

# Facial Emotion Recognition using Deep Learning

Emilia Basioli Kirkvik



Thesis submitted for the degree of  
Master in Applied Computer and Information  
Technology (ACIT)  
Program option: Robotics and Control  
30 credits

Department of Computer Science  
Faculty of Technology, Art and Design

OSLO METROPOLITAN UNIVERSITY

Spring 2022



# Facial Emotion Recognition using Deep Learning

Emilia Basioli Kirkvik

© 2022 Emilia Basioli Kirkvik

Facial Emotion Recognition using Deep Learning

<http://www.oslomet.no/>

Printed: Oslo Metropolitan University



# Abstract

Rapid advancements in Machine Learning (ML) have made it possible to equip computers with the ability to analyze, recognize and understand emotions. Facial Emotion Recognition (FER) is a technology that analyzes facial expressions in images to reveal information about a person's emotional state. Researchers from a variety of sectors are becoming increasingly interested in FER as it has a wide range of applications, but of special importance is human-computer interaction.

In this thesis, a main purpose is to evaluate how can transfer learning influence model performance, and can traditional sequential-built models be reduced by transfer learning and still obtain similar accuracies. Seven categories of emotions are categorized from the The Facial Expression Recognition 2013 (FER-2013) database containing a total of 35887 images. The dataset is divided with a k-fold split method of 10 folds in order to separate independent training and testing indices. Among many techniques for FER, deep learning models have shown a great potential for powerful automatic feature extractions and computational efficiencies, especially the Convolutional Neural Network (CNN)'s. This thesis will investigate the performance of one sequential-built CNN model, one pre-trained Resnet50 model and one pre-trained VGG16 model. Each model is compared and evaluated based on chosen metrics such as f1 and confusion matrices. Finally, a few selected images from the Japanese Female Facial Expression (JAFFE) database is presented to the models as an experiment to introduce unseen facial expression images.

Results show that a sequential base model of only a few convolutions ran on 50 epochs gave accuracy result of 89.7%, with an average time of training per k-fold of 21 minutes. The two pre-trained models had fewer convolutions and gave accuracy results of 86.5% without fine-tuning after only 20 epochs. Average training time was 12 minutes per k-fold. Use of slightly more epochs would have given more similar results. A pre-trained model with the use of transfer learning is therefore a recommended choice for saving computational training time and model building, but still receive similar accuracies in results.



# Preface

This thesis is the end of a two year long master degree in Applied Computer and Information Technology, under the Robotics and Control program at Oslo Metropolitan University (OsloMet). The work has been carried out during the spring of 2022.

I would like to thank my supervisor Evi Zouganeli for her excellent guidance, quick response time and her availability during thesis term. I would also like to express my gratitude to my co-supervisor Nasos Lentzas for all the help regarding Python programming and knowledge in Machine Learning techniques I have received during this project.

Oslo, May 16, 2022

---

Emilia Basioli Kirkvik



# Acronyms

Descriptions are itemized in alphabetical order and only specifies the most important ones

**AI** Artificial Intelligence.

**ANN** Artificial Neural Networks.

**API** Application Program Interface.

**AUC** Area Under the Curve.

**CNN** Convolutional Neural Network.

**DL** Deep Learning.

**FACS** Facial Action Coding System.

**FER-2013** The Facial Expression Recognition 2013.

**GPU** Graphics Processing Unit.

**ILSVRC** ImageNet Large Scale Visual Recognition Challenge.

**JAFFE** Japanese Female Facial Expression.

**KDEF** Karolinska Directed Emotional Faces.

**ML** Machine Learning.

**OsloMet** Oslo Metropolitan University.

**R&D** Research and Development.

**ResNet** Residual Neural Network.

**SVM** Support Vector Machine.



# Contents

<b>Abstract</b>	<b>i</b>
<b>Preface</b>	<b>iii</b>
<b>Acronyms</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background & Motivation . . . . .	1
1.2 Goal . . . . .	2
1.3 Problem formulation . . . . .	2
1.4 Scope & Limitations . . . . .	2
1.5 Research method . . . . .	3
1.6 Thesis outline . . . . .	3
<b>2 Theoretical background</b>	<b>5</b>
2.1 Emotional categories . . . . .	5
2.2 Digital Images . . . . .	8
2.2.1 Data processing . . . . .	10
2.3 Artificial intelligence . . . . .	11
2.3.1 Machine Learning . . . . .	11
2.3.2 Deep Learning . . . . .	12
2.4 Selecting ML dataset . . . . .	14
2.4.1 Subdividing data sets . . . . .	14
2.4.2 Splitting datasets . . . . .	15
2.4.3 Ethical considerations on dataset . . . . .	16
2.5 Artificial Neural Networks . . . . .	17
2.5.1 Convolutional Neural Network . . . . .	17
2.5.2 CNN Parameters . . . . .	18
2.6 Analysis Metrics . . . . .	18
2.7 CNN models . . . . .	22
2.7.1 LeNet . . . . .	22
2.7.2 AlexNet . . . . .	22
2.7.3 VGG16 . . . . .	23
2.7.4 Inception . . . . .	23
2.7.5 ResNet . . . . .	23
2.8 Transfer learning . . . . .	24
2.8.1 Pre trained models . . . . .	24

<b>3</b>	<b>Literature Review</b>	<b>27</b>
<b>4</b>	<b>Methodology</b>	<b>31</b>
4.1	Existing software . . . . .	31
4.1.1	Python . . . . .	31
4.1.2	Keras . . . . .	32
4.1.3	Scikit-learn . . . . .	32
4.1.4	Tensorflow . . . . .	32
4.2	Metrics used . . . . .	32
4.3	Datasets . . . . .	33
4.4	Implementation . . . . .	35
4.4.1	Database collection . . . . .	36
4.4.2	Dataset Directory Structure . . . . .	37
4.4.3	Loading datasets and images . . . . .	38
4.4.4	Analyzing dataset . . . . .	40
4.4.5	Data preprocessing . . . . .	44
4.4.6	Defining callbacks . . . . .	45
4.4.7	Building Sequential model . . . . .	46
4.4.8	Building VGG16 model . . . . .	49
4.4.9	Building Resnet50 model . . . . .	51
4.4.10	Tracking and running experiments . . . . .	52
4.4.11	Extracting results . . . . .	53
4.4.12	Predicting on new dataset . . . . .	54
<b>5</b>	<b>Results</b>	<b>55</b>
5.1	Sequential model . . . . .	55
5.1.1	.jpg dataset . . . . .	55
5.1.2	.csv dataset . . . . .	58
5.1.3	Data format assessment . . . . .	61
5.2	Pre-trained VGG16 model . . . . .	61
5.3	Pre-trained ResNet50 model . . . . .	64
5.4	Predictions from testing . . . . .	67
5.5	Predictions on unseen datasets . . . . .	68
<b>6</b>	<b>Discussion</b>	<b>75</b>
6.1	Metric results . . . . .	75
6.2	Choosing datasets . . . . .	76
6.3	Splitting in train and test . . . . .	77
6.4	Model parameters . . . . .	77
6.5	Predictions on unseen datasets . . . . .	78
6.6	Challenges with unbalanced datasets . . . . .	78
6.7	Challenges with implementation of ML program . . . . .	79
6.8	Further work . . . . .	79
<b>7</b>	<b>Conclusion</b>	<b>81</b>
	<b>Bibliography</b>	<b>85</b>
	<b>Appendices</b>	<b>89</b>



<b>A</b>	<b>Training results for each fold</b>	<b>89</b>
A.1	VGG16 model . . . . .	89
A.2	Resnet50 model . . . . .	105
A.3	Sequential model . . . . .	121



# List of Figures

2.1	Ekman's six Basic Emotions . . . . .	6
2.2	Muscles of Facial Expression . . . . .	6
2.3	Action Units of Facial Expressions . . . . .	7
2.4	Creation of Digital Images . . . . .	9
2.5	RGB color channels of an image . . . . .	9
2.6	Correlation between AI, ML and DL . . . . .	11
2.7	ML in contrast to classical programming . . . . .	12
2.8	ML with feature extraction compared to DL . . . . .	13
2.9	Splitting Train and Test datasets by 80/20 % . . . . .	15
2.10	Splitting Train and Test datasets with k folds . . . . .	16
2.11	List of available models in Keras . . . . .	25
4.1	Extraction of FER dataset . . . . .	34
4.2	Extraction of JAFFE dataset . . . . .	35
4.3	Emotion Recognition system architecture . . . . .	36
4.4	Dataset directory structure . . . . .	38
4.5	Print first row of dataset . . . . .	39
4.6	Count from Training, PublicTest and PrivateTest . . . . .	39
4.7	Example images from dataset . . . . .	41
4.8	Emotional categories in Piechart . . . . .	42
4.9	Distribution of train, validation test dataset . . . . .	42
4.10	Emotional categories in bar plot . . . . .	43
4.11	Image shape structure . . . . .	44
4.12	Structure of Sequential model . . . . .	48
4.13	Structure of VGG16 model . . . . .	51
4.14	Structure of Resnet50 model . . . . .	52
5.1	Sequential model: Train and loss . . . . .	56
5.2	Sequential model: Precision, recall F1 . . . . .	56
5.3	Sequential model: Confusion matrix 1 . . . . .	57
5.4	Sequential model: Classification report 1 . . . . .	58
5.5	Sequential model: History of training acc and loss . . . . .	59
5.6	Sequential model: History of training auc, recall, precision, f1 . . . . .	59
5.7	Sequential model: Confusion matrix 2 . . . . .	60
5.8	Sequential model: Classification report 2 . . . . .	61
5.9	VGG16 model: History of training acc and loss . . . . .	62
5.10	VGG16 model: History of training auc, recall, precision, f1 . . . . .	62
5.11	VGG16 model: Confusion matrix . . . . .	63

5.12	VGG16 model: Classification report . . . . .	64
5.13	Resnet50 model: History of training acc and loss . . . . .	65
5.14	Resnet50 model: History of training auc, recall, precision, f1 . . . . .	65
5.15	Resnet50 model: Confusion matrix . . . . .	66
5.16	Resnet50 model: Classification report . . . . .	67
5.17	Sequential model: predicting on test image with expression <i>happy</i> from JAFFE dataset . . . . .	68
5.18	Sequential model: predicting on test image with expression <i>sad</i> from JAFFE dataset . . . . .	69
5.19	Sequential model: predicting on test image with expression <i>angry</i> from JAFFE dataset . . . . .	69
5.20	Resnet50 model: predicting on test image expression <i>happy</i> from JAFFE dataset . . . . .	70
5.21	Resnet50 model: predicting on test image expression <i>sad</i> from JAFFE dataset . . . . .	71
5.22	Resnet50 model: predicting on test image expression <i>angry</i> from JAFFE dataset . . . . .	71
5.23	VGG16 model: predicting on test image expression <i>happy</i> from JAFFE dataset . . . . .	72
5.24	VGG16 model: predicting on test image expression <i>sad</i> from JAFFE dataset . . . . .	72
5.25	VGG16 model: predicting on test image expression <i>angry</i> from JAFFE dataset . . . . .	73
6.1	Tensorflow deprecated classes . . . . .	79

# List of Tables

2.1	Table showing Emotion Action Units . . . . .	7
3.1	KDEF dataset performance . . . . .	27
3.2	KDEF dataset performance with Transfer Learning . . . . .	28
5.1	Test set results from FER-2013 dataset over average metrics .	67
6.1	Test set results from FER-2013 dataset after 20 epochs . . . .	75



# Chapter 1

## Introduction

The background for the thesis, formulation of a problem statement and research questions, as well as goal for the project and the outline of this Master Thesis is first presented.

This thesis is the final work for a Master in Applied Computer and Information Technology at the Oslo Metropolitan University (OsloMet) in Oslo, Norway.

### 1.1 Background & Motivation

Communication between robots and humans are crucial when we look at the fast development of technology in the direction of Artificial Intelligence (AI). One important part of human communication is the emotions behind expressed words. The goal of emotion recognition is to extract, evaluate, and comprehend people's emotional behavior traits (Tian, 2021). Emotion detection can be utilized for a variety of purposes, including enhancing people's health, improving human to machine communication, and identifying customers' emotional states in order to provide them with more tailored and high-quality services. Emotions can be detected from both speech, gestures and body language, but this thesis will purely focus on facial expressions, a non-verbal communication technique.

Facial expressions are unique in that they are the only part of the face that changes rapidly and in a number of complex ways. We can impressively identify diverse expressions and set them apart with only a quick glance, without necessarily knowing the person (Kaulard et al., 2012). Human emotions by facial expressions and gestures differ by very small facial changes. In recent years, the cross-cultural communication has increased with more international students, travelers, and migrants today than ever before (Fang et al., 2021). It is crucial to comprehend how gender and cultural differences impact the recognition of the same facial expression.

## 1.2 Goal

The purpose of this thesis is to develop a program or model that can distinguish various human emotions. Using Machine Learning (ML), the program should be able to distinguish between human emotions and classify each feeling into one of seven categories, including neutral.

Another goal is to optimize the model by finding and combining algorithms that give a satisfactory prediction percentage. It is of specific interest to run the program on a training dataset and then examine its effectiveness and subsequently test it on other datasets. Different ethnic identities should be present in the other datasets than in the original training dataset. The current work will design a ML model using software development based on prior research on relevant materials.

## 1.3 Problem formulation

The project will investigate ML methods for emotion recognition, i.e. a system that can automatically recognize the emotions of an individual person. The work will use a dataset containing six different facial expressions in addition to neutral, to train the Artificial Intelligence (AI) model, namely:

Sadness, Anger, Disgust, Fear, Happiness, Surprise and Neutral.

Research questions to be asked and answered are:

- Which Deep Learning model is better for facial emotion recognition?
- How can transfer learning influence model performance?
- How well does the model perform on a different dataset containing facial expressions other than the one it was trained on?

## 1.4 Scope & Limitations

This thesis will not cover all facial emotions. A selection of emotions will be focused on since there are too many different emotions. The context of the emotions can not be taken into account, only facial emotions found in the database are to be considered. Not all ML models will be compared, only a selection of the most common ML models used in previous research papers focusing on emotion recognition or facial expressions will be considered. One ML model will be chosen from the comparison and used for the subsequent research questions. The emphasis on programming language and development environment will be limited to suit a one-semester thesis.



## 1.5 Research method

The method for carrying out this project follows the principles of a Research and Development (R&D) project. An initial task is to acquire a wide range of knowledge within the chosen topic by reading literature and searching information from relevant research papers.

The next part of the method was to divide the thesis tasks into smaller parts, examine what needs to be done, and describe the goal of each task. The thesis is then divided into smaller parts with these main themes:

- Investigate common types of ML algorithms used in research articles focusing on FER
- Research on building a ML environment
- Collecting several databases containing different emotional expressions
- Build and implement a ML program
- Create one DL model and two pre-trained models for transfer learning. Compare model performance and their average percentage results
- Test the trained ML models on unseen image samples

In this context, an algorithm is the logic to solve a task that can be programmed in any programming language. For our purpose, algorithms will be combined such that feature recognitions can be learned.

A model is the outcome of predictive algorithms used on relevant datasets.

## 1.6 Thesis outline

As an overview for the reader, the main structure of the report is as outlined below in a stepwise manner to clarify which factors are needed to answer the problem statement and fulfill the scope of the current work.

**Chapter 1** is the Introduction and provides information on the background for the assignment, goal formulation and problem statement, scope of work and structure of the thesis report.

**Chapter 2** is on Theoretical background, covering basics of emotions, digital images, AI, ML and DL. Database selection, CNN models, metric definitions and transfer learning are also covered.

**Chapter 3** is the Literature Review present and critically analyse relevant work and their findings.

**Chapter 4** is on Methods used, describing database collections, data pre-processing and implementation of ML program.

**Chapter 5** is on Results from model performance.

**Chapter 6** is on Discussion of the results and parameters involved.

**Chapter 7** is on the Conclusions for findings and work done to conclude on research questions. The thesis work will seek improvement in methods to extend the current knowledge and will identify the most efficient methods to answer research questions and solve the scope of the work.

**Chapter 8** is the Bibliography used as background for the research.

Appendices are added for detailed descriptions as referred to in the main sections.

## Chapter 2

# Theoretical background

This chapter will provide a short overview of the basic concepts of emotions, digital images and data pre-processing. A quick comparison of Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) is presented. Then, CNN models, metrics and transfer learning are presented.

### 2.1 Emotional categories

Classification of emotions have a long history and may date as far back as to the first century where a Chinese encyclopedia called the "Book of Rites" described seven feelings, i.e. joy, anger, sadness, fear, love, disliking, and liking. In 1972, Ekman published a list of universal facial emotions (Landowska, 2018 p. 14), consisting of six emotions: anger, disgust, happiness, sadness, fear, and surprise. Others like Robert Plutchik (Rodrigues and Pereira, 2018 p. 850) used eight, which he grouped into four pairs of polar opposites. These are: joy-sadness, anger-fear, trust-distrust, surprise-anticipation. However, with Ekman's definitions, it became easier to process emotions when the six universal facial expressions were described more specifically, as:

**Anger** Lowered eyebrows, glared eyes, tightened lower eyelids, tightened and narrowed lips, and thrust jaw

**Fear** Raised, pulled together eyebrows, tensed lower eyelids, and a lightly opened mouth

**Disgust** Narrowed eyebrows, wrinkled nose, and a curled upper lip

**Happiness** Smile and wide eyes

**Surprise** Raised eyebrows, wrinkled forehead, widely opened eyes, dropped jaw, and an opened mouth

**Sadness** Dropped eyelids, lowered corners of the mouth, downcasted eyes,

and pouted lips.

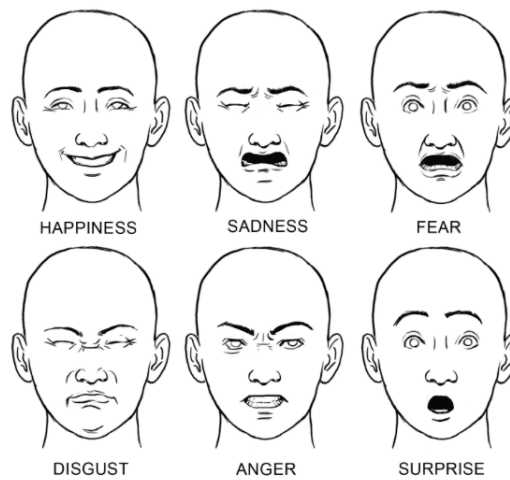


Figure 2.1: Ekman's 6 Basic Emotions. From *Ekman's six Basic Emotions and How They Affect Our Behavior*, by Harrison 2021.

Above is an illustration of Ekman's six basic emotions in Figure 2.1.

Later in 1978, based on work of anatomists, Paul Ekman and Wallace V. Friesen translated emotions into muscle movements (Ekman and Friesen, 1978).

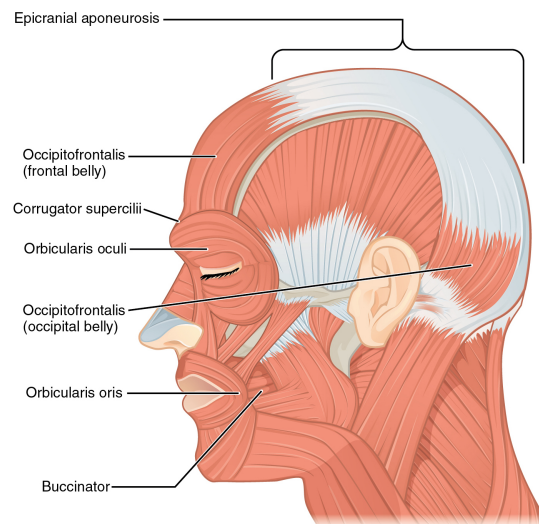
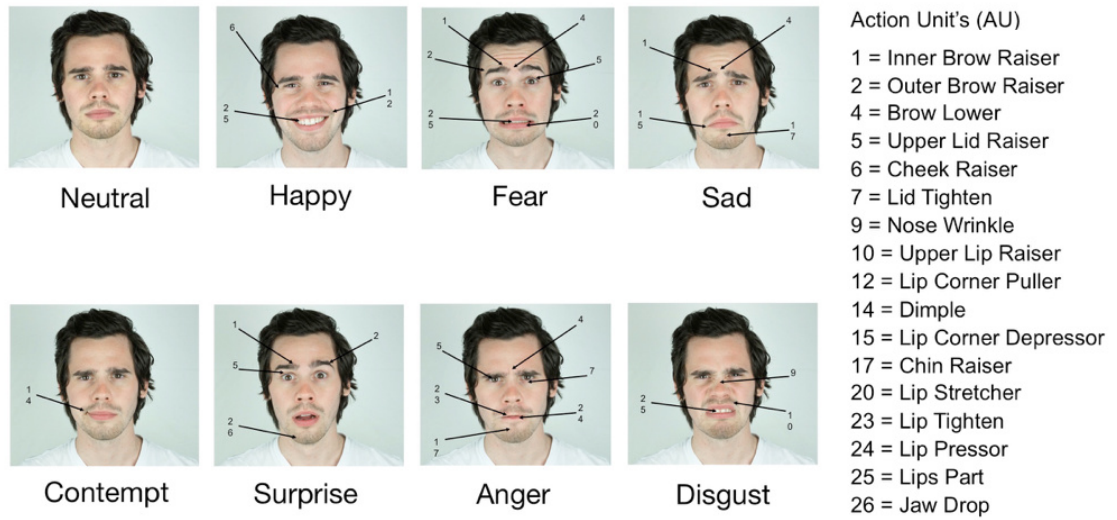


Figure 2.2: Muscles of Facial Expression, lateral view. From *Anatomy and physiology*, by Betts et al. 2013.

In 2002 Joseph C. Hager contributed to a significant update to create a Facial Action Coding System (FACS) in *"Facial Action Coding System: Salt Lake City: A Human Face"* (2002).



© 2019 EIAGroup.com

Figure 2.3: Action Units of facial expressions. From *Facial expressions*, by Lansley 2022.

If no specific expression was identified, a neutral expression was added in addition to the six universal facial expressions. Psychologists have over time added several other expressions, like the shown disgust.

Individual facial muscles and their movements are numbered and called Action Units by FACS. FACS is now a common standard for categorizing physical facial expressions that describe an emotion (Paul Ekman Group LLC, 2020).

By identifying muscle movements by specific numbers or action units, one can digitally represent emotions by concluding on which muscles are activated. E.g.

Emotion Action units	
Happiness	6+12
Sadness	1+4+15
Surprise	1+2+5B+26
Fear	1+2+4+5+7+20+26
Anger	4+5+7+23
Disgust	9+15+17

Table 2.1: Table showing Emotion Action Units

The intensity of each indicated action can also be graded from weak to strong by characters A to E in order to arrive at the most likely conclusion for the expressed emotion.

FACS also has additional codes for eye positions, instant behavioural ac-

tions like speaking and also head movements like tilting, looking sideways, up/ down or other combinations of these.

The latter classification of head movements can easily be used for augmenting training- or test data-sets by flipping, mirroring and rotating original pictures with known expressions. A large increase in samples is beneficial for numerical training in a machine learning algorithm.

From the combined sciences of anatomists and psychologist it is shown that human emotions can be broken down to specific and manageable information items that are suitable for digitization. Once a numerical representation of a facial emotion is obtained, further computerization with numerous algorithms is made possible.

The current work will apply algorithms for the purpose of identifying expressed emotions in a test dataset, and eventually from a random facial picture. For this purpose, so called Artificial Intelligence (AI) is used for identifying muscle movements via machine learning rules for facial expressions, as defined by the FACS standard. The machine learning algorithms employ AI techniques within the category of ANN.

## 2.2 Digital Images

A natural image taken with a camera, telescope, microscope, or other optical device exhibits a constantly changing array of colors and color tones. A natural image must first be translated into a computer-readable form or digital format before it can be processed or displayed by a computer. This procedure is applicable to all photos, regardless of their origin or intricacy, or whether they are in black and white (grayscale) or full color (Spring et al., n.d.).

To convert a continuous-tone image into a digital format, the analog image is divided into individual brightness values. The analog representation of a miniature young starfish imaged with an optical microscope is presented in Figure 2.4(a). The image is then sampled in a two-dimensional array Figure 2.4(b). Brightness levels at specific locations in the analog image are stored and converted into integers Figure 2.4(c). The goal is to turn the image into an array of discrete points, each of which contains unique brightness or tonal range information and may be defined by a specific digital data value in a given place. The output is a numerical representation of the intensity for each sampled data point in the array, which is generally referred to as a picture element or pixel.

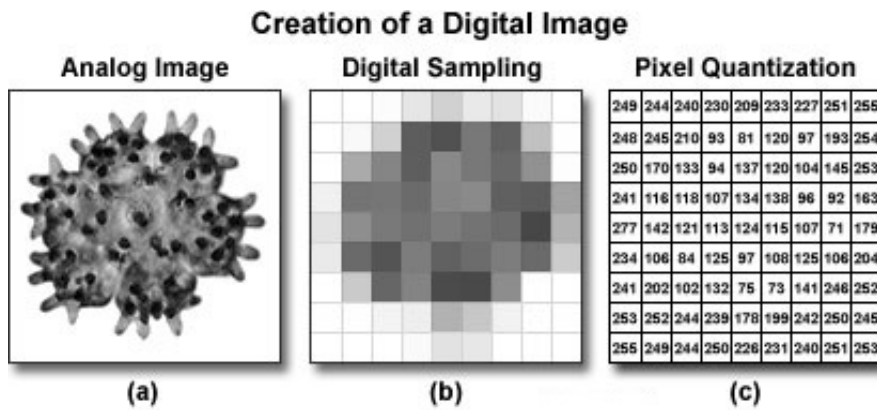


Figure 2.4: Creation of Digital Images. the arrow shows the progression in the level of detail of information which can be extracted from the images. From *Basic Properties of Digital Images*, by Spring et al. n.d.

The Figure 2.4 above represents a greyscale image. When we look at a digital image, we normally see three color channels, which are the Red-Green-Blue channels, sometimes known as the "RGB" values. It has been demonstrated that combining these three can make any color palette. When working with a color image, we must remember that the image is made up of many pixels, each of which contains three separate RGB channel values (Das, 2021).

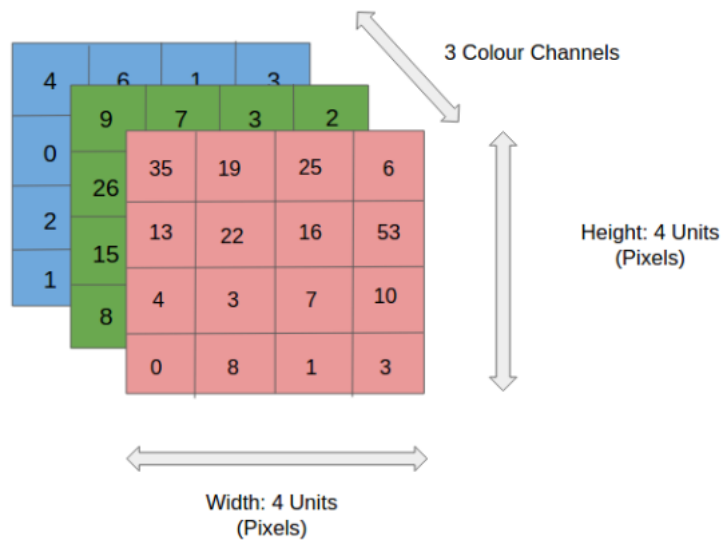


Figure 2.5: RGB color channels of an image. From *Convolution Neural Network for Image Processing — Using Keras*, by Das 2021

Figure 2.5 above illustrates the data structure of an RGB image. The difficulty with photos with several color channels is that we have massive amounts of data to work with, making the procedure computationally costly. In other words, imagine a difficult process in which a Neural Network or other machine learning algorithm must deal with three

different types of data (in this case, R-G-B values) to extract characteristics from photos and categorize them into relevant categories.

### 2.2.1 Data processing

This section will look at data processing from the perspective of an image. In most circumstances, many preprocessing steps are required before the dataset can be fed into the network. This is required in order to prepare the data for training.

The network design as well as the input data type must be carefully considered when creating an effective neural network model. The number of images, image height, image width, number of channels, and number of levels per pixel are the most frequent image data input parameters. A colored image has three data channels that correlate to the hues Red, Green, and Blue (RGB), and a greyscale image has one data channel. The most common pixel levels are  $[0,255]$  (Nikhil, 2019).

**Uniform aspect ratio:** The majority of neural network models assume a square-shaped input image, which means that each image must be evaluated for squareness and cropped accordingly. Cropping can be used to choose a square portion of an image.

**Image Scaling:** Resizing photos to a lesser resolution is a standard process when working with an image-based dataset. This is especially important if the computer lacks a cutting-edge Graphics Processing Unit (GPU), as the model would otherwise be extremely slow. The model will be more effective and more efficient as a result of the down scaling, but some details from the original resolution will be lost. As a result, it's critical to experiment with various scales in order to discover the best balance between maintaining enough vital elements and having an effective model.

**One Hot Encoding:** Another important data pre-processing of images is called one hot encoding. A one hot encoding is a binary vector representation of categorical information. Many machine learning algorithms are unable to operate directly with categorical data. The categories must be numerically transformed (Brownlee, 2019).

**Normalizing image inputs:** Data normalization is a crucial step that assures that each input parameter (in this case, pixel) has a consistent data distribution. This allows for faster convergence while training the network. By removing the mean from each pixel and dividing the result by the standard deviation, data is normalized. A Gaussian curve centered at zero would be the distribution of such data. We need positive pixel numbers for picture inputs, so we might scale the normalized data in the range  $[0, 1]$  or  $[0, 255]$  (Nikhil, 2019).

**Dimensionality reduction:** The RGB channels could be combined into a



single gray-scale channel. When the neural network performance is allowed to be invariant to that dimension, there are typically considerations to lower other dimensions or make the training problem more manageable. If color has no significance in your images to classify then its better to go for grey scale images to avoid false classification and complexities.

## 2.3 Artificial intelligence

Artificial Intelligence is an area of computer science that tries to develop programs that have some intelligence, such as performing tasks based on multiple and different or stochastic inputs, with results that resemble reasoning, concluding or planning.

Intelligence also involves learning, classification and storing memories or saved data for future reference by processing e.g. pictorial images.

Artificial intelligence is in this context an application of machine learning, or perhaps the application is better described as artificial learning.

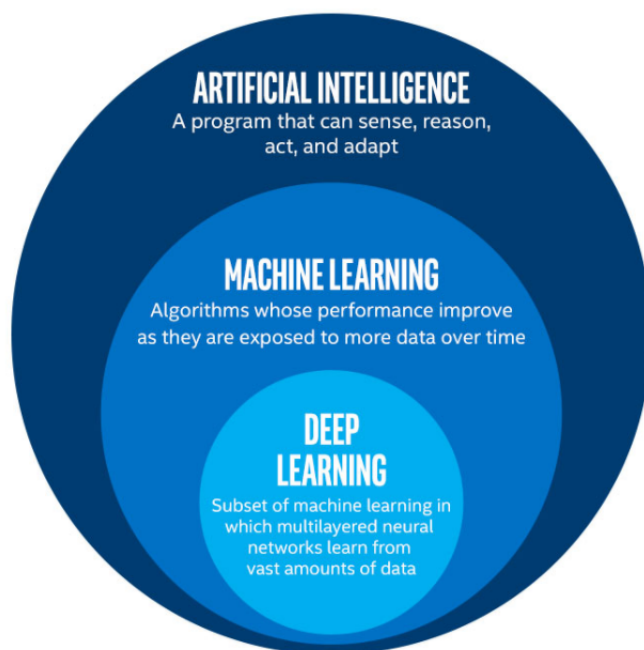


Figure 2.6: Correlation between AI, ML and DL. From *Is machine learning required for deep learning?*, by Suman 2019.

### 2.3.1 Machine Learning

Machine learning is an integrated part of AI. The field of machine learning is concerned with the question of how to construct computer programs that

automatically improve with experience (Mitchell, 1997).

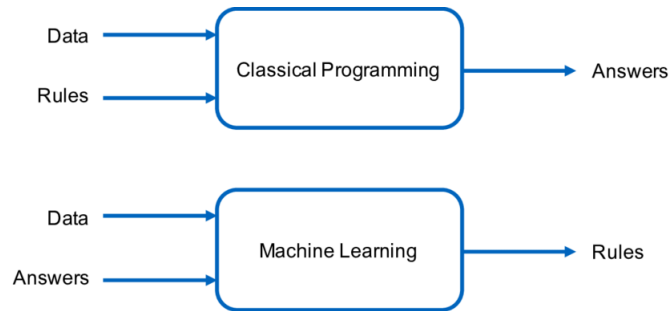


Figure 2.7: Machine Learning in contrast to classical programming. From *Autonomous Driving: Object Detection using Neural Networks for Radar and Camera Sensor Fusion*, by Geisslinger 2019.

Machine Learning may be an algorithm or procedure that uses sets of input data with known outputs or answers to create rules. These are further used to predict outputs for new and random data for a specific area under investigation. As shown in Figure 2.7, compared to classical programming, one must know and create the rules themselves to be able to output answers from programmed rules.

### Supervised learning

Supervised machine learning will learn relationships between data inputs and outputs by a set of rules.

All data is labelled and the algorithms learn to predict the correct output from processing and interpretation of known input data (Brownlee, 2016).

