University of St Andrews

Explore Sequential Relationships Between Facial Expressions in a Dynamical Situation

by Inthuch Therdchanakul

Student ID: 120022067

University of St Andrews

CS5099 Dissertation in Computer Science

Supervisor: Dr Juan Ye

Submitted 21st August 2018

Abstract

One of non-verbal method used to communicate emotional state of a person is the expression of face. Some of the application areas of automatic facial expression recognition (FER) are sociable robotics, medical treatment, driver fatigue surveillance, and many other human-computer interaction systems (Li & Deng 2018). The primary goal of this project is to build a model that is able to learn sequential relationship between facial expressions in video frames. Facial feature extraction techniques such as using pre-trained convolutional neural network (CNN) and facial action units (AU) detector are also explored. The EU-Emotion Stimulus Set (EESS) was provided by Autism research centre (arc). It contains various emotions and mental states that are represented through facial expressions, vocal expressions, body gestures and contextual social scenes (O'Reilly et al. 2016). For basic emotion classification, the best classifier had an accuracy of 99.69%. For complex emotion classification, the best classifier achieved an accuracy of 95.71%. Both of these classifiers use features extracted by a pre-trained CNN as inputs. The dissertation concludes with the discussion on applicability of the classifiers in real-time emotion recognition and potential future works.

Acknowledgements

I would like to thank my project supervisor Dr Juan Ye and Esma Mansouri Benssassi for providing support and guidance throughout the duration of the project. I would also like to thank Autism Research Centre for providing the dataset.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 9,574 words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Inthuch Therdchanakul
Signature

21/08/2018
Date

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

Automatic facial expression recognition (FER) has become an interesting and challenging research topic for the computer vision field (Kumari et al. 2015). Some of its application areas are sociable robotics, medical treatment, driver fatigue surveillance, and many other human-computer interaction systems (Li & Deng 2018). Facial expression is one of the most powerful signals for humans to communicate their emotional states and intentions (Li & Deng 2018). Basic emotions commonly used in FER systems are anger, disgust, fear, happiness, neutral, sadness, and surprise.

There are two main categories of FER systems. These categories are divided according to the feature representations: static image FER and dynamic sequence FER (Li & Deng 2018). The feature representation is encoded with only spatial information from the current image in static-based methods, whilst dynamic-based methods consider the temporal relation among adjacent frames in the input facial expression sequence (Li & Deng 2018).

The focus of this project is on inferring different emotional types (basic and complex emotions) using dynamic sequence FER. With the advances in computer vision and machine learning techniques, it is becoming more common to see that studies in various fields have begun to transfer to deep learning methods, which have achieved the state-of-the-art classification accuracy and exceeded previous results by a large margin (Li & Deng 2018). Specifically, Recurrent Neural Network (RNN) have proven to be capable of learning long-term dependencies. It possess the capability to perform well in a dynamic sequence FER system.

In 2015, Autism Research Centre (ARC) published a dataset called The EU-Emotion Stimulus Set (EESS) where various emotions and mental states are represented through facial expressions, vocal expressions, body gestures and contextual social scenes (O'Reilly et al. 2016). The primary goal of this project is to design and implement Recurrent Neural Network (RNN) to infer different emotional types from real-time video. In this case, the RNN will be used to enable sequential learning on consecutive video frames. The RNN will be trained on the videos from EESS dataset.

1.2 Problem Statement

The aim of this project is to build a dynamic sequence FER system that can detect emotion from video frames. In order to do this, a machine learning system needs to be designed, implemented and trained on the EESS dataset. Recurrent Neural Network (RNN) have proven to be effective in sequential learning tasks over the past decade. The hypothesis is that RNN should be able to outperform classifiers used in static image FER because its capability to learn long-term dependencies, which means that it can infer emotional types based on the relationship learned from dynamic sequence. Various steps must be taken in order to develop such a system including:

- Evaluate the dataset to identify what pre-processing techniques are required.
- Research and evaluate pre-trained Convolutional Neural Network model that can be used to perform feature extraction
- Research and evaluate tools that can be used extract facial action units (AU)
- Evaluate current RNN implementations and their performance in order to come up with an initial structure to start experimenting with configurations and parameters.
- Research and evaluate hyperparameters and network configurations for training the models
- Analyze and evaluate trained models, using appropriate error measures and a separate test set to determine the best hyperparameters and configurations

1.3 Dissertation Structure

Each section in this report contributes in some way to the final proposal and implementation of emotion recognition system. Each section in this report are summarized below.

Introduction This section briefly describes automatic facial expression recognition, introduces deep networks and explains the scope of this dissertation.

Background This section covers main aspects and the background information of methods and techniques used throughout the project as well as investigating current implementations and work.

Dataset This section analyses and describes the dataset used to train the models in detail. This section also covers data cleaning, handling insufficient training samples, and dataset splitting methodology.

Implementation In this section, various hyperparameters and network configurations are explored and evaluated in order to come to a selection of models that can be compare against each other in the Experiment and Results section.

Experiments and Results This section takes the models from the Implementation section and evaluates extensively using various techniques. Once evaluated, a final model for each classification task is selected.

Conclusion and Future Work This section concludes what had been done as well as identifying key learning aspects and determine if the aims and objectives have been satisfied. It also identifies potential extensions to the project.

2 Background

2.1 Facial Expression Recognition

One of non-verbal method used to communicate emotional state of a person is the expression of face. According to (Kumari et al. 2015), FER consists of five steps as illustrated in figure 1. The pre-processing step involves noise-removal/enhancement by taking image or image sequence as an input and outputs the face for further processing. Then, the facial components detection detects the ROI for eyes, nose, cheeks, mouth, eye brow, ear, fore-head,

etc (Kumari et al. 2015). After that, the feature extraction step handles the extraction of features from the Region of Interest (ROI). Some of the most popular feature extraction techniques are Gabor filters, Local Binary Patterns (LBP), Principal Component Analysis (PCA), Independent Component Analysis (ICA), Linear Discriminant Analysis (LDA), Local Gradient Code (LGC), and Local Directional Pattern (LDP) (Kumari et al. 2015). Finally, the classifier classifies the features into the respective facial expression classes. some of the most popular classification methods are SVM (Support Vector Machine) and NN (Nearest Neighbor) (Kumari et al. 2015). However, Li & Deng (2018) points out that these handcrafted features appear to be incapable of addressing the great diversity of factors unrelated to facial expressions.

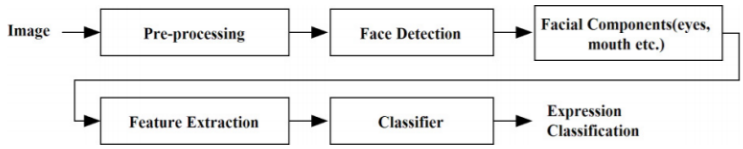


Figure 1: Facial expression classification steps (Kumari et al. 2015).

Recently with the emergence of new computer vision and machine learning techniques, specifically deep neural networks, feature extraction and detection is becoming more sophisticated, ideal for solving issues related to emotion recognition.

2.2 Machine Learning

Machine learning is the practice of programming computers so they can learn from data (Géron 2017). A machine learning system can learn patterns and trends using training algorithms. With the increase in computing power and more data become available, the use of machine learning methods across multiple domain including FER have drastically increased over the past decades because of their proven ability to solve classification problems.

2.3 Multilayer Neural Network

One of the most basic neural network architecture is the multilayer feed forward neural network (see figure 2). Inspired by the functionality of human brains, A typical multilayer feed forward neural network usually consists of an input layer, one or more hidden layers and an output layer (Russell & Norvig 2009). The input layer consists of neurons that represents the features or input data. Russell & Norvig (2009) explains that the input data passes through the neurons in the hidden layer and the weights and biases are used to modify these input values in order to produce a predicted output at the output layer.

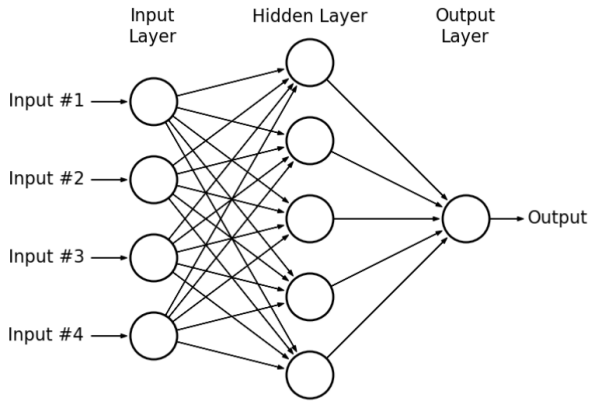


Figure 2: Multilayer feed forward neural network illustration (Vanderplas 2012).

2.4 Backpropagation

A neural network needs to be trained on training data before it can start generating useful predictions. Backpropagation is one of the most popular algorithms for training feed forward neural networks. The algorithm has 3 main step; forward propagation, backward propagation, and weight update. Training involves inputting data through the network and continuously modifying the weights based on error of the network prediction in comparison to the expected output.

An important hyperparameter used in backpropagation algorithm is learn-

ing rate. It determines the change in weights and biases during training. If the learning rate is too small, then the network will take a long time to converge. If the learning rate is too large, then the network will not reach the point where the error is minimised.

Each sample in the training data x_i is fed through the input layer where it will propagate forward through the network by computing the weighted sums s_j on incoming connections using equation 1 (Russell & Norvig 2009).

$$s_j = \sum_{i=0}^{n} w_{i,j} u_i \tag{1}$$

After that, an activation function g needs to be applied to the weighted sum in order to derive the output. This is shown in equation 2 (Russell & Norvig 2009).

$$u_j = g(s_j) = g(\sum_{i=0}^{n} w_{i,j} x_i) \tag{2}$$

Once the output is calculated, the network performs a backward pass by calculating the error value of all neurons using the following equation (Russell & Norvig 2009):

$$\delta_i = \begin{cases} (y_i - u_i) g'(s_i), & \text{if } u_i \text{ is an output neuron} \\ \sum\limits_{m:m>1}^{n} w_{m,i} \delta_m, & \text{otherwise} \end{cases} \tag{3}$$

Where $w_{m,i}$ is the weight values between neuron i and m, δ_i is the error for neuron i and y_i is the expected output. After calculating the errors, the weights are updated using the equation (Russell & Norvig 2009):

$$w^*_{i,j} = w_{i,j} + \alpha \delta_i u_j \tag{4}$$

Where α is the learning rate. These steps are repeated until specific terminating condition is met (e.g. maximum number of epochs reached, minimum error value reached, or validation error starts to increase).

2.5 Activation Functions

The purpose of an activation function is to determine the output of neural network. It can also introduce non-linearity to the model. This allows the

model to generalize or adapt with variety of data and to differentiate between the output (Sharma 2017). An example of activation function used in neural network is sigmoid. It takes a number as input and outputs a value in the range 0 to 1, the output value increases as the input approaches $\pm\infty$. This makes the sigmoid function suitable for modeling probability. There are other activation functions including hyperbolic tangent (tanh), rectified linear unit (ReLU), etc. The illustration of each activation function can be found in figure 3.

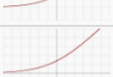
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figure 3: Activation functions (Sharma 2017).

2.6 Regularization

Regularization is a good way to reduce overfitting. Regularization typically involves constraining the weights of the model by penalizing large weights (Géron 2017). Some regularization techniques are discussed in the subsection below.

2.6.1 L1 and L2 Regularization

L1 and L2 regularization applies penalty to large weights. The key difference between L1 and L2 is the penalty term. L1 adds "absolute value of magnitude" of coefficient as penalty term to the loss function, whereas L2 adds "squared magnitude" of coefficient as penalty term to the loss function (Nagpal 2017). The loss function $C(y, \hat{y})$ becomes $C(y, \hat{y}) + \lambda \sum_{i=0}^{n} ||w||$, where w is the vector of model coefficients, $||.||$ is L1 or L2 norm and λ is a parameter that specifies the amount of regularization. More detailed explanation of L1 and L2 regularization can be found below.

L1 regularization (also referred to as lasso regression) adds penalty term $\lambda \sum_{i=1}^{n} |w_i|$ to the loss function. Less important features are forced to have coefficient 0. This can also be useful feature selection technique when there is a huge number of features (Nagpal 2017). The loss function with L1 regularization is shown in equation 5.

$$C^* = C + \lambda \sum_{i=1}^{n} |w_i| \tag{5}$$

L2 regularization (also referred to as ridge regression) adds the penalty term $\lambda \sum_{i=1}^{n} w_i^2$ to the loss function. This means the learning algorithm will not only fit the data but also keep the model weights as small as possible (Géron 2017). The loss function with L1 regularization is shown in equation 6.

$$C^* = C + \lambda \sum_{i=1}^{n} w_i^2 \tag{6}$$

2.6.2 Dropout

Dropout is a regularization technique developed by Srivastava et al. (2014). The main idea of this technique is to randomly turn off units (and their connections) from the neural network during training (Srivastava et al. 2014). This prevents the neural network from co-adapting too much and overfitting the data. Srivastava et al. (2014) states that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology. The results

from Srivastava et al. (2014) also shows that models with dropout applied performed better than models with L1 or L2 regularization. The illustration of dropout regularization is shown in figure 4.

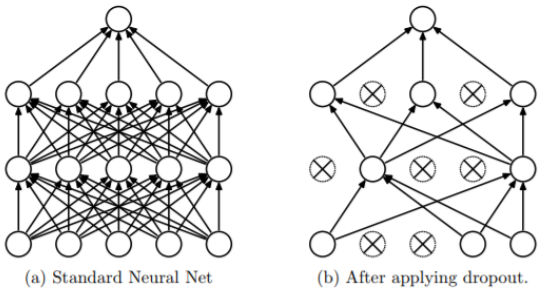


Figure 4: Neural network before and after applying dropout (Srivastava et al. 2014).

2.7 Performance Measures

When measuring the performance of a model, it is important that the loss function represents the nature of the problem. There are many types of loss functions, some of which are explained in the subsections below.

