv-cpp-python/.

[3] scikit-learn. URL: https://scikit-learn.org/.

[4] Advances in Deep Learning Methods for Visual Tracking: Literature Review and Fundamentals. *International Journal of Automation and Computing*, 2021. doi:10.1007/s11633-020-1274-8.

[5] Tadas Baltrušaitis, Amir Zadeh, Yao Chong Lim, and Louis-Philippe Morency. OpenFace 2.0: Facial Behavior Analysis Toolkit. *IEEE International Conference on Automatic Face and Gesture Recognition*, 2018.

[6] Luay Alawneh, Belal Mohsen, Mohammad Al-Zinati, Ahmed Shatnawi, and Mahmoud Al-Ayyoub. A Comparison of Unidirectional and Bidirectional LSTM Networks for Human Activity Recognition. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 1–6, 2020. doi:10.1109/PerComWorkshops48775.2020.9156264.

[7] Wissam J. Baddar and Yong Man Ro. Learning Spatio-temporal Features with Partial Expression Sequences for on-the-Fly Prediction, 2017. arXiv:1711.10914.

[8] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *CoRR*, abs/1803.01271, 2018. URL: http://arxiv.org/abs/1803.01271, arXiv:1803.01271.

[9] Lisa Feldman Barrett and Tor D. Wager. The structure of emotion: Evidence from neuroimaging studies. *Current Directions in Psychological Science*, 15(2):79–83, 2006. doi:10.1111/j.0963-7214.2006.00411.x.

[10] Sven Buechel and Udo Hahn. Emotion Analysis as a Regression Problem — Dimensional Models and Their Implications on Emotion Representation and Metrical Evaluation. 08 2016. doi:10.3233/978-1-61499-672-9-1114.

[11] Thi Huyen Cao. Bachelor thesis: German Word Level Lip Reading with Deep Learning. URL: https://reposit.haw-hamburg.de/handle/20.500.12738/8807.

[12] Thi Huyen Cao. Hauptprojekt: Implementation of an end-to-end Deep Learning pipeline for Facial Expression Recognition. URL: https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2021-proj/cao_hp.pdf.

[13] Thi Huyen Cao. Seminar: Facial Expression Recognition. URL: https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2020-proj/cao.pdf.

[14] ChangRak Yoon and DoHyun Kim. Mobile Convolutional Neural Networks for Facial Expression Recognition. 2020.

[15] Bincy Chellapandi, M. Vijayalakshmi, and Shalu Chopra. Comparison of Pre-Trained Models Using Transfer Learning for Detecting Plant Disease. In *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pages 383–387, 2021. doi:10.1109/ICCCIS51004.2021.9397098.

[16] Jiankang Deng, Jia Guo, Yuxiang Zhou, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. RetinaFace: Single-stage Dense Face Localisation in the Wild. *CoRR*, abs/1905.00641, 2019. URL: http://arxiv.org/abs/1905.00641, arXiv:1905.00641.

[17] Qingyan Duan and Lei Zhang. Look more into occlusion: Realistic face frontalization and recognition with boostgan. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):214–228, 2021. doi:10.1109/TNNLS.2020.2978127.

[18] Tuomas Eerola and Jonna K. Vuoskoski. A comparison of the discrete and dimensional models of emotion in music. *Psychology of Music*, 39(1):18–49, 2011. arXiv:https://doi.org/10.1177/0305735610362821, doi:10.1177/0305735610362821.

[19] Ekman, P., and Friesen, W. V. Constants across cultures in the face and emotion. *Journal of Personality and Social Psychology, 17(2), 124–129*, 1971. URL: https://doi.org/10.1037/h0030377.

[20] Olufisayo Ekundayo and Serestina Viriri. Facial Expression Recognition: A Review of Trends and Techniques. 2021.

[21] Yin Fan, Xiangju Lu, Dian Li, and Yuanliu Liu. Video-based emotion recognition using cnn-rnn and c3d hybrid networks. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, ICMI '16, page 445–450, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2993148.2997632.

[22] Yarin Gal and Zoubin Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks, 2016. arXiv:1512.05287.

[23] Mudasir A. Ganaie, Minghui Hu, Mohammad Tanveer, and Ponnuthurai N. Suganthan. Ensemble deep learning: A review. *CoRR*, abs/2104.02395, 2021. URL: https://arxiv.org/abs/2104.02395, arXiv:2104.02395.

[24] F.A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194 vol.3, 2000. doi:10.1109/IJCNN.2000.861302.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, 2015. arXiv:1512.03385.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. doi:10.1162/neco.1997.9.8.1735.