### Unsupervised learning

In unsupervised machine learning, all data is unlabelled and the algorithm learn an underlying pattern in input data by its enhanced complexity and skills to sort and classify processed data. The purpose of unsupervised learning is to detect common patterns in the data.

Unsupervised learning can be used to reduce large dimensions of complicated dataset by sorting similar data to a set of classes before supervised learning is performed on separate classes of data.

### 2.3.2 Deep Learning

Deep Learning is a set of algorithms that tries to resemble the structure of a human brain with its numerous interconnectivities, and is thus called an artificial neural network.

As opposed to Machine Learning that use sets of input data with known answers for features, deep learning algorithms may extract features from raw pixel data, called feature learning. This is illustrated in Figure 2.8, where DL can skip feature extraction compared to ML. Traditional machine learning approaches require the manual implementation of various computer vision techniques in order to extract the appropriate features prior to classification. Convolution layers from CNN's are used to extract features from the layers automatically.

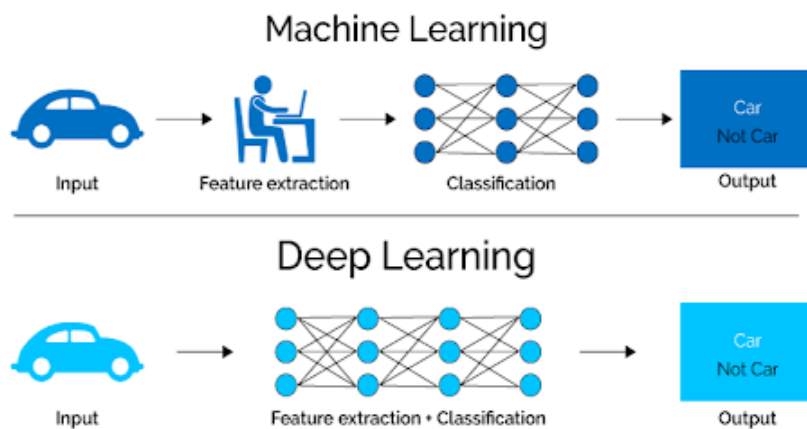


Figure 2.8: In a DL model, the feature extraction step is unnecessary. The model can recognize these unique characteristics of a car and make correct predictions without the help of a human doing feature extraction. From *Artificial Intelligence vs. Machine Learning vs. Deep Learning: What's the Difference?*, by Oppermann 2022.

A deep learning algorithm is tweaked for the task at hand such that results and predictions will improve by training on more and more examples. Parameters are introduced for measuring the accuracy and precision of algorithms.

### Benefits of using CNN versus traditional ML

Deep learning algorithms scale source data in different ways. By expanding or augmenting datasets by e.g. flipping, mirroring and rotating original pictures with known expressions, more and more picture patterns or features may match patterns found in new pictures and give correct interpretations.

Augmenting datasets by artificially changing source data in a machine-like procedure, this will increase the capacity to recognize and classify features in new pictures. Classifications are made in stages of different resolutions, in blocks connected as a neural network.

## 2.4 Selecting ML dataset

Datasets can be loaded directly from places like the UCI Machine Learning repository.

Pandas may be used to load the data as well as to explore data by statistics and visualizations. UCI datasets have been assembled over a long time and have smaller but some famous datasets like the iris flower dataset from 1936. Dataset assemblies are often referred to as repositories.

OpenML is another and newer repository that can be searched by wanted names, and it has a web API to easily retrieve data.

Kaggle has many large datasets that also can be searched by name. The ImageNet image recognition tasks, where architectures of VGG, Inception and ResNet were trained have datasets of photographic and video-graphic content with a size of more than 160 GB.

As the number of datasets increase, and have different free content, information on availability may be updated in a "List of datasets for machine-learning research" on Wikipedia.

Datasets of many GB are normally not downloaded; they are loaded as network libraries, accessed by requests in Python functions, however provided that a good internet connection is available.

Scikit-learn can download dataset data using a built in real-time API interface.

TensorFlow can also be used for download in machine learning projects, however an API need to be installed separately.

Each dataset have different APIs that may return slight differences in formats such that it may be required that e.g. Scikit-learn must be used to re-format or create the dataset to be processed.

### 2.4.1 Subdividing data sets

A dataset usually contain samples of a similar nature. We want to train a candidate model to predict a specific feature of these samples, and we want a training set to check if the model is fit for purpose. Metrics are used to evaluate the model, and if found reasonable, the model is 'fitted' for use with this training dataset.

We also may wish to validate the model on a different part of the training dataset and have a data/validation/ folder with a separate validation dataset. This will enable further optimizations on parameters like augmentations, blocking, dropouts, sizing etc to obtain optimum weights or

strengths for each feature recognition. Results from the validation dataset may give new candidate models that may need to be trained all over again from scratch, without use of previous weight result.

After studying the validation metrics, this may result in a better and new candidate model that may be chosen for further testing. A combination of the train and validation dataset may be used on the chosen best model for this purpose. Comparing metrics in this way is called cross validation.

In addition, we need to keep a part of the complete dataset separated for use in the testing process. A test dataset is not used in previous training and/ or validation steps since test data shall be unseen data for the model. Testing will give an estimate on how the model will behave when it is deployed for public use. If a candidate model is trained correctly, test scores are not expected to be very different from the previous cross validation results.

## 2.4.2 Splitting datasets

A common approach for splitting datasets is to use a 80 % train- and 20% test-split for cross-validation (Feick, 2022). Usually, datasets do not contain thousands of features in each class and are much smaller than required for getting unbiased estimates. Re-sampling strategies like validation testing that result in new candidate models may be necessary. The common division of train and test is illustrated in Figure 2.9 below.

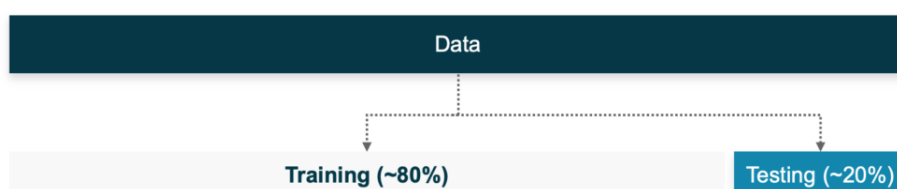


Figure 2.9: A common approach of dividing train test split with 80% and 20%. From *Evaluating Model Performance by Building Cross-Validation from Scratch*, by Feick 2022.

A more advanced procedure for splitting datasets is to use a k-fold strategy where the whole dataset is divided into k folds, and where the last fold is the test-data section. The model is run repeatedly on the dataset, however a different fold is defined as the test fold each time. This gives a total of k models to be fitted and evaluated. The procedure is illustrated in Figure 2.10 below. Metrics for the tested model can be calculated as the mean of all runs. If the dataset is split into folds of 20%, a 5-fold setup will enable the test fold to visit 100% of the dataset.

	Data				
1.	Validate	Train	Train	Train	Train
2.	Train	Validate	Train	Train	Train
3.	Train	Train	Validate	Train	Train
...	Train	Train	Train	Validate	Train
k	Train	Train	Train	Train	Validate

Figure 2.10: A more advanced approach of dividing train test split with k folds. From *Evaluating Model Performance by Building Cross-Validation from Scratch*, by Feick 2022.

It is noted that randomly splitting datasets requires that the number of examples in each class is quite even. Otherwise, results can be misleading if the classification problem has a severe class imbalance, where e.g. one class has less than 10% of what other large samples have.

To prevent inaccuracies, the original class distribution from the whole dataset must be preserved in each split. A technique to modify splits with a class label that preserves the distribution is called stratified train-test splits, or stratified k-fold cross-validation.

Another method of preventing inaccuracies from class imbalance is use of under- or oversampling. This is just to reduce or create multiple copies of pictures in an overpopulated class or a minority class in the training dataset. If one class has only 10% of what other classes have, oversampling can be done one or more times to increase the occurrence 10x.

### 2.4.3 Ethical considerations on dataset

Ethical considerations play a role while recognizing human emotions. When it comes to Machine Learning, it is especially important to evaluate on which data the software is trained on. How does this training dataset function on real-life situations, where e.g. male European facial expressions are processed, if the dataset used to train the model only comprises female Japanese facial expressions?

By experience, ethical considerations may be a major challenge for universal software in this category. It is crucial to not misinterpret human images with other species, especially if minorities or groups with challenges are involved. However, in the subgroup of species emotions, misinterpretation may be of less importance than identification of different species.

## 2.5 Artificial Neural Networks

From the concepts of DL, AI and ML, the outcome of ML is to get a model that can do predictions on data that is introduced to the model. Data may either be of the form where membership to a class shall be predicted, or where a value shall be predicted from other variables.

As opposed to predicting values as a regression problem, sorting pictures into classes of emotions is a classification problem. Classification for these purposes can be made with a Decision Tree model, a Support Vector Machine model or a Neural Network model.

Algorithms based on Artificial Neural Networks have proven useful in interpretation of pictures and gives a satisfactory percentage for prediction accuracy of e.g. general face recognition in pictures. Algorithms for face recognition are widely applied in consumer product like digital cameras and smartphones, where a frame can be seen to appear around faces during picture-taking.

Of special interest in this study is the application of ANN on photographic methods that may be used to interpret facial emotions in pictures, or emotions in face recognitions processed from real-life situations.

### 2.5.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a class of ANN algorithms. A convolutional neural network uses several blocks of layers during the process of filtering, sorting and classifying information from the source. These layers may be a convolutional layer, a pooling layer, and a fully connected classifying output layer (Brownlee, 2020).

A convolutional layer is first used to specify the amount of filters to be used in the learning process of creating feature maps. In addition, properties of filters are defined, like pixel size used in each filter. A kernel shape is defined.

A feature map is a two-dimensional map of activated defined features, with position and strength of a detected feature in an image, in our case a muscle reaction in an emotion.

Pooling layers are used to downsample feature maps in an attempt to more easily identify and sort features from patches of the full original picture, by use of fewer pixels. Calculations may return the maximum, or the most probable feature value from each patch of the full feature map.

Classifier Layer are used after features have been extracted. The feature map is interpreted to arrive at a prediction, in our case an emotion of a face in a photograph. Feature value from each patch can be assembled to a

representative full feature map that is interpreted by a classification output layer. The classification layer will typically predict a value or a set of values that is used for identification of an emotion.

The key to create effective convolutional neural networks is to find an architectural design that best utilize these individual layers, and the combination and number of layers. Deep CNN's have e.g. several convolutional and / or pooling layers.

### 2.5.2 CNN Parameters

Some functions and parameters are often encountered in a description of a Convolutional Neural Network and are listed below;

**Activation function:** The activation functions are located between the network layers and determine which nodes should be fired, that is, nodes that are close to one another.

**Optimization function:** To minimize the error, this function is used to change weights of a recognition.

**Regularization:** L1 regularization L2 regularization

**Learning rate:** This parameter controls how much the weights change between iterations, or how quickly the network evolves.

**Batch size:** When working with big training sets, it is common to divide them into different batches and train each one in turn. The number of samples in a subset, or batch, is referred to as batch size.

**Epochs:** An epoch is a complete data cycle. One pass is a prediction, cost calculation, and weight update learning cycle.

## 2.6 Analysis Metrics

### Regression metrics

A big part of applied machine learning is selection of models, i.e. of several machine learning methods on a dataset, which model has the best skills to make good predictions on unseen data? Shall classification based on maximum accuracy or minimum loss determine how much an estimate can be trusted? Or is the difference purely from statistical differences? Statistical hypothesis testing can be used for this purpose, however use of common test metrics may give sufficiently good results.



A number of common analysis metrics may be used to diagnose model performance on different datasets. In cases where a value shall be predicted from other variables as a regression problem, commonly used metrics are:

- Mean squared error (MSE)
- Root mean squared error (RMSE)
- Mean absolute error (MAE)

### **Classification metrics**

In cases where membership to a class shall be predicted as a classification problem, commonly used metrics are:

- Accuracy
- Logarithmic-loss
- Area Under the Curve (AUC)
- Precision
- Recall
- F-measure

Predicting pictures into classes of emotions is a classification problem. Plots of accuracy, loss and other metrics, and their development over processed epochs give good indications of model performance.

Blue lines will indicate model performance on a training dataset while orange lines will indicate performance on a test dataset. The relative behaviour of these plots give info on model overfitting or underfitting, i.e. if lines for train- and test-results separate and depart from each other. A well-fit model will give the same trend for both datasets.

Another useful measure of performance is using a Confusion matrix. It will in detail show how classifications were made for each feature.

A summary of all analysis metrics can be tabulated in a Classification report.

The purpose of common metrics suitable for judging a classification problem is summarized:

### **Accuracy**

The most common evaluation metric for classification problems is accuracy. Accuracy is the number of correct predictions in relation to all predictions made for a feature, disregarding whether the prediction was correct

or wrong. An accuracy score of 75% to 95% is desirable. A prerequisite is actually that there are an equal number of observations in each class to be evaluated, and that prediction errors are equal in importance.

Classification accuracy is the number of correct predictions relative to the total number of predictions, simply by dividing 100% correct class predictions by the total number of predictions for that class, whether they were right or wrong. A model build for a classification problem will almost always have an accuracy test as a measure. Classification accuracy alone is however often not enough to make a decision on whether it is a good model for solving a specific task. In our case with classifying seven emotions, there will be a problem if features like 'happy' and 'sad' consistently falls into 'neutral' or if one feature like 'disgust' only have 100 pictures while other emotions occur more than thousand times in the dataset. The model may predict a high accuracy value for the majority of classes but will generally not be good at predicting all emotions. Precision and Recall performance are measures that can be used to better evaluate a classification problem.

### **Loss**

Loss of membership to a class is a predicted probability between 0 and 1 of belonging to a given class. Correct predictions are rewarded while incorrect predictions are punished exponentially after a logarithmic scale. A low log-loss is desirable, where zero is a perfect log loss.

Loss is a function for calculating the model error in training of deep learning neural networks during predictive modeling. The train dataset has 0 or 1 for belonging to a given feature class while the model will predict a value between 0 and 1 for pictures in the validation or test dataset. The model architecture will estimate weight parameters that best map examples to known output. Predictions close to the correct answer are rewarded while incorrect predictions are punished exponentially as a logarithmic loss. Loss function may be different for loss of membership to a logistic class and for loss of numerical regression. A low loss is desirable, where zero is a perfect loss.

### **AUC**

AUC is the area-under-curve, where the curve is Sensitivity or a true positive rate on the y-axis and a false positive rate on the x-axis. An ideal curve will look like the letter L upside-down. The area under the curve is regarded to be a summary of the model skill.

### **Precision**

Precision will express the number of correct predictions of all predictions, including False Positives. A high precision express that recognition of the

class was very exact.

Precision will express the number of correct predictions of e.g. 'angry' i.e. True Positives, divided not only by the number of correct predictions of 'angry'- but also including the incorrect predictions of 'angry' i.e. from False Positives of cases that really were other emotions like 'sad' or 'neutral'. Low precision will indicate a large number of False Positives. High precision means less false classifications and indicate that a large amount of specific feature classifications were actually correct. A high precision express that recognition of the class was very exact.

### **Recall**

Recall will express the number of correct predictions of an emotion relative to the total number of only those emotions. A high recall express that recognition of the class was almost complete.

Recall will express the number of correct predictions of e.g. 'angry' i.e. True Positives, divided not only by the number of correct predictions of 'angry'- but also including the incorrect rejections of 'angry' i.e. False Negatives. Recall is the number of correct predictions of an emotion relative to the total number of only those emotions. A low recall indicates many False Negatives. High recall means less incorrect rejections and indicate that a large amount of a specific feature was correctly recognized. A high recall express that recognition of the class was almost complete.

### **F-measure**

An F-measure will consider the relationship between the number of True/ False Positives and True/ False Negatives and make functions of combinations.

The F1 Score may be seen as a model that will express the balance between precision and recall. While precision may summarize the 'exactness' of the model, recall may summarize the 'completeness' of the model. The F1 Score is expressed as equation (2.1).

$$F_1 = 2 \left( \frac{precision * recall}{precision + recall} \right) \quad (2.1)$$

If both have the same value, F1 will also get that value.

Precision and Recall may be an average of results obtained for each classified emotion or may have weights added. Results obtained from Precision and Recall may show if predictions are more exact than they are complete.

### **Confusion matrix**

For a classification problem it is also very useful to create a two-dimensional cross matrix of number of features that are looked at, with

known features in one direction and the same estimated features in the other direction. A populated diagonal with numbers for correctly interpreted features will then appear, hopefully with as few as possible incorrectly interpreted features outside the diagonal. As an example, disgust may easily be interpreted as angry, fearful, neutral or even happy. Such a matrix is called a Confusion Matrix.

Presentation of results from testing the model is done in a main Chapter 5, and results are discussed in Chapter 6.

## 2.7 CNN models

Before use of effective methods can be employed for image interpretations in this thesis, it is necessary to investigate which innovations have been made lately in the field of Artificial Intelligence (AI), Artificial Neural Networks (ANN) and Convolutional Neural Network (CNN).

A CNN is an assembled architecture consisting of several layers that filter, sort and classify information from a digital source like a picture.

Each of the currently widely used architectures brought significant scientific advances to the market, especially during the yearly ImageNet Large Scale Visual Recognition Challenge (ILSVRC) challenges, the ImageNet Large Scale Visual Recognition Challenge. The ILSVRC challenges resulted in rapid advancement in computer vision interpretations and the development of successful architectures as convolutional neural network models.

Great advances in creating functional CNN's were made through innovations in projects like LeNet and ImageNet, that resulted in architectures like AlexNet, VGG, Inception, and ResNet. These successful architectural innovations for computer vision tasks were made fairly recent, mainly in the period of 2012 to 2016.

### 2.7.1 LeNet

LeNet-5 describes the network as having seven layers with input of grayscale images with 32×32 pixels. A pattern with a convolutional layer followed by an average pooling layer is repeated two and a half times before output feature maps are transferred to connected layers for classification and a final prediction (Lecun et al., 1998).

### 2.7.2 AlexNet

The AlexNet architecture has 5 convolutional layers in the feature extraction part of the model and 3 layers in the classifier part of the model. Convolutional layer were necessarily not followed by pooling layers but were fed directly to another convolutional layer (Krizhevsky et al., 2017).

### 2.7.3 VGG16

The VGG architecture makes use of a large number of small filters, down to the size  $3 \times 3$  and  $1 \times 1$ . This is different from use of large sized filters in LeNet-5 and relatively smaller filters in AlexNet. VGG also stack convolutional layers after each other, however by use of smaller pooling filters to simulate the effect of a single convolutional layer with a larger sized pooling filter. Different versions of the architecture are deep with use of 16 or 19 layers (Simonyan and Zisserman, 2014).

### 2.7.4 Inception

The Inception/GoogLeNet architecture use a set of parallel LeNet convolutional layers, however consisting of different sized pooling filters (e.g.  $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$  in each channel). When large filters from several layers or channels are used, resource use becomes excessive, especially if modules are stacked. Additional convolutions to smaller filters are used to reduce computations and temporary output classifications are used as error feedback at intermediate layers. Development of a very deep network up to 22-layers has been used (Szegedy et al., 2014).

### 2.7.5 ResNet

ResNet or the Residual Network introduced the idea of residual blocks in pairs that make use of shortcut connections where the input is passed on to a deeper layer, skipping the next, without use of step-wise pooling filters, as is used in the VGG architecture. Repetition of pairs of residual block are stacked to a very deep model of up to 152-layers, with a pooling filter at the end (He et al., 2015).

For reference, error rate during the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) challenges for the described architectures were; AlexNet obtained an error rate of 0.154, the derived ZF-Net got 0.112, VGG19 0.073, Inception/GoogLeNet 0.067 and ResNet obtained a very low 0.036 .

The Residual Neural Network (ResNet) is one of the latest CNN that were developed for Artificial Neural Networks (ANN). The ResNet architecture consists of more than hundred layers, which is considered to be a very deep CNN.

ResNet is a particularly suited learning architecture for Image Recognition, and often outperforms other popular Convolutional Neural Networks. This was demonstrated during the ILSVRC challenges, where it won in 2015.

More specifically, ResNet is a CNN design that overcomes the "vanishing gradient" problem, by creating a network of numerous convolutional layers that numerically outperform more shallow networks. The model makes

predictions with a given set of weights and the error is calculated in each layer.

A gradient descent algorithm will change weights and gradually reduce the error, meaning that the error will continue improvement and still have a gradient to better results.

## 2.8 Transfer learning

Normally, an added convolutional layer close to your Baseline Model input layer may learn simple one-dimensional features such as lines. Layers in the middle may combine extracted low level features and learn more complex features, and layers closer to the output may classify the extracted feature and give weights to different types of recognition. To train and optimize your own model on your own database may be very time-consuming.

The motivation for the transfer learning process was to decrease the requirement for extended training sets. The approach of modifying the weights and parameters of an already trained network to meet your goal is known as transfer learning. It is sometimes referred to as using a pre-trained network.

### 2.8.1 Pre trained models

A large number of image classification models have been developed over time, such as for the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

It would be a great advantage for development of a new deep convolutional neural network model if these innovations in architectures and training could be utilized. Models are already trained on millions of images in numerous categories and have found feature weights that give excellent performance for specific tasks.

These pre-trained models are usually freely accessible by API's and parts of them can be used as-is as an integrated part of new models. Parts will act as a feature extraction algorithm, or will use found feature weights as starting points for new interpretation problems. Fortunately, Keras are available and give access to pre-trained CNN's and will download model weights from their libraries. Suitable CNN's are of the type that use repeating structures (VGG), inception modules (GoogLeNet) or residual modules (ResNet).

From the keras applications (Keras Documentation, n.d.), we can pick and select any of the state-of-the-art models and use it for our problem.

Pictures from CNN models have a predefined format. When models are loaded into a new Baseline Model from their database, images may be

## Available models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201

Figure 2.11: List of available models in keras. From *Keras Documentation*, by Keras Documentation n.d.

specified to have a new pixel input shape or even be in a grey-scale format to suit training or model weight calculations in your own model.

If however you would like to benefit from what these architectures already have obtained in feature recognition or in model weight calculations, your own picture format must be transformed to what these models require:

- The VGG16 Pre-trained Model is expected to get square shaped color images of size 224×224
- The ResNet50 Pre-trained Model is expected to get square shaped color images of size 224×224

Taking advantage of these pre-trained CNNs as a component for feature or weight calculation in your own computer vision code for e.g. image- and emotion recognition is called Transfer Learning.



## Chapter 3

# Literature Review

A Literature Review will show that use of Convolutional Neural Network (CNN) is widely used for image interpretations. Specifically for the purpose of facial emotion recognition by use of CNN's, a selection of papers are reviewed.

The paper *ResNet-50 and VGG-16 for recognizing Facial Emotions* by Dhankhar from 2019, explored the VGG-16 and ResNet50 architectures for recognizing facial emotions using deep learning. They also developed a Support Vector Machine (SVM) classifier baseline model for comparison purposes. The article used both the ( ) dataset and the Karolinska Directed Emotional Faces (KDEF) dataset for testing of their two models. In the introduction they state that they expect their best model to achieve at least 60% test set validation because the winner of Kaggle's Facial Expression Recognition Challenge achieved 71.2% accuracy and the top ten contestants achieved at least 60% accuracy.

On the KDEF dataset, overall results on accuracies, precision, and recall with their models were higher than on the dataset. Surprisingly, all four models gave better results on the KDEF dataset compared to the dataset, which is a much smaller dataset. However, the photos in the KDEF data collection are of greater quality and the dataset contained examples of text overlay in the image's background. Below, the results on the KDEF dataset will be shown and what results application of transfer learning could give. The results of the SVM (baseline), VGG-16, ResNet50 on the KDEF dataset are shown in Table 3.1 below.

	Accuracy	Precision	Recall
SVM (baseline)	37.9 %	50.1 %	54.9 %
VGG-16	71.4 %	81.9 %	79.4%
ResNet50	73.8 %	83.3 %	80.7%

Table 3.1: KDEF dataset performance. From *ResNet-50 and VGG-16 for recognizing Facial Emotions* by Dhankhar 2019.

Applying transfer learning further improved the results. The results

after transfer learning was implemented is shown in Table 3.2 below.

	Accuracy	Precision	Recall
VGG-16	73.6%	84.2%	81.1%
ResNet50	76.0%	86.1%	82.5%

Table 3.2: KDEF dataset performance with Transfer Learning. From *ResNet-50 and VGG-16 for recognizing Facial Emotions* by Dhankhar 2019.

The findings were even better when transfer learning was used. They fixed the layer weights of the pre-trained models except for the last few layers and retrained them on the KDEF dataset. Resnet-50 improved from 73.8% to 76.0%. Precision and recall both improved in the same way. This demonstrates that their model was able to take learnings from datasets, and apply them to the KDEF dataset.

The paper *Deep learning-based facial emotion recognition for human-computer interaction applications* by Chowdary et al. from 2021, studies emotion recognition techniques by using transfer learning approaches. In this work, pre-trained networks of Resnet50, VGG19, Inception V3, and Mobile Net are used. The fully connected layers of the pre-trained ConvNets are eliminated, and we add our fully connected layers that are suitable for the number of instructions in our task. Finally, the newly added layers are only trainable to update the weights. The experiment was conducted by using the CK+database and achieved an average accuracy of 96% for emotion detection problems.

The paper *Face Recognition using Transfer Learning on VGG16* by Mittal from 2020, used VGG16 and was trained for weeks, using an NVIDIA Titan Black GPUs. Applying transfer learning gave results presented in 3.3

	Accuracy	Loss
VGG16	97.3 %	0.63 %

Table 3.3: From *Face Recognition using Transfer Learning on VGG16* by Mittal 2020.

The paper *Automatic facial recognition using VGG16 based transfer learning model* by Dubey and Jain from 2020, used a VGG16 based transfer learning model that showed 94.8% accuracy on CK+ and 93.7% on the JAFFE dataset, and was superior to existing techniques. The proposed technique was implemented on Google Colab-GPU that was helpful for processing these data.

The paper *Improvement of emotion recognition from facial images using deep learning and early stopping cross validation* by Mohamed et al. from 2022, present a new approach for emotion recognition from facial images. The proposed method is based on the association of a pretrained CNN models

(VGG16, ResNet50) with a multilayer perception (MLP) classifier. The pre-trained CNN model is used as a feature extractor. They adapt the original architecture by adding a global average pooling layer (GAP) without any fine tuning of the network parameters. In order to avoid overfitting for the MLP classifier, the early stopping criterion was introduced.



# Chapter 4

## Methodology

### 4.1 Existing software

In order to start working with machine learning in Python, a platform for programming must be created.

Python for the operating system in use must be downloaded and installed (Linux, OS X or Windows). A package manager may be needed to also install the SciPy platform and the scikit-learn library - or everything may be installed at once with Anaconda.

The code then becomes rather simple, starting with Python version;

#### **Numpy**

```
import numpy
```

#### **Matplotlib**

```
import matplotlib
```

#### **Pandas**

```
import pandas
```

#### **Scikit-learn**

```
import sk-learn
```

#### 4.1.1 Python

Python is one of the largest programming languages for applied machine learning, available for the Linux-, OS X- and Windows-operating systems. It has proven efficient for building accurate numerical models for use in AI, ANN and machine learning projects.

Of great importance to a working Python environment is the availability of compatible software packages. A package like Anaconda is made for the

development environment to ease implementations, a main library used for machine learning is scikit and libraries used for deep learning are Keras, Theano and TensorFlow.

Use of these packages are via free open-source licenses and may be downloaded to a computer in ready-made bundles for ease of installation. Many packages are made as an Application Program Interface (API) that may plug-in as ready made for other packages.

#### **4.1.2 Keras**

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. Keras is a more high-level API of TensorFlow 2 that makes it a user friendly interface for solving machine learning problems by reducing the number of user actions required for common programming tasks and by providing more understandable error messages.

The focus is on creating building blocks to enable rapid development and deploying machine learning solutions to many platforms, including mobile phones.

#### **4.1.3 Scikit-learn**

The main library for machine learning in Python is scikit-learn. It will allow saving and loading the machine learning models that make numerous predictions.

A disadvantage with interdependent software environments and contributions from many developers is that the Python versions used in every part of the assembled code must carefully be noted in order to avoid compatibility issues in the software development process.

#### **4.1.4 Tensorflow**

TensorFlow is an open source library for fast numerical computing created by Google, especially made for Python programming, although it is created in the more low-level language C++.

The TensorFlow installation is often made via packages like PyPI and comes with a number of Deep Learning models that can be experimented on directly in a web browser. It also has an examples directory and a list of tutorials to show how different network types and datasets can be used.

### **4.2 Metrics used**

In our case where membership to a class shall be predicted, the following metrics are used:

### **Accuracy**

Accuracy is used to measure the number of correct predictions in relation to all predictions made for a feature, disregarding whether the prediction was correct or wrong. An accuracy score of 75% to 95% is desirable.

### **Loss**

Loss is used to check the predicted probability between 0 and 1 of belonging to a given class. A low log-loss is desirable.

### **AUC**

AUC is used to check the model skill, where the area-under-curve is a true positive rate versus a false positive rate. The result should approach the looks of a letter L upside-down.

### **Precision**

Precision is used to check if recognition of the class is very exact.

### **Recall**

Recall is used to check how complete the recognition of the class is.

### **F-measure**

The F-measure displays the relationship between Precision and Recall.

### **Confusion matrix**

A Confusion matrix is used to check if the matrix diagonal is populated with numbers for correctly interpreted features.

## **4.3 Datasets**

We shall introduce a few datasets in this part. The datasets are chosen to be as large and diversified as feasible in terms of quantity and size.

### **FER-2013**

The Facial Emotions Recognition dataset is available as both .csv file and .jpg folders.



Figure 4.1: Extraction of a few images from FER dataset with unique faces and emotions

## JAFFE

The Japanese Female Facial Expression dataset is available as a .csv folder. JAFFE consists of 213 images of the 7 facial expressions from 10 different Japanese females.



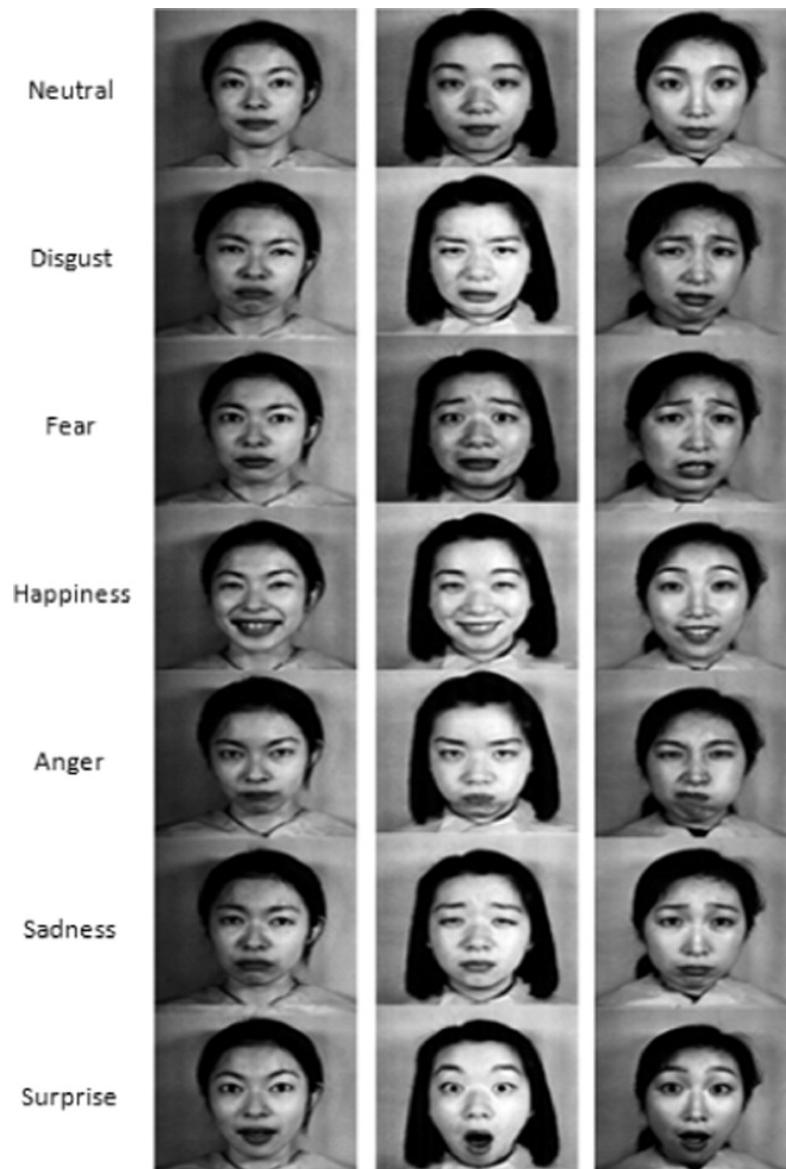


Figure 4.2: Extraction of a few images from JAFFE dataset with corresponding labels to the left and three unique faces on the right

## 4.4 Implementation

Implementation of a model can be done after a database has been made available on a computer by

1. Collecting a specific set of data
2. Saving data in a data structure
3. A model can then be developed to a stage for testing when the following elements are completed:
  - (a) Loading of the dataset

- (b) Preparation of the dataset
- (c) Definition of the model
- (d) Evaluation of the model
- (e) Presentation of performance results

Based on theories as outlined in Chapter 2, a walk-through of the implementation is presented in the following subsections. For further details, the model code and algorithms can be consulted. The figure below shows the data flow of how the models will be trained. All models will get raw images, pre-processed images, but only the pre-trained models will be used with transfer learning. A trained prediction model is the result.