2.7.1 Mean Squared Error

Mean squared error (MSE) is an important criterion that is used to measure the performance of a model (TutorVista 2018). It can relay the concepts of precision, bias and accuracy during the statistical estimation (TutorVista 2018). The MSE can be formulated using equation below.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2 \tag{7}$$

Where y is the observed values and $\hat{y}$ is the predicted values.

2.7.2 Cross Entropy

Cross entropy measures the distance between two probability distributions. In classification task, it measures the distance between the probability dis-

tribution output by the model and the true distribution of the labels (Dahal 2017). By minimizing the distance between these two distributions, the model that minimizes cross entropy loss will output something as close as possible to the labels. Cross entropy loss can be calculated with the following:

$$H(y,p) = -\sum_{i} y_i log(p_i) \tag{8}$$

Where y is the observed values and p is the predicted values.

2.8 Gradient Descent Optimization

The purpose of gradient descent is to find the global minimum of a loss function. Initially, the neural network's weight matrices are filled with small random values. It can use the gradient to gradually adjusting these weights until it has a very low loss on its training data. This is when the network has learned to map its inputs to correct outputs (Chollet 2017*a*). When using gradient descent algorithm, it should be taken into consideration that the algorithm can converge to local minimum as illustrated in figure 5. Various optimizers have been proposed to resolve issues related to local minimum and optimize gradient decent. Some of the optimizers are described in the subsections below.

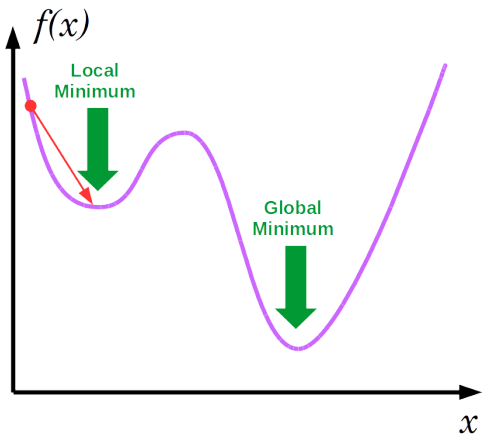


Figure 5: Gradient descent with local minimum and global minimum (Neural-network.io 2018).

2.8.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) picks a random sample in the training set and at every step and calculates the gradients based only on that single sample (Géron 2017). It is clear that this method is faster than calculating the gradients on the whole training set because it has very little data to compute at every epoch. It is also suitable for training on huge training set due to the fact that only one sample needs to be in memory at each epoch (Géron 2017).

When using SGD, the loss function tends to bounce up and down due to SGD's stochastic nature. The loss value will end up very close to the global minimum and continue to bounce around (see figure 6). This means the final weight values are good, but not optimal (Géron 2017).

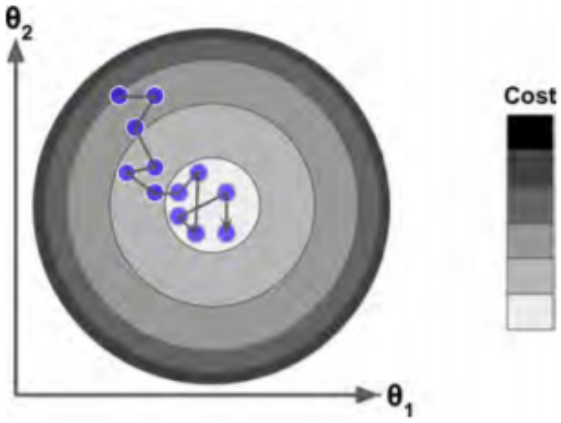


Figure 6: Stochastic Gradient Descent (Géron 2017).

2.8.2 RMSprop

It is difficult to choose a single global learning rate because the magnitude of the gradient can be very different for different weights and can change during training (Geoffrey Hinton 2014). One way to deal with this issue is to use the sign of the gradient and adapt the learning rate separately for each weight (Geoffrey Hinton 2014). RMSprop keeps a moving average of the squared gradient for each weight (see equation 9).

$$MeanSquare(w,t) = 0.9MeanSquare(w,t-1) + 0.1(\frac{\delta E}{\delta w}t)^2 \quad (9)$$

Geoffrey Hinton (2014) states that computing $\sqrt{MeanSquare(w,t)}$ makes the learning work much better.

2.8.3 Adam

Adam is an algorithm for first-order gradient-based optimization of stochastic cost functions, based on adaptive estimates of lower-order moments (Kingma & Ba 2014). It is often used to introduce adaptive learning rates for each parameter. The algorithm updates exponential moving averages of the gradient (m_t) and the squared gradient (v_t) with equation 10 and equation 11.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t \quad (10)$$

$$v_t = \beta_2 m_{t-1} + (1-\beta_2)g_t^2 \quad (11)$$

Where $\beta_1, \beta_2 \in [0,1)$ are exponential decay rates for the moment estimates and g_t is the gradients w.r.t. stochastic objective at timestep t (Kingma & Ba 2014). During the initial time steps, Adam update tends to be biased towards 0 as m_t and v_t are initialized as vectors of 0s. These biases are corrected by calculating bias-corrected first moment estimate (equation 12) and second (equation 13) moment estimates.

$$\hat{m_t} = \frac{m_t}{(1-\beta_1^t)} \quad (12)$$

$$\hat{v_t} = \frac{v_t}{(1-\beta_2^t)} \quad (13)$$

The parameters are updated with the following:

$$\theta_t = \theta_{t-1} - \alpha\frac{\hat{m_t}}{(\sqrt{\hat{v_t}}+\epsilon)} \quad (14)$$

Where α is the learning rate. Full Adam algorithm is shown in figure 7.

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 7: Adam algorithm (Kingma & Ba 2014).

2.9 Convolutional Neural Network

This section introduces Convolutional Neural Network (CNN), which in the last decade has become the most popular variation of a neural network when it comes to image classification tasks.

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

Figure 8: LeNet-5 architecture (Géron 2017).

The CNN is a variant of a neural network. The studies of visual cortex inspired the neocognitron, introduced in 1980, which gradually evolves into CNNs (Géron 2017). An important milestone is the work of LeCun et al. (1998), which introduced the LeNet-5 architecture (see figure 8), widely used to recognize handwritten check numbers. LeNet-5 introduces two new components of the neural network: convolutional layers and pooling layers.

2.9.1 Convolutional Layer

The purpose of convolutional layer is to learn feature representation of the input. It consists of filters alongside several feature maps. The units in a feature map are not connected to every single pixel in the input image (like in the fully connected networks), but only to pixels in their receptive fields (Géron 2017). This is illustrated in figure 9. This architecture allows the network to focus on low-level features in the previous hidden layer, then combine them into higher level features in the next hidden layer (Géron 2017). This structure is common in the images, which is why CNNs work so well for image recognition.

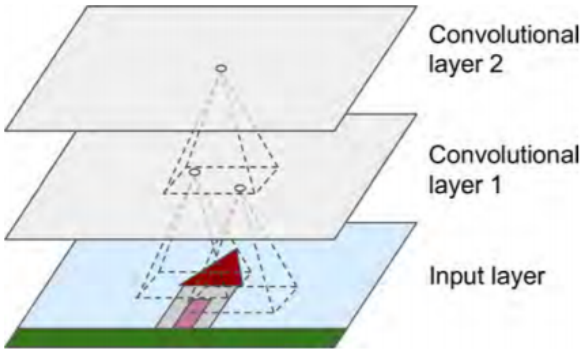


Figure 9: Convolutional layers with rectangular local receptive fields (Géron 2017).

2.9.2 Pooling Layer

The aim of a pooling layer is to reduce the spatial size of the inputs to minimise the computational load, the memory usage, and the number of parameters (thereby limiting the risk of overfitting) (Géron 2017). Similar

to convolutional layers, each unit in a pooling layer is partially connected with outputs of units in the previous layer. A pooling unit does not have weights. Its job is to aggregate the inputs using an aggregation function such as the max or mean (Géron 2017). An example of max pooling layer is shown in figure 10. It is common to add a pooling layer in between multiple convolution layers.

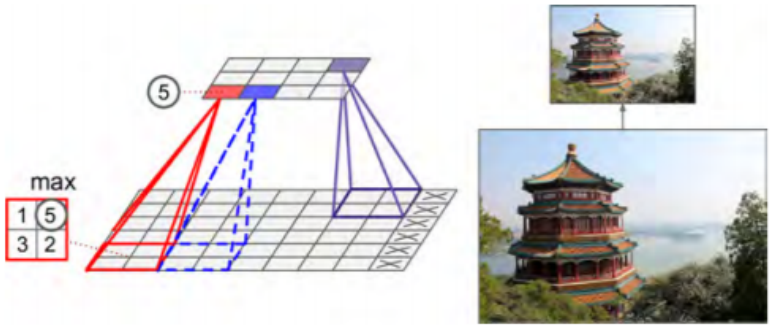


Figure 10: Max pooling layer (Géron 2017).

2.9.3 CNN Architectures

Typical CNN architectures stack a few convolutional layers (with ReLU activation function), then a pooling layer. This stacking of layers can repeat multiple times. The image gets smaller and deeper (more feature maps) as it progresses through the network (Géron 2017). A fully connected network is located at the top of the stack. Its purpose is to output the prediction (normally the output layer is a softmax layer that outputs estimated class probabilities). A typical CNN architecture is illustrated in figure 11.

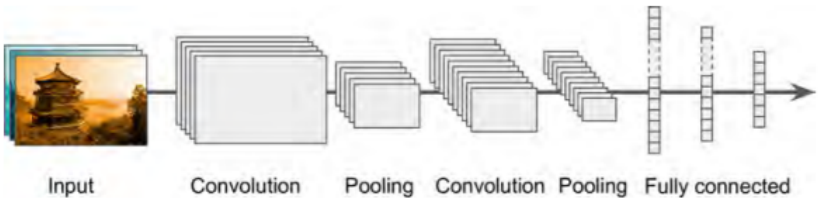


Figure 11: Typical CNN architecture (Géron 2017).

There are many CNN architectures developed in the past. Some example of the popular ones are AlexNet (2012), GoogLeNet (2014), and ResNet (2015). Those CNN architectures are shown in figure 12, 13, and 14.

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

Figure 12: AlexNet architecture (Géron 2017).

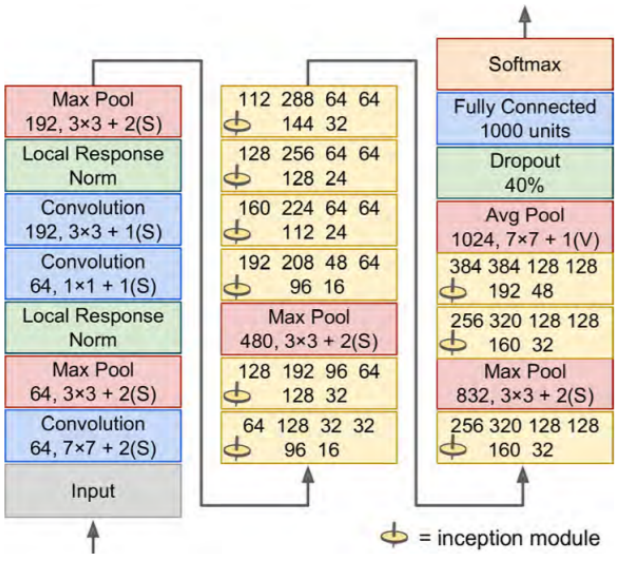


Figure 13: GoogLeNet architecture (Géron 2017).

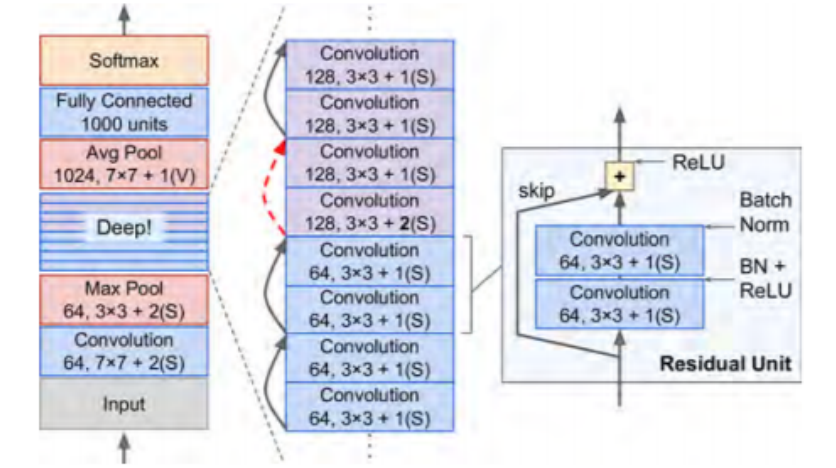


Figure 14: ResNet architecture (Géron 2017).

2.9.4 Feature Extraction with CNN

Chollet (2016) demonstrated that a pre-trained CNN can be used to perform feature extraction from images. He extracted the features from cats and dogs images with VGG16 (see figure 15) pre-trained on imagenet dataset. Then, these features are fed into a fully connected network and obtained 94% accuracy on the cats vs. dogs kaggle challenge. The feature extraction was done by initializing the model with everything up to the fully-connected layers. The features are the predictions of the model.

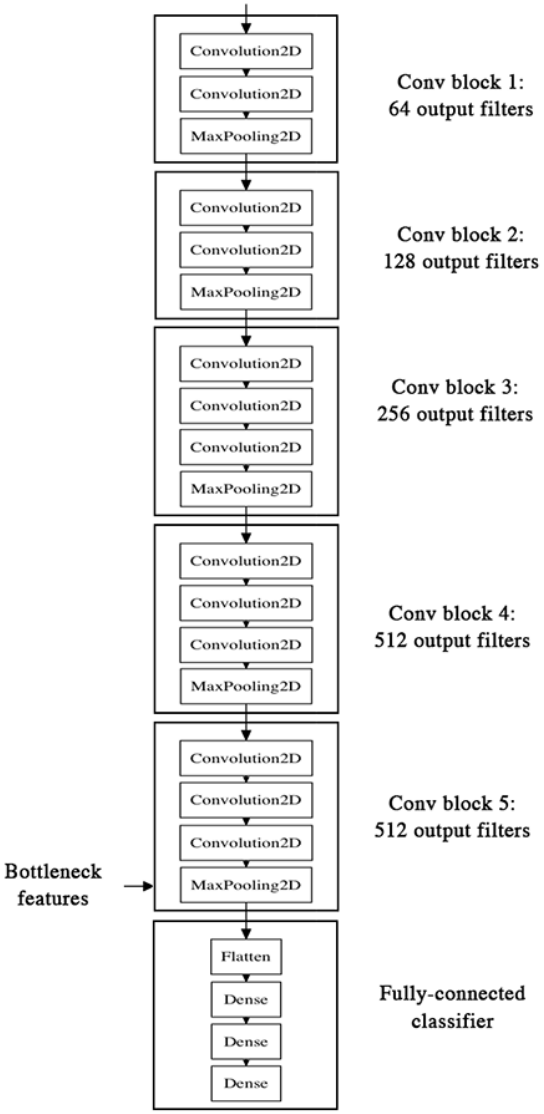


Figure 15: VGG16 architecture (Chollet 2016).