[27] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks, 2018. arXiv:1608.06993.

[28] Huang, SC., Pareek, A., Seyyedi, S. et al. Fusion of medical imaging and electronic health records using deep learning: a systematic review and implementation guidelines. 2020. URL: https://doi.org/10.1038/s41746-020-00341-z.

[29] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. URL: http://arxiv.org/abs/1502.03167v3.

[30] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013. doi:10.1109/TPAMI.2012.59.

[31] JIANNAN YANG , TIANTIAN QIAN, FAN ZHANG , AND SAMEE U. KHAN, (Senior Member, IEEE). Real-Time Facial Expression Recognition Based on Edge Computing. 2021.

[32] Jan Kukacka, Vladimir Golkov, and Daniel Cremers. Regularization for Deep Learning: A Taxonomy. *CoRR*, abs/1710.10686, 2017. URL: http://arxiv.org/abs/1710.10686, arXiv:1710.10686.

[33] Shan Li and Weihong Deng. Facial Expression Recognition: A Survey. 2018. URL: http://arxiv.org/abs/1804.08348.

[34] Prof. Dr.-Ing. Andreas Meisel. HAW Deep Learning Project.

[35] Shervin Minaee, Ping Luo, Zhe Lin, and Kevin W. Bowyer. Going Deeper Into Face Detection: A Survey. *CoRR*, abs/2103.14983, 2021. URL: https://arxiv.org/abs/2103.14983, arXiv:2103.14983.

[36] Andrea Vedaldi Omkar M. Parkhi and Andrew Zisserman. Deep face recognition. 2015. URL: https://www.robots.ox.ac.uk/~vgg/publications/2015/Parkhi15/parkhi15.pdf.

[37] James Russell. A circumplex model of affect. *Journal of Personality and Social Psychology*, 39:1161–1178, 12 1980. doi:10.1037/h0077714.

[38] James A. Russell and Albert Mehrabian. Evidence for a three-factor theory of emotions. *Journal of Research in Personality, 11(3), 273–294*, 1977.

[39] Pei-Wei Tsai Rusul L. Abduljabbar, Hussein Dia. Unidirectional and Bidirectional LSTM Models for Short-Term Traffic Prediction. 2021. doi:https://doi.org/10.1155/2021/5589075.

[40] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. doi:10.1109/78.650093.

[41] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *CoRR*, abs/1506.04214, 2015. URL: http://arxiv.org/abs/1506.04214, arXiv:1506.04214.

[42] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. The performance of lstm and bilstm in forecasting time series. In *2019 IEEE International Conference*

on Big Data (Big Data), pages 3285–3292, 2019. doi:10.1109/BigData47090.2019.9005997.

[43] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[44] Valentin Vielzeuf, Stéphane Pateux, and Frédéric Jurie. Temporal multimodal fusion for video emotion classification in the wild. *CoRR*, abs/1709.07200, 2017. URL: http://arxiv.org/abs/1709.07200, arXiv:1709.07200.

[45] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001.

[46] Paul Viola and Michael Jones. Robust real-time face detection. *International Journal of Computer Vision*, 2004.

[47] Yingchun Wang, Jingyi Wang, Weizhan Zhang, Yufeng Zhan, Song Guo, Qinghua Zheng, and Xuanyu Wang. A survey on deploying mobile deep learning applications: A systemic and technical perspective. *Digital Communications and Networks*, 8(1):1–17, 2022. URL: https://www.sciencedirect.com/science/article/pii/S2352864821000298, doi:https://doi.org/10.1016/j.dcan.2021.06.001.

[48] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014. URL: http://arxiv.org/abs/1411.1792, arXiv:1411.1792.

[49] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016. doi:10.1109/LSP.2016.2603342.

[50] Zhang, Wen and Deng, Lingfei and Zhang, Lei and Wu, Dongrui. A Survey on Negative Transfer, 2020. URL: https://arxiv.org/abs/2009.00909, doi:10.48550/ARXIV.2009.00909.

[51] Yanjia Zhu, Hongxiang Cai, Shuhan Zhang, Chenhao Wang, and Yichao Xiong. TinaFace: Strong but Simple Baseline for Face Detection. *CoRR*, abs/2011.13183, 2020. URL: https://arxiv.org/abs/2011.13183, arXiv:2011.13183.

[52] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning. *Proceedings of the IEEE*, 109(1):43–76, 2021. doi:10.1109/JPROC.2020.3004555.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original