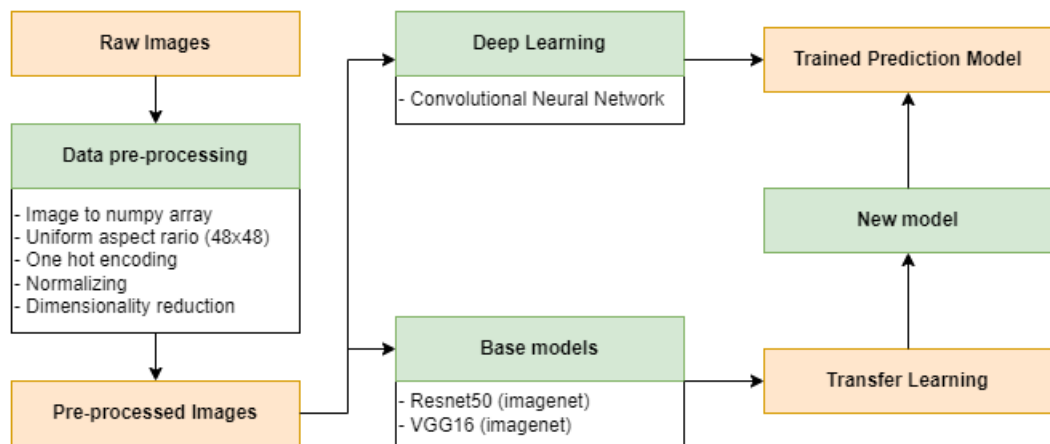


Figure 4.3: Emotion Recognition system architecture. Arrows show the work flow in sequence starting with loading Raw images and ending with a trained prediction model. By E. Kirkvik

#### 4.4.1 Database collection

This research will collect datasets from databases that are organized in a comprehensive, systematic and sufficiently detailed manner. Based on literature studies in Chapter 3, the FER-2013 database will be used as a training, validation and test set. There are two types of the FER-2013 database available online, where one database contains folders of .jpg images of faces (Kaggle, 2020) and the other one is a .csv file (Kaggle, 2018) where the images have been converted to an array.

If pictures only exist as .jpg, pixels can be converted to a NumPy array for treatment with Keras. A function `img-to-array()` exist for this task.

The images will need to be reshaped to a 4-dimensional format of the form `reshape(samples, pixelrows, pixelcolumns, colors)`. If image pixels shall be treated by pre-trained models, they need to be prepared to the same ImageNet format the models used. Keras also provides a function `preprocess-input(image)` for use with the network models.

To distinguish the two datasets, they will further be mentioned as *FER-2013 .jpg* and *FER-2013 .csv*.

#### 4.4.2 Dataset Directory Structure

Subdividing data sets is done in a structured manner. According to Brownlee (2019), there is a standard way to lay out image data for modeling. After collecting the database containing the .jpg images, *FER-2013 .jpg*, they are sorted first by dataset, such as train, validation and test, and secondly by their feature class.

First, we create a `data/` directory to store all of the image data.

Next, the training dataset will be stored in a `data/train/` directory, while the test dataset will be stored in a `data/test/` directory. During training, we may also have a `data/validation/` folder for a validation dataset, but this must be created manually as the database downloaded did not contain a separate validation folder.

Based on this we get the following directory structure on the images:

```
data/
├── data/train
├── data/test
└── data/validation
```

For the dataset *FER-2013 .jpg*, it has seven different classes, which are sadness, anger, disgust, fear, happiness, neutral and surprise. Under each of the dataset directories, we will have subdirectories, one for each class. For my dataset there will therefore be seven class directories under each dataset directory. Images of for example angry faces would then be placed in the appropriate class directory.

An overview of the final directory structure:

```
data/
├── data/train
│   ├── data/train/angry/
│   │   └── data/train/angry/Training3908.jpg
│   ├── data/train/disgust/
│   ├── data/train/fear/
│   ├── data/train/happy/
│   ├── data/train/neutral/
│   ├── data/train/sad/
│   └── data/train/surprise/
├── data/test
│   ├── data/train/angry/
│   └── data/train/disgust/
```

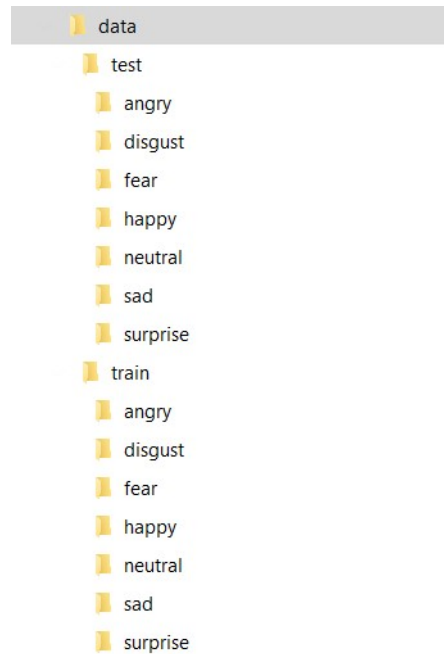
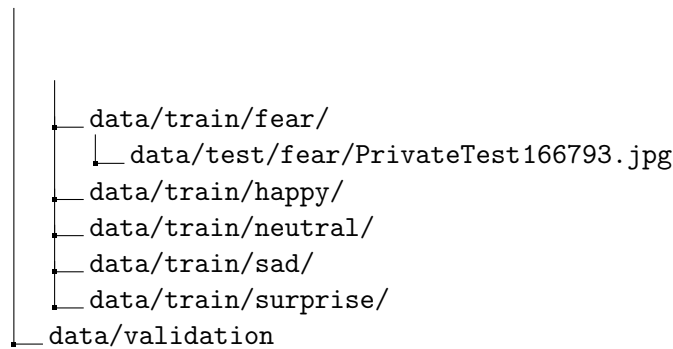


Figure 4.4: Image of computer folders illustrating dataset directory structure

An alternative to Subdividing data sets is to split and analyse the whole dataset in k-folds.

#### 4.4.3 Loading datasets and images

Manually loading image data and returning data appropriate for modeling is done with the Python code. This would e.g. be navigating within a dataset's directory structure, loading image data, print the input as checks of correct loading, giving info on pixel arrays such as range for colors between 0 and 255 and returning image data with feature class integers.

The *FER-2013 .csv* database contains the same amount of images, only it is converted to an array of images in a .csv folder. It contains the columns with names: *'emotion'*, *'pixels'* and *'Usage'*. I download the *FER-2013 .csv* to my computer locally and add a path to it in the Python code. The dataset is then defined with the variable *data*, and the dataset can be visually presented by calling *data.head()* which prints the first rows and corresponding rows. This is done with the following code below.

```

1 # Create path to dataset
2 path='C:/data3/fer2013.csv'
3 # Load the dataset
4 data=pd.read_csv(path)
5 # Look at 5 first rows and corresponding columns
6 print(data.head())
7 # More about the dataset
8 print(data['Usage'].value_counts())

```

Listing 4.1: Create dataset path and verify database structure

We can verify the database structure by visually looking at the output which is shown in Figure 4.5 below.

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98 9	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 154 153 164 173 11	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38 39 74 138 161 1	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 19 43 52 13 26 40 5	Training
4	6	4 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84 115 127 137 142 151	Training
5	2	55 55 55 55 55 54 60 68 54 85 151 163 170 179 181 185 188 188 1	Training
6	4	20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 108 118 130 139 134	Training
7	3	77 78 79 79 78 75 60 55 47 48 58 73 77 79 57 50 37 44 56 70 80 8	Training
8	3	85 84 90 121 101 102 133 153 153 169 177 189 195 199 205 207 2	Training
9	2	255 254 255 254 254 179 122 107 95 124 149 150 169 178 179 17	Training
10	0	30 24 21 23 25 25 49 67 84 103 120 125 130 139 140 139 148 171	Training
11	6	39 75 78 58 58 45 49 48 103 156 81 45 41 38 49 56 60 49 32 31 28	Training
12	6	219 213 206 202 209 217 216 215 219 218 223 230 227 227 233 2	Training
13	6	148 144 130 129 119 122 129 131 139 153 140 128 139 144 146 14	Training
14	3	4 2 13 41 56 62 67 87 95 62 65 70 80 107 127 149 153 150 165 16	Training
15	5	107 107 109 109 109 109 110 101 123 140 144 144 149 153 160 16	Training
16	3	14 14 18 28 27 22 21 30 42 61 77 86 88 95 100 99 101 99 98 99 99	Training
17	2	255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 2	Training

Figure 4.5: Image of PyCharm console of first rows of dataset and its corresponding columns: emotion, pixels, usage. Can be shown in console by running data.head()

By printing the 'Usage' column, we can see that it is already divided into three categories; 'Training', 'PublicTest' and 'PrivateTest'.

```

In [116]: data['Usage'].value_counts()
Out[116]:
Training      28709
PublicTest    3589
PrivateTest   3589
Name: Usage, dtype: int64

```

Figure 4.6: Image of console output returning total amount of images under columns 'Usage' where it is counted 28709 with label 'Training', 3589 with label 'PublicTest' and 3589 with 'PrivateTest'

This shows us that the total dataset consists of a total amount of data

values:

$$28709 + 3589 + 3589 = 35887$$

And 10% PublicTest and 10% PrivateTest, resulting in 80% training data:

$$35887 * 0.1 = 3588.7$$

We want to use this categorization further and define it in the code by these simple lines:

```
1 # Define train images to be the ones under 'Usage' Training
2 train_data = data.loc[data.Usage=="Training"]
3 # Define validation images to be the ones under 'Usage'
  PublicTest
4 val_data = data.loc[data.Usage=="PublicTest"]
5 # Define test images to be the ones under 'Usage' PrivateTest
6 test_data = data.loc[data.Usage=="PrivateTest"]
```

Listing 4.2: Separating Usage by Training PublicTest and PrivateTest

Because we want to use the k fold split function for splitting train and test, it is preferred to use the whole dataset. We will therefore first combine the validation and test dataset which are 10% each of the total dataset to use as a validation set when fitting the model. This means it would have 20% of the total data for validation. And further combine the test, validation and train set together with a *np.concatenate* function to merge the whole dataset to be used in the k fold function. The code doing this is presented below:

```
1 # Combining validation and test data to be test dataset
2 test_data = pd.concat([test_data, val_data])
3 ...
4 # Merge inputs and targets to use the whole dataset
5 inputs = np.concatenate((X_train, X_test), axis=0)
6 #100% dataset is used
7 targets = np.concatenate((y_train, y_test), axis=0)
8 #100% dataset is used
```

Listing 4.3: Separating Usage by Training PublicTest and PrivateTest

#### 4.4.4 Analyzing dataset

Before pre- processing and presenting the dataset to an ML algorithm, it is vital to look at the downloaded dataset to verify its content and get an idea of what type of pre-processing or image augumentation needs to be done.

##### Show images

The following function will show one random image from each emotion folder to illustrate how the images in the database look. The function will use the total dataset, including test and train folders. This is done by calling the *plot\_emotions()* function below:

```
1 ##### Example images from dataset for each emotion
  #####
2 def plot_emotions():
```

```

3  fig, axis = plt.subplots(1, 7, figsize=(30, 12)) # create
    figure box
4  fig.subplots_adjust(hspace = .2, wspace=.2) #make spaces
    between each image shown
5  axis = axis.ravel() #create a contiguous flattened array
6  for j in range(7): #loop through each of the 7 emotions and
    show the image
7      number = data[data['emotion']==j].index[j]
8      axis[j].imshow(X_train[number][:,:,0], cmap='gray')
9      axis[j].set_title(emotions[y_train[number].argmax()])
10     axis[j].set_xticklabels([])
11     axis[j].set_yticklabels([])

```

Listing 4.4: plot\_emotions function showing one random image from each emotion

Figure 4.7 below shows the output from the console running the function above. One image for each of the emotional labels.



Figure 4.7: Results from running function plot\_emotions() showing an image for each of the seven emotions; angry, disgust, fear, happy, sad, surprise and neutral

Having a further look at the dataset, it is interesting to see the distribution of datapoints for each emotional label. This can be visualized by running the function develop\_piechart. The function will count amount of data points for each emotion and visualize it by use of matplotlib library. The function will use the total dataset, including test and train folders. The code for develop\_piechart is shown below:

```

1  ##### Show distribution of each emotion in a piechart
    #####
2  def develop_piechart():
3      ...
4      plt.figure(1)
5      plt.pie(x=la_num, labels=lab, autopct='%3.1f %%',
        startangle=90)
6      plt.title('Size of each emotion in dataset')
7      plt.axis('equal') # Equal aspect ratio ensures that pie is
        drawn as a circle
8      plt.show()

```

Listing 4.5: develop\_piechart function illustrating distribution of emotions in dataset

The output from running the develop\_piechart() function above will result in a figure plot shown in Figure 4.8 below.

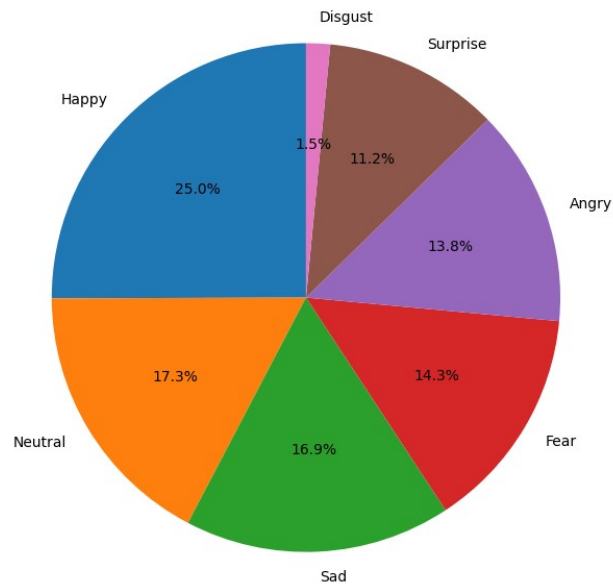


Figure 4.8: Results from running function `develop_piechart()` showing a piechart of the distribution for each of the seven emotions followed by a percentage. These are from lowest to highest: disgust (1.5%), surprise (11.2%), angry (13.8%), fear (14.3%), sad (16.9%), neutral (17.3%) and happy (25.0%)

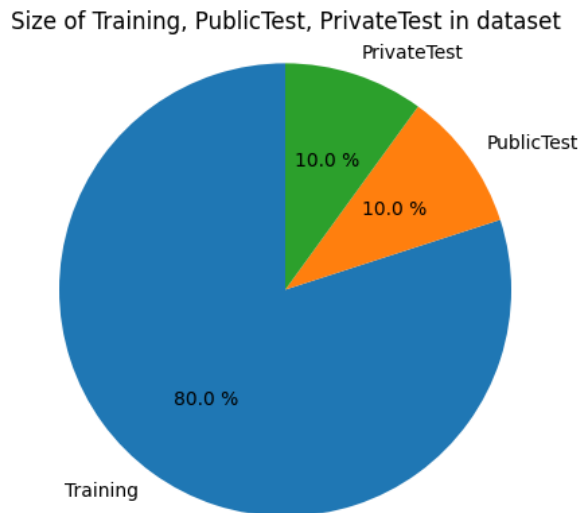


Figure 4.9: A Piechart from running function `develop_piechart()` function. It shows a distribution of 80% train data (Train), 10% validation data (PublicTest) and 10% test data (PrivateTest).



## Dataset category count

As stated in the Kaggle documentation where the *FER-2013* dataset was downloaded from, it contains 7 emotions. It is important to plot a category count of the amount of each category to verify if the dataset is balanced or not. I created a function in my code that prints out the amount of images for each of the seven emotional categories by use of the matplotlib library. The code is attached below:

```
1 def plot_bar():
2     "Function that plots the distribution of image amount for
3     each emotional category"
4     plt.bar(range(len(la_num)), la_num, tick_label=lab)
5     for a, b in zip(lab_no, la_num):
6         plt.text(a, b + 0.05, '%.0f' % b, ha='center', va='
7         bottom', fontsize=10)
8     plt.title('Size of each emotion in whole dataset')
9     plt.show()
```

Listing 4.6: plot\_emotions function showing one random image from each emotion

The output is shown in Figure 4.10 below:

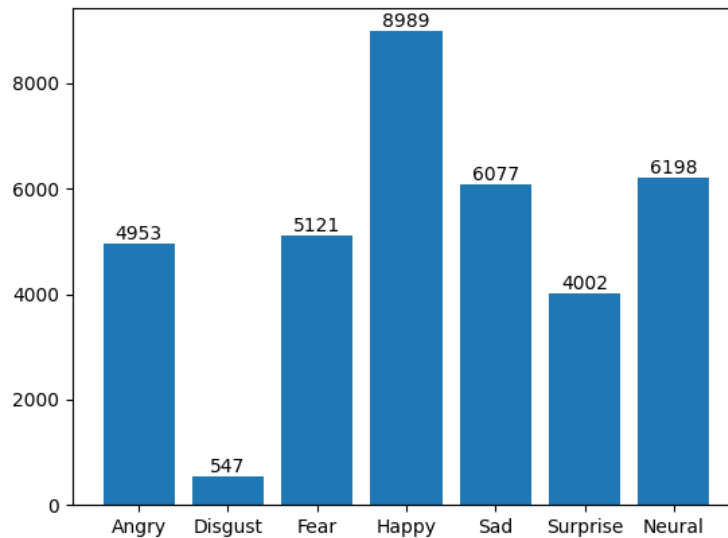


Figure 4.10: The horizontal X-axis shows each emotional feature category, named; angry, disgust, fear, happy, neutral, sad and surprise. The vertical Y-axis shows number of counted images, ranging from approximately 500 images to 9000 images.

The horizontal X-axis shows each emotional feature category, named; angry, disgust, fear, happy, neutral, sad and surprise. The vertical Y-axis shows number of counted images, ranging from approximately 500 images to 9000 images.

#### 4.4.5 Data preprocessing

For pre-processing of data, Keras utilities for pre-processing datasets can be utilized, located in the database `tf.keras.preprocessing`. Utilities can be used to go from raw data on disk to a set of `tf.data`, i.e. dataset that can be used to train a model. For the *FER-2013 .csv* dataset We have the following values of the images:

- Number of images = 35887
- image width, image height = 48 x 48
- 3 channels, pizel levels in range [0-255]

This can be confirmed by running code: `"data.shape()"` in the program and getting the output:

```
In [198]: data.shape
Out[198]: (35887, 3)
```

Figure 4.11: Running `data.shape()` in the program returning (number of images, number of channels), in this case: (35887, 3)

The images already have the same height and width so they have the same aspect ratio. The neural network will expect images of same square shape input, so I do not need to do any form of cropping. I also chose to not do any form of pixel resizing by up-scaling or down-scaling because of the already low image resolution.

The first part of data processing is to convert the images to `nparrays` and 32-bit floating number because the weights of a Neural Network are all `float32`. The next step is to convert the `nparray` to One-hot encoding, the correct input format for an ML algorithm.

Next, the images are normalized by the pixel values for colors, by rescaling 256 colors to be represented by range of [0,1]. This will transfer colors from integers to real numbers. The actual data and the image lables will be extracted separately for the program to be able to run correctly. The variables are defined such as:

- **X\_train** = all train images without corresponding labels. these will be used to train the model
- **y\_train** = all train labels without corresponding data
- **X\_test** = all test images without corresponding labels. Will not be used in the training phase but will be used to make predictions to test the accuracy of the model.
- **y\_test** = all test labels without corresponding data

```

1 # Convert images to an ndarray and 32-bit floating number.
   Extract only data, without labels
2 X_train = np.array(list(map(str.split, train_data.pixels)), np.
   float32)
3 X_test = np.array(list(map(str.split, test_data.pixels)), np.
   float32)
4
5 # Converting to One-hot encoding
6 X_train = X_train.reshape(X_train.shape[0], 48, 48, 1)
7 X_test = X_test.reshape(X_test.shape[0], 48, 48, 1)
8
9 # Normalizing
10 X_train=X_train/255 # Can also be called train images, without
   corresponding labels
11 X_test=X_test/255 # Can also be called test images, without
   corresponding labels
12
13 # Extracting only labels not containing data
14 y_train = train_data.emotion # Can also be called train labels,
   without corresponding data
15 y_test = test_data.emotion # Can also be called test labels,
   without corresponding data
16
17 # Converting to One-hot encoding
18 y_train = np_utils.to_categorical(y_train, 7)
19 y_test = np_utils.to_categorical(y_test, 7)
20
21 # Look at the final formatted data
22 print(f"X_train shape: {X_train.shape} - y_train shape: {
   y_train.shape}")
23 print(f"X_test shape: {X_test.shape} - y_test shape: {y_test.
   shape}")

```

Listing 4.7: Separating Usage by Training PublicTest and PrivateTest

#### 4.4.6 Defining callbacks

Before a neural network can undergo training, a decision must be made on analysis steps and how many epochs shall be used.

Too many epochs can lead to overfitting while too few may result in an underfit model.

Functions presented below measure the model performance during analysis and they may stop the training if further improvements are not found. Checkpoints are set such that the model at best performing epoch is saved and can be called for later use.

```

1 ##### Callbacks function #####
2 lrd = ReduceLROnPlateau(monitor='val_loss', patience=20,
   verbose=1, factor=0.50, min_lr=1e-10)
3
4 mcp = ModelCheckpoint('model.h5')
5
6 es = EarlyStopping(verbose=1, patience=20)

```

Listing 4.8: Separating Usage by Training PublicTest and PrivateTest

#### 4.4.7 Building Sequential model

A Baseline convolutional neural network Model will be established as a typical classification model since emotions are divided into classes.

Since all reviewed literature studies confirm that transfer learning give a clear improvement in predictions for base sequential models, an alternative would then be to investigated how much a complicated baseline model can be reduced to obtain the same accuracy as a less advanced model that use transfer learning.

By building and analysing a rather complicated baseline model and comparing results to a less advanced model that use transfer learning, it will be checked if transfer learning can be used to obtain the same accuracy with significantly smaller models. Transfer learning can then be used as a 'performance optimizer' on smaller baseline models that give sufficiently accurate results with a minimum use of resources.

First, a rather complex baseline sequential model is built. Transfer Learning on a smaller model will later be used in comparisons with this model, and results will show how much benefit this gave to a new and modified model. Transfer Learning from several architectures can be used but VGG and ResNet will be used. The baseline model built from scratch is illustrated below.

The model is called Sequential due to its sequence of convolutions and pooling layers. A CNN network structure is defined as described below.

A structure with four convolutional layers followed by max pooling and dropout definitions is used, with a flattening out of the network to fully connected layers for making predictions.

The Sequential() structure is summarized as follows:

- **Convolutional input layer** 32 feature maps with a size of  $3 \times 3$ , a rectifier activation function and a weight constraint of max norm set to 3, for  $48 \times 48$  monocrome images.
- **Convolutional layer** 64 feature maps with a size of  $3 \times 3$ , padding and a rectifier activation function.
- **Max Pool layer** with size  $2 \times 2$ .
- **Dropout** set to 25%.
- **Convolutional layer** 128 feature maps with a size of  $3 \times 3$ , padding, a rectifier activation function and kernel regularisers.
- **Max Pool layer** with size  $2 \times 2$ .
- **Dropout** set to 25%.
- **Convolutional layer** 256 feature maps with a size of  $3 \times 3$ , padding, a rectifier activation function and kernel regularisers.
- **Max Pool layer** with size  $2 \times 2$ .
- **Dropout** set to 25%.

- **Flatten layer** Included
- **Fully connected layer** with 512 units and a rectifier activation function.
- **Dropout** set to 50
- **Fully connected output layer** with 7 units and a softmax activation function.

A logarithmic loss function is used with the cross entropy algorithm and defined metrics are used.

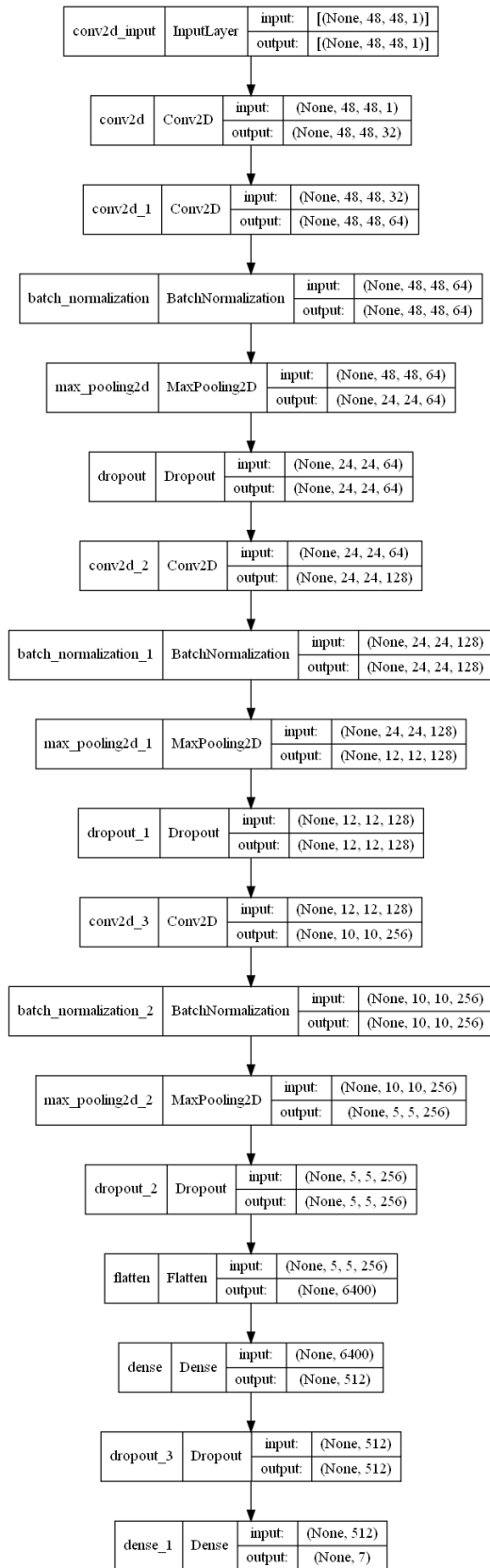


Figure 4.12: Illustration of the CNN Sequential model

```

1  model = Sequential()
2  model.add(Conv2D(32, kernel_size=(3, 3), padding='same',
3  activation='relu', input_shape=(48, 48, 1)))
4  model.add(Conv2D(64, kernel_size=(3, 3), activation='relu',
5  padding='same'))
6  model.add(BatchNormalization())
7  model.add(MaxPooling2D(pool_size=(2, 2)))
8  model.add(Dropout(0.25))
9
10 model.add(
11     Conv2D(128, kernel_size=(3, 3), activation='relu',
12     padding='same', kernel_regularizer=regularizers.l2(0.01))
13 model.add(BatchNormalization())
14 model.add(MaxPooling2D(pool_size=(2, 2)))
15 model.add(Dropout(0.25))
16
17 model.add(Conv2D(256, kernel_size=(3, 3), activation='relu',
18     kernel_regularizer=regularizers.l2(0.01))
19 model.add(BatchNormalization())
20 model.add(MaxPooling2D(pool_size=(2, 2)))
21 model.add(Dropout(0.25))
22
23 model.add(Flatten())
24 model.add(Dense(512, activation='relu'))
25 model.add(Dropout(0.5))
26
27 model.add(Dense(7, activation='softmax'))
28 model.compile(optimizer='adam',
29               loss='categorical_crossentropy',
30               metrics=METRICS)

```

Listing 4.9: Separating Usage by Training `PublicTest` and `PrivateTest`

If a larger network should be designed for the problem, more feature maps could be used closer to the input and less pooling could then be used.

Simple procedures for choosing the right loss exist. If there is a classification problem with more than two categories, a loss called "categorical\_crossentropy" is a natural choice. This is the same for all the models as it will load a dataset with 7 emotional categories.

The model is later fitted with the train dataset, using a batch size of 32 and with 20-50 epochs.

Better end results will normally be obtained with more epochs. A run with up to 150 epochs was at one time tested to see if the rate of convergence improved significantly. If there is no significant improvement, fewer epochs are used.

#### 4.4.8 Building VGG16 model

The model's architecture cannot be changed because the weights have been trained for a specific input configuration. This is why a property of the image must be changed from 1 channel grey scale to a 3 channel rgb image.

I also run the pre-trained model on only 20 epochs compared to the sequential model that is run on 50 epochs. This is because the VGG and Resnet models are already pre-trained and does not need as many epochs as a non-pre-pretrained models.

The ImageNet collection is known to contain photos of various vehicles (sports cars, pick-up trucks, minivans, etc.). We can still import a model that has been pre-trained on the ImageNet dataset and extract features using its pre-trained layers. We can however not use the complete architecture of the pre-trained model. Our Target Dataset has only seven classes for prediction, while the Fully-Connected layer generates 1,000 possible for output labels. So we will take a pre-trained model like VGG16 and "chop off" the Fully-Connected layer (also known as the "top" model).

```
1 def VGG_model():
2     # Loading the vgg16 model
3     # VGG model must receive 3 channel data. Resizing 48x48x1
4     # to 48x48x3
5     img_input = Input(shape=(48, 48, 1))
6     img_conc = Concatenate()([img_input, img_input, img_input])
7     base_model = VGG16(include_top=False, weights="imagenet",
8     input_tensor=img_conc, classes=7)
9
10    for layer in base_model.layers:
11        layer.trainable = False
12
13    # Create a new 'top' of the model (i.e. fully-connected
14    # layers).
15    # This is 'bootstrapping' a new top_model onto the
16    # pretrained layers.
17    model = Sequential()
18    model.add(base_model)
19    model.add(Flatten())
20    model.add(Dense(4096, activation='relu'))
21    model.add(Dense(1072, activation='relu'))
22    model.add(Dropout(0.2))
23    model.add(Dense(7, activation='softmax'))
24
25    # summarize the model
26    #model.summary()
27
28    plot_model(model, to_file='VGG_model.png', show_shapes=True
29    , show_layer_names=True)
30
31    # Compiles the model for training.
32    model.compile(optimizer='sgd',
33                  loss='categorical_crossentropy',
34                  metrics=METRICS)
35
36    return model
```

Listing 4.10: Separating Usage by Training PublicTest and PrivateTest



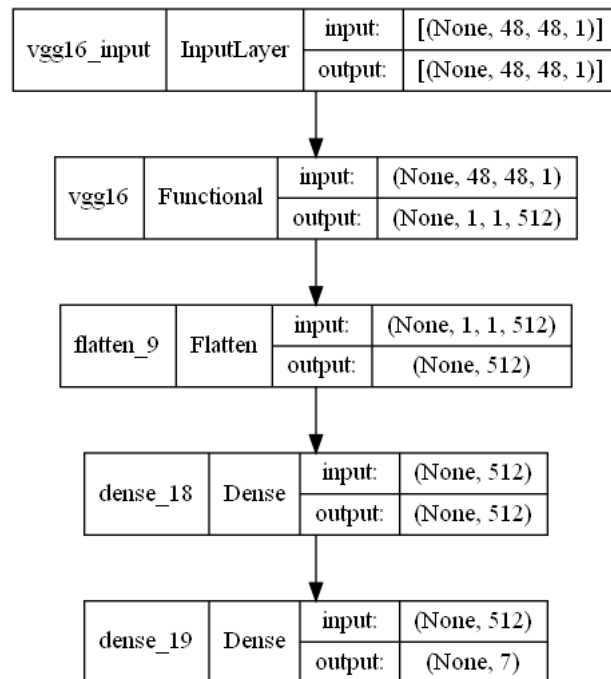


Figure 4.13: The pre-trained VGG16 model with weights 'imagenet' illustration of each layer. The trainable layers are frozen, and one Flatten, Dense(512) and Dense(7) layers are added at the end.

#### 4.4.9 Building Resnet50 model

As an example of existing model architectures, the Resnet50 model used in pre-training is illustrated. The weight "imagenet" is implemented in the model. The model's architecture cannot be changed because the weights have been trained for a specific input configuration. This is why a change of a property of the image from 1 channel grey scale to a 3 channel rgb image must be done.