2.10 Facial Action Coding System

Facial Action Coding System (FACS) is a system designed to categorize human facial movements based on their appearance on the face (Baltrusaitis 2018*a*). FACS can be decomposed into Action Units (AU) that formed the expression. Examples of AU can be found in figure 16.

AU	Description	Facial muscle	Example image
1	Inner Brow Raiser	*Frontalis, pars medialis*	
2	Outer Brow Raiser	*Frontalis, pars lateralis*	
4	Brow Lowerer	*Corrugator supercilii, Depressor supercilii*	
5	Upper Lid Raiser	*Levator palpebrae superioris*	
6	Cheek Raiser	*Orbicularis oculi, pars orbitalis*	
7	Lid Tightener	*Orbicularis oculi, pars palpebralis*	
9	Nose Wrinkler	*Levator labii superioris alaquae nasi*	

Figure 16: Examples of AU (CMU n.d.).

A paper by Khorrami et al. (2015) shows that deep neural networks can learn AU when doing expression recognition. This finding is an indication that AU can be used as feature in FER systems.

2.11 Recurrent Neural Network

Traditionally, neural networks accept a fixed-sized vector as input and produce a fixed-sized vector as output. The issue with traditional networks is that the information from previous data points do not persist. Recurrent Neural Networks (RNN) can address the issue with their ability to operate over sequences of vectors (Karpathy 2015).

As illustrated in figure 17, RNN A, takes some input x_t and outputs a value h_t. The loop allows information to be passed on to the next step (see unrolled RNN in figure 17). This means that the outputs are influenced not only by the current input, but also on the entire history of inputs (Karpathy 2015). RNNs have been successfully applied to a variety of problems such as speech recognition, language modeling, translation and image captioning (Olah 2015).

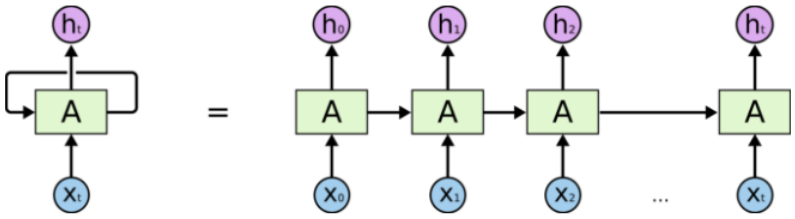


Figure 17: RNN illustration (Olah 2015).

2.11.1 Long Short Term Memory Networks

Long Short Term Memory Networks (LSTM) are a special kind of RNN proposed by Hochreiter & Schmidhuber (1997). It is capable of learning long-term dependencies (Olah 2015). This means that LSTM can perform sequential learning on consecutive video frames. This can be useful in real-time emotion recognition task because previous video frames might inform the understanding of the present frame.

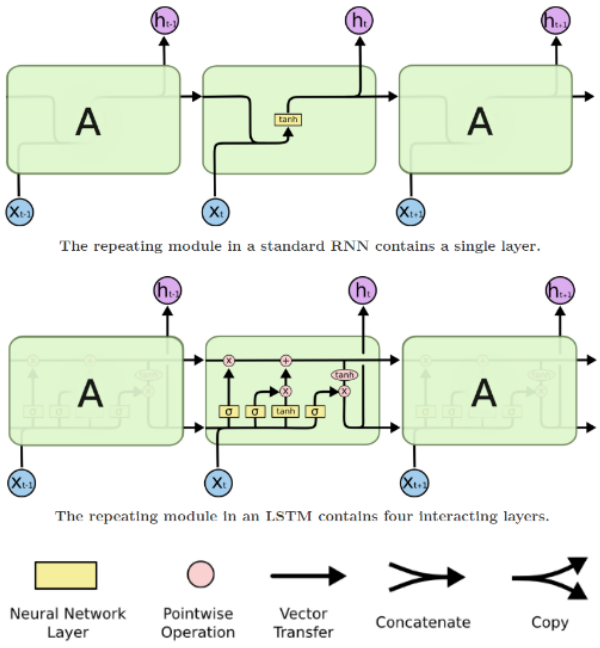


Figure 18: Comparison between standard RNN and LSTM (Olah 2015).

A comparison of standard RNN and LSTM can be found in figure 18. An important component of LSTMs is the cell state, the horizontal line running through the top of figure 19. LSTM can remove or add information to the cell state. This interaction is controlled by structures called gates (Olah 2015). Olah (2015) states that gates consist of a sigmoid neural network layer and a pointwise multiplication operation (see figure 20). The sigmoid neural network layer produces numbers between 0 and 1, indicating how much information should be let through (Olah 2015). In this case, a value of 0 means no information should be let through, whereas a value of 1 means let everything through. According to Olah (2015), there are three types of gates: an input gate, an output gate and a forget gate.

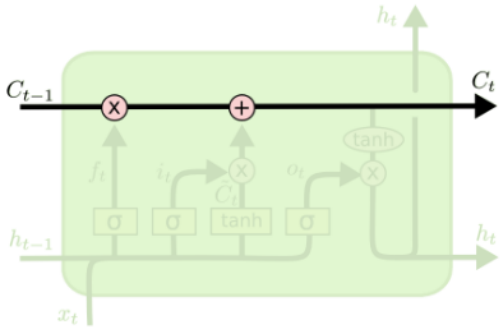


Figure 19: LSTM cell state (Olah 2015).

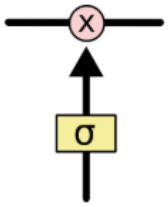


Figure 20: LSTM gate (Olah 2015).

The LSTM must first decide what information needs to be thrown away from the cell state (Olah 2015). This is determined by a sigmoid layer called "forget gate" (see figure 21). If forget gate outputs 0, then the LSTM will

get rid of this information. If forget gate outputs, then the LSTM will keep the information. The calculation at forget gate is shown in equation 15.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{15}$$

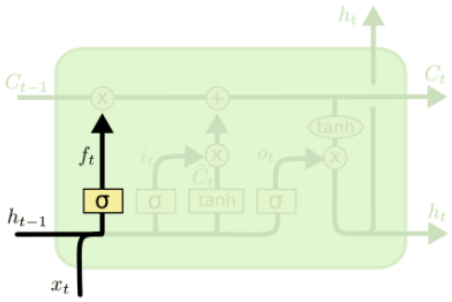


Figure 21: Forget gate used to decide what information needs to be thrown away from the cell state (Olah 2015).

As shown in figure 22, the LSTM decides what new information will be stored in the cell state (Olah 2015). This step consists of two parts. First, a sigmoid layer called the "input gate" determines which values need to be updated (see equation 16). Second, a tanh layer outputs a vector of new candidate values $\widetilde{C}_t$ (see equation 17). These are the values that can be added to the state (Olah 2015).

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{16}$$

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{17}$$

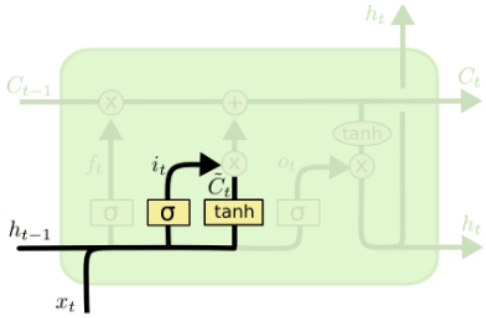


Figure 22: Input gate used to decide what new information will be stored in the cell state (Olah 2015).

Once the values from the two parts are calculated, equation 18 can be used to update old cell state C_{t-1} into the new cell state C_t. This computation step is shown in figure 23.

$$C_t = f_t C_{t-1} + i_t \widetilde{C}_t \tag{18}$$

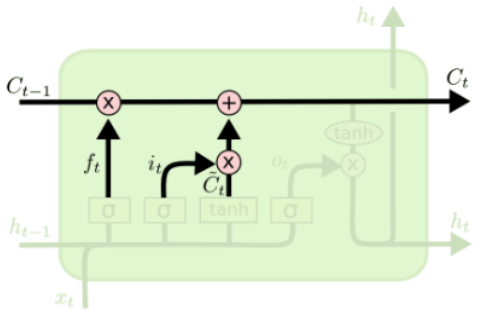


Figure 23: Computing state update (Olah 2015).

The last step is to determine the output. Olah (2015) points out that the output is a filtered version of the cell state. This step consists of two parts. First, a sigmoid layer called the "output gate" decides what parts of the cell state to output (see equation 19). Second, the cell state is put through tanh and multiply it by the output of the output gate (see equation 20). This computation step is illustrated in figure 24.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{19}$$

$$h_t = o_t * tanh(C_t) \tag{20}$$

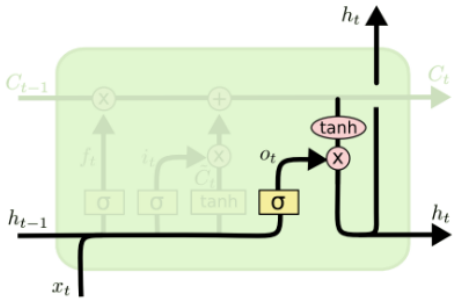


Figure 24: Computing output of LSTM (Olah 2015).

3 Dataset

The EU-Emotion Stimulus Set (EESS) was provided by Autism research centre (arc). It consists of various emotions and mental states that are represented through facial expressions, vocal expressions, body gestures and contextual social scenes (O'Reilly et al. 2016). This dataset is chosen because it is (to the author's knowledge) the only publicly available dataset that contains complex emotion samples. This project uses videos that show various emotions through facial expressions from the EESS dataset (see figure 25).

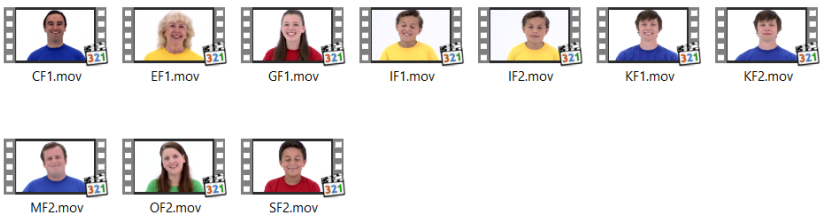


Figure 25: Videos in EESS dataset.

Within this dataset, there are 7 basic and 27 complex emotion labels. These are summarized in table 1. The actor demographics can be found in figure 26.

Emotion	Labels	Environment
Basic	Angry, Disgust, Fear, Happy, Neutral, Sad, Surprise	Lab
Complex	Afraid, Afraid Low Intensity, Angry, Angry Low Intensity, Ashamed, Bored, Disappointed, Disgusted, Disgusted Low Intensity, Excited, Frustrated, Happy, Happy Low Intensity, Hurt, Interested, Jealous, Joking, Kind, Neutral, Proud, Sad, Sad Low Intensity, Sneaky, Surprised, Surprised Low Intensity, Unfriendly, Worried	Lab

Table 1: EESS emotion labels

Table 1 Actor demographics and modalities completed

Actor	Gender	Age	Ethnicity	Facial Expressions	Body Gestures	Social Scenes
A	Female	19	Mixed Afro-Caribbean/Asian British	X	✓	✓
B	Female	37	White British	✓	✓	✓
C	Male	31	White British	✓	✓	✓
D	Female	27	Black British	✓	✓	✓
E	Female	70	White British	✓	✓	✓
F	Female	10	Black British	✓	✓	✓
G	Female	15	White British	✓	✓	✓
H	Male	62	White British	✓	✓	✓
I	Male	11	White British	✓	X	✓
J	Male	12	Mixed White/Asian British	✓	X	✓
K	Male	30	White British	✓	X	✓
L	Male	12	White British	✓	X	✓
M	Male	37	White British	✓	X	✓
N	Female	42	White British	✓	X	✓
O	Female	21	White British	✓	X	✓
P	Male	12	White British	✓	X	✓
Q	Female	10	Mixed White/Asian British	X	X	✓
R	Female	11	White British	✓	X	✓
S	Male	11	Mixed Mediterranean/Asian British	✓	X	✓

Figure 26: Actor demographics and modalities completed (O'Reilly et al. 2016).

3.1 Cleaning Data

There are some video frames where the actor has neutral expression but the frames are labeled as something else. This makes it impossible for the classifiers to recognize neutral expression. Therefore, these frames has to be deleted first. There are 797 and 1174 video frames deleted for basic emotion part and complex emotion part respectively. Some examples of these video frames are shown in figure 27.

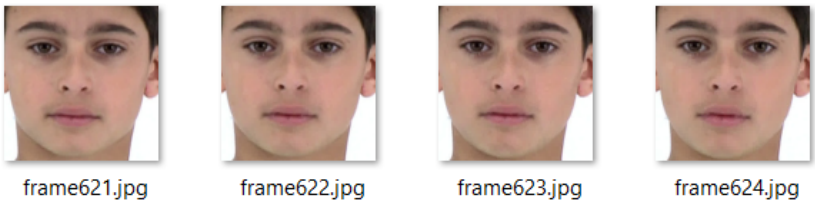


Figure 27: Neutral facial expression labeled as 'Angry'.

3.2 Insufficient Training Samples

When the video frames are converted to sequential data, the number of sample reduces. This means the sequential classifiers like LSTM will have worse performance than other classifiers like SVM due to having insufficient training samples. The EESS dataset is affected by this problem since each video only last around 10 seconds. One way to deal with this issue is to have overlapping between sequences. This means a sequence of video frames will contain some of the frames from the previous sequence. The illustration of overlapping frames is shown in figure 28. This project use 90% overlapping frames in order to maximize the number of training samples. This means that a sequence of video frames will include 90% of the frames from the previous sequence.

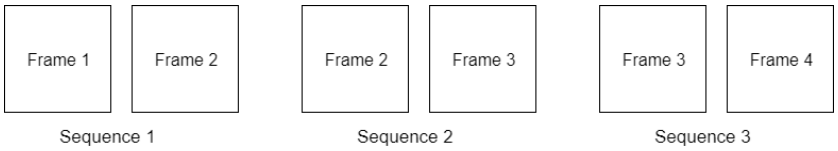


Figure 28: Maximizing number of training samples with overlapping video frames.

3.3 Splitting into Subsets

For the LSTM models, the full dataset was split into training/validation/test sets with a ratio of 80%/10%/10% respectively. For baseline SVM models, the full dataset was split into training/test sets with a ratio of 80%/20% respectively. This set up applies to both basic emotion classification and complex emotion classification tasks.

4 Implementation

The goal of this project is to build a model that is able to learn sequential relationship between facial expressions in video frames. If a system is to be considered a success, it must be able classify emotions at a higher level of accuracy than that of a static image classifier. In this project, Support Vector Machine (SVM) is the chosen static image classifier used to obtain baseline accuracy. It is selected due to its success in Michel & El Kaliouby (2003). They managed to achieve 86% accuracy with SVM. Therefore, the sequential model has to achieve higher accuracy in order to confirm the theory that sequential FER systems are the superior ones. In this section, various training parameters and network configurations are explored and evaluated in order to obtain a selection models to test in the final model selection stage.

4.1 Resources

This section covers all software and hardware components that were used to create and train the LSTMs in order to classify emotions. These include hardware specifications, third party libraries and software that were used and their purpose.

4.1.1 Software and Libraries

Due to time constraints, LSTM models cannot be built from scratch. Therefore, third party libraries were used to enable the construction of LSTMs and training within the time provided for this dissertation. The chosen language was Python. This is because Python has wide variety of deep learning libraries available. The purpose of each software and library is explained in more detail below.

1. **FFmpeg:** The classifiers accept features extracted from video frames as inputs. This software is used to extract frames from videos.

2. **OpenCV:** The nature of the problem involves processing video frames and classifying them. OpenCV is an open source computer vision and machine learning library that allows the task of video frame processing to be completed with relative ease.

3. **Dlib:** In order to achieve the best performance from the classifiers, the faces of the actors in EESS dataset need to be cropped before performing facial feature extraction. Dlib made this possible with its frontal face detector model.

4. **Sklearn:** This is one of the most popular Python machine learning library. Sklearn package provides several classification algorithms including SVM. This allows training of baseline models to be performed with ease. It also contains many useful utility functions that assist in machine learning tasks such as confusion matrix, train test split, cross validation, etc.

5. **OpenFace:** OpenFace is a facial behavior analysis toolkit developed by Baltrusaitis (2018*b*). It allows Facial Action Units (AU) extraction to be performed with ease.

6. **Tensorflow:** TensorFlow (2018) is a Python library that makes it possible to define, optimize, and evaluate tensor operations efficiently. It serves as backend in order to build a machine learning model, or in the case of this dissertation, a deep learning model. The library also provides the ability to train the models on GPU in order to minimize training time.

7. **Keras:** This library is chosen due to its user friendliness. Keras is a Python high-level neural networks API (Keras 2018*a*). It is can run on top of TensorFlow, CNTK, or Theano (Chollet 2017*a*). It also contains several pre-trained CNNs which can be used to perform feature extraction. In this project, Keras uses Tensorflow as backend.

4.1.2 Hardware

In order to create a high performance model, various LSTM networks were created with different hyperparameters and configurations. The nature of this project involves processing huge amount of data in the video frames, which could take a very long time if training was done on a CPU. For this reason, the LSTMs were trained on a NVIDIA GeForce 940 MX GPU with 2GB GDDR5 graphics memory. This allows more LSTM architectures to be tested within the time provided for this project.

4.2 Video Frame Extraction

As mentioned before, classifiers accept features extracted from video frames as inputs. Therefore, the frames have to be extracted using FFmpeg. Once the frames are extracted, the face from each frame needs to be cropped in order to ensure that only meaningful features are extracted when performing feature extraction. This was done with the help of OpenCV and dlib frontal face detector. The whole process is illustrated in figure 29.

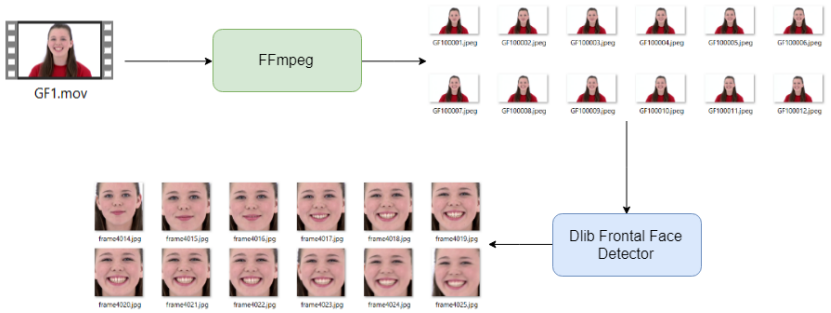


Figure 29: Video pre-processing pipelines.

4.3 Feature Extraction

In this project, two feature extraction methods were implemented. The first method uses pre-trained CNN called VGG16. It was trained on imagenet dataset. VGG16 was chosen because of high classification accuracy achieved by Chollet (2016) as discussed in section 2.9.4. Another feature extraction method involves extracting AU from sequences of video frames. This was done using OpenFace, a facial behavior analysis toolkit developed by Baltrusaitis (2018*b*). The two feature extraction methods are described in more detail in the subsections below.

4.3.1 VGG16

For features used in LSTM networks, the first step is to initialize VGG16 pre-trained on imagenet dataset with fully connected layers removed (see figure 30). After that, the video frames are fed through the convolutional base of the VGG16 model. It outputs vectors of feature for each frame. Then, these vectors are built into sequences and saved to the local disk. This allows the classifiers to load the data without having to repeat the feature extraction process again. The extracted features have the shape (samples, sequence length, 3, 3, 512). Since LSTM excepts 3-dimensional vectors as inputs, the sequences are flatten to (samples, sequence length, 4608). The feature extraction process is illustrated in figure 31.

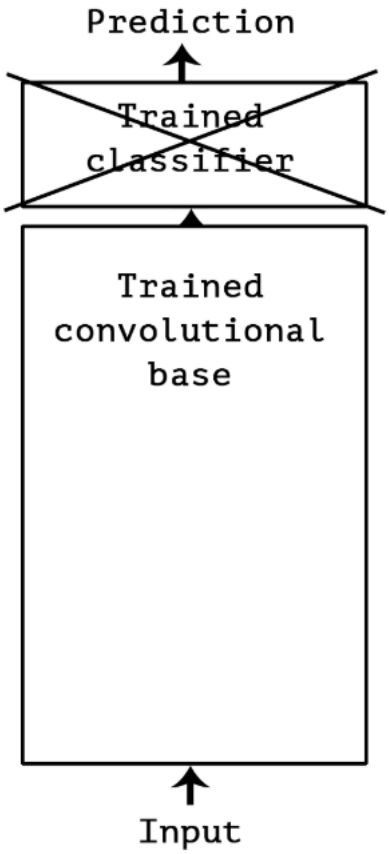


Figure 30: Initializing a pre-trained CNN with fully connected layers removed (Chollet 2017*b*).

Figure 31: Feature extraction with VGG16.

The feature extraction process for baseline SVM models is similar to the process described above. The only different is that the feature vectors are not built into sequences. The extracted features have the shape (samples, 3,

3, 512). Since SVM excepts 2-dimensional vectors as inputs, the sequences are flatten to (samples, 4608).

4.3.2 Facial Action Units

For features used in LSTM models, the first step is to pass video frame sequences into OpenFace (see figure 32). Then, OpenFace outputs AU intensities of each frame as CSV files. The AU information in CSV files built into sequences and saved in the local disk in order for the classifiers to load the AU data without having to repeat the AU extraction process again. The extracted features have the shape (samples, sequence length, 17). The number 17 is the number of AU that OpenFace can extract. The subset of AU that OpenFace can recognize are: 1, 2, 4, 5, 6, 7, 9, 10, 12, 14, 15, 17, 20, 23, 25, 26, 28, and 45 (Baltrusaitis 2018*a*). The detail of each AU can be found in table 2. The AU extraction process is illustrated in figure 33.

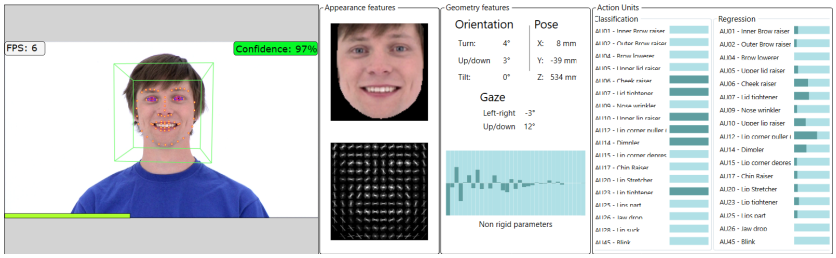


Figure 32: AU extraction with OpenFace.

AU	Description	Facial muscle
1	Inner Brow Raiser	Frontalis, pars
2	Outer Brow Raiser	Frontalis, pars lateralis
4	Brow Lowerer	Corrugator supercilii, Depressor supercilii
5	Upper Lid Raiser	Levator palpebrae superioris
6	Cheek Raiser	Orbicularis oculi, pars orbitalis
7	Lid Tightener	Orbicularis oculi, pars palpebralis
9	Nose Wrinkler	Levator labii superioris alaquae nasi
10	Upper Lip Raiser	Levator labii superioris
12	Lip Corner Puller	Zygomaticus major
14	Dimpler	Buccinator
15	Lip Corner Depressor	Depressor anguli oris (a.k.a. Triangularis)
17	Chin Raiser	Mentalis
20	Lip stretcher	Risorius w/ platysma
23	Lip Tightener	Orbicularis oris
25	Lips part	Depressor labii inferioris or relaxation of Mentalis, or Orbicularis oris
26	Jaw Drop	Masseter, relaxed Temporalis and internal Pterygoid
28	Lip Suck	Orbicularis oris
45	Blink	Relaxation of Levator palpebrae superioris; Orbicularis oculi, pars palpebralis

Table 2: Subset of AU that OpenFace can recognize (CMU n.d.).

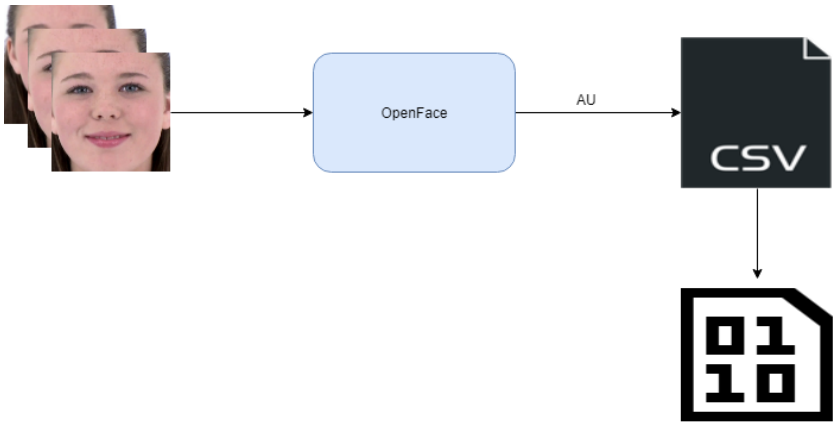


Figure 33: AU extraction with pipeline.

The AU extraction process for baseline SVM models is similar that of LSTM models. The only different is that the AU vectors are not built into sequences. The extracted AUs have the shape (samples, 1, 17). Since SVM excepts 2-dimensional vectors as inputs, the sequences are flatten to (samples, 17).

4.4 Baseline Models

SVM models used to obtain baseline accuracy are shown in figure 34. Model SVM+VGG16 accepts features extracted from VGG16 as inputs, whereas model SVM+AU accepts AUs extracted with OpenFace as inputs. Both models use linear kernel. Here, the validation accuracy is obtained by evaluating the models with 10-fold cross validation. Then, the models make predictions on the test set to obtain test accuracy.

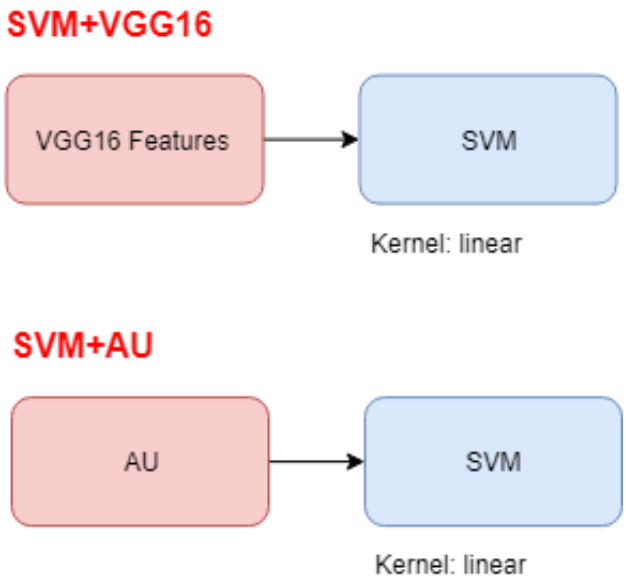


Figure 34: Baseline SVM models.