I also only run the pre-trained model on 20 epochs compared to the sequential model that was run on 50 epochs. This is because the Resnet model is already pre-trained and does not need as many epochs as a non-pre-trained model.

```

1 def resnet_model():
2     #from: https://chroniclesofai.com/transfer-learning-with-keras-resnet-50/
3     img_input = Input(shape=(48, 48, 1))
4     img_conc = Concatenate()([img_input, img_input, img_input])
5
6     base_model = tf.keras.applications.ResNet50(weights='
7     imagenet', include_top=False,
8         input_tensor=img_conc)
9
10    for layer in base_model.layers:
11        layer.trainable = False

```

```

12 model = Sequential()
13 model.add(base_model)
14 model.add(Flatten())
15 model.add(Dense(512, activation='relu'))
16 model.add(Dense(7, activation='softmax'))
17
18 model.summary()
19 plot_model(model, to_file='resnet_model.png', show_shapes=
20 True, show_layer_names=True)
21
22 model.compile(optimizer='adam', loss='
23 categorical_crossentropy', metrics=METRICS)
24 return model

```

Listing 4.11: Separating Usage by Training PublicTest and PrivateTest

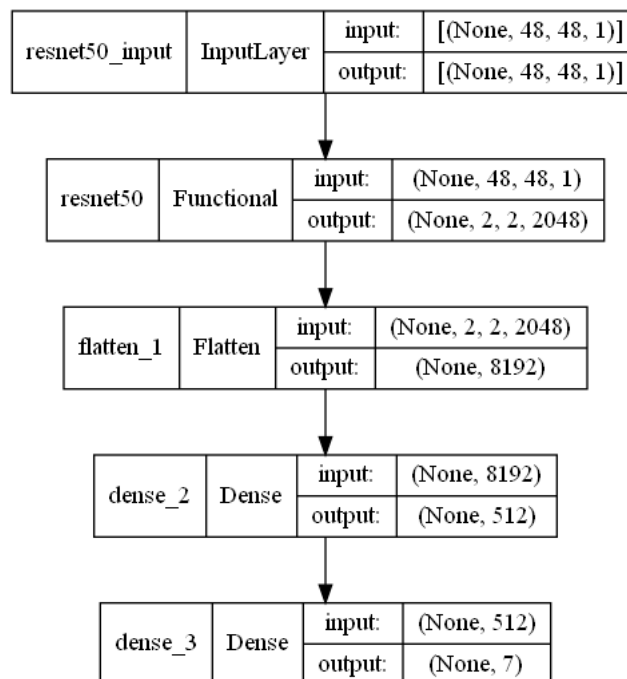


Figure 4.14: The pre-trained Resnet50 model with weights 'imagenet' illustration of each layer. The trainable layers are frozen, and one Flatten, Dense(512) and Dense(7) layers are added at the end.

#### 4.4.10 Tracking and running experiments

```

1 # Define the K-fold Cross Validator
2 kfold = KFold(n_splits=10, shuffle=True)
3
4 # K-fold Cross Validation model evaluation
5 fold_no = 1
6 for train, test in kfold.split(inputs, targets):
7     model = Sequential_model() #call on Sequential_model(),
8     resnet_model() or VGG_model()
9
10    # Generate a print

```

```

10     print('-----')
11     print(f'Training for fold {fold_no} ...')
12
13     #steps_per_epoch = train_dataset.n // train_dataset.
14     batch_size
15     #validation_steps = valid_dataset.n // valid_dataset.
16     batch_size
17
18     # Fit data to model
19
20     history = model.fit(x=inputs[train], y=targets[train],
21                       batch_size=32,
22                       callbacks=[lrd, mcp, es],
23                       epochs=50,
24                       validation_data=(inputs[test], targets[
25     test]),
26                       verbose=1)
27
28     ...
29     # Increase fold number
30     fold_no = fold_no + 1

```

Listing 4.12: Separating Usage by Training PublicTest and PrivateTest

#### 4.4.11 Extracting results

Using `model.evaluate()` from Keras requires the x values: (testing) and y values (testinglabel) from the data set and will produce the loss value and metrics values for the model in test mode. This is used during my testing and validation phase.

```

1 # Generate metrics
2 Train_Val_Plot(history.history['accuracy'], history.history
3 ['val_accuracy'],
4               history.history['loss'], history.history['
5 val_loss']
6               )
7
8 Train_Val_Plot_All(history.history['auc'], history.history[
9 'val_auc'],
10                  history.history['recall'], history.
11 history['val_recall'],
12                  history.history['precision'], history.
13 history['val_precision'],
14                  history.history['f1_score'], history.
15 history['val_f1_score']
16                  )
17
18 # Evaluate model on test set
19 scores = model.evaluate(inputs[test], targets[test],
20 verbose=0)
21 ...
22 # Predict on test set
23 results = model.predict(inputs[test], batch_size=128)
24 predicted_classes = np.argmax(results, axis=1)
25 y_classes = np.argmax(targets[test], axis=1)
26 ...
27 # Create confusion matrix per fold

```

```

22 cm = confusion_matrix(y_classes, predicted_classes,
23                        normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=lab)

```

Listing 4.13: Separating Usage by Training PublicTest and PrivateTest

#### 4.4.12 Predicting on new dataset

Keras model.predict() only get the input data (testing) and produces the output from the trained model. It does not know anything about the actual expected value (testinglabel).

Use model.predict() in production, to just get the model output.

When predicting on a new dataset that has not been used to testing, it is important that the data pre-processin is done in exactly the same way as on the testing dataset. I therefore initialize all the same pre-processing steps on the image as mentioned earlier, before calling it to function *plot\_image\_and\_emotion*

```

1 pred1_test = list(modelVGG.predict(img1_batch))
2
3 def plot_image_and_emotion(img_show, img_label, pred_test):
4     """ Function to plot the image and compare the prediction
5     results with the label """
6
7     fig, axs = plt.subplots(1, 2, figsize=(12, 6), sharey=False
8     )
9
10    bar_label = lab
11
12    axs[0].imshow(img_show, 'gray')
13    axs[0].set_title(img_label)
14
15    axs[1].bar(bar_label, pred_test[0], color='orange', alpha
16    =0.7)
17    axs[1].grid()
18
19    plt.show()

```

Listing 4.14: Def plot image and emotion for predicting a test image from unseen dataset

# Chapter 5

## Results

This chapter presents results of the research made. Focus is set on the obtained database, on the type of data it contained and how it was split in a training and test dataset.

Since the database was fairly large, augmentation of original data was not needed to increase the number of samples.

However the internal distribution of features was uneven with large differences in representative numbers, such that the split into a training- and test dataset had to be carefully selected by use of a stratified split that preserved the original class distribution between features.

The model is shown in its entirety with a logical presentation of the data sequence necessary to form a model for type classifications.

Results from training and testing the model are analysed and categorised after type of test. Test results are presented in diagrams and tables to provide a good overview of obtained results.

Test results and differences between various test types are explained, and the reliability and validity of findings are assessed, interpreted and evaluated. A separate chapter is added for discussion of results.

Code with standard algorithms are presented by text and explanations of results are presented in figures and tables, such that a clear, objective and unbiased judgement of the model can be made.

### 5.1 Sequential model

#### 5.1.1 .jpg dataset

##### Metric results

Below is the plotted training history with data from the .jpg dataset. Training history for model accuracy and model loss from the *Fer2013.jpg*

dataset is presented in Figure 5.1.

The sequential model is used, running 150 epochs on a train and test dataset. Accuracy and Loss is first shown.

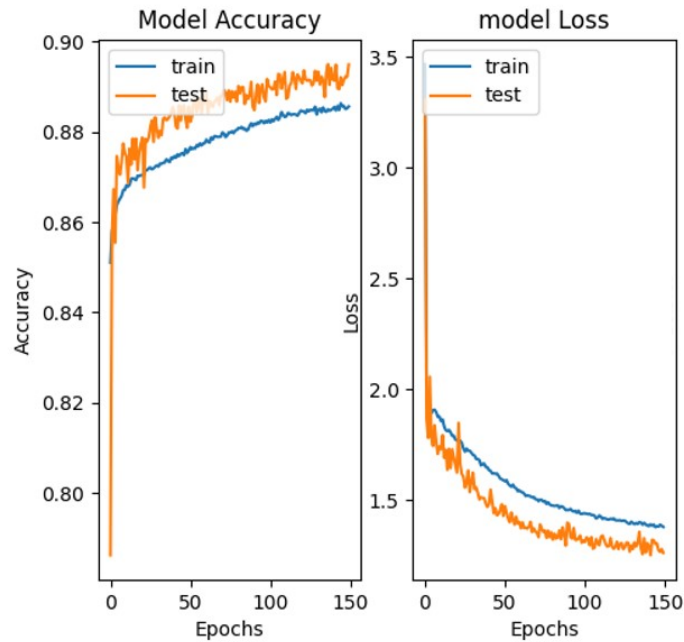


Figure 5.1: Training accuracy and loss for 150 epochs with *Fer2013 .jpg* dataset and the Sequential model

The model accuracy is relatively high, up to 89%. But both the train and test dataset oscillates through epochs.

In Figure 5.2, other metrics are shown for training on the *Fer2013 .jpg* dataset with 150 epochs .

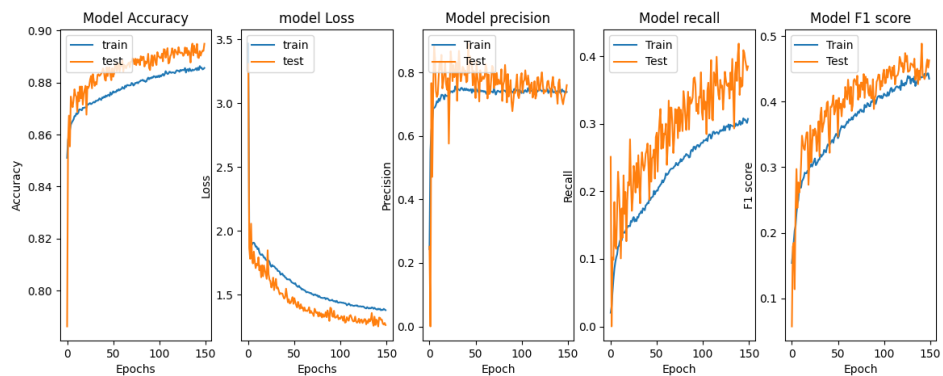


Figure 5.2: All training metrics for 150 epochs with *Fer2013 .jpg* dataset and the Sequential model

Based on Recall and Precision, the model can be said to find only 30-40%

of the correct emotions but will classify up to 70-80% of those emotions in an exact and correct manner.

### Confusion matrix

A Confusion Matrix will show a breakdown of all predictions made for an unseen dataset and is very useful for presenting the accuracy of a model with several feature classes.

The table below presents cells for predictions on the horizontal x-axis and cells for groups of correct answers on the vertical y-axis. Contents of cells are fractions of predictions for each emotion. If all predictions were correct, all numbers on the matrix diagonal would be 1.0.

It can be seen that except for the emotion of 'disgust' with ten times less representation in the dataset, the sum of each row is around 1.0 and that predictions have a spread away from the true prediction at the diagonal.

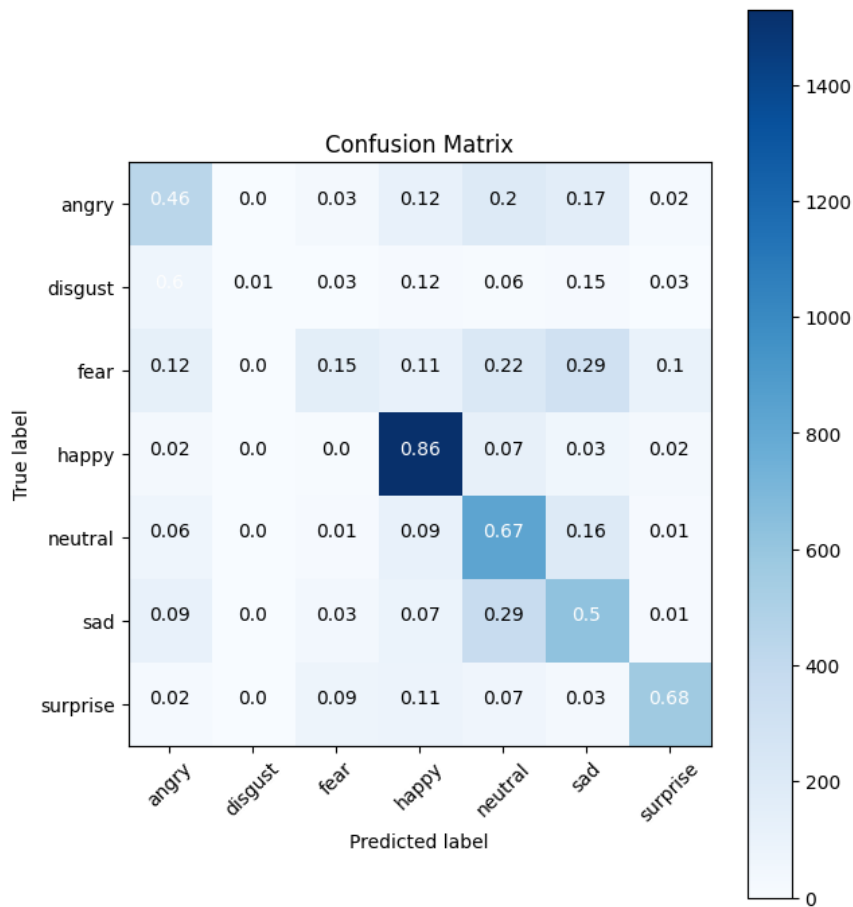


Figure 5.3: Image of confusion matrix for the Sequential model with 150 epochs and *Fer2013 .jpg* dataset

## Classification report

From Scikit-learn, a standardized report for working with classification problems is easily generated. It gives a quick overview of the model accuracy by using standardized measures.

The classification report function displays numerical representations of F-measures such as precision, recall, f1-score and support for each feature class.

```
Normalized confusion matrix
Classification Report
      precision    recall  f1-score   support

 angry         0.50      0.46      0.48       958
 disgust        1.00      0.01      0.02       111
  fear         0.48      0.15      0.23      1024
  happy         0.74      0.86      0.80      1774
 neutral        0.46      0.67      0.55      1233
   sad         0.45      0.50      0.47      1247
 surprise       0.75      0.68      0.71       831

 accuracy                   0.58      7178
 macro avg         0.63      0.48      0.47      7178
 weighted avg      0.58      0.58      0.55      7178

Accuracy on test set= 57.61%
```

Figure 5.4: Image illustrating the classification report for the Sequential model with 150 epochs and *Fer2013 .jpg* dataset

### 5.1.2 .csv dataset

#### Metric results

Below is the plotted training history with data from the .csv dataset. Training history for model accuracy and model loss from the *Fer2013 .csv* dataset is presented in Figure 5.5.

The same sequential model is used here, running 50 epochs on a train and test dataset. Accuracy and Loss is first shown.



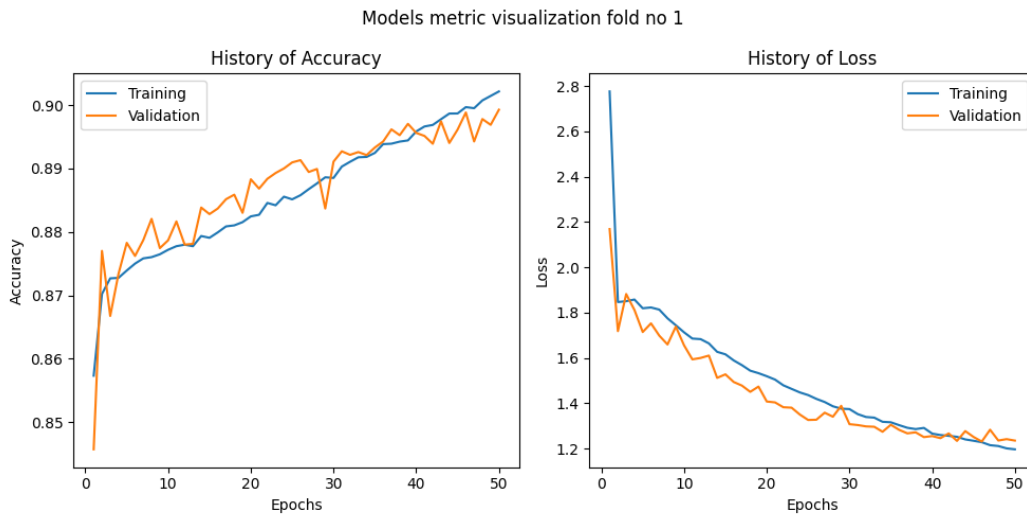


Figure 5.5: Training accuracy and loss for 50 epochs with *Fer2013 .csv* dataset and the Sequential model.

The model accuracy is relatively high, up to 90%, however both the train and test dataset still oscillates during epochs, training to a lesser degree. 90% was obtained after only 50 epochs compared to an accuracy of 88% in the previous .jpg dataset.

In Figure 5.6, other metrics are shown for training on the *Fer2013 .jpg* dataset with 50 epochs.

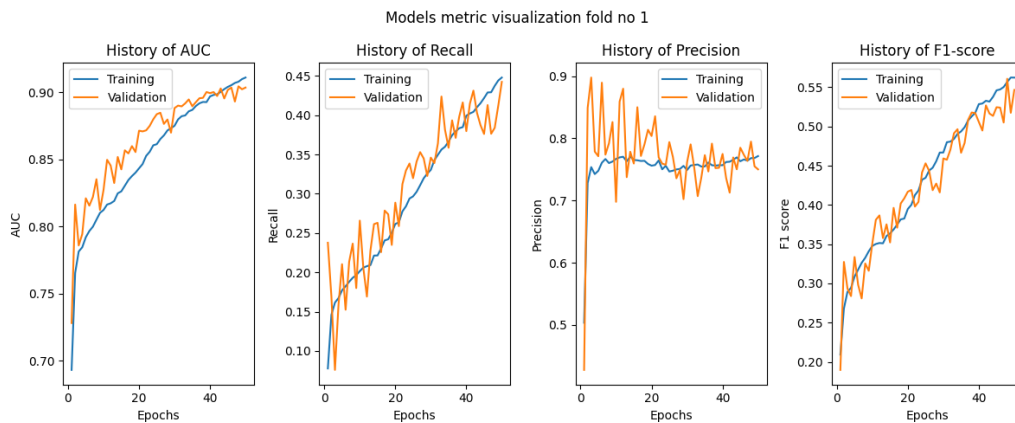


Figure 5.6: History of training auc, recall, precision, f1 for 50 epochs with *Fer2013 .csv* dataset and the Sequential model.

Based on Recall and Precision, the model can be said to find only 45% of the correct emotions but will classify up to 77% of those emotions in an exact and correct manner.

### Confusion matrix

It can be seen also here that except for the emotion of 'disgust' with ten times less representation in the dataset, the sum of each row is around 1.0 and that predictions have a spread away from the true prediction at the diagonal.

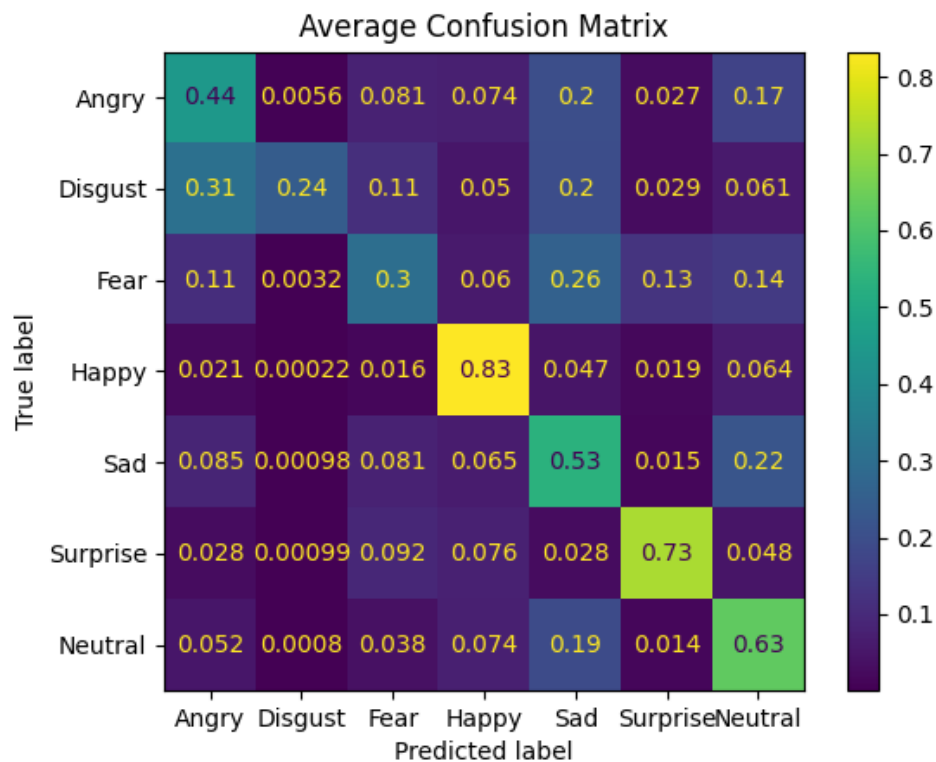


Figure 5.7: Image of average confusion matrix for the Sequential model with 50 epochs and *Fer2013 .csv* dataset

The fold with the best percentages is chosen to be presented here, results from other 10 k folds are found in the Appendix.

### Classification report

The classification report function display numerical representations of F-measures such as precision, recall, f1-score and support for each class.

```

Classification Report
      precision    recall  f1-score   support

   Angry         0.53      0.43      0.48       479
   Disgust        0.57      0.23      0.33        52
   Fear          0.56      0.22      0.31       547
   Happy         0.77      0.82      0.80       876
   Sad           0.45      0.56      0.50       597
   Surprise       0.77      0.66      0.71       418
   Neutral       0.47      0.71      0.57       619

 accuracy                   0.59      3588
 macro avg                 0.59      0.52      0.53      3588
 weighted avg              0.60      0.59      0.58      3588

 Mean time for training per fold: 1259.8 seconds

```

Figure 5.8: Image illustrating the classification report for the Sequential model with 50 epochs and *Fer2013 .csv* dataset

Mean time for the model to train on the train set was 1259.8 seconds = 21 minutes.

### 5.1.3 Data format assessment

Based on numbers for Accuracy, Precision and higher numbers on the diagonals of the confusion matrices, a significant improvement is seen by use of the .csv dataset, and significantly fewer epochs are needed to obtain similar metrics.

It is concluded that the .csv dataset shall be used in further training and testing.

## 5.2 Pre-trained VGG16 model

### Metric results

The same typical metrics for judging the pre-trained classification problem are used, but with an added AUC. These are: Accuracy, Loss, AUC, Recall, Precision and F1 score.

The training history for model accuracy and model loss from the *Fer2013 .csv* dataset with a pre-trained VGG16 model over 20 epochs is presented in Figure 5.9 below. The training history for only fold 1 is shown here, and all the remaining are attached in the Appendix.

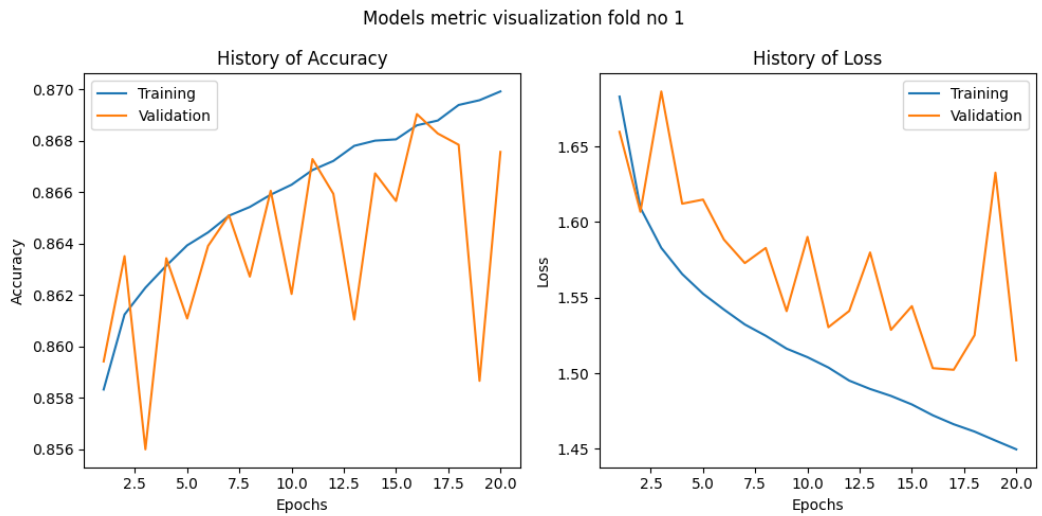


Figure 5.9: Training accuracy and loss for 20 epochs with *Fer2013 .csv* dataset and the pre-trained VGG16 model Fold 1

The model accuracy on the training set is up to 87%, while somewhat lower for the validation set. The training dataset show no oscillations, although some during validation. This was obtained after only 20 epochs, and further improvements can be expected with use of more epochs.

In Figure 5.10, the resulting metrics for training on the pre-trained VGG16 dataset with 20 epochs are shown. Also Fold number 1 is chosen here to correspond with the Figure above.

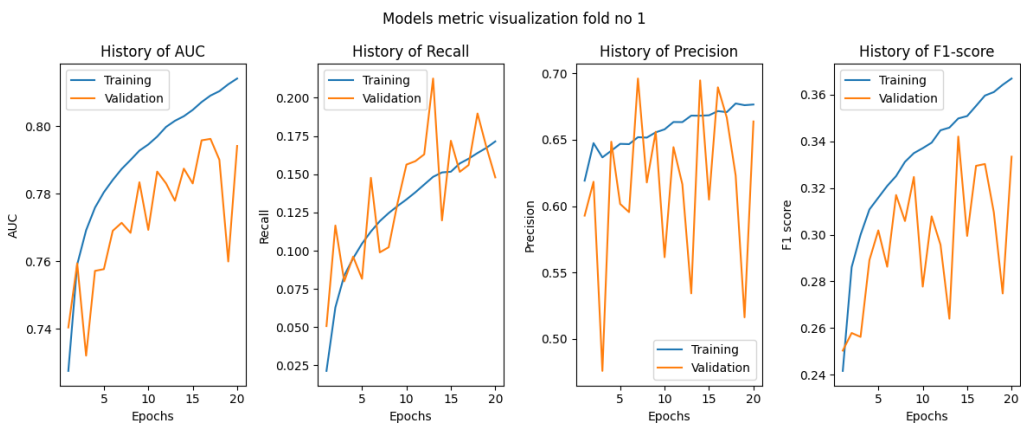


Figure 5.10: Training auc, recall, precision, f1 for 20 epochs with *Fer2013 .csv* dataset and the pre-trained VGG16 model Fold 1

Based on Recall and Precision, the model can be said to find an average of 17% of the correct emotions and will classify up to 67% of those emotions in an exact and correct manner. This was obtained after only 20 epochs, and further improvements can be expected with use of more epochs.

## Confusion matrix

Figure 5.11 is Confusion matrix averaging for all 10 k folds

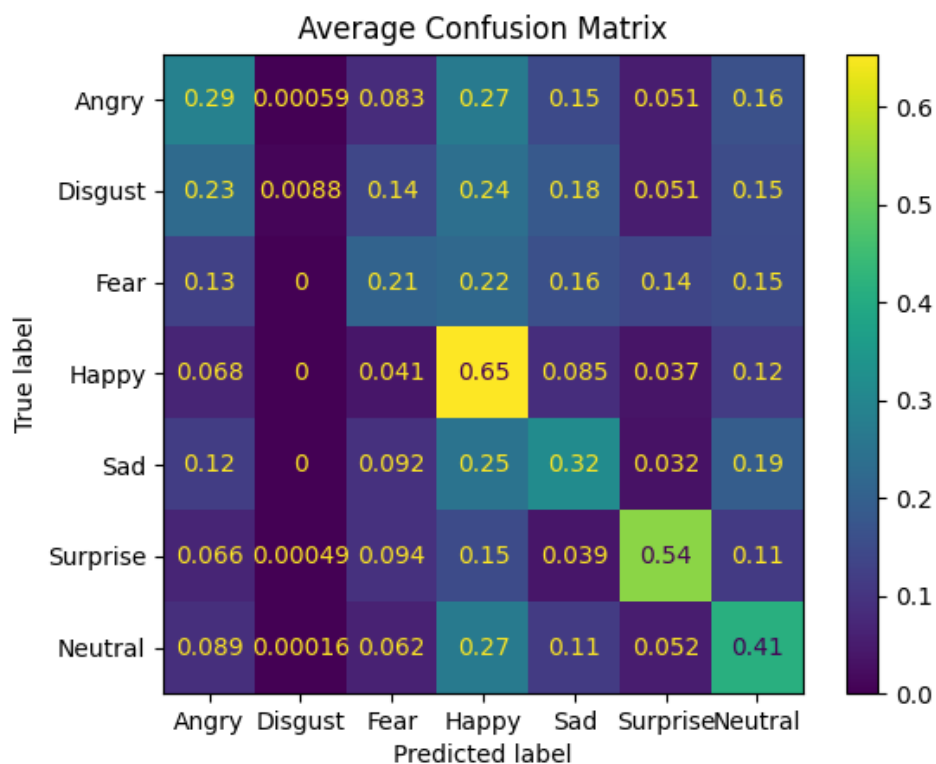


Figure 5.11: Image of confusion matrix for the pre-trained VGG16 model with 20 epochs and *Fer2013 .cvs* dataset. Average confusion matrix is calculated from the 10 k folds

## Classification report

The classification report function display numerical F-measures such as precision, recall, f1-score and support for each feature class.

	precision	recall	f1-score	support
Angry	0.35	0.25	0.29	488
Disgust	0.00	0.00	0.00	49
Fear	0.26	0.34	0.30	516
Happy	0.51	0.66	0.58	921
Sad	0.37	0.23	0.28	628
Surprise	0.43	0.65	0.52	394
Neutral	0.42	0.29	0.34	592
accuracy			0.41	3588
macro avg	0.34	0.35	0.33	3588
weighted avg	0.40	0.41	0.39	3588
Mean time for training per fold: 675.0 seconds				

Figure 5.12: Image illustrating the classification report for the pre-trained VGG16 model with 20 epochs and *Fer2013 .csv* dataset

The classification reports shows metric results from actual classes vs predicted classes from the last k fold. Happy and surprise receiving the highest scores of around 0.5 for precision, recall and f1 score. The emotion disgust gives the lowest score of of 0.0 for all metrics.

Mean time for the model to train on the train set was 675 seconds = 11.3 minutes.

### 5.3 Pre-trained ResNet50 model

#### Metric results

The same typical Metrics for judging the pre-trained classification problem are used, but with an added AUC:

Accuracy, Loss, AUC, Recall, Precision and F1 score

The training history for model accuracy and model loss from the *FER 1* dataset in presented in Figure 5.13 below.

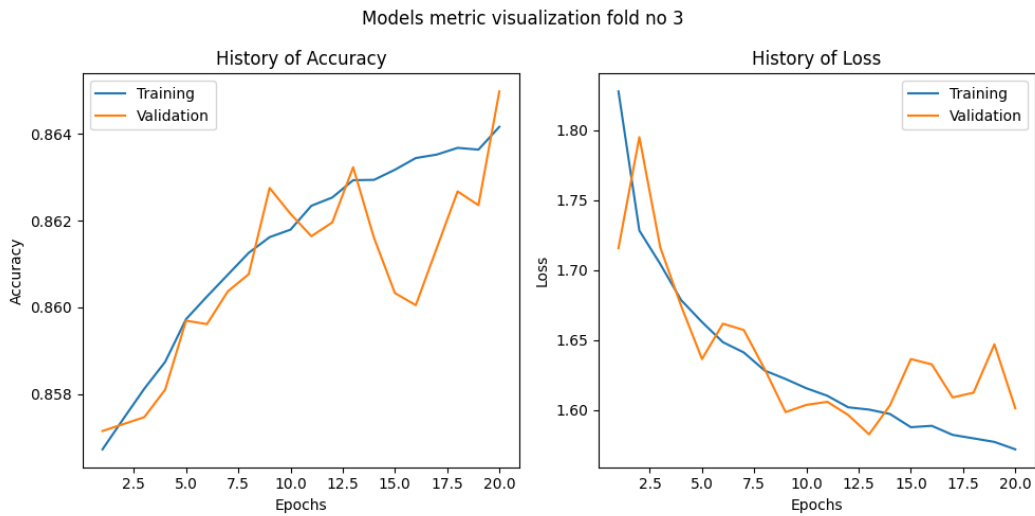


Figure 5.13: Training accuracy and loss for 20 epochs with *Fer2013 .csv* dataset and the pre-trained Resnet50 model

The sequential model was pre-trained using the ResNet model with 'imagenet' weights, running 20 epochs. The model accuracy on the training and validation set is high, up to 86%. The training dataset show small oscillations, somewhat more during validation. As an average, 86.4% was obtained after only 20 epochs.

In Figure 5.14, the resulting metrics for training on the ResNet dataset with 20 epochs are shown.

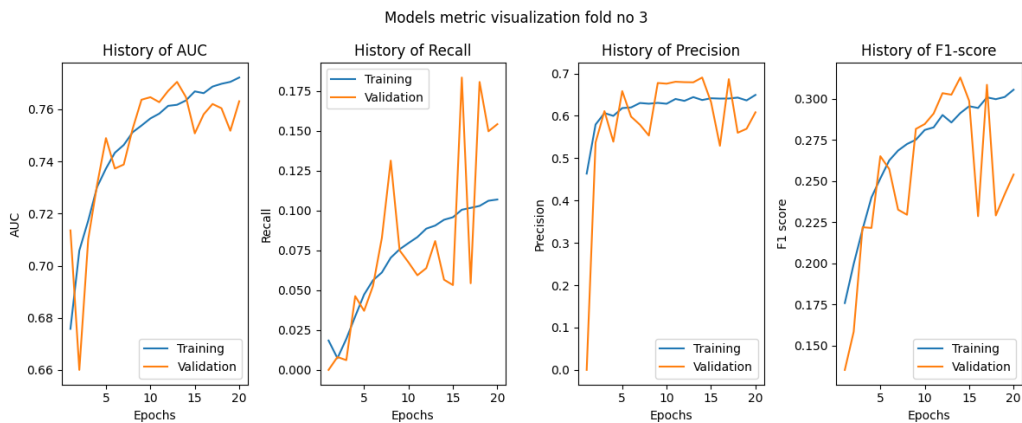


Figure 5.14: Training auc, recall, precision, f1 for 20 epochs with *Fer2013 .csv* dataset and the pre-trained Resnet50 model

Based on Recall and Precision, the model can be said to find an average of 12% of the correct emotions and will classify up to 65% of those emotions in an exact and correct manner.