4.4.1 Basic Emotion

Model	Validation	Test
SVM+VGG16	98.51%	98.61%
SVM+AU	52.34%	53.43%

Table 3: Basic emotion baseline accuracy of SVM models

The basic emotion classification results of both SVM models can be found in table 3. These are the scores that sequential models have to beat. It is very surprising to see SVM+VGG16 achieved such a high accuracy. It surpasses the 86% accuracy achieved by Michel & El Kaliouby (2003). The results

indicate that features extracted from pre-trained VGG16 are better than AU features. The confusion matrices are shown in figure 35 and figure 36.

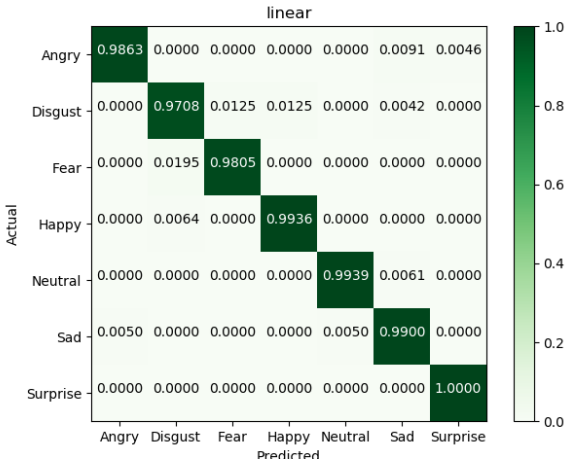


Figure 35: Confusion matrix of SVM+VGG16 (basic emotions).

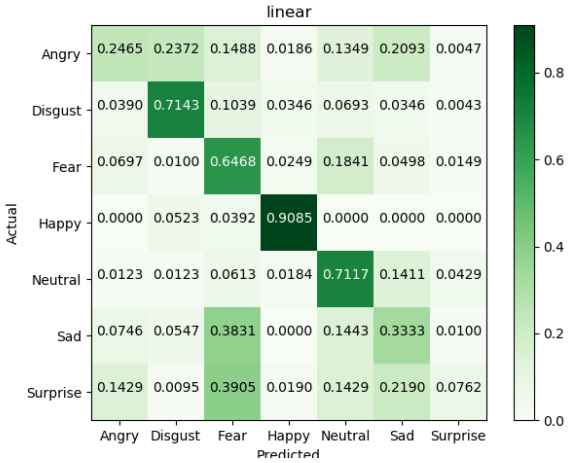


Figure 36: Confusion matrix of SVM+AU (basic emotions).

4.4.2 Complex Emotion

Model	Validation	Test
SVM+VGG16	88.63%	90.07%
SVM+AU	23.10%	21.26%

Table 4: Complex emotion baseline accuracy of SVM models

The complex emotion classification results of both SVM models can be found in table 4. Model SVM+VGG16. This is expected because the complex emotion has 27 emotion categories. Some of the emotions such as afraid, sad, and happy have low intensity version of themselves as well. This makes it even harder for SVM to distinguish between these emotions. There is no confusion matrix for complex emotion classification task because matrix of shape (27 x 27) is too big to plot.

4.5 Basic Emotion Models

As discussed in the previous sections, the sequential model that will be explored is a special kind of Recurrent Neural Network (RNN) called Long Short Term Memory network (LSTM) because capable of learning long-term dependencies.

4.5.1 Parameter Selection

There are many parameters that need to be optimized in order to create a classifier for emotion recognition. Various parameters that were explored are:

1. Number of layers and layer sizes

2. Activation functions
3. Regularization
4. Optimizer
5. Loss functions
6. Epochs
7. sequence length

Number of layers and layer sizes Training a model can take a long time. Therefore, only limited number of layers and layer sizes can be explored. After experimenting with a few number of layers, it appears that one layer is enough for basic emotion classification task. The number of hidden units explored are ranging from 32 to 512.

Activation functions Since LSTM does not suffer from the vanishing gradient problem because of its gate architecture, the default activation function (tanh) is left unchanged. The activation function at output layer is softmax function. This means that the network will output a probability distribution over seven emotion classes i.e. for every input sample, output vector, where output[i] is the probability that the sample belong to class i. The seven scores will sum up to 1.

Regularization As discussed in section 2.6.2, dropout can help reduce overfitting and improve the performance of deep neural networks. Dropout value of 0.5 is chosen.

Optimizer The models that adam optimizer in the video classification tasks of Harvey (2016) and Harvey (2017) achieved high accuracy. Therefore, adam optimizer is selected.

Loss functions Since this is a multiclass classification task, the best loss function to use in this case is categorical cross entropy. It measures the distance between two probability distributions. In this case, it measures the distance between the probability distribution output by the network and the

true distribution of the labels. By minimizing the distance between these two distributions, the network will output something as close as possible to the labels.

Epochs In this task, the number of epoch is 250. This number is chosen so that the models can be trained within the time provided. ModelCheckpoint callback is also used to ensure that the model with the lowest validation loss is saved at the end of each training session (Keras 2018*b*).

Sequence length This is the number of video frames in each training sample. The chosen sequence length is 2. It appears that longer sequence length results in lower classification accuracy. This happens because longer sequence means having less training samples. In the future, longer sequence length can be experimented on larger dataset.

4.5.2 Model Architectures

Based on the parameter choices in section 4.5.1, the model architectures are shown in figure 37 and figure 38.

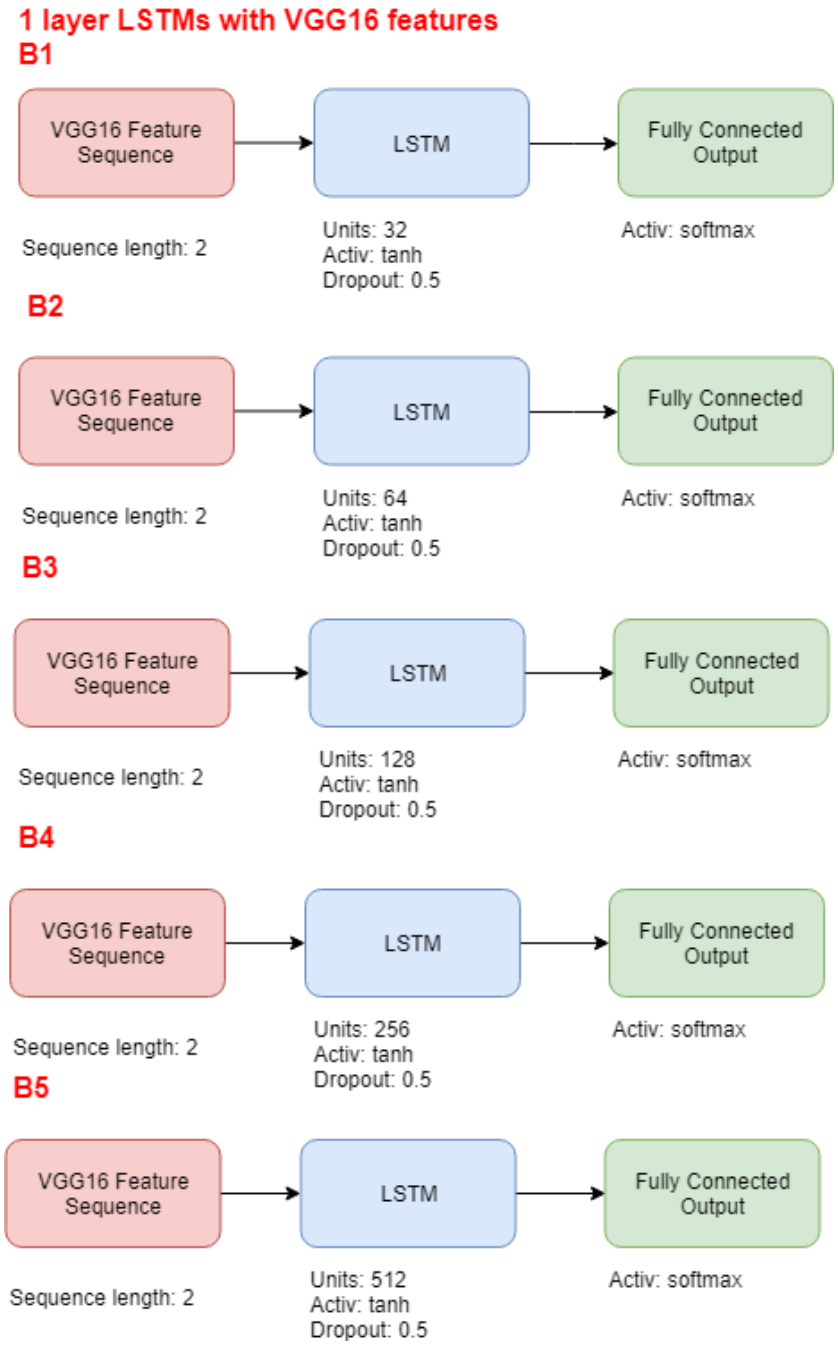


Figure 37: Basic emotion models with VGG16 features.

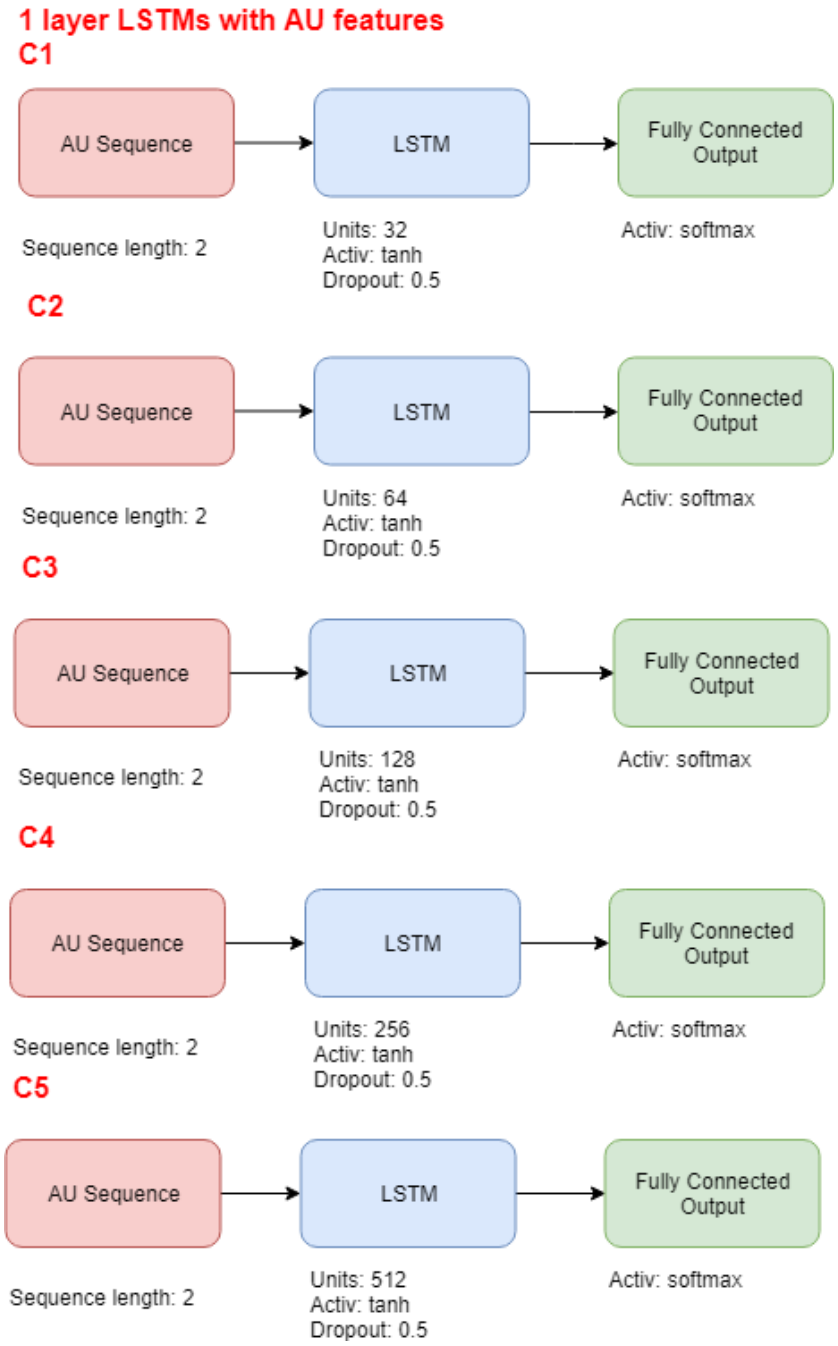


Figure 38: Basic emotion models with AU features.

4.5.3 Chosen Models

Model	Validation loss	Validation accuracy
B1	0.0233	100%
B2	0.0128	100%
B3	0.0129	100%
B4	**0.0054**	**100%**
B5	0.0109	100%

Table 5: Basic emotions classification performance of models with VGG16 features.

Model	Validation loss	Validation accuracy
C1	0.3705	88.17%
C2	0.2391	92.01%
C3	0.1676	94.88%
C4	0.1133	95.68%
C5	**0.0951**	**96.80%**

Table 6: Basic emotions classification performance of models with AU features.

In this section, all model architectures from section 4.5.2 are trained on the training set and evaluated on the validation set. The performance of each model are shown in table 5 and table 6. The models with lowest validation loss are chosen to be evaluated on the test set in section 5. The results indicate that bigger layer size can lead to higher accuracy of the networks. However, it is not possible to investigate more network configurations within the time provided for this project. In this case, the chosen basic emotion models are model B4 and C5.

4.6 Complex Emotion Models

4.6.1 Parameter Selection

For complex emotion classification task, the model parameters are the same as the ones in section 4.5.1 except the number of layers. The number of layers is increased to 2 in order to model more complex relationship in this task. The selection is based on the result shown in figure 39.

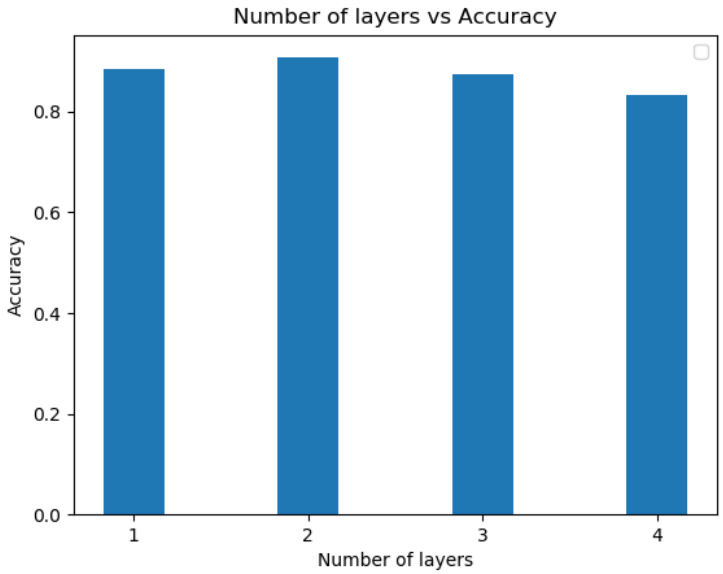


Figure 39: Number of layers vs. Validation accuracy.

4.6.2 Model Architectures

Based on the parameter choices in section 4.6.1, the model architectures are shown in figure 40 and figure 41.

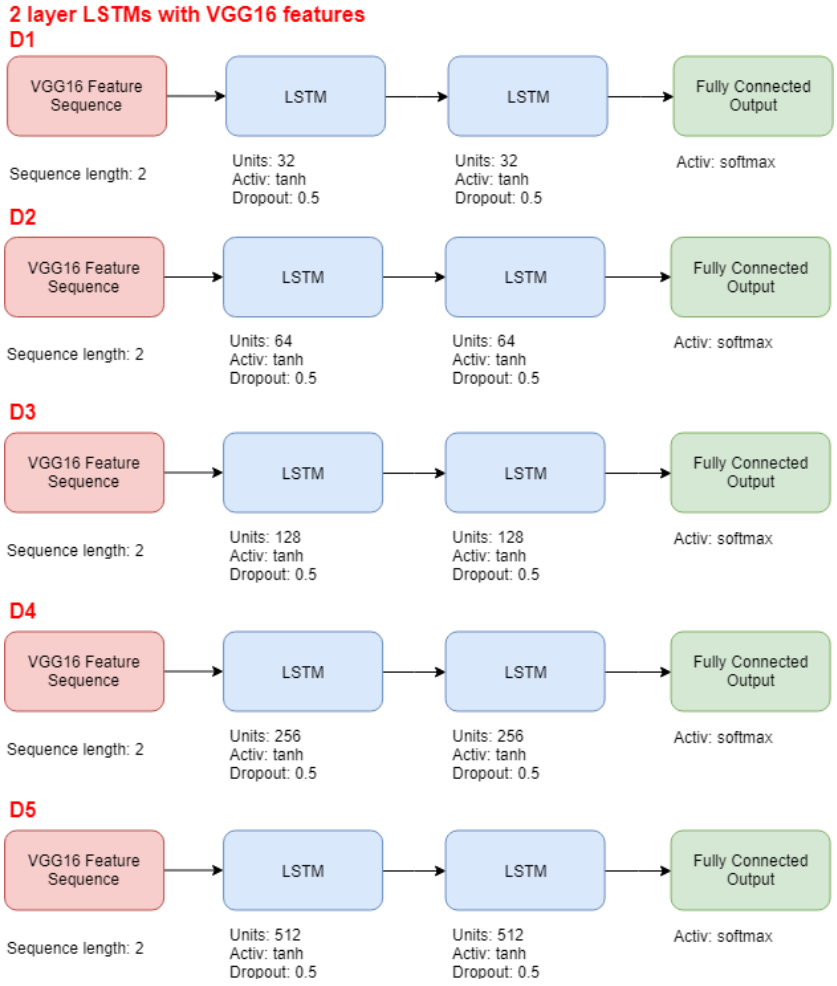


Figure 40: Complex emotion models with VGG16 features.

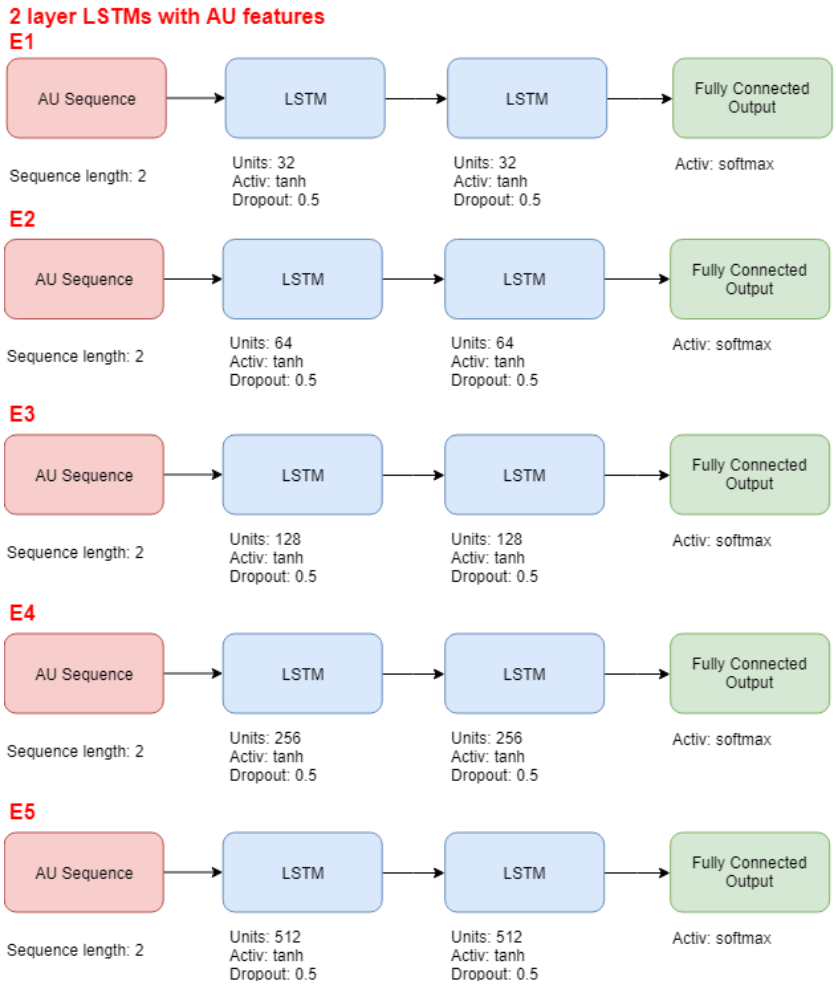


Figure 41: Complex emotion models with AU features.

4.6.3 Chosen Models

In this section, all model architectures from section 4.6.2 are trained on the training set and evaluated on the validation set. The performance of each model are shown in table 7 and table 8. The models with lowest validation loss are chosen to be evaluated on the test set in section 5. These results indicate that bigger layer size can increase the accuracy of the networks. Due to time constraints, it is not possible to investigate more network configurations. In this case, the chosen complex emotion models are model D5 and E5.

Model	Validation loss	Validation accuracy
D1	0.3079	90.76%
D2	0.2510	92.57%
D3	0.1592	95.26%
D4	0.1420	95.80%
D5	**0.1283**	**95.93%**

Table 7: Complex emotions classification performance of models with VGG16 features.

Model	Validation loss	Validation accuracy
E1	1.7742	44.52%
E2	1.1932	62.37%
E3	0.6547	79.64%
E4	0.3643	88.60%
E5	**0.3292**	**91.43%**

Table 8: Complex emotions classification performance of models with AU features.

5 Experiments and Results

In this section, the network configurations and parameters chosen in the previous section are experimented using the full training dataset.The validation loss and validation accuracy is calculated using a separate validation set at the end of each epoch. The weights are also saved to the hard drive when the validation loss improves. Once training had completed, the set of weights that generates the lowest validation loss is used as the final weights for that model. This is based on the principle that as soon as the validation loss started to increase, it is very likely that the network is beginning to overfit the training data.

5.1 Experiments with EESS Dataset

5.1.1 Basic Emotion Classification

frame	face_id	timestamp	confidenc	success	AU01_r	AU02_r	AU04_r	AU05_r	AU06_r	AU07_r	AU09_r	AU10_r	AU12_r	AU14_r	AU15_r	AU17_r
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0.46	0	0.21	1.22	0.17	0.28	0	0	0.09	0.33

Figure 42: Unsuccessful AU extraction by OpenFace.

Basic emotion classification results are shown in table 9. The first model trained on the training set is model B4. It is a 1-layer LSTM network with 256 hidden units that takes VGG16 features as inputs, and outputs the predictions through its fully connected softmax layer. It managed to achieve the accuracy score of 99.69% on the test set, which is higher than the scores

achieved by the baseline SVM models. The second model is model C5. It is a 1-layer LSTM network with 512 hidden units that takes AU features as inputs, and outputs the predictions through its fully connected softmax layer. It achieved 96.81% accuracy on the test set. It is higher than the baseline SVM model that uses the same features as inputs. However, model B4 and SVM+VGG16 managed to obtain slightly higher accuracy scores. The results indicate that features extracted by pre-trained VGG16 tend to achieve better results than AU. One explanation for this is that VGG16 can extract more detailed features because it was trained on the imagenet dataset, which contains hundreds and thousands of images. However, OpenFace can only recognize a subset of AU and it cannot extract AU from some video frames. Some examples of failed AU extraction are shown in figure 42. Therefore, it is not possible to conclude that pre-trained CNN feature extraction method will always achieve higher accuracy than the AU. From these results, it is clear that model B4 is the best basic emotion classification model. Therefore, it is chosen as the final model for basic emotion classification task. The confusion matrices of both LSTM models are shown in figure 43 and figure 44.

Model	**Test accuracy**
SVM+VGG16	98.61%
SVM+AU	53.43%
B4	**99.69%**
C5	96.81%

Table 9: Basic emotions classification results

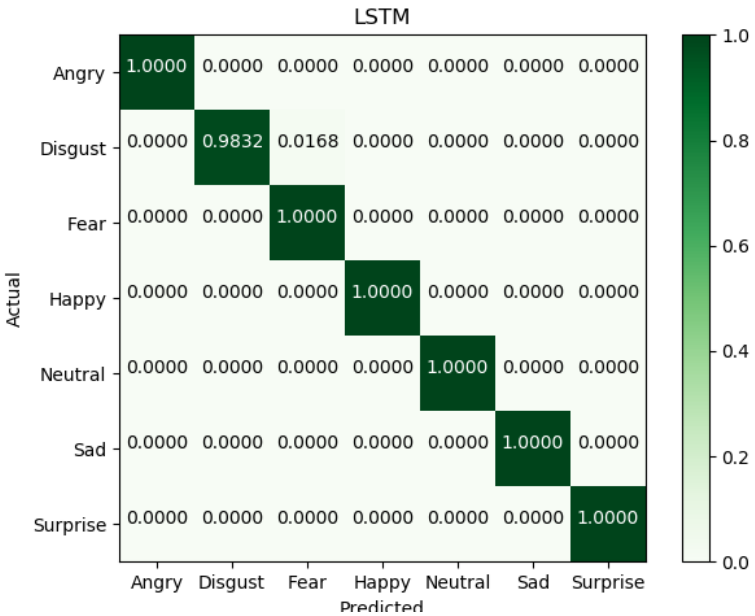


Figure 43: Confusion matrix of model B4.

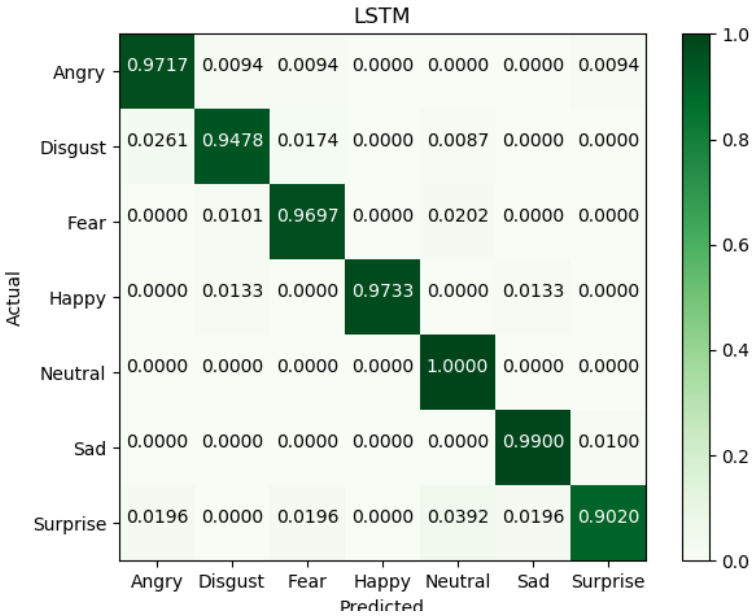


Figure 44: Confusion matrix of model C5.

5.1.2 Complex Emotion Classification

Complex emotion classification results are shown in table 10. The first model trained on the training set is model D5. It is a 2-layer LSTM network with 512 hidden units that takes features extracted by pre-trained VGG16 as inputs, and outputs the predictions through its fully connected softmax layer. Similar to the basic emotion classification task, the LSTM model that use VGG16 features outperforms all other models. It achieved 95.71% accuracy on the test set. The second model is model E5. It is a 2-layer LSTM network with 512 hidden units that takes AU features as inputs, and outputs the predictions through its fully connected softmax layer. The accuracy achieved by this model is 91.18%. The results indicate that the LSTM models always perform better than the baseline SVM models when they use the same type of feature as inputs. This is because LSTMs also consider information from previous video frames when making prediction, whereas SVMs make predictions without any knowledge about the past history (it starts thinking from scratch). It is also worth mentioning that the accuracies achieved by both LSTM models are higher than 47% accuracy achieved by the Euclidean distance-based classifier in the paper by Adams & Robinson (2015). This means that recurrent neural networks are likely to produce good results when used in video classification tasks. From these results, it is clear that model D5 is the best complex emotion classification model. Therefore, it is chosen as the final model for complex emotion classification task.