This was obtained after only 20 epochs, and further improvements can be expected with use of more epochs.

### Confusion matrix

Figure 5.15 below is the developed average confusion matrix from 10 folds with the pre-trained Resnet50 model with weights "imagenet" and 20 epochs.

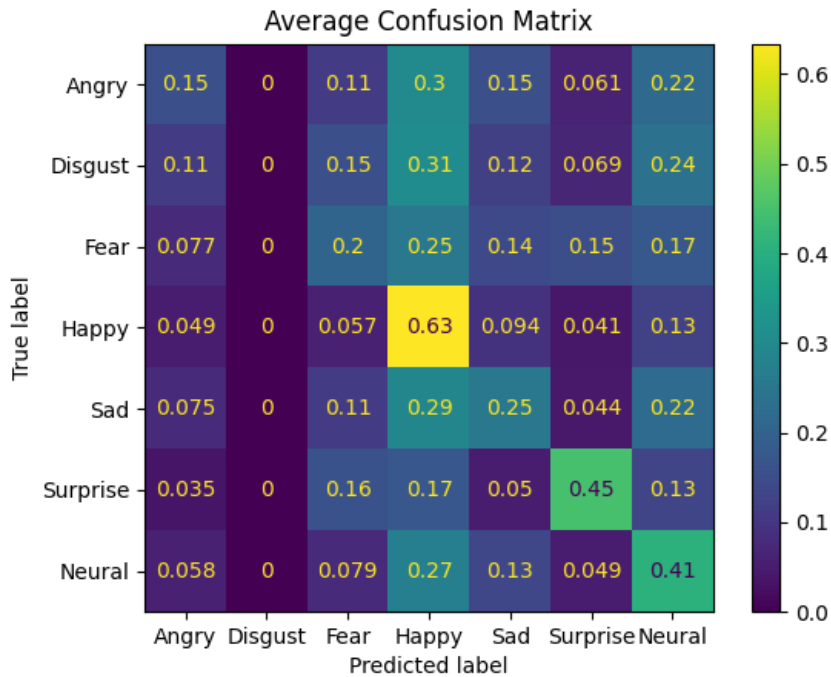


Figure 5.15: Average confusion matrix for 10 k folds with pre-trained Resnet50 model during 20 epochs. Confusion matrix predicted on test dataset showing highest 0.63 % accuracy of predicting happy-happy labels

It can be seen that except for the emotion of 'disgust' with ten times less representation in the dataset, the sum of each row is around 1.0 and that predictions have a spread away from the true prediction at the diagonal.

### Classification report

The classification report function show numerical F-measures such as precision, recall, f1-score and support for each class.



	precision	recall	f1-score	support
Angry	0.35	0.16	0.22	482
Disgust	0.00	0.00	0.00	56
Fear	0.25	0.32	0.28	503
Happy	0.54	0.44	0.48	898
Sad	0.28	0.35	0.31	589
Surprise	0.54	0.41	0.47	405
Neutral	0.35	0.51	0.41	655
accuracy			0.37	3588
macro avg	0.33	0.31	0.31	3588
weighted avg	0.39	0.37	0.37	3588

Mean time for ██████████ on all folds: 687.1 seconds

Figure 5.16: Classification report from using the pre-trained ResNet50 model on 20 epochs on test dataset. Mean time for training on all folds is 687.1 seconds

The classification reports shows metric results from actual classes vs predicted classes from the last k fold. Happy and surprise receiving the highest scores of around 0.5 for precision, recall and f1 score. The emotion disgust gives the lowest score of of 0.0 for all metrics.

Mean time for the model to train on the train set was 687.1 seconds = 11.5 minutes.

## 5.4 Predictions from testing

	Accuracy	St.Deviation	Loss	Epochs
Sequential .jpg	88 %	+- %	- %	150
Sequential .csv	89.67%	+- 0.193%	1.27%	50
VGG16 pre-trained	86.70%	+- 0.131%	1.51%	20
Resnet50 pre-trained	86.26%	+- 0.171%	1.61%	20

Table 5.1: Test set results from FER-2013 dataset over average metrics

Judging by these results one may conclude that models are quite equal in accuracies.

The VGG and ResNet pre-trained models do have a somewhat lower accuracy and would clearly have benefited from use of more epochs since most metrics did not reach their asymptotic value. Table values for the pre-trained models can be said to be conservative estimates.

Low values for pre-trained models may also arise from use of too few convolution sequences, however results are not far from results of the more complex sequential base model.

But it is also very important to consider the other metrics to arrive at a conclusion, as will be discussed in the next Discussion chapter.

Note however that models were run with a different number of epochs and that results are for the last epoch. Performance of models should be evaluated at the same epoch.

Using only 20 epochs may give inaccurate results for the sequential model without any pre-trained weights, however a low number may be sufficient when pre-trained models are optimally configured.

## 5.5 Predictions on unseen datasets

After the model has trained on the train dataset and receiving data of images and its corresponding labels, it was run on pictures from an unseen dataset. Examples of how the different models predict a random image is presented as a demonstration.

### Sequential

The Sequential model was used for prediction of the same selection from the JAFFE dataset.

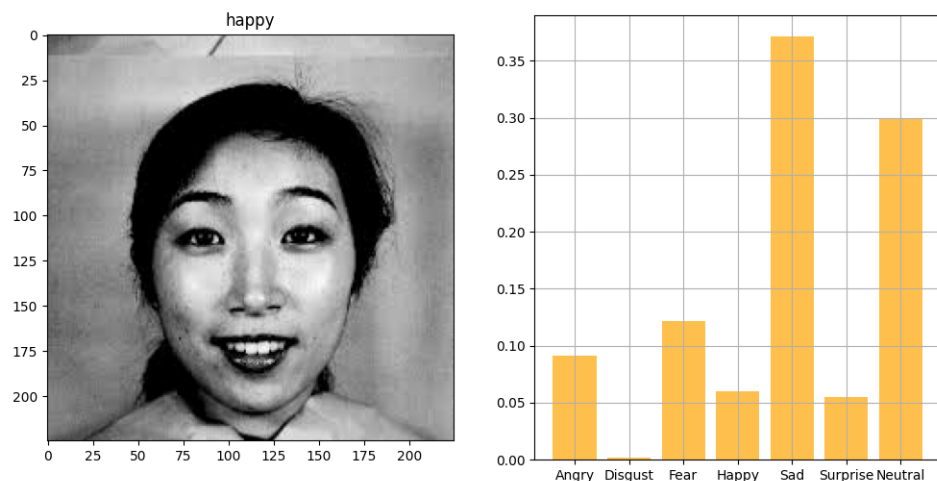


Figure 5.17: Sequential predicting on a Happy test image from the JAFFE dataset, with results to the right.

When the sequential model makes predictions on a Happy test image, the predicted results are highest for Sad and Neutral while Happy is among

the lowest features, i.e. not an accurate prediction.

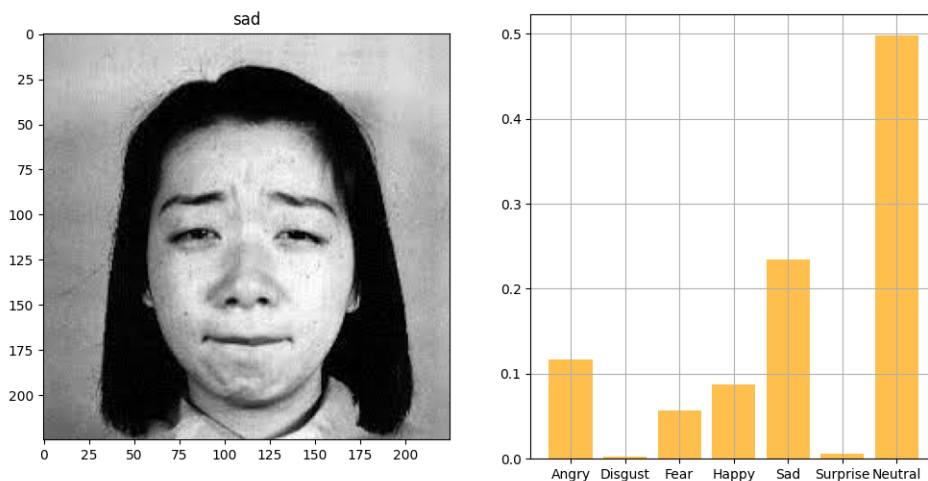


Figure 5.18: Sequential predicting on a Sad test image from the JAFFE dataset, with results to the right.

When the sequential model makes predictions on a Sad test image, the predicted results are highest for Neutral, however Sad is the second most likely feature, i.e. still not an accurate prediction.

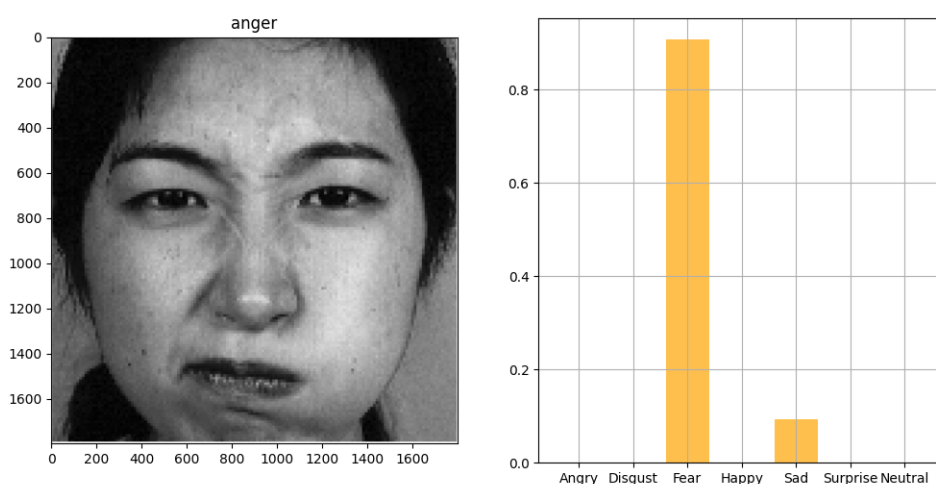


Figure 5.19: Sequential predicting on an Angry test image from the JAFFE dataset, with results to the right.

When the sequential model makes predictions on an Anger test image, the predicted results are highest for fear and sad while Anger is not

predicted, i.e. a very inaccurate prediction. However, anger, fear and sad can be considered related emotions..

### ResNet50 pre-trained

The ResNet pre-trained model was used for prediction of the same selection from the JAFFE dataset.

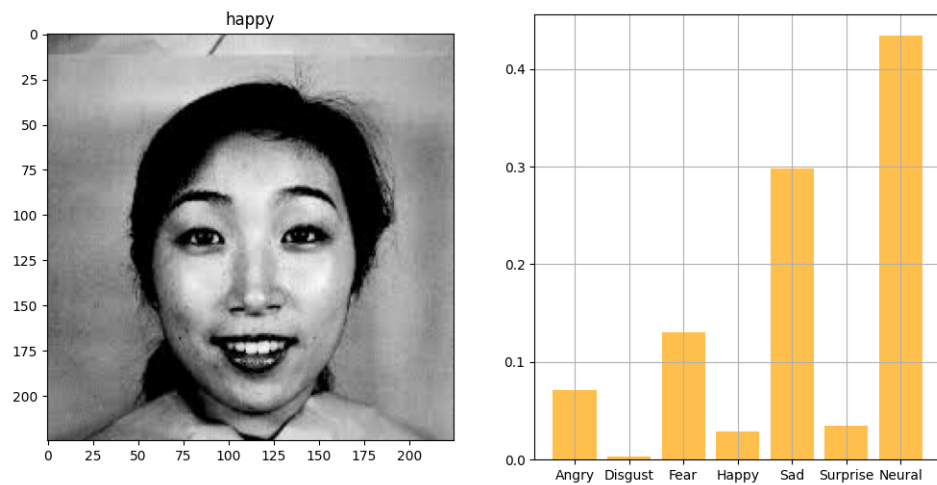


Figure 5.20: Pre-trained Resnet50 predicting on a Happy test image from the JAFFE dataset, with results to the right.

When the pre-trained Resnet50 model makes predictions on a Happy test image, the predicted results are highest for Sad and Neutral while Happy is among the lowest features, i.e. not an accurate prediction.

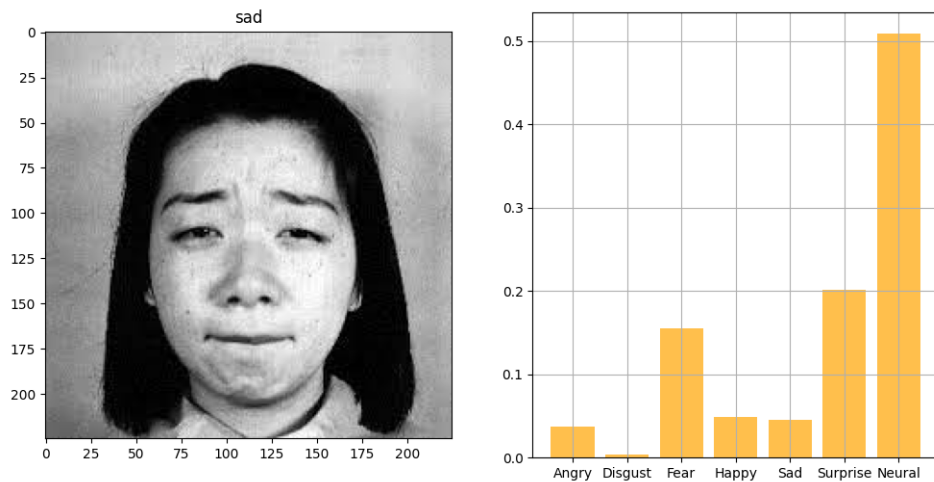


Figure 5.21: Pre-trained Resnet50 predicting on a Sad test image from the JAFFE dataset with results to the right.

When the pre-trained Resnet50 model makes predictions on a Sad test image, the predicted results are highest for Neutral and Surprise while Sad is among the lowest features, i.e. not an accurate prediction.

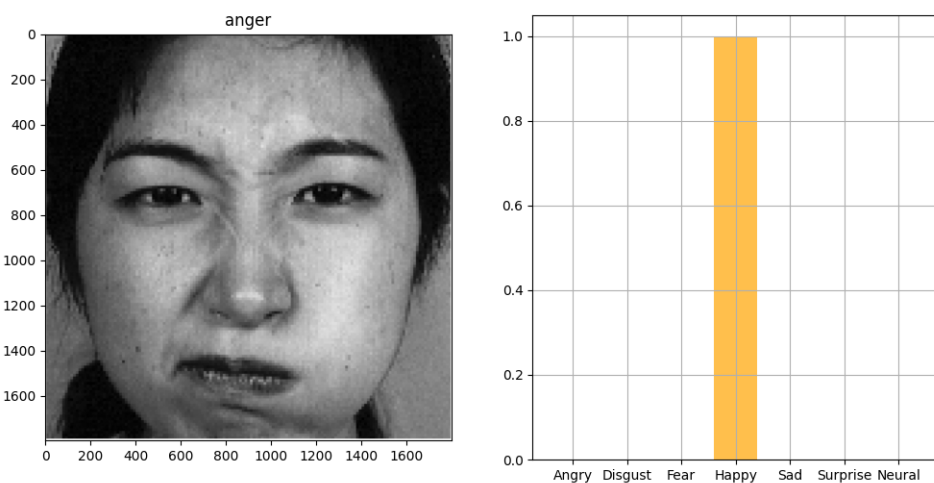


Figure 5.22: Pre-trained Resnet50 predicting on a angry test image from the JAFFE dataset with results to the right.

When the pre-trained Resnet50 model makes predictions on an Anger test image, the predicted results are highest Happy while Anger is not predicted, i.e. a very inaccurate prediction. However, mouth movements may have triggered a happy prediction.

## VGG16 pre-trained

Then the VGG16 pre-trained model was used for prediction of the same selection from the JAFFE dataset.

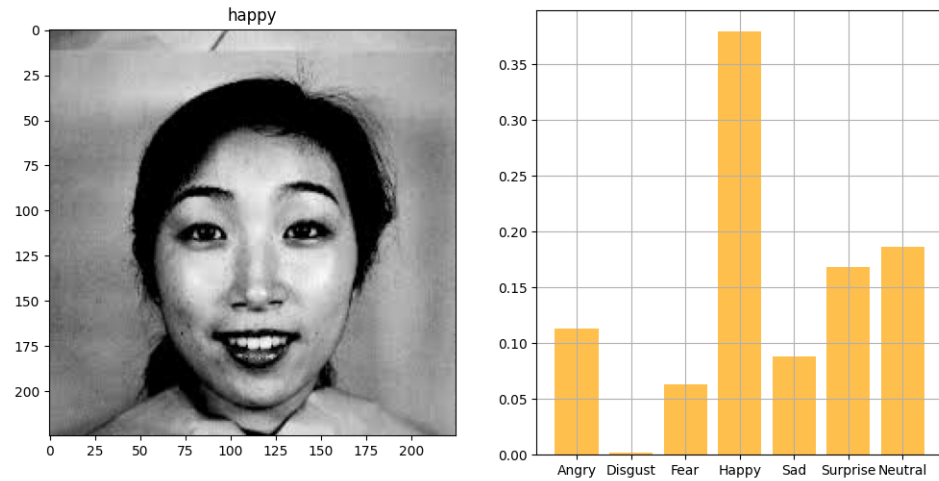


Figure 5.23: Pre-trained VGG16 predicting on a Happy test image from the JAFFE dataset, with results to the right.

When the pre-trained VGG16 model makes predictions on a Happy test image, the predicted result was actually highest for Happy.

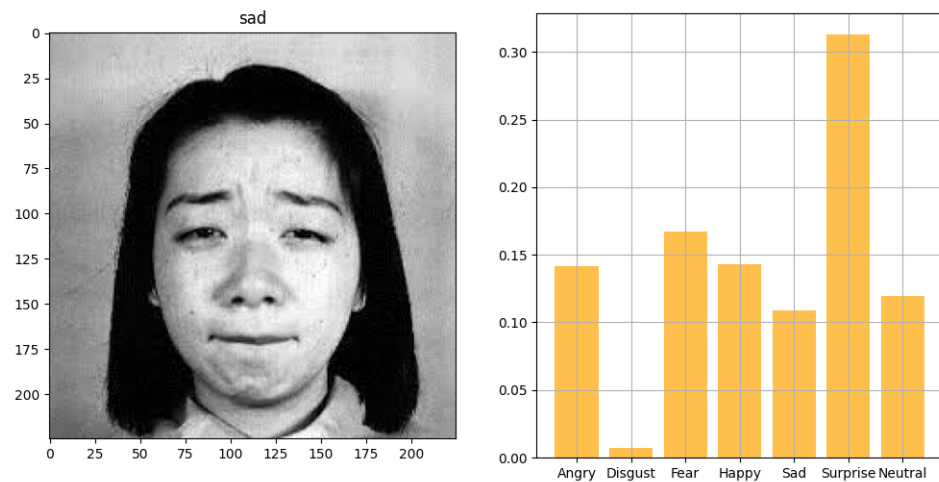


Figure 5.24: Pre-trained VGG16 predicting on a Sad test image from the JAFFE dataset with results to the right.

When the pre-trained VGG16 model makes predictions on a Sad test

image, the predicted results are highest for Surprise, while Sad is among the lowest features, i.e. not an accurate prediction.

A sad situation is usually due to a disappointment and can be said to be closely related to surprise.

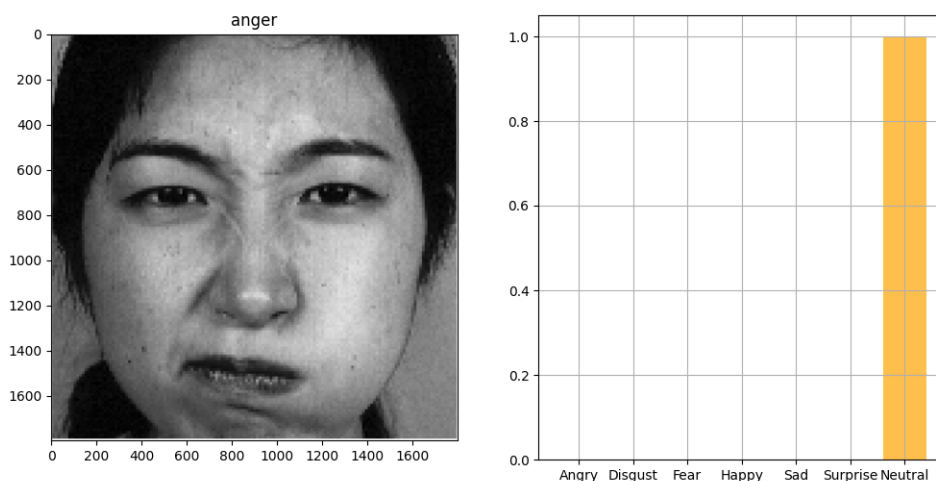


Figure 5.25: Pre-trained VGG16 predicting on a Angry test image from the JAFFE dataset with results to the right.

When the pre-trained VGG model makes predictions on an Anger test image, the predicted results are highest for Neutral, while Anger is not predicted, i.e. a very inaccurate prediction.

### Comparison

In this test of random images, none of the models gave good predictions. They did however give similarly bad predictions, proving that small and simple pre-trained models give similar results as use of more complex models with several convolutional layers.





# Chapter 6

## Discussion

This chapter discusses analytical and critical thinking in relation to results and analyses, also with reference to arguments described in the literature review.

### 6.1 Metric results

Accuracies will improve if more epochs are run. The current results are the status after only 20 epochs for all models.

	Accuracy	Loss	Recall	Precision	f1
Sequential .jpg	88 %	1.75 %	20 %	75 %	35%
Sequential .csv	89.60%	1.21 %	41 %	76 %	54%
VGG16 pre-trained	86.70%	1.51 %	17 %	67 %	35%
ResNet50 pre-trained	86.26%	1.61 %	12 %	60 %	29%

Table 6.1: Test set results from FER-2013 dataset after 20 epochs

It was earlier concluded that models were quite equal in accuracies and other analysis metrics.

Judging by these more complete results, a clear and significant difference is seen for the Recall parameter for the sequential .csv dataset, which also has a direct influence on the f1-parameter. Recall is the parameter that indicate the number of true correct predictions, which means that this is a better model.

It is seen that use of additional analysis metrics is very important to consider in order to arrive at a conclusion. In this case, indications are that the Sequential .csv model is better while the VGG16 model is the best of the pre-trained models when a small base sequential model is used.

## 6.2 Choosing datasets

From researching possible datasets containing emotional expressions, I found a total of five datasets. These are:

- FER-2013 dataset is labelled with one of seven categories: angry, disgust, fear, happy, sad, surprise, neutral
- JAFFE dataset mentions that it contains anger, happiness, fear, surprise, disgust and sadness + neutral, but not sure if they are already pre-labelled
- RAF-DB dataset is labelled with 6 expressions: angry, disgust, fear, happy, sad, surprise (plus seems like one more, neutral?)
- CK+ dataset is labelled with one of eight expression classes: anger, disgust, fear, happiness, sadness, neutral and surprise, plus the contempt expression

The FER-2013 dataset was the easiest available as it was publicly posted on the Kaggle platform. The dataset was available both as a dataset containing folders with images, or a dataset in a .csv format. It was not much information posted on Kaggle about the .csv dataset, but it could seem like it used the image dataset and just converted it to an array of images in digital format. I would probably be able to use the image folder dataset and converted it myself to a .csv file by running `imgtoarray` in a python function. For this time, I found it easier to download an already converted dataset from Kaggle.

The reason for changing the dataset from FER image folders to a .csv file was that it was easier for me to work with `np.arrays` than working with an Image classifier. Due to less experience working with Image classifiers earlier and encounter of several error messages when processing the dataset into labels for visualization, using the planned image structure for the ML models often gave in/out errors, meaning conversion of images in my program did not function correctly. I therefore decided to use a .csv file where documentation of image processing with arrays was more easily available.

As well as encountering problems with image processing from the FER-2013 image folders, the highest accuracy percentage score for predicting the test set would not exceed about 60%. This can be considered quite low, when I now in retrospect have obtained results as high as 90% accuracy for prediction on a test set. I also looked at Kaggle to see what others have posted from their classification matrices or classification reports of predicted metrics. Many of the models I found did however also report accuracy percentages of about 60% from running predictions on a test set. This, in combination with the processing errors, made me consider another dataset that also did not contain image augmentation or had a pre-defined train/test split.

### 6.3 Splitting in train and test

I first started out by using the already created train/test split in the downloaded dataset. The split was set to a classic 80% for training and 20% for testing. However, after running the program and using the pre-defined splits I was encouraged from my supervisors to use a more advanced k-fold cross validation split method on the whole dataset.

### 6.4 Model parameters

A sequential CNN model was established and thereafter subjected to transfer learning. Two main approaches to implementing transfer learning can be employed: Weight Initialization Feature Extraction.

The Weight Initialization method was used and final model behaviour was assessed by its analysis metrics.

Accuracy is the number of correct predictions in relation to all predictions made for a feature, disregarding whether the prediction was correct or wrong. An accuracy score for the best model of up to 90% is found acceptable. The prerequisite that there should be an equal number of observations in each evaluated class was circumvented by performing a stratified train-test split, or a stratified k-fold cross-validation.

Loss of membership to a class was found to be acceptably low, down to 1.5% for the best model. Correct predictions were rewarded more than incorrect predictions were punished. A lower logarithmic loss is however desirable, as close to zero as possible, as for a perfect log loss of zero.

For reference, of architectures described in the section describing Convolutional Neural Network (CNN)'s, the AlexNet architecture obtained an 'error rate' of 0.154, the derived ZF-Net had 0.112, VGG19 0.073, Inception/GoogLeNet 0.067 and ResNet obtained a very low 0.036.

An F-measure considered the relationship between the number of True/False Positives and True/False Negatives. The F1 Score expresses the balance between 'precision' and 'recall', where precision summarizes the 'exactness' of the model and recall summarizes the 'completeness' of the model. Results obtained from the runs showed that estimates were more exact than they were complete. The best model found around 30% of the correct emotions and classified up to 70-80% of those emotions in an exact and correct manner. Improving the Recall rate will give an overall improvement to the models,

The Confusion Matrix in the form of a two-dimensional cross matrix of number of features showed that the diagonal with numbers for correctly interpreted features was reasonably well populated with few incorrectly interpreted features outside the diagonals. It was clearly seen that the disgust feature was scarce and was easily interpreted as angry or fear. The

best model had more than 65% of cases on the correctly indicated diagonal.

## 6.5 Predictions on unseen datasets

After the model has trained on the train dataset and receiving data of images and its corresponding labels, it was run on pictures from an unseen dataset. A random picture can never be interpreted 100% correctly unless Accuracy, Recall and Precision are extremely high. These numbers were below 76% such that some misinterpretations had to be expected.

By selecting only a few random images and performing testing with the selected models, it was seen that models were working well but they did not give reasonably good predictions.

Models did however give similarly bad predictions. Analysis metrics also showed similar behaviour could be expected from all three models. It is also indicated that especially the pre-trained models could have benefited from more training, including use of a larger number of epochs.

Considering the different model architectures, it is indicated and confirmed that small and simple pre-trained models give similar results as will be obtained from use of more complex models with several convolutional layers. This may be of importance if a model shall be implemented on small handheld devices.

Tests were also only made on emotions from a female Asian dataset. Other datasets were not readily available in an easily usable format, such that conclusions on estimates for people with other origins could not be made.

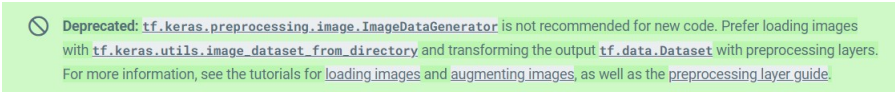
## 6.6 Challenges with unbalanced datasets

It was found that six of the emotions were quite evenly represented in the dataset, except for the emotion of 'disgust' that had ten times less representation for its class among the data. Results for the confusion matrix of the worst model also show that 'disgust'-cases were identified correctly only 1% of the time, however the classification report show that these few identifications were indeed correct. This have likely influenced the overall results of the poor model where the model could be said to 'recall' only 35% of the correct emotions but managed to classify up to 80% of those emotions in an exact and correct manner. An improvement in Recall from a low 35% will have a direct effect on the total F1-number for this model.

Correcting the mismatch in representation among emotions is likely to improve the overall performance for the whole model. A solution could be to oversample or augment the dataset for this emotion only such that its representation becomes more equal to the others, or it could be left out altogether, by removing it and use only six emotions.

## 6.7 Challenges with implementation of ML program

After collecting the desired dataset an online search was made to find how to best load the image dataset into the program. From Brownlee (2019) I found an example that used the ImageDataGenerator class provided by Keras. After fully implementing this into the code, I found that the class has been deprecated. From Tensorflow (2022) I see the following message on the website:



ⓘ Deprecated: `tf.keras.preprocessing.image.ImageDataGenerator` is not recommended for new code. Prefer loading images with `tf.keras.utils.image_dataset_from_directory` and transforming the output `tf.data.Dataset` with preprocessing layers. For more information, see the tutorials for [loading images](#) and [augmenting images](#), as well as the [preprocessing layer guide](#).

Figure 6.1: Image taken from TensorFlow Documentation web page stating the ImageDataGenerator class from Keras is deprecated.

I then changed my code to use the recommended class instead, to `imagedatasetfromdirectory`.

It is important to use time on investigating a preferred class method for a code before implementing it. It is recommended to visit the documentation website and check if the module is still active and recommended by the company that created it.

## 6.8 Further work

Further work directly connected to the current thesis could be conducted on the FER-2013 dataset by removing or balancing the "disgust" emotional category. In all results, it is clearly seen that disgust was poorly trained, and thus could not be predicted with a sufficiently high accuracy.

If more high-performance hardware had been available, e.g. as in a research project with more resources available, finding and testing additional datasets with a high mix of people's origin could be done to see if interpretations of the same feature would vary significantly between people's origin.



## Chapter 7

# Conclusion

FER is a technology that analyzes facial expressions in images to reveal information about a person's emotional state. Rapid advancements in ML have made it possible to equip computers with the ability to analyze, recognize and understand emotions.

This thesis investigates which DL model give the best results in FER. Among many techniques for FER, deep learning models, especially CNN's have shown great potential due to their powerful automatic feature extraction and computational efficiency. In this thesis, one sequential built CNN model, one pre-trained Resnet50 model and one pre-trained VGG16 model were created. Each model were compared and evaluated.

Seven categories of emotions were categorized from the The Facial Expression Recognition 2013 (FER-2013) database containing a total of 35887 images. The sequential built CNN model of only a few convolutions run on 50 epochs gave accuracy result of 89.7% with an average time of training per k fold of 21 minutes. The two pre-trained models with imagenet weights run on 20 epochs gave accuracy results of 86.5% without fine-tuning and average training time of 12 minutes per k fold.

By analysing a rather complicated baseline model and comparing results to a less advanced model that used transfer learning, it showed that transfer learning can be used to obtain the same accuracy with significantly smaller models. Transfer learning can then be used as a performance optimizer on a smaller baseline model to give a sufficiently accurate model with minimum use of resources. Even though a limited number of epochs were run on the pre-trained models, it was seen that validation and test curves quickly approached training curves at a high level of accuracy. It is concluded that the base sequential model and the pre-trained models all performed well and converged towards better values.

Lastly, example images from the Japanese Female Facial Expression (JAFFE) database were introduced to the trained model to evaluate testing predictions. A random picture can never be interpreted 100% correctly unless Accuracy, Recall and Precision are extremely high. These numbers were below 76% such that some misinterpretations had to be expected. By selecting only a few random images and performing testing with the selected models, it was seen that models were working well but they did

not give reasonably good predictions on the unseen dataset.

Recommendations suggested in the field of Facial Expression Recognition could be to investigate if other pre-trained models will identify features to an even higher degree, and with less use of required resources, also on more diverse datasets.