Model	Test accuracy
SVM+VGG16	90.07%
SVM+AU	21.26%
D5	**95.71%**
E5	91.18%

Table 10: Complex emotions classification results

5.2 Experiments with Game Data

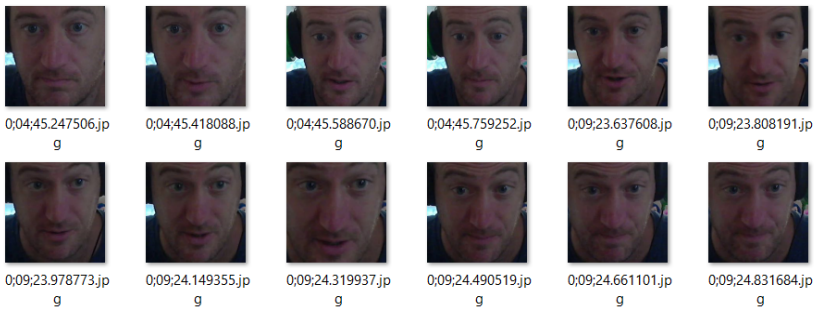


Figure 45: Frames extracted from video recordings of people playing video game.

In this section, the basic emotion classifier from section 5.1.1 (model B4) is experimented on video recordings of people playing video game (see figure 45). The purpose of this experiment is to see how well the model perform under an environment that is considerably different from the one it was trained on. Only the basic emotion model is used because there is no complex emotion label for these videos. The results can be found in table 11. First, model

B4 is used to classify the players' emotion without further training on the game videos. This time it only managed to achieve 21.08% accuracy. Then, the model is trained from scratch on half of the video frames from game videos and the training set of EESS dataset. It achieved 94.05% accuracy when it was re-evaluated with the game data. The confusion matrices before and after training can be found in figure 46 and figure 47. The results are consistent with the findings in the paper by Michel & El Kaliouby (2003). They found that training and classification be should be performed under similar conditions in order to achieve high accuracy. Michel & El Kaliouby (2003) also mentioned that classification accuracy is greatly affected by the distance between the person and the camera as well.

Model	**Test accuracy**
B4 (no training)	21.08%
B4 (with training)	94.05%

Table 11: Game video classification results

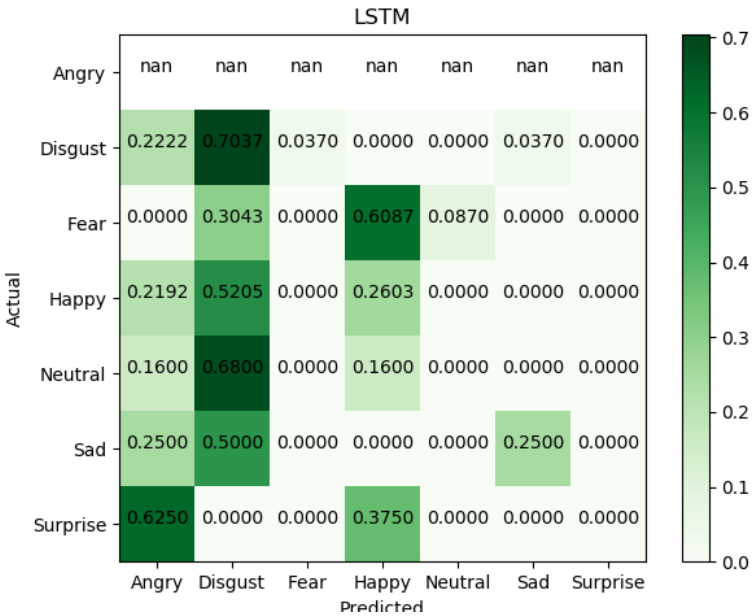


Figure 46: Confusion matrix of model B4 without any further training.

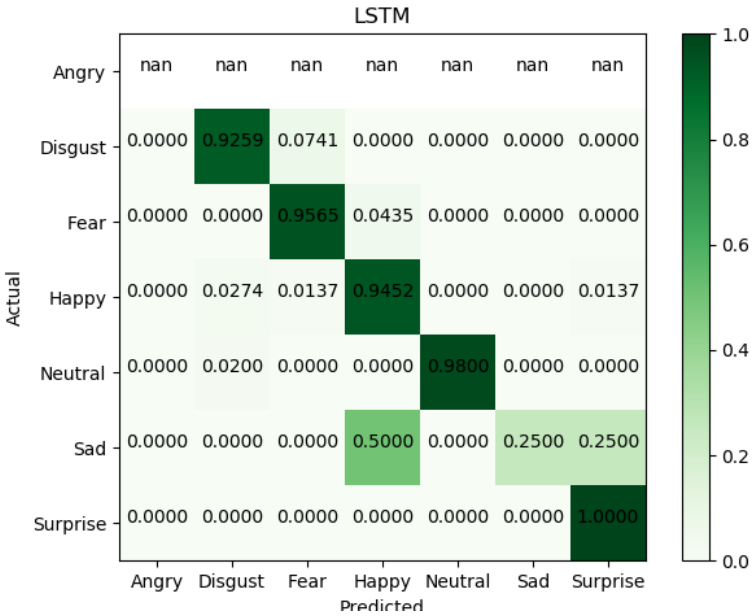


Figure 47: Confusion matrix of model B4 trained on EESS and game data.

5.3 Experiments with Realtime Videos

5.3.1 Basic Emotions

In this section, model B4 was used to perform real-time recognition on several videos. The purpose of this experiment is to see how well the model perform in real-time application, and to identify areas that can be improved.

The first video is a video that the model has been trained on. In this case, the correct label is "Angry". As illustrated in figure 48, the model was able to classify the person's emotion as "Angry" throughout the video. This is expected since the model has seen parts of this video before.

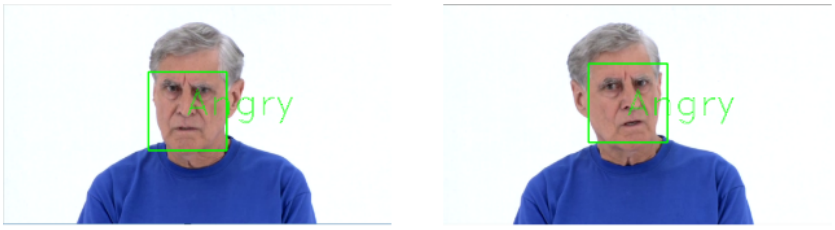


Figure 48: Real-time basic emotion recognition with a video that the model has been trained on. In this case, the correct label is "Angry".

The second video features a person that the model has never seen before. Like the previous video, the correct label is "Angry". This time the model classify the person's emotion as "Angry" most of the time. However, it occasionally switches to "Happy" and "Sad" (see figure 49). It appears that the model performance is slightly worse when it encounters a person that is not included in the training set.

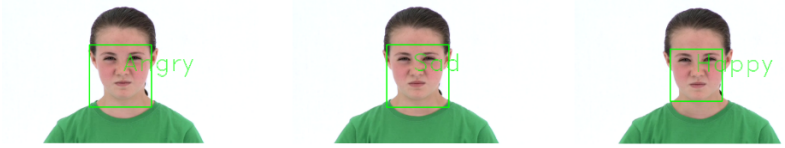


Figure 49: Real-time basic emotion recognition with a video that the model has never seen before. In this case, the correct label is "Angry".

The third video is a YouTube video shown in figure 50.There is no label

for this video. The model assigns "Happy" label most of the time, which seems to be correct because the person in the video always smile when she talks to the viewers. However, the label occasionally switches to "Sad" as well. It is clear that the model performance will drop when it encounters an unseen person. The video used in this experiment can be found at ***https://www.youtube.com/watch?v=yvYUhY4nY2g&t=5s***.

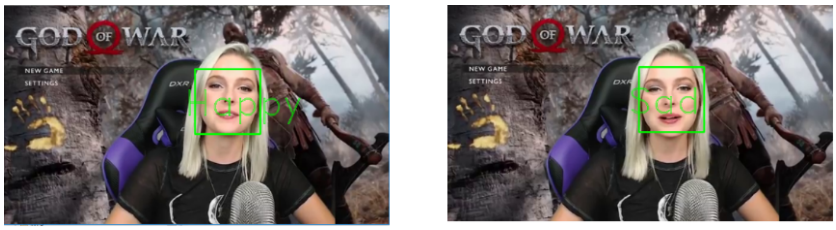


Figure 50: Real-time basic emotion recognition with a YouTube video. This video is unlabeled.

The last experiment involves using the model to classify the author's emotion in real-time. The video was captured with the laptop webcam. Unfortunately, the classifier always classify the author's emotion as "Angry" or "Disgust" even though the author has neutral expression. The cause of this could be the position of the webcam on Dell Inspiron 7460. Since the webcam is located at the bottom of the screen, it is not possible to capture the full frontal view of the face like in the EESS videos. Another explanation could be that the model performance is worse when used on a person from ethnic groups that are not included in the training set. One way to cope with this is to train the model with more diverse training samples.

5.3.2 Complex Emotions

In this section, model D5 was used to perform real-time recognition on several videos. The first video is a video that the model has been trained on. In this case, the correct label is "Angry". As illustrated in figure 52, the model was able to classify the person's emotion as "Angry" and similar emotions such as "Angry low intensity" and "Frustrated". This is a reasonable performance considering that there are 27 complex emotion categories.

Figure 51: Real-time complex emotion recognition with a video that the model has been trained on. In this case, the correct label is "Angry".

The second video is a YouTube video shown in figure **??**.There is no label for this video. The model switches between "Happy", "Happy low intensity" and "Proud" throughout the video. This is consistent with the result from the previous video. It seems that complex emotion classifier usually assign the right label, but it occasionally get confused with similar emotions. The video used in this experiment can be accessed by following the link provided in section 5.3.1.

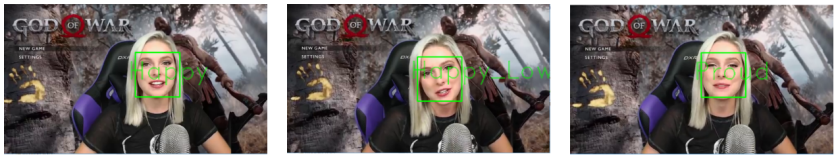


Figure 52: Real-time complex emotion recognition with a YouTube video. This video is unlabeled.

The last experiment involves using the model to classify the author's emotion in real-time. The result is similar to the webcam test in section 5.3.1. The classifier always classify the author's emotion as "Sneaky" even though the author has neutral expression. The performance is likely to improve if the classifier was trained on a dataset with more diverse actor demographics.

6 Conclusion and Future Work

6.1 Conclusion

The objectives of this dissertation are to design and implement Recurrent Neural Network (RNN) to enable sequential learning on consecutive video

frames and infer basic and complex emotional types, and perform quantitative, comparative analysis of feature extraction techniques.

The dataset was provided by Autism research centre (arc). It contains various emotions and mental states that are represented through facial expressions, vocal expressions, body gestures and contextual social scenes (O'Reilly et al. 2016). There were several video frames where the actor has neutral expression but the frames were labeled as something else. This affects the classifiers' ability to recognize neutral expression. Therefore, these frames were deleted from the dataset. The classifiers' performance also suffered from insufficient training samples. To tackle with this problem, overlapping between sequences was implemented. This means that a sequence of video frames will contain some of the frames from the previous sequence. The dataset was split into a 80%/10%/10% ratio of training/validation/test subsets.

VGG16 model pre-trained on imagenet dataset and OpenFace were chosen to perform feature extraction, while LSTM was chosen to perform basic and complex emotion classification on sequences of features. A total of 4 different network architectures (B4 and C5 for basic emotion classification, D5 and E5 for complex emotion classification) were chosen to be tested on the test set from EESS. Model B4 managed to obtained the highest classification accuracy (99.69%) for basic emotion classification task, whereas model D5 achieved the highest accuracy (95.71%) for complex emotion classification task. From the classification results, it is clear that models capable of sequential learning always perform better than the ones that only use feature from the static images. This is because recurrent classifiers such as LSTM also consider information from previous video frames when making prediction, whereas traditional classifiers such as SVM make predictions without any knowledge about the previous inputs. The results also show that features extracted by pre-trained VGG16 tend to achieve better results than AU.

The results from the experimentations on videos outside of the training set shows that the model performance is worse under an environment that is considerably different from the one it was trained on. The recognition accuracy also decreases when the classifier was used on the person that is not part of the dataset. This is more noticeable with the person from ethnic groups that are not included in the dataset. It is clear that the classifier needs

to be trained on datasets that has more diverse environments than the EESS dataset in order for it to be applicable for real-time emotion recognition.

Overall, the objectives of this dissertation have been fulfilled since the LSTM models can classify emotions at a higher level of accuracy than that of the baseline static image classifiers. Two feature extraction techniques were also analyzed and it appears that features extracted by a pre-trained CNN tend to achieve better results than AU features.

6.2 Future Work

Due to time constraints, there were many functionalities that could not be implemented and tested. Here, potential future works are explored.

Increase training samples The most common way of improving classifier performance is by increasing the size of the training set. With more training samples, the recurrent networks with longer sequence length can also be experimented with.

More diverse training samples The experimentation on several real-time videos shows that the model performance is worse under an environment that is considerably different from the one it was trained on. It also had a hard time trying to classify emotion of a person from ethnic groups that are not included in the dataset. It is clear that the model needs to be trained on more diverse training samples before it can be used in real-time emotion recognition applications.

Experiment with other pre-trained CNN architectures In this project, the most successful feature extraction method involves using outputs of a pre-trained CNN called VGG16. It is possible that other popular CNN architectures such as AlexNet, GoogLeNet, or ResNet can extract better features than the ones extracted from VGG16.

Experiment with other variants of LSTM This project only used standard LSTM to perform sequential learning. One variation of LSTM that can potentially be used with this classification is the Gated Recurrent Unit (GRU), introduced by Cho et al. (2014). GRU merges the forget and input

gates into a single gate called “update gate”, and combines the cell state and hidden state (Olah 2015). As a result, GRU is simpler than standard LSTM networks.

Bibliography

Adams, A. & Robinson, P. (2015), Automated recognition of complex categorical emotions from facial expressions and head motions, *in* 'Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on', IEEE, pp. 355–361.

Baltrusaitis, T. (2018*a*), 'Action units', https://github.com/TadasBaltrusaitis/OpenFace/wiki/Action-Units. Accessed: 2018-08-06.