# Bibliography

- Betts, J. G., Young, K. A., Wise, J. A., Johnson, E., Poe, B., Kruse, D. H., Korol, O., Johnson, J. E., Womble, M. & DeSaix, P. (2013, April 25). *Anatomy and physiology*. <https://openstax.org/books/anatomy-and-physiology/pages/1-introduction>
- Brownlee, J. (2016). *Supervised and unsupervised machine learning algorithms*.
- Brownlee, J. (2019). *How to load large datasets from directories for deep learning in keras*.
- Brownlee, J. a. s. o. n. (2019, August 14). *How to one hot encode sequence data in python*. <https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/>
- Brownlee, J. (2020). How to get started with deep learning for computer vision (7-day mini-course). <https://machinelearningmastery.com/how-to-get-started-with-deep-learning-for-computer-vision-7-day-mini-course/>
- Chowdary, M., Nguyen, T. & Hemanth, D. (2021). Deep learning-based facial emotion recognition for human–computer interaction applications. *Neural Computing and Applications*, 1–18. <https://doi.org/10.1007/s00521-021-06012-8>
- Das, A. n. g. e. l. (2021, December 15). *Convolution neural network for image processing — using keras*. <https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306>
- Dhankhar, P. (2019). Resnet-50 and vgg-16 for recognizing facial emotions.
- Dubey, A. & Jain, V. (2020). Automatic facial recognition using vgg16 based transfer learning model. *Journal of Information and Optimization Sciences*, 41, 1–8. <https://doi.org/10.1080/02522667.2020.1809126>
- Ekman, P. & Friesen, W. V. (1978). Facial action coding system: A technique for the measurement of facial movement. *Palo Alto, CA: Consulting Psychologists Press. Environmental Psychology & Nonverbal Behavior*.
- Ekman, P., Friesen, W. V. & Hager, J. C. (2002). Facial action coding system: The manual. *Salt Lake City, UT: Research Nexus*.
- Fang, X., van Kleef, G. A., Kawakami, K. & Sauter, D. A. (2021). Cultural differences in perceiving transitions in emotional facial expressions: Easterners show greater contrast effects than westerners. *Journal of experimental social psychology*, 95, 104143.
- Feick, L. (2022). Evaluating Model Performance by Building Cross-Validation from Scratch. <https://www.statworx.com/en/content->

hub / blog / evaluating - model - performance - by - building - cross - validation - from - scratch /

- Geisslinger, M. (2019). *Autonomous driving: Object detection using neural networks for radar and camera sensor fusion* (Doctoral dissertation). <https://doi.org/10.13140/RG.2.2.10949.47840>
- Harrison, T. h. e. o. (2021, November 5). *Ekman's 6 basic emotions and how they affect our behavior*. <https://themindsjournal.com/basic-emotions-and-how-they-affect-us/>
- He, K., Zhang, X., Ren, S. & Sun, J. (2015). Deep residual learning for image recognition. <https://doi.org/10.48550/ARXIV.1512.03385>
- Kaggle. (2018). FER2013. <https://www.kaggle.com/datasets/deadskull7/fer2013>
- Kaggle. (2020). FER-2013. <https://www.kaggle.com/datasets/msambare/fer2013>
- Kaulard, K., Cunningham, D. W., Bülthoff, H. H. & Wallraven, C. (2012). The mpi facial expression database - a validated database of emotional and conversational facial expressions. *PloS one*, 7(3), e32321–e32321.
- Keras Documentation. (n.d.). Keras documentation: Keras applications. <https://keras.io/api/applications/>
- Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6), 84–90. <https://doi.org/10.1145/3065386>
- Landowska, A. (2018). Towards new mappings between emotion representation models. *Applied sciences*, 8(2), 274.
- Lansley, J. . (2022, March 2). *Facial expressions*. <https://www.eiagroup.com/knowledge/facial-expressions/>
- Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. <https://doi.org/10.1109/5.726791>
- Mitchell, T. M. (1997). *Machine learning*. MacGraw-Hill.
- Mittal, R. . (2020, May 19). *Face recognition using transfer learning on vgg16*. <https://www.linkedin.com/pulse/face-recognition-using-transfer-learning-vgg16-rohit-mittal>
- Mohamed, B., Daoud, M., Mohamed, B. & Ahmed, A. (2022). Improvement of emotion recognition from facial images using deep learning and early stopping cross validation. *Multimedia Tools and Applications*. <https://doi.org/10.1007/s11042-022-12058-0>
- Nikhil, B. (2019, January 18). *Image data pre-processing for neural networks - becoming human: Artificial intelligence magazine*. <https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258>
- Opperma, A. . (2022, February 9). *Artificial intelligence vs. machine learning vs. deep learning: What's the difference?* <https://builtin.com/artificial-intelligence/ai-vs-machine-learning>
- Paul Ekman Group LLC. (2020, January 30). *Facial action coding system*. <https://www.paulekman.com/facial-action-coding-system/>

- Rodrigues, N. & Pereira, A. (2018). A user-centred well-being home for the elderly. *Applied sciences*, 8(6), 850.
- Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. <https://doi.org/10.48550/ARXIV.1409.1556>
- Spring, Russ & Davidson. (n.d.). *Hamamatsu learning center: Basic properties of digital images*. <https://hamamatsu.magnet.fsu.edu/articles/digitalimagebasics.html>
- Suman. (2019, October 11). *Is machine learning required for deep learning?* <https://ai.stackexchange.com/questions/15859/is-machine-learning-required-for-deep-learning>
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. & Rabinovich, A. (2014). Going deeper with convolutions. <https://doi.org/10.48550/ARXIV.1409.4842>
- Tensorflow. (2022). *Tensorflow core v2.8.0; tf.keras.preprocessing.image.imagedatagenerator*.
- Tian, W. (2021). Personalized emotion recognition and emotion prediction system based on cloud computing. *Mathematical Problems in Engineering*, 2021, 1–10.



# Appendices



# Appendix A

## Training results for each fold

### A.1 VGG16 model

Training of pre-trained VGG16 model with weights "imagenet" during 20 epochs and 10 k folds.

#### Fold 1



Figure A.1: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 1

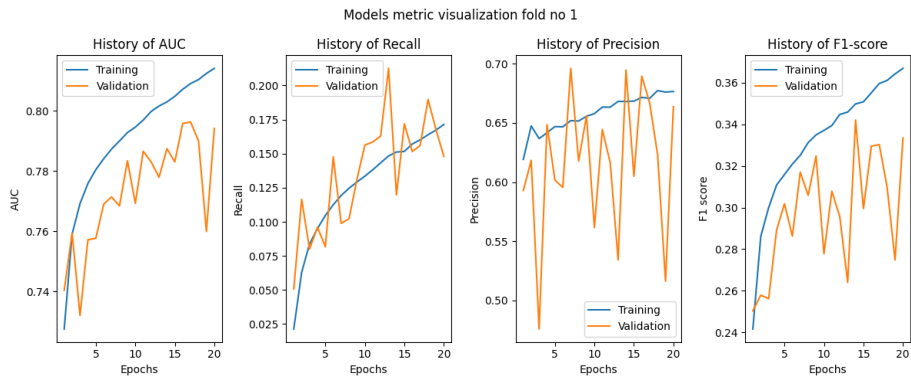


Figure A.2: All metrics over 20 epochs with pre-trained VGG16 model fold 1

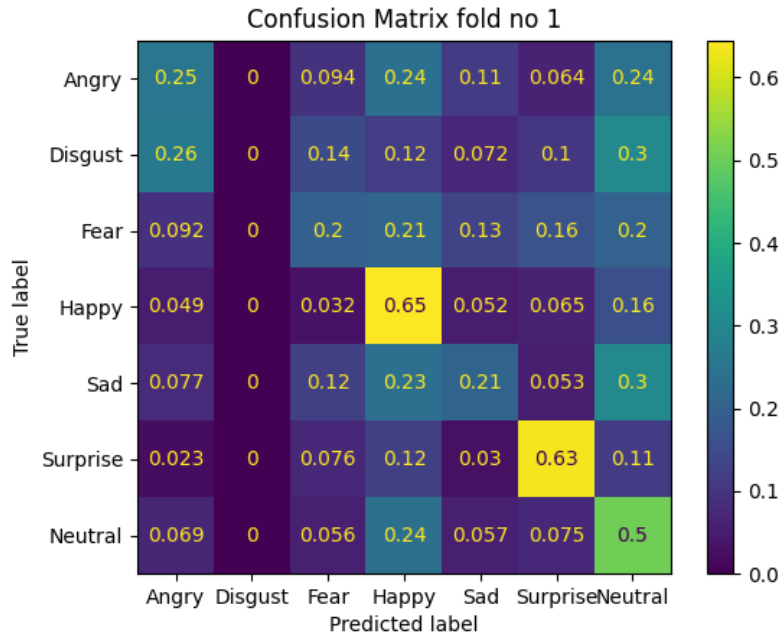


Figure A.3: Confusion matrix over 20 epochs with pre-trained VGG16 model fold 1



## Fold 2



Figure A.4: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 2

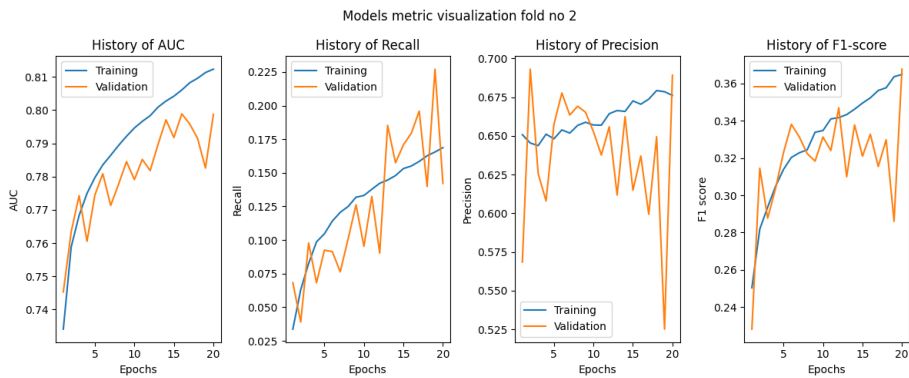


Figure A.5: All metrics over 20 epochs with pre-trained VGG16 model fold 2

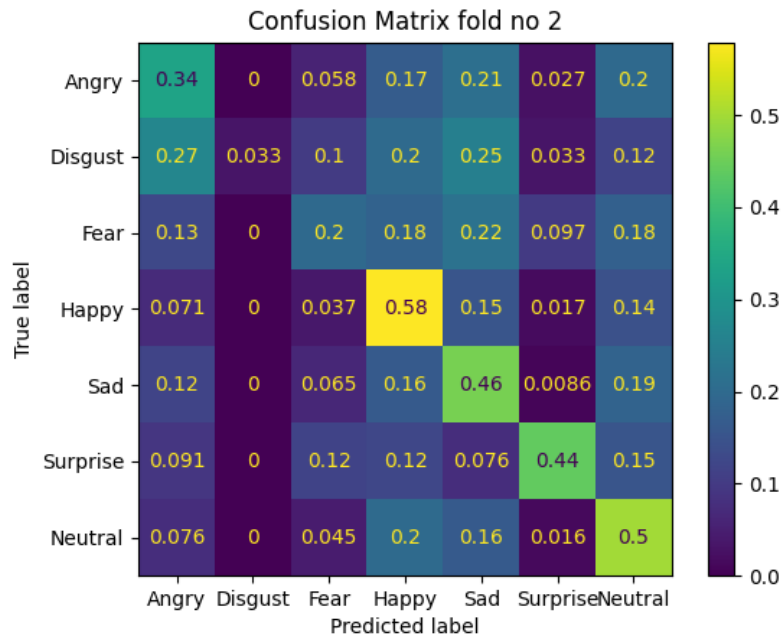


Figure A.6: Confusion matrix over 20 epochs with pre-trained VGG16 model fold 2

### Fold 3

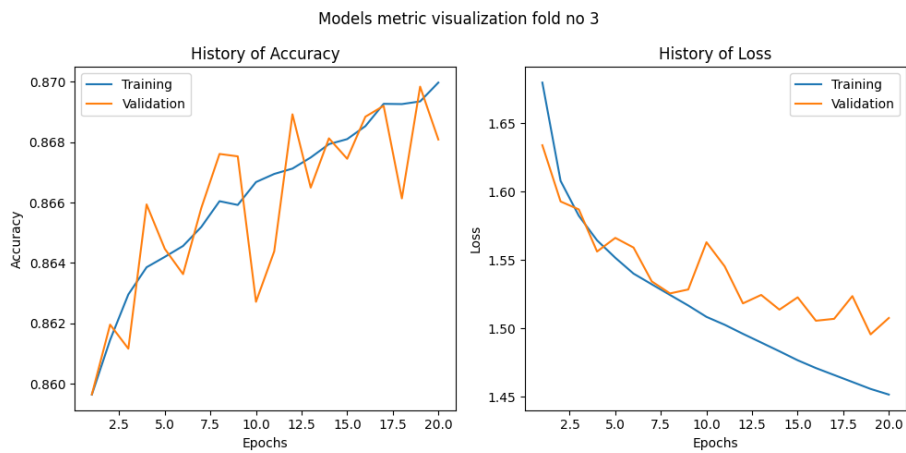


Figure A.7: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 3

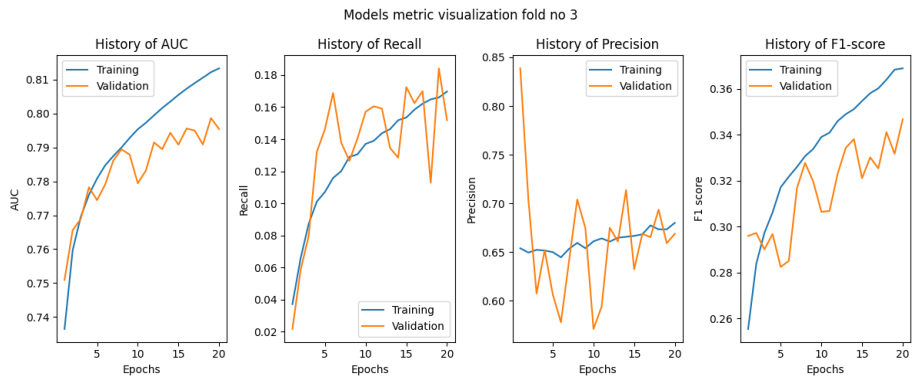


Figure A.8: All metrics over 20 epochs with pre-trained VGG16 model fold 3

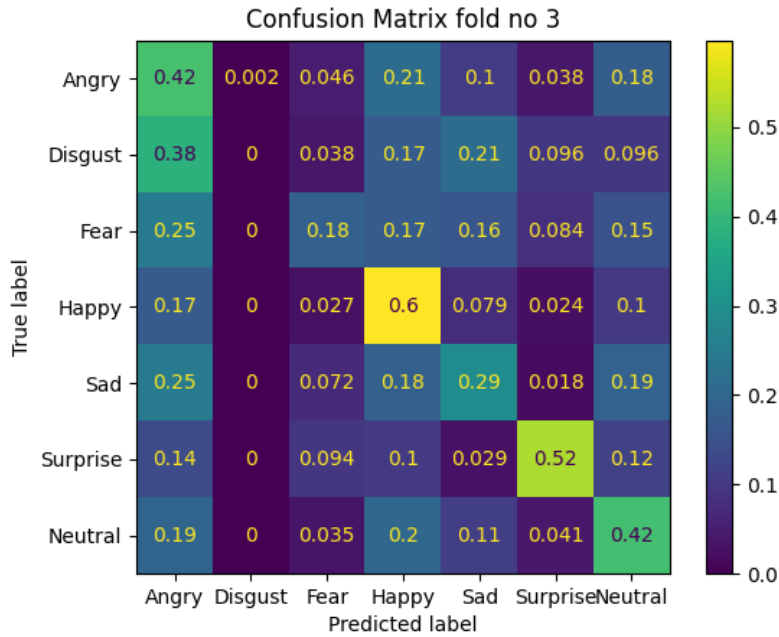


Figure A.9: Confusion matrix over 20 epochs with pre-trained VGG16 model fold 3

## Fold 4



Figure A.10: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 4

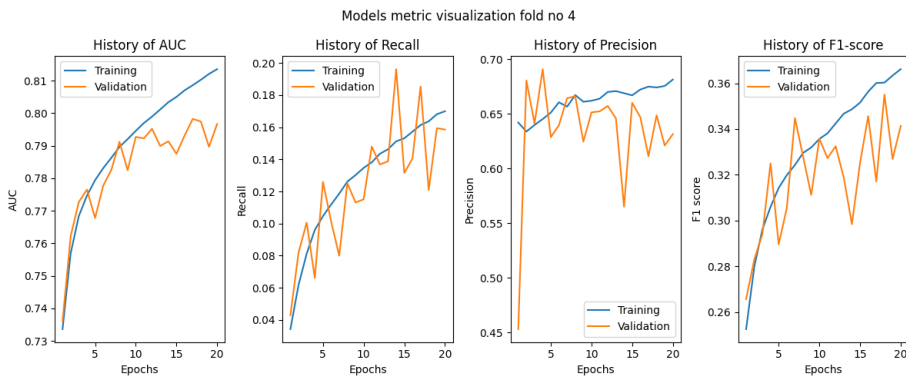


Figure A.11: All metrics over 20 epochs with pre-trained VGG16 model fold 4

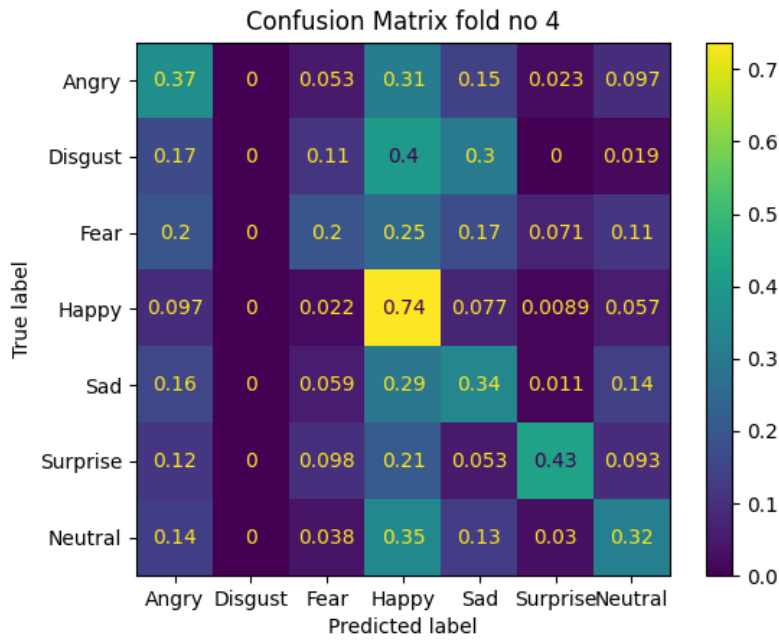


Figure A.12: Confusion matrix over 20 epochs with pre-trained VGG16 model fold 4

### Fold 5

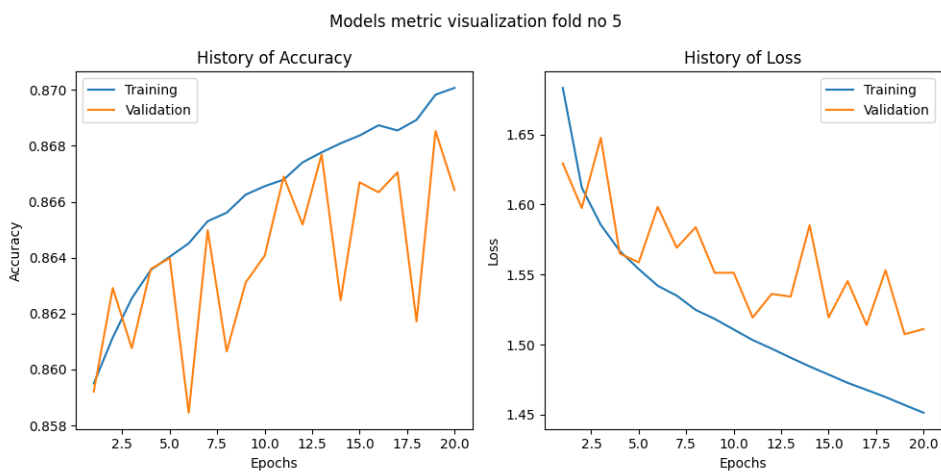


Figure A.13: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 5

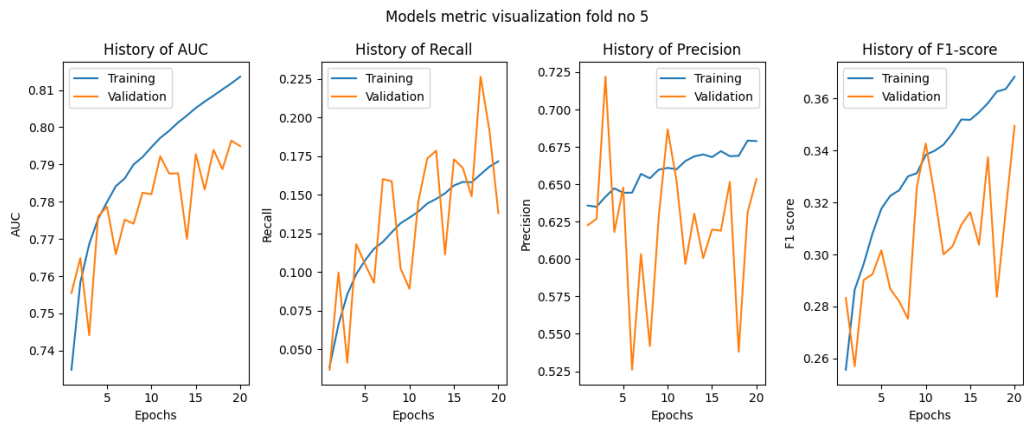


Figure A.14: All metrics over 20 epochs with pre-trained VGG16 model fold 5

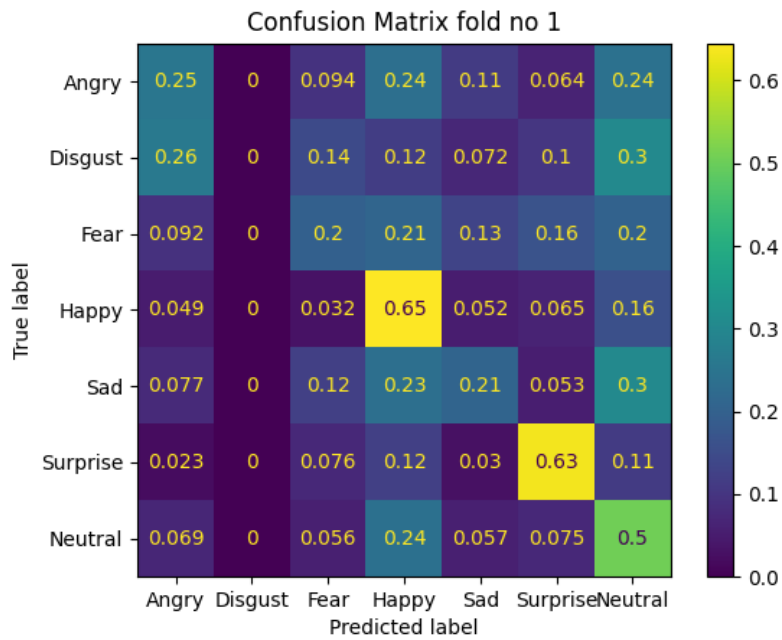


Figure A.15: Confusion matrix over 20 epochs with pre-trained VGG16 model fold 5

## Fold 6

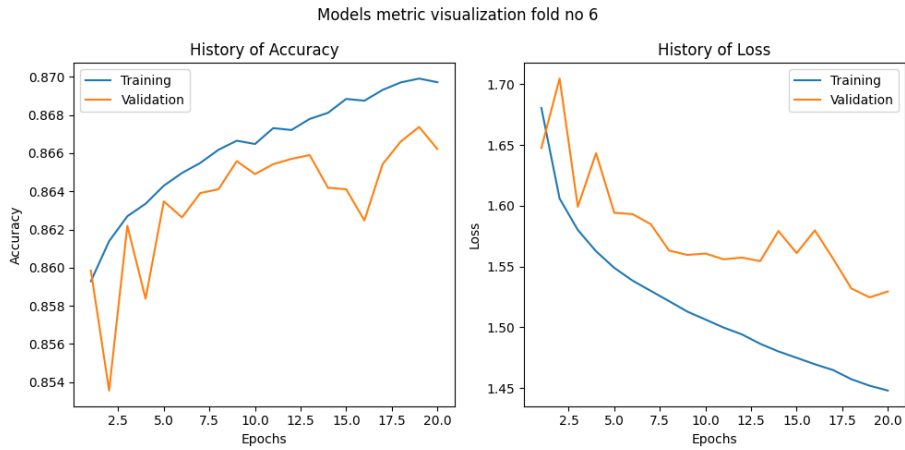


Figure A.16: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 6

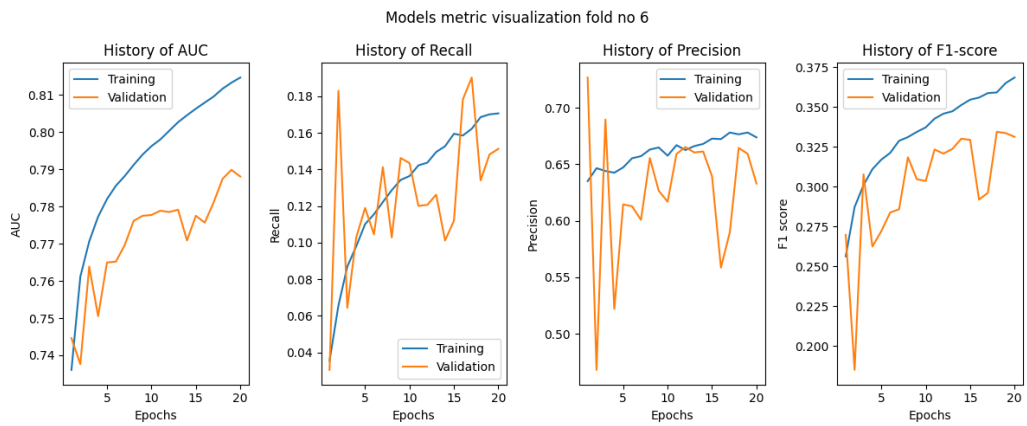
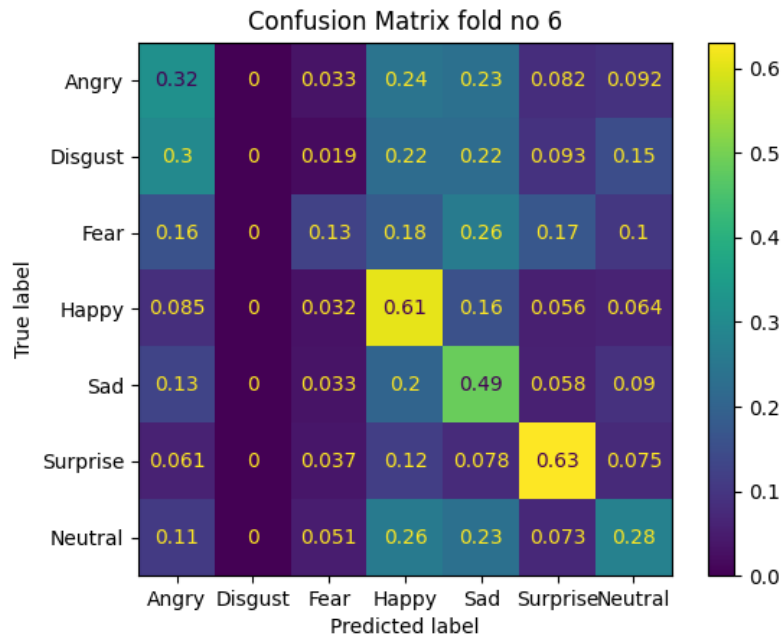


Figure A.17: All metrics over 20 epochs with pre-trained VGG16 model fold 6



Confusion matrix over 20 epochs with pre-trained VGG16 model fold 6

### Fold 7

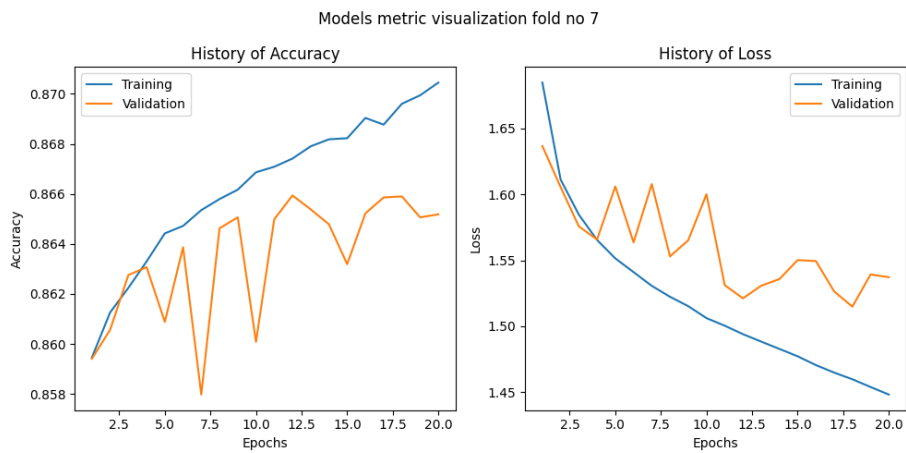


Figure A.18: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 7



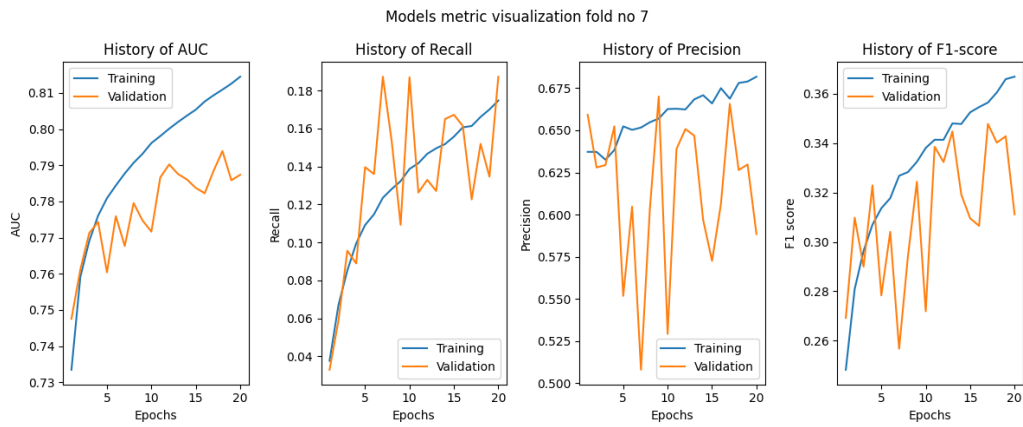


Figure A.19: All metrics over 20 epochs with pre-trained VGG16 model fold 7

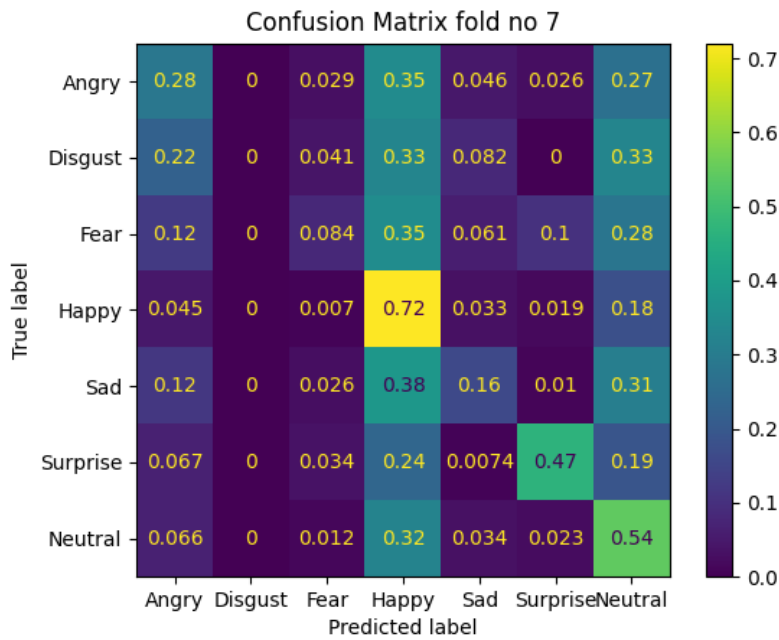


Figure A.20: Confusion matrix over 20 epochs with pre-trained VGG16 model fold 7

## Fold 8

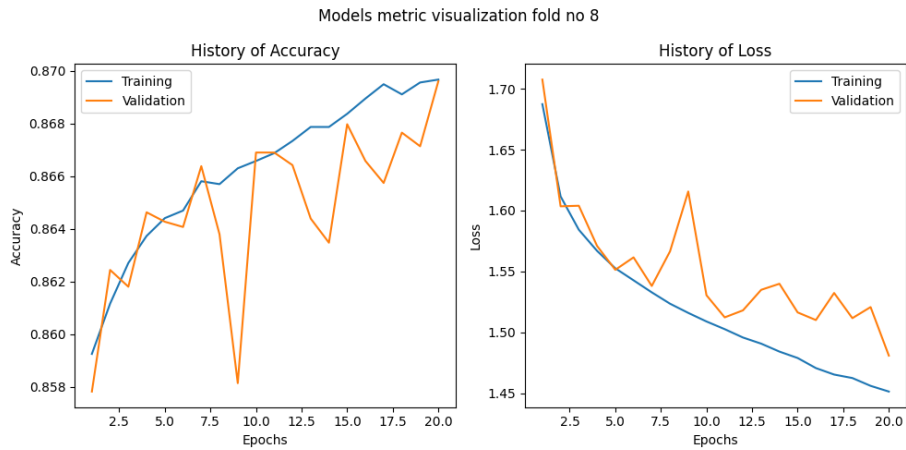


Figure A.21: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 8

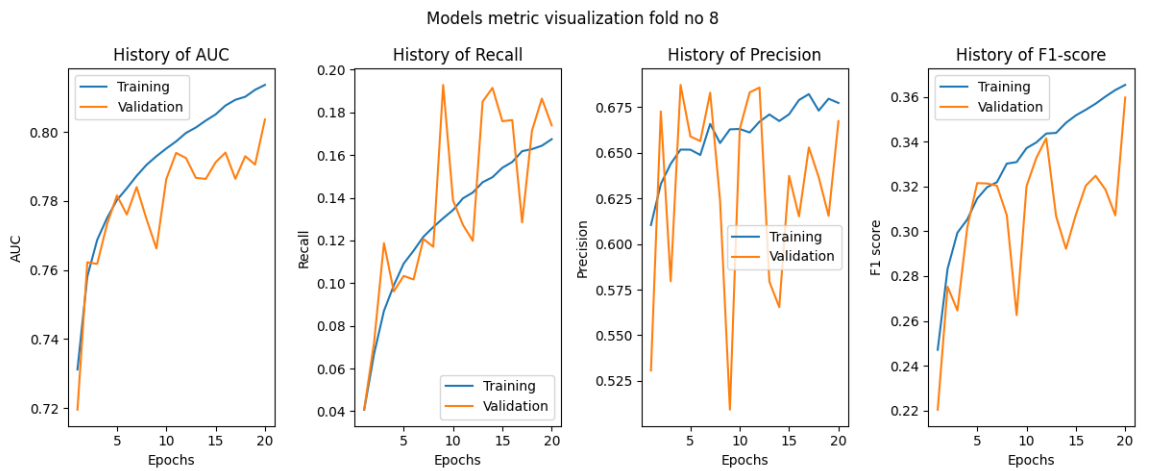


Figure A.22: All metrics over 20 epochs with pre-trained VGG16 model fold 8

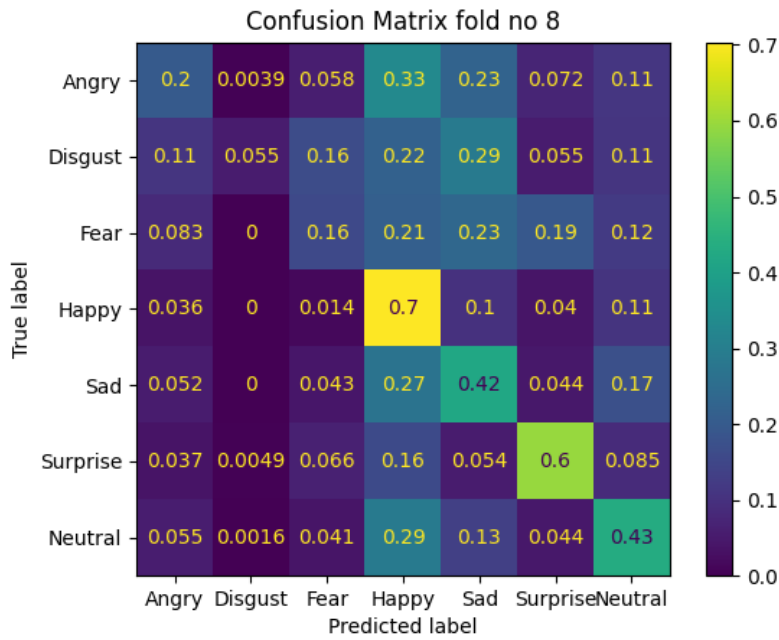


Figure A.23: Confusion matrix over 20 epochs with pre-trained VGG16 model fold 8

### Fold 9

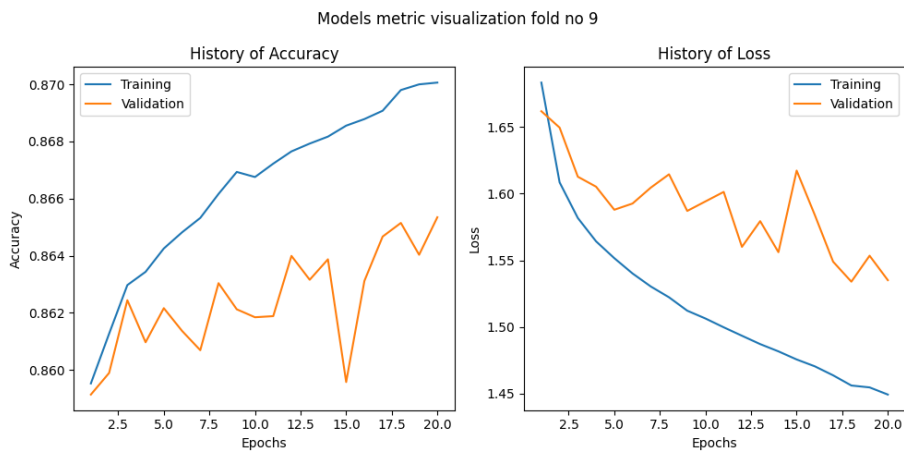


Figure A.24: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 9

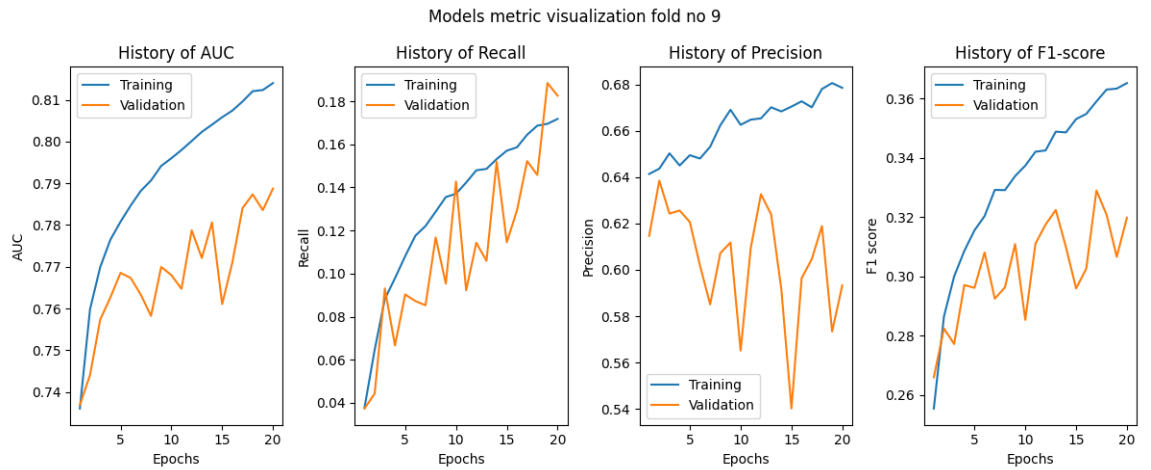


Figure A.25: All metrics over 20 epochs with pre-trained VGG16 model fold 9

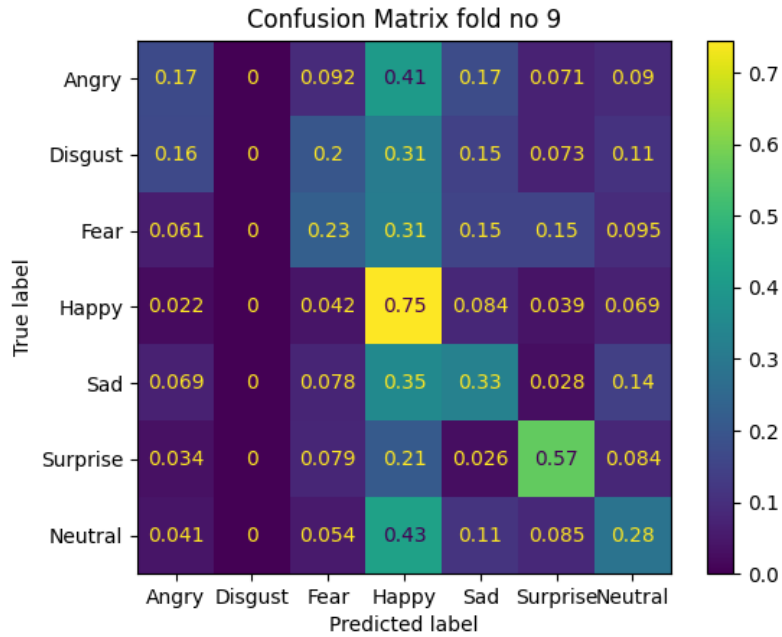


Figure A.26: Confusion matrix over 20 epochs with pre-trained VGG16 model fold 9

## Fold 10

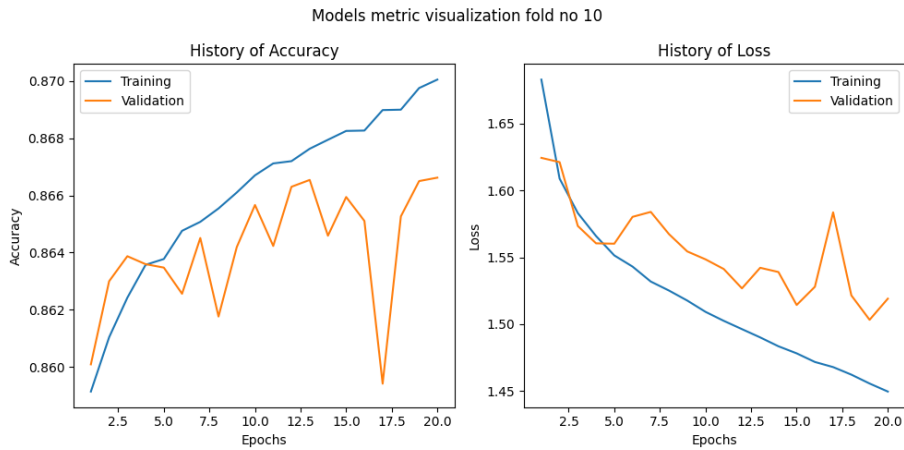


Figure A.27: Train and validation accuracy over 20 epochs with pre-trained VGG16 model fold 10

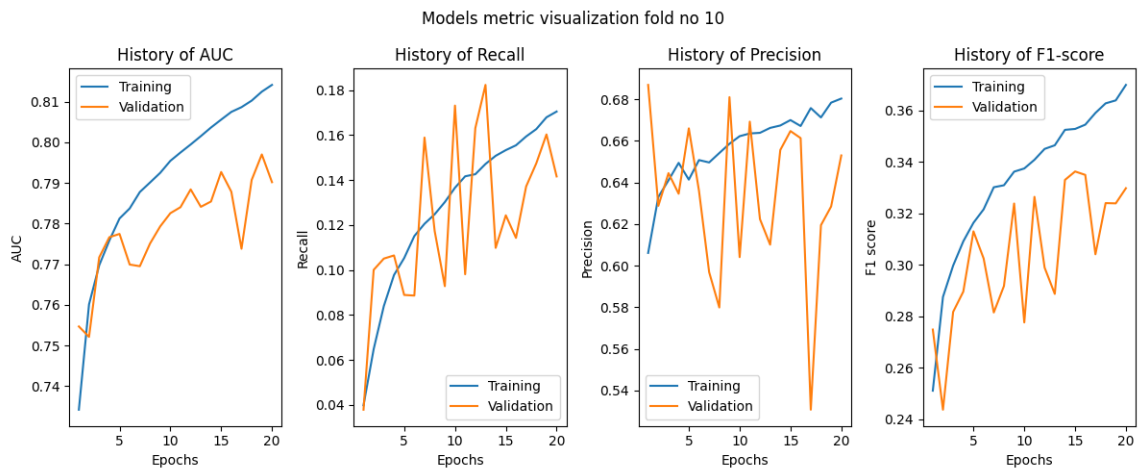
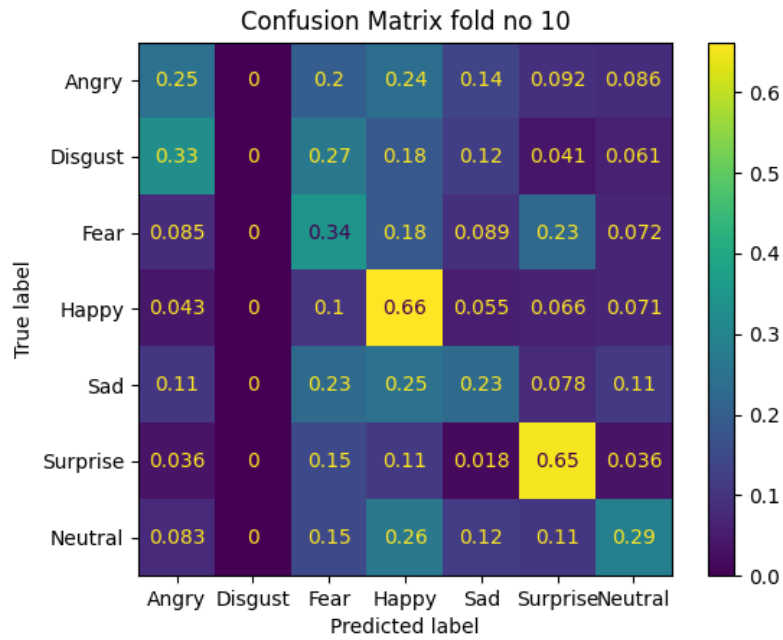


Figure A.28: All metrics over 20 epochs with pre-trained VGG16 model fold 10



Confusion matrix over 20 epochs with pre-trained VGG16 model fold 10

### Average scores all folds

```

Average scores for all folds:
> Accuracy: 86.69912874698639 (+- 0.13142458272266172)
> Loss: 1.5132954001426697

```

Figure A.29: Average scores for accuracy and loss with pre-trained VGG model for all 10 folds on train dataset

```

> Fold 1 - Loss: 1.5085598230361938 - Accuracy: 86.75716519355774%
-----
> Fold 2 - Loss: 1.4998388290405273 - Accuracy: 86.82879209518433%
-----
> Fold 3 - Loss: 1.5076884031295776 - Accuracy: 86.8088960647583%
-----
> Fold 4 - Loss: 1.5041308403015137 - Accuracy: 86.65763139724731%
-----
> Fold 5 - Loss: 1.511084794998169 - Accuracy: 86.64173483848572%
-----
> Fold 6 - Loss: 1.5295354127883911 - Accuracy: 86.62183284759521%
-----
> Fold 7 - Loss: 1.5371716022491455 - Accuracy: 86.51833534240723%
-----
> Fold 8 - Loss: 1.4807922840118408 - Accuracy: 86.96051836013794%
-----
> Fold 9 - Loss: 1.535049557685852 - Accuracy: 86.53448820114136%
-----
> Fold 10 - Loss: 1.5191024541854858 - Accuracy: 86.66189312934875%

```

Figure A.30: Calculated accuracy and loss with pre-trained VGG model on train dataset per fold

## A.2 Resnet50 model

Training of pre-trained Resnet50 model with weights "imagenet" during 20 epochs and 10 k folds.