Baltrusaitis, T. (2018*b*), 'Openface 2.0.4: a facial behavior analysis toolkit', https://github.com/TadasBaltrusaitis/OpenFace/wiki. Accessed: 2018-08-06.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014), 'Learning phrase representations using rnn encoder-decoder for statistical machine translation', *arXiv preprint arXiv:1406.1078* .

Chollet, F. (2016), 'Building powerful image classification models using very little data', https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html. Accessed: 2018-08-03.

Chollet, F. (2017*a*), *Deep learning with python*, Manning Publications Co.

Chollet, F. (2017*b*), 'Using a pre-trained convnet', https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.3-using-a-pretrained-convnet.ipynb. Accessed: 2018-07-31.

CMU (n.d.), 'Facs - facial action coding system', https://www.cs.cmu.edu/~face/facs.htm. Accessed: 2018-08-06.

Dahal, P. (2017), 'Classification and loss evaluation - softmax and cross entropy loss', https://deepnotes.io/softmax-crossentropy. Accessed: 2018-08-03.

Geoffrey Hinton, Nitish Srivastava, K. S. (2014), 'rmsprop: Divide the gradient by a running average of its recent magnitude', http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accessed: 2018-08-05.

Géron, A. (2017), *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*, " O'Reilly Media, Inc.".

Harvey, M. (2016), 'Continuous video classification with tensorflow, inception and recurrent nets', https://blog.coast.ai/continuous-video-classification-with-tensorflow-inception-and-recurrent-nets-250ba9ff6b85. Accessed: 2018-08-12.

Harvey, M. (2017), 'Five video classification methods implemented in keras and tensorflow', https://blog.coast.ai/five-video-classification-methods-implemented-in-keras-and-tensorflow-99cad29cc0b5. Accessed: 2018-08-17.

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780.

Karpathy, A. (2015), 'The unreasonable effectiveness of recurrent neural networks', http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Accessed: 2018-08-05.

Keras (2018*a*), 'Keras: The python deep learning library', https://keras.io/. Accessed: 2018-08-08.

Keras (2018*b*), 'Usage of callbacks', https://keras.io/callbacks/. Accessed: 2018-08-17.

Khorrami, P., Paine, T. & Huang, T. (2015), Do deep neural networks learn facial action units when doing expression recognition?, *in* 'Proceedings of the IEEE International Conference on Computer Vision Workshops', pp. 19–27.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Kumari, J., Rajesh, R. & Pooja, K. (2015), 'Facial expression recognition: A survey', *Procedia Computer Science* **58**, 486–491.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998), 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324.

Li, S. & Deng, W. (2018), 'Deep facial expression recognition: A survey', *arXiv preprint arXiv:1804.08348* .

Michel, P. & El Kaliouby, R. (2003), Real time facial expression recognition in video using support vector machines, *in* 'Proceedings of the 5th international conference on Multimodal interfaces', ACM, pp. 258–264.

Nagpal, A. (2017), 'L1 and l2 regularization methods', https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c. Accessed: 2018-08-03.

Neural-network.io (2018), 'Gradient descent', https://www.neural-networks.io/en/single-layer/gradient-descent.php. Accessed: 2018-08-05.

Olah, C. (2015), 'Understanding lstm networks', http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2018-07-31.

O'Reilly, H., Pigat, D., Fridenson, S., Berggren, S., Tal, S., Golan, O., Bölte, S., Baron-Cohen, S. & Lundqvist, D. (2016), 'The eu-emotion stimulus set: a validation study', *Behavior research methods* **48**(2), 567–576.

Russell, S. J. & Norvig, P. (2009), *Artificial Intelligence: A Modern Approach*, 3 edn, Pearson Education.

Sharma, S. (2017), 'Activation functions: Neural networks', https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Accessed: 2018-08-05.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: a simple way to prevent neural networks from overfitting', *The Journal of Machine Learning Research* **15**(1), 1929–1958.

TensorFlow (2018), 'Tensorflow', https://www.tensorflow.org/. Accessed: 2018-08-08.

TutorVista (2018), 'Mean squared error', https://math.tutorvista.com/statistics/mean-squared-error.html. Accessed: 2018-08-03.

Vanderplas, J. (2012), 'Neural network diagram', http://www.astroml.org/book_figures/appendix/fig_neural_network.html. Accessed: 2018-07-31.

A Confusion Matrices

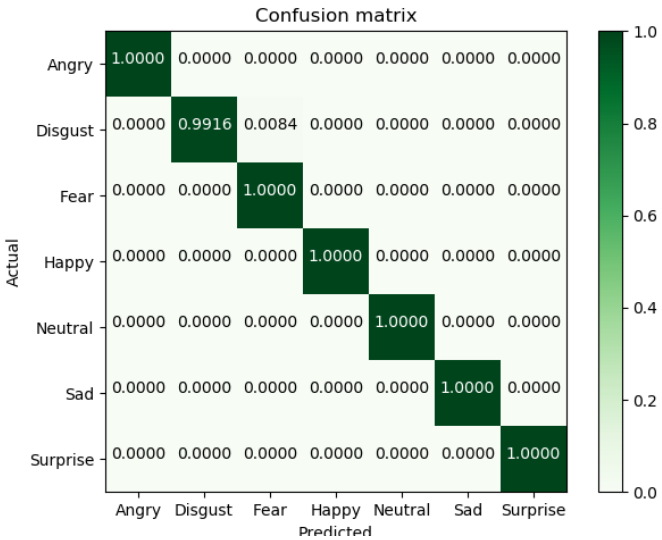


Figure 53: Confusion matrix of model B1.

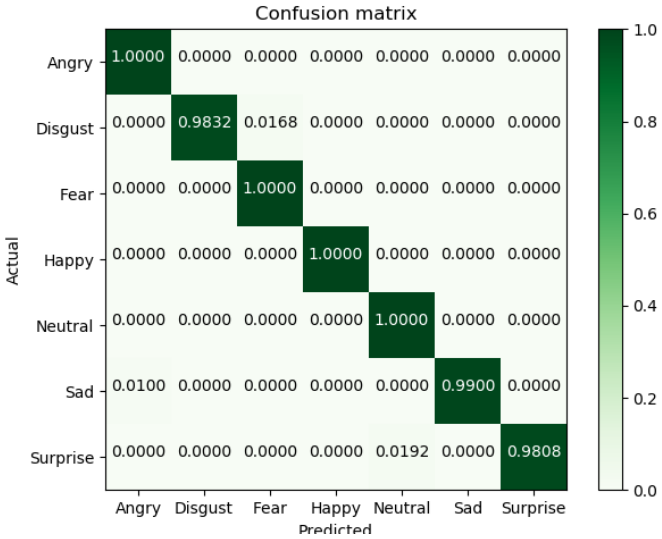


Figure 54: Confusion matrix of model B2.

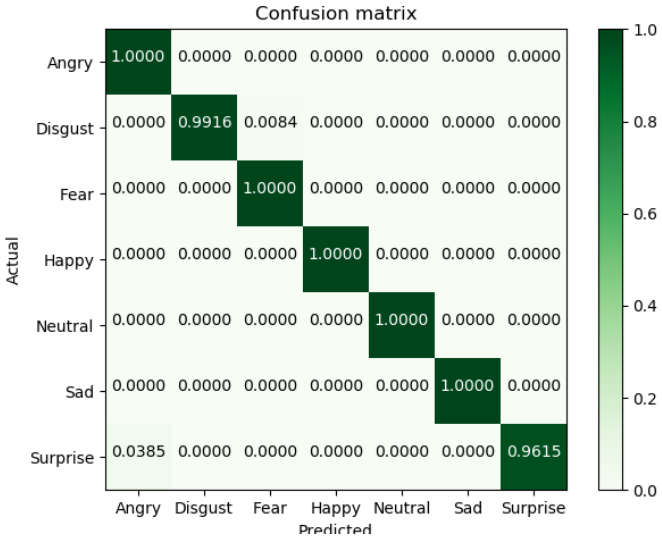


Figure 55: Confusion matrix of model B3.

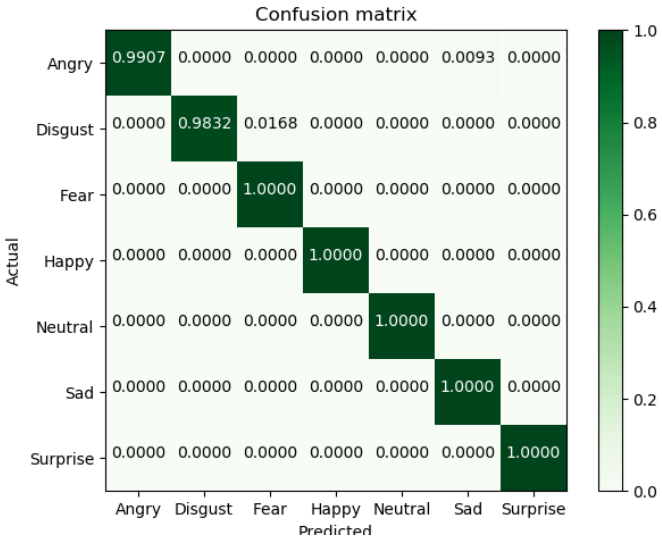


Figure 56: Confusion matrix of model B5.

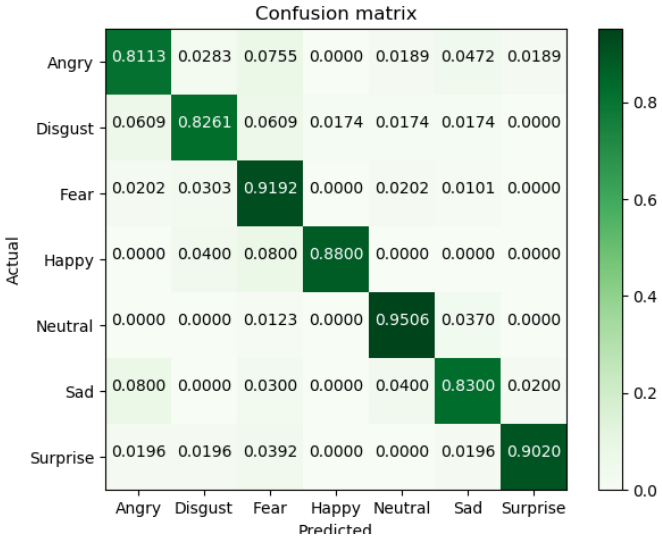


Figure 57: Confusion matrix of model C1.

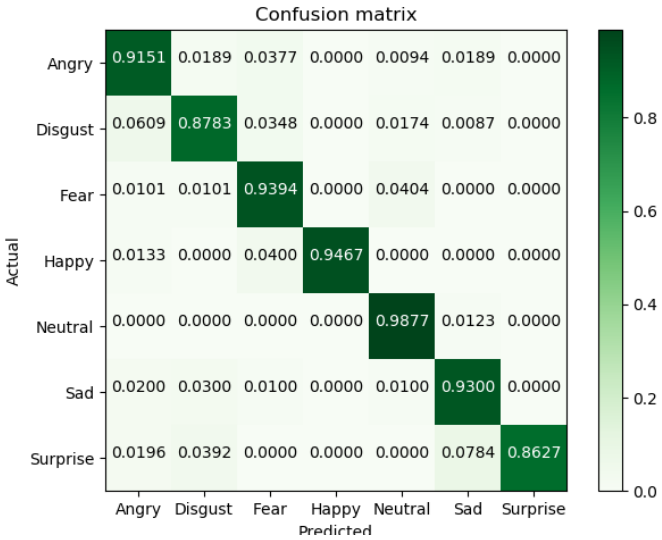


Figure 58: Confusion matrix of model C2.

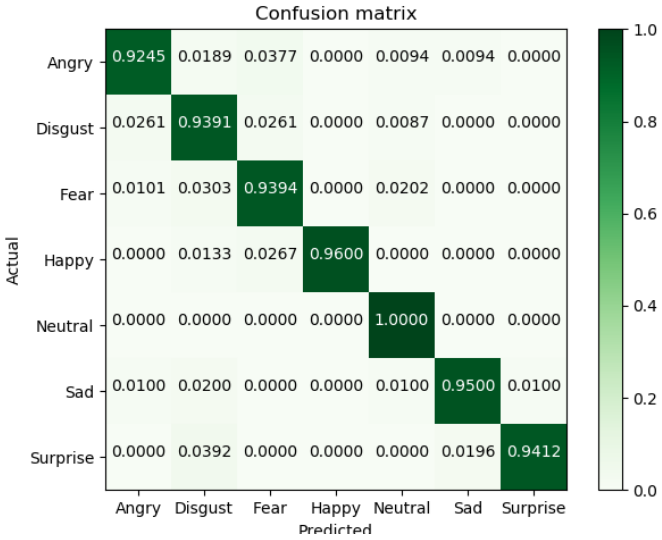


Figure 59: Confusion matrix of model C3.

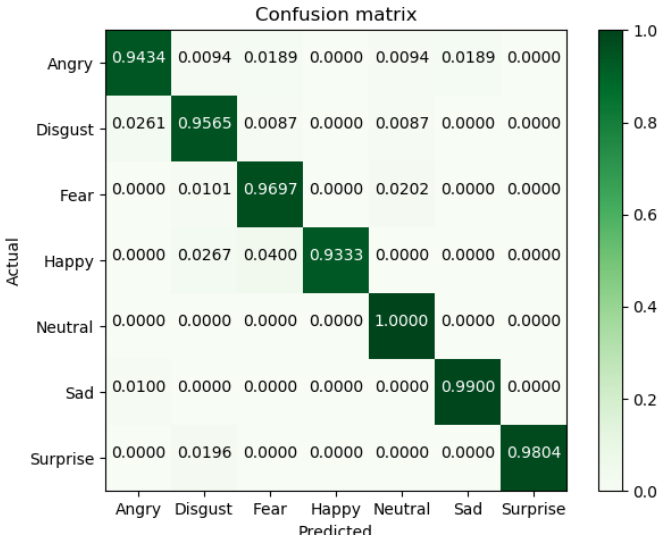


Figure 60: Confusion matrix of model C4.