### Fold 1

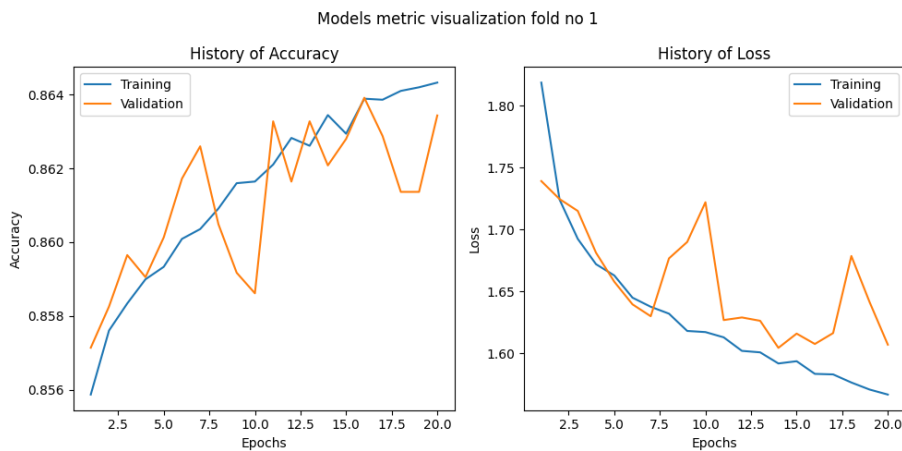


Figure A.31: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 1

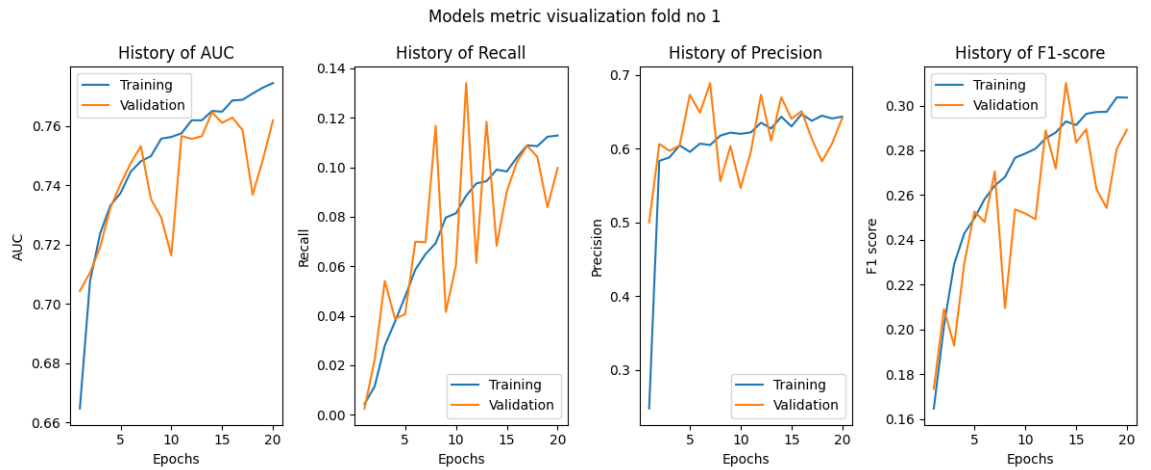


Figure A.32: All metrics for 20 epochs with pre-trained Resnet50 model fold 1

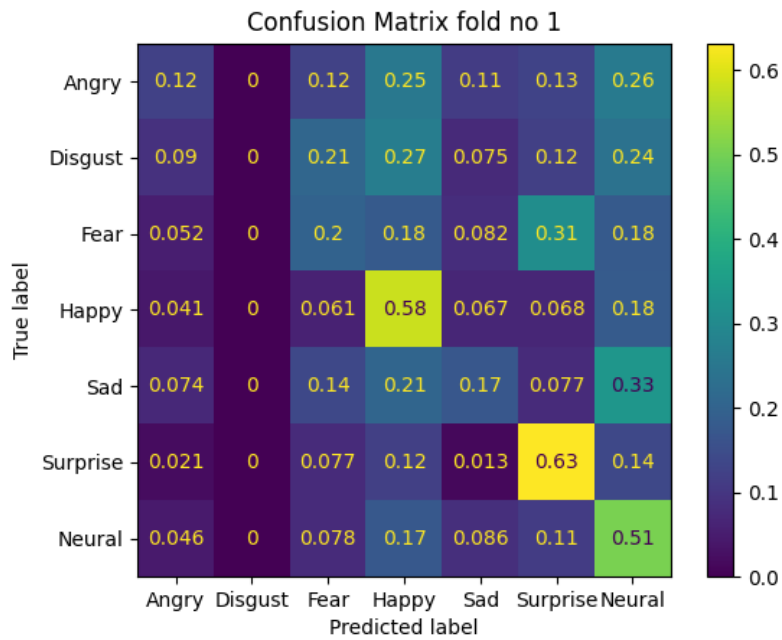


Figure A.33: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 1





Figure A.34: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 2

## Fold 2

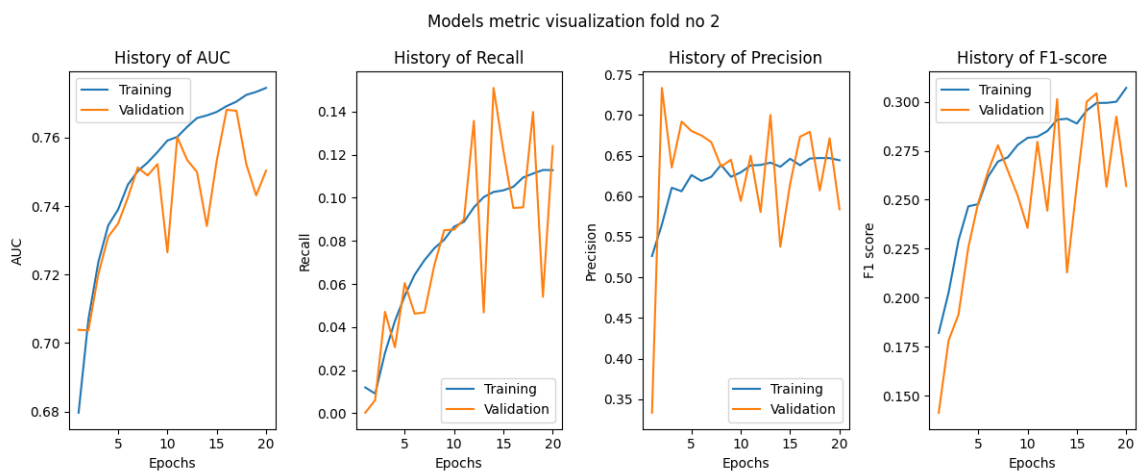


Figure A.35: All metrics for 20 epochs with pre-trained Resnet50 model fold 2

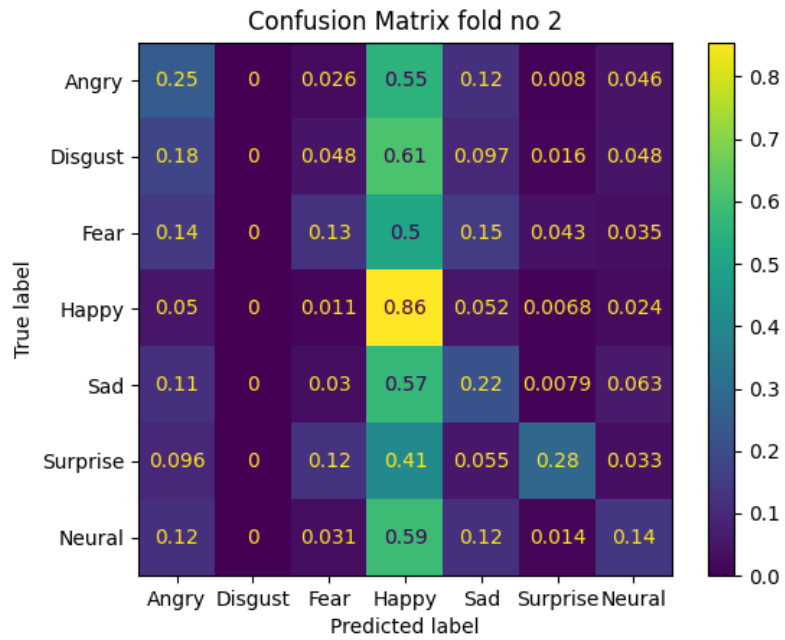


Figure A.36: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 2

### Fold 3

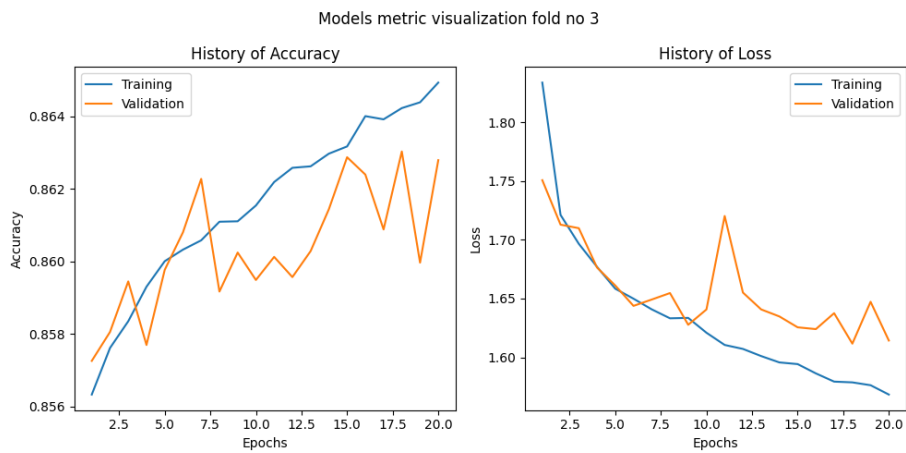


Figure A.37: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 3

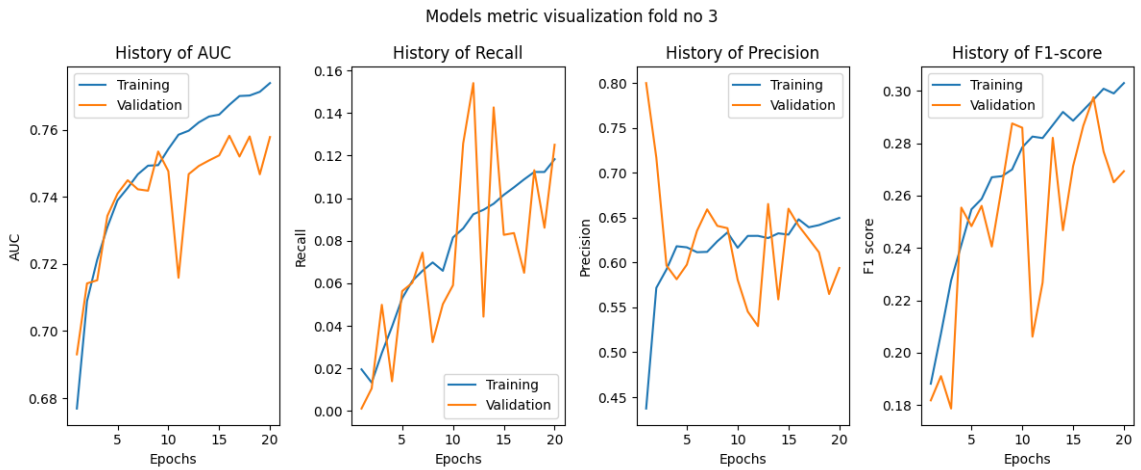


Figure A.38: All metrics for 20 epochs with pre-trained Resnet50 model fold 3

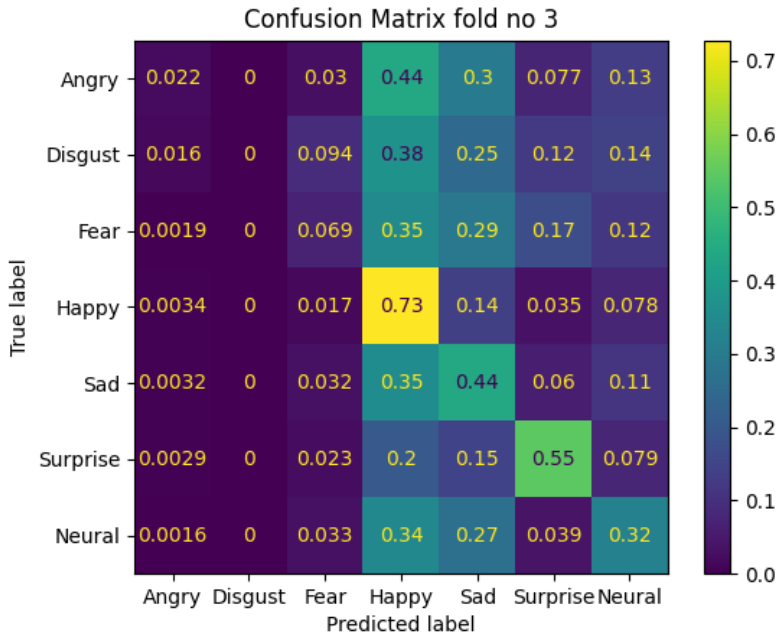


Figure A.39: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 3

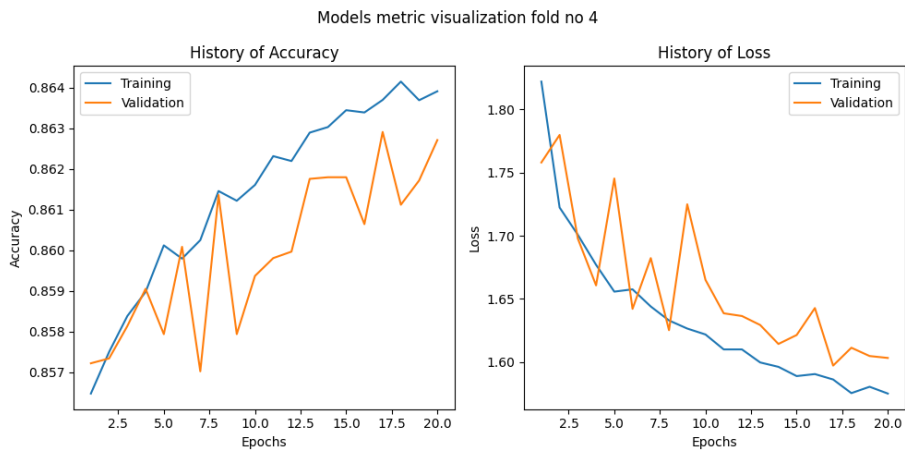


Figure A.40: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 4

#### Fold 4

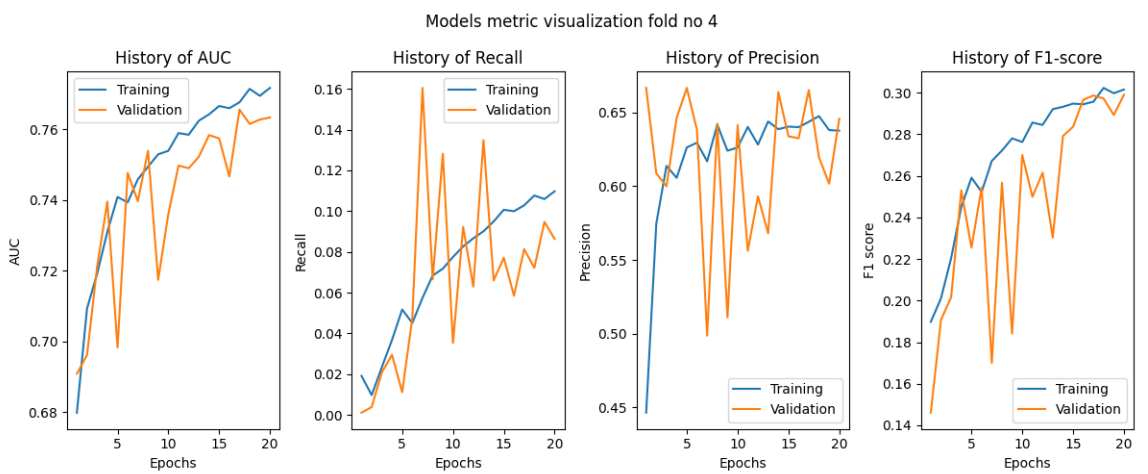


Figure A.41: All metrics for 20 epochs with pre-trained Resnet50 model fold 4

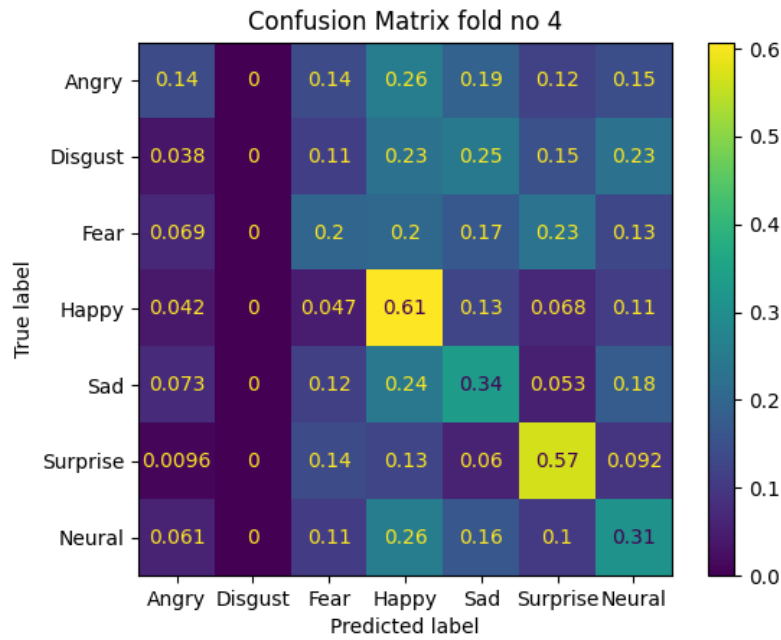


Figure A.42: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 4

### Fold 5

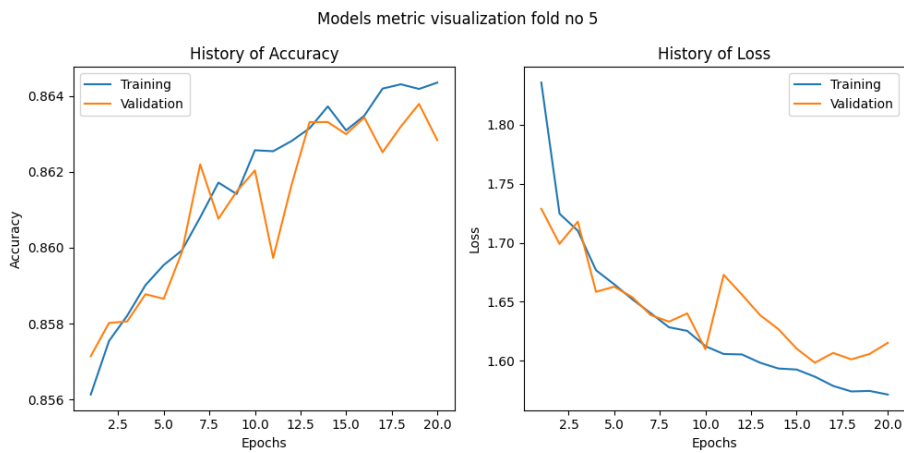


Figure A.43: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 5

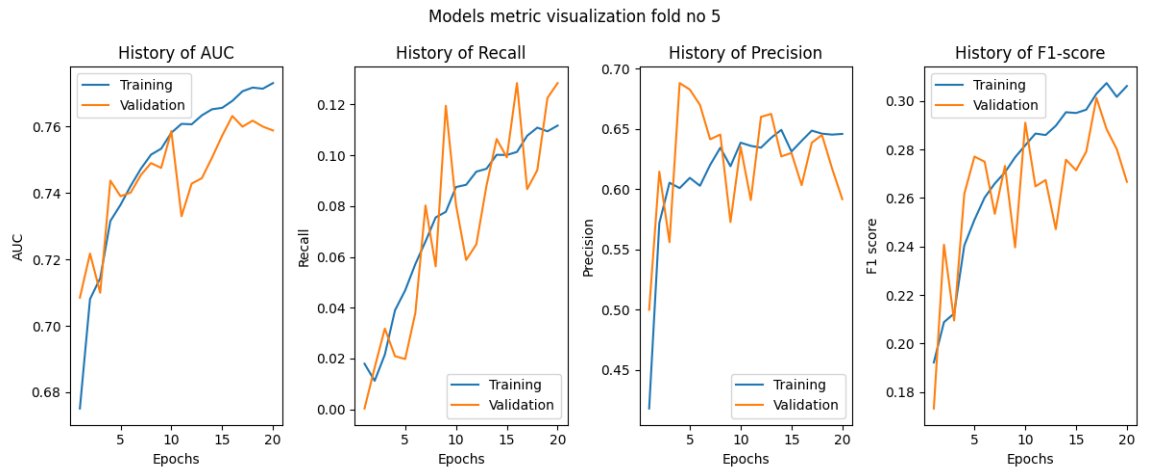


Figure A.44: All metrics for 20 epochs with pre-trained Resnet50 model fold 5

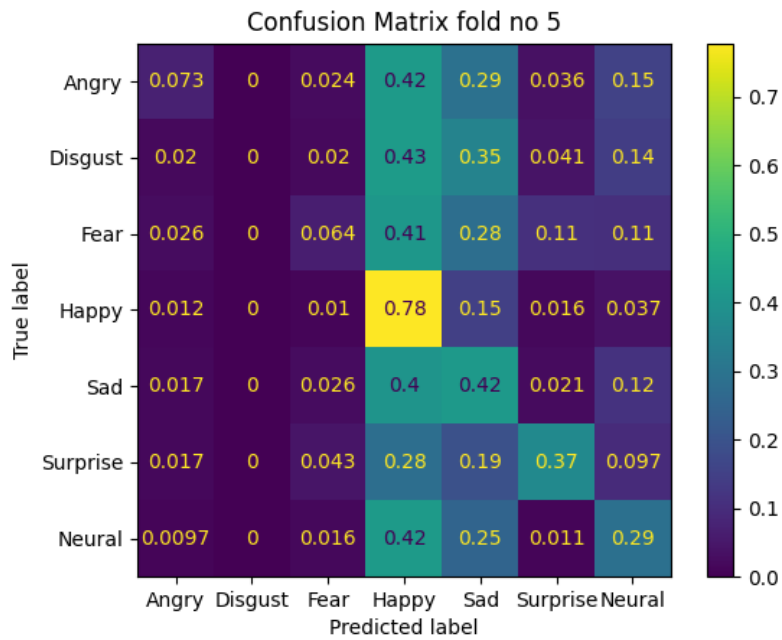


Figure A.45: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 5



Figure A.46: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 6

### Fold 6

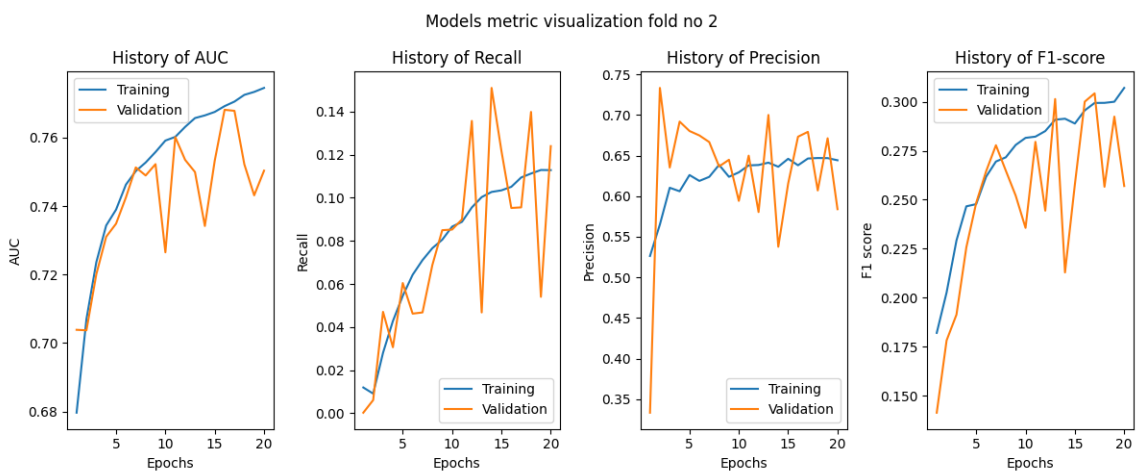


Figure A.47: All metrics for 20 epochs with pre-trained Resnet50 model fold 6

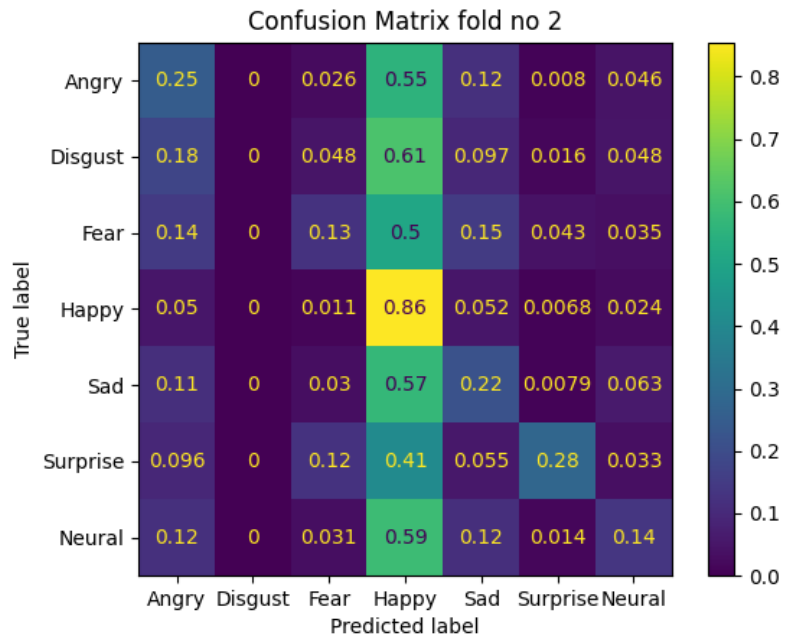


Figure A.48: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 6

### Fold 7



Figure A.49: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 7



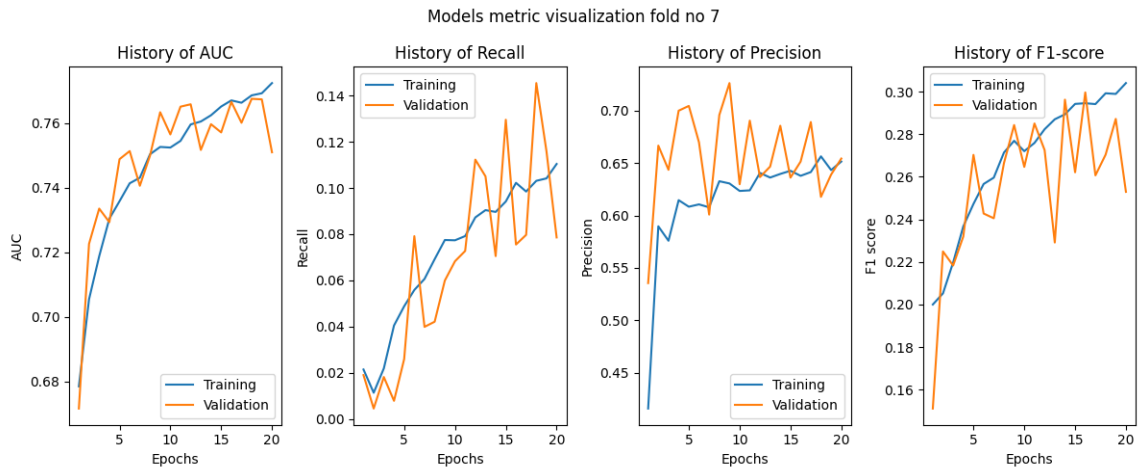


Figure A.50: All metrics for 20 epochs with pre-trained Resnet50 model fold 7

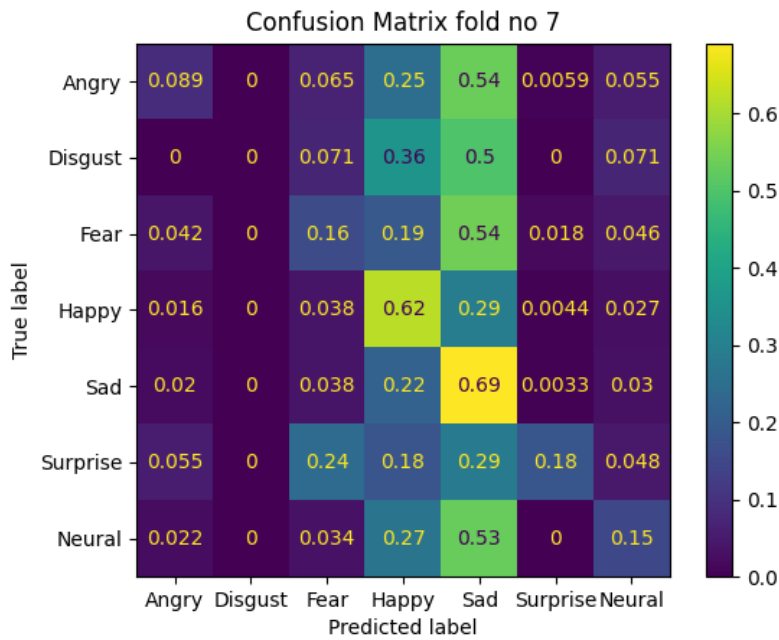


Figure A.51: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 7

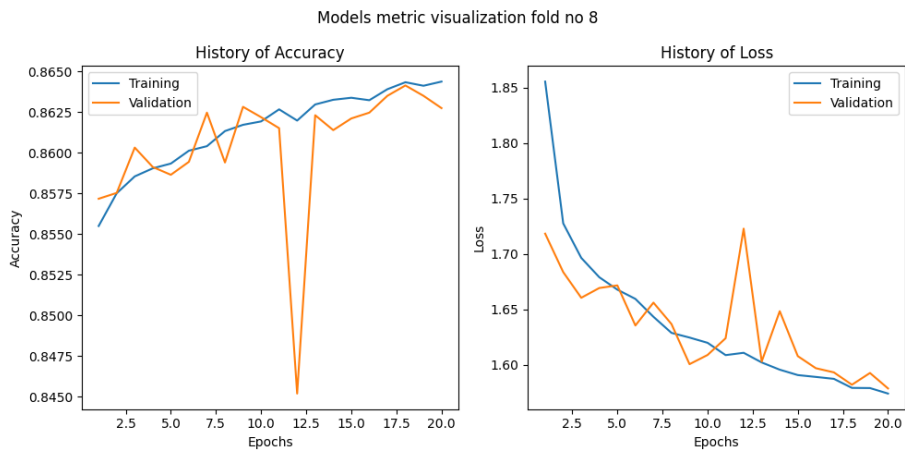


Figure A.52: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 8

### Fold 8

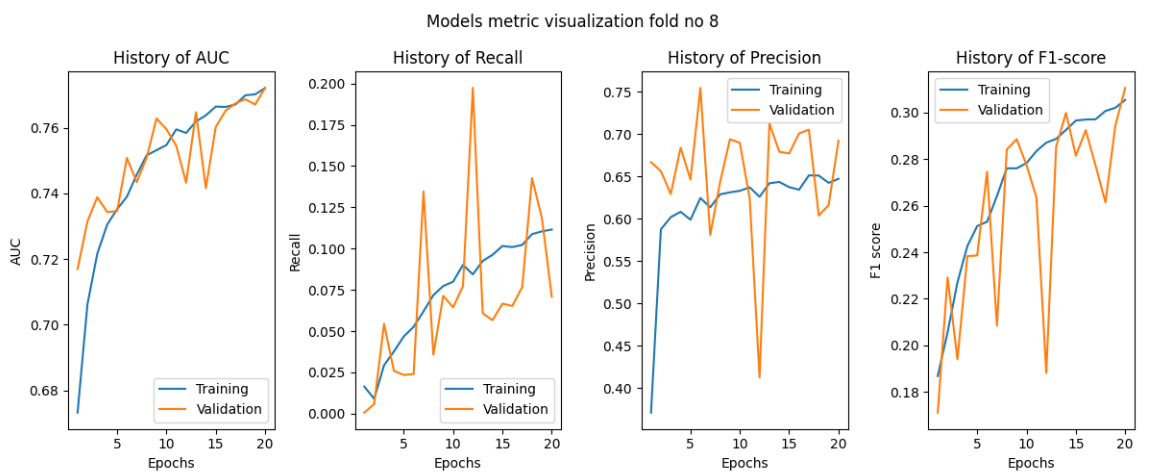


Figure A.53: All metrics for 20 epochs with pre-trained Resnet50 model fold 8

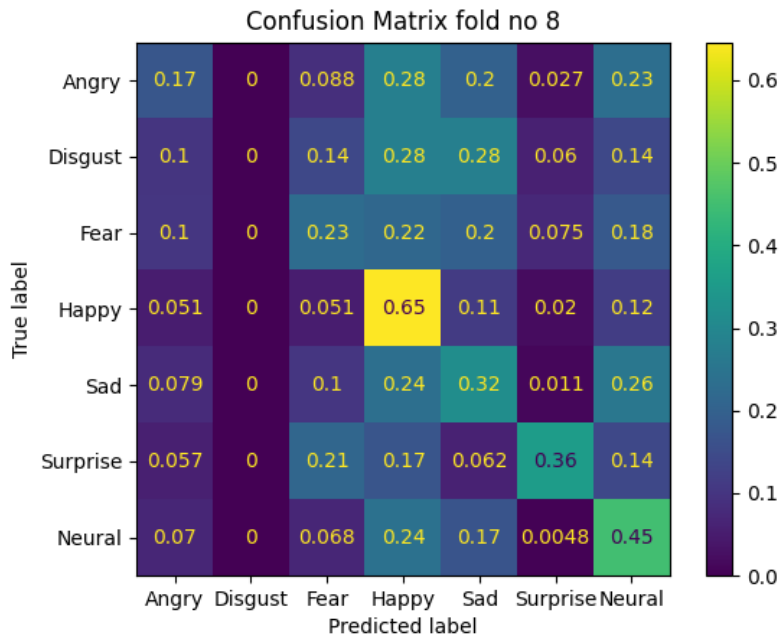


Figure A.54: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 8

### Fold 9

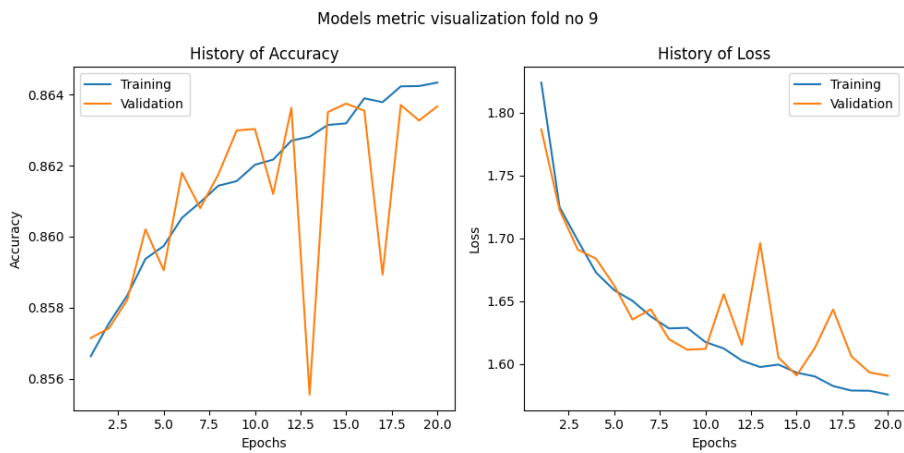


Figure A.55: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 9

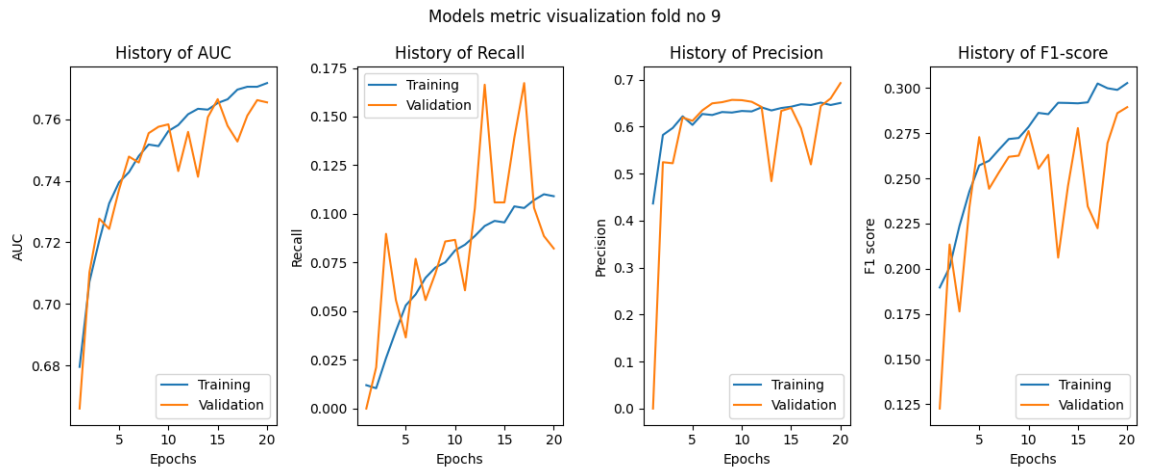


Figure A.56: All metrics for 20 epochs with pre-trained Resnet50 model fold 9

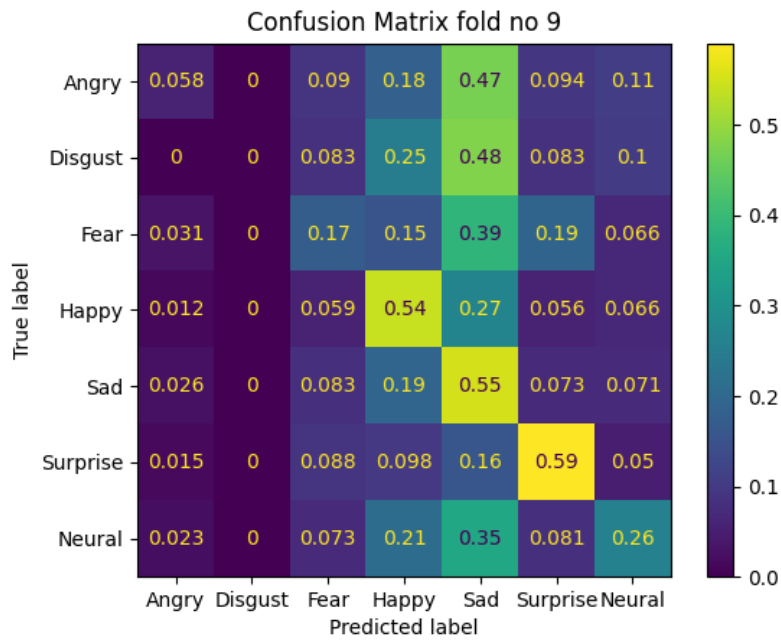


Figure A.57: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 9



Figure A.58: Train and validation accuracy over 20 epochs with pre-trained Resnet50 model fold 10

### Fold 10

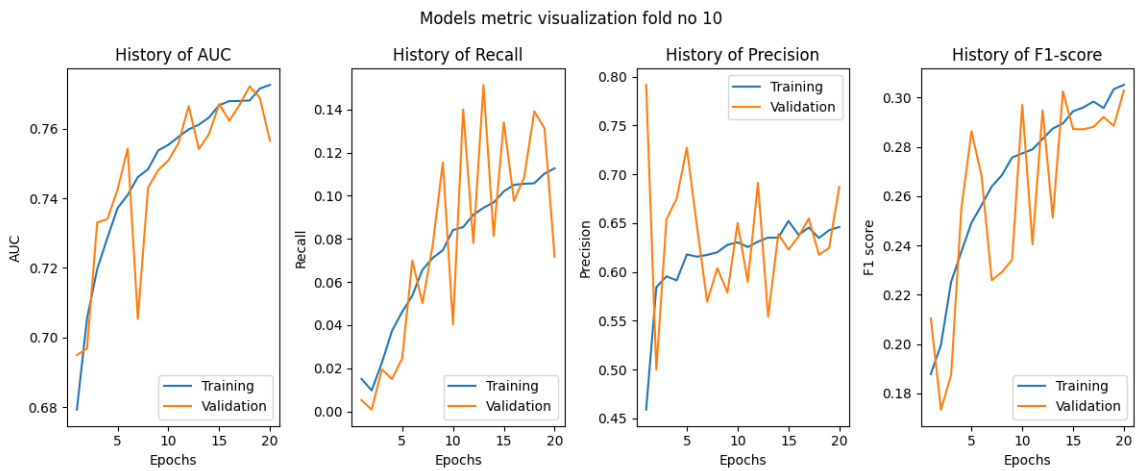


Figure A.59: All metrics for 20 epochs with pre-trained Resnet50 model fold 10

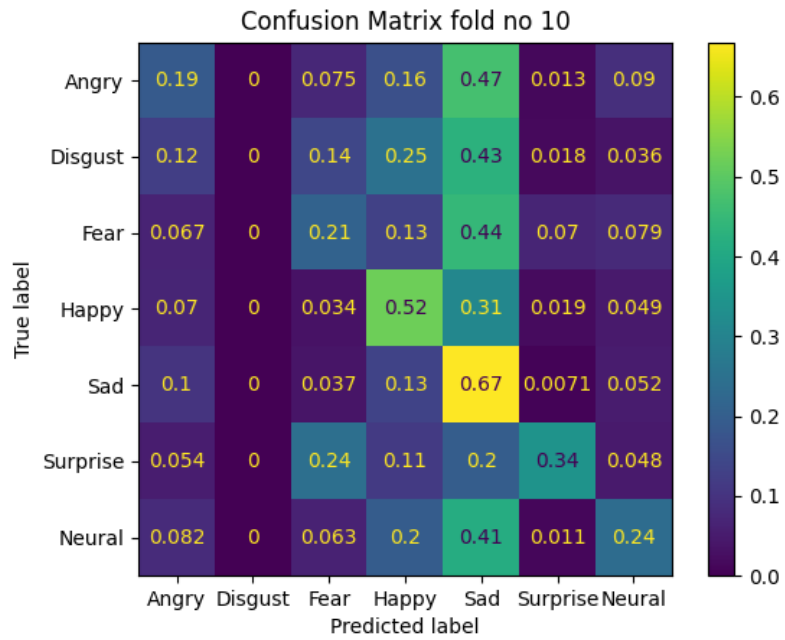


Figure A.60: Confusion matrix for 20 epochs with pre-trained Resnet50 model fold 10

### Average scores all folds

```

Average scores for all folds:
> Accuracy: 86.27955377101898 (+- 0.04267890817100542)
> Loss: 1.609612262248993

```

Figure A.61: Average scores for accuracy and loss with pre-trained Resnet50 model for all 10 folds on train dataset

```

Score per fold
-----
> FoId 1 - Loss: 1.6070618629455566 - Accuracy: 86.3431990146637%
-----
> FoId 2 - Loss: 1.6340068578720093 - Accuracy: 86.2237811088562%
-----
> FoId 3 - Loss: 1.6143078804016113 - Accuracy: 86.27947568893433%
-----
> FoId 4 - Loss: 1.6033358573913574 - Accuracy: 86.27153635025024%
-----
> FoId 5 - Loss: 1.6151304244995117 - Accuracy: 86.28345727920532%
-----
> FoId 6 - Loss: 1.592206358909607 - Accuracy: 86.23573780059814%
-----
> FoId 7 - Loss: 1.6421787738800049 - Accuracy: 86.24370694160461%
-----
> FoId 8 - Loss: 1.5789225101470947 - Accuracy: 86.27567291259766%
-----
> FoId 9 - Loss: 1.5904293060302734 - Accuracy: 86.36726140975952%
-----
> FoId 10 - Loss: 1.6185427904129028 - Accuracy: 86.27170920372009%
-----

```

Figure A.62: Calculated accuracy and loss with pre-trained Resnet50 model on train dataset per fold

### A.3 Sequential model

Training of Sequential model without weights during 50 epochs and 10 k folds.

#### Fold 1

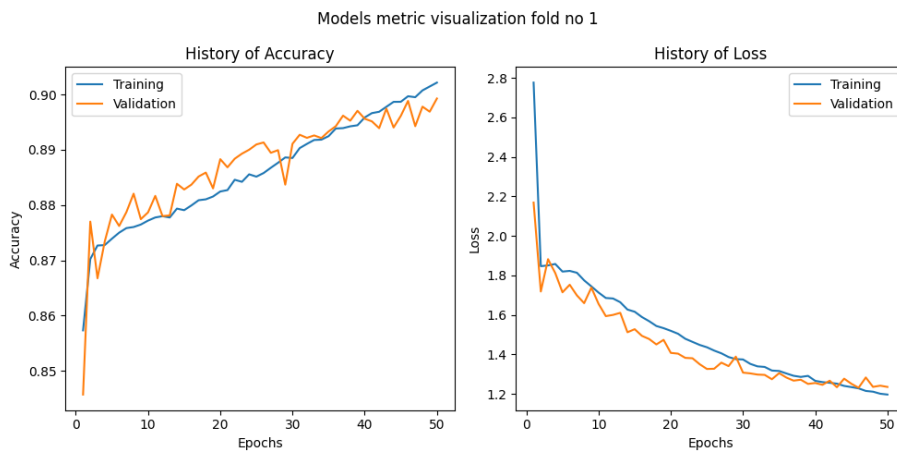


Figure A.63: Train and validation accuracy over 50 epochs with Sequential model fold 1

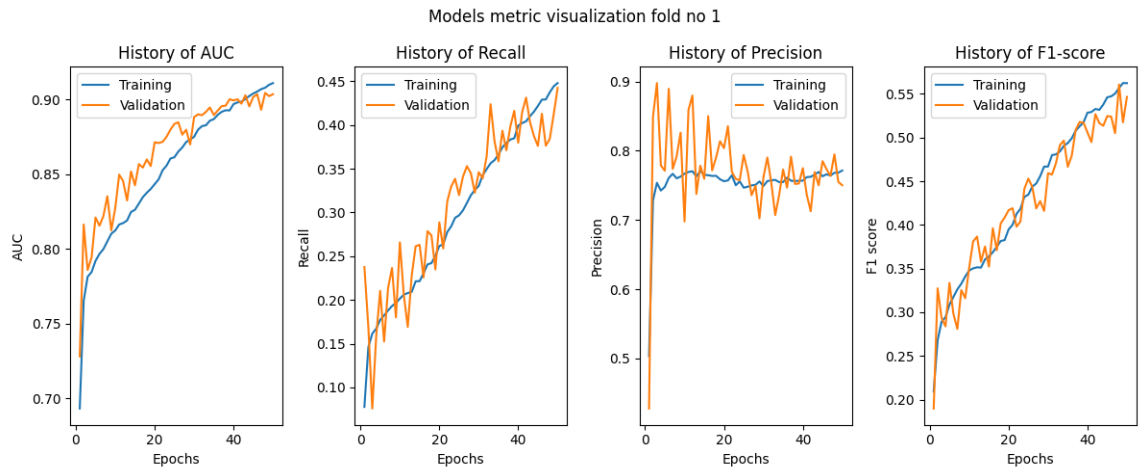


Figure A.64: All metrics over 50 epochs with Sequential model fold 1

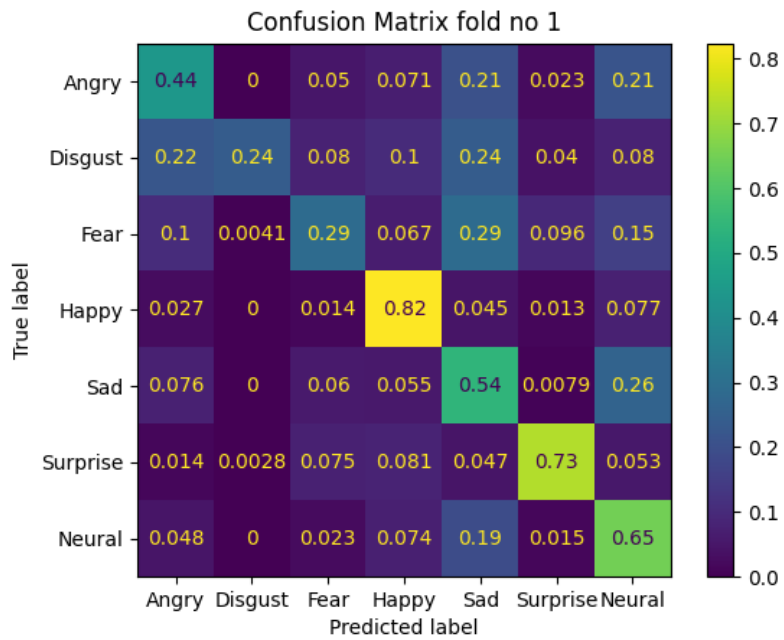


Figure A.65: Confusion matrix over 50 epochs with Sequential model fold 1



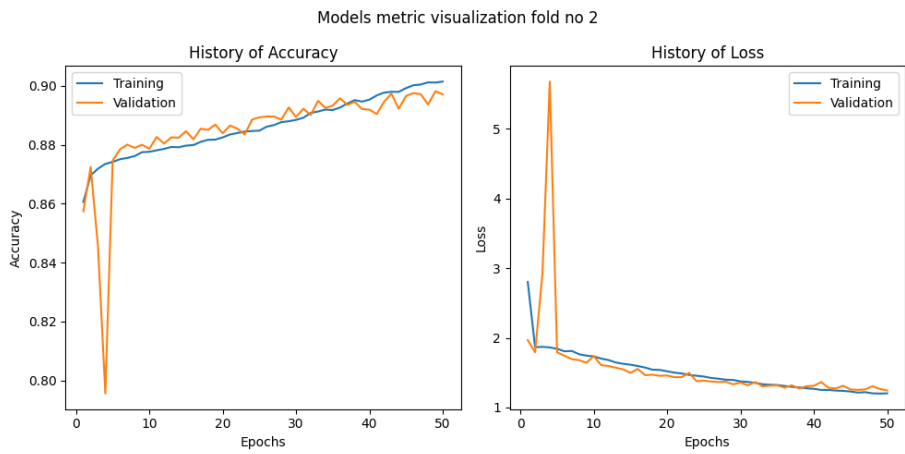


Figure A.66: Train and validation accuracy over 50 epochs with Sequential model fold 2

## Fold 2

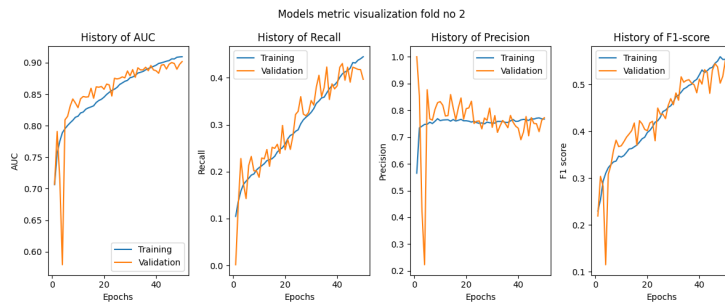


Figure A.67: All metrics over 50 epochs with Sequential model fold 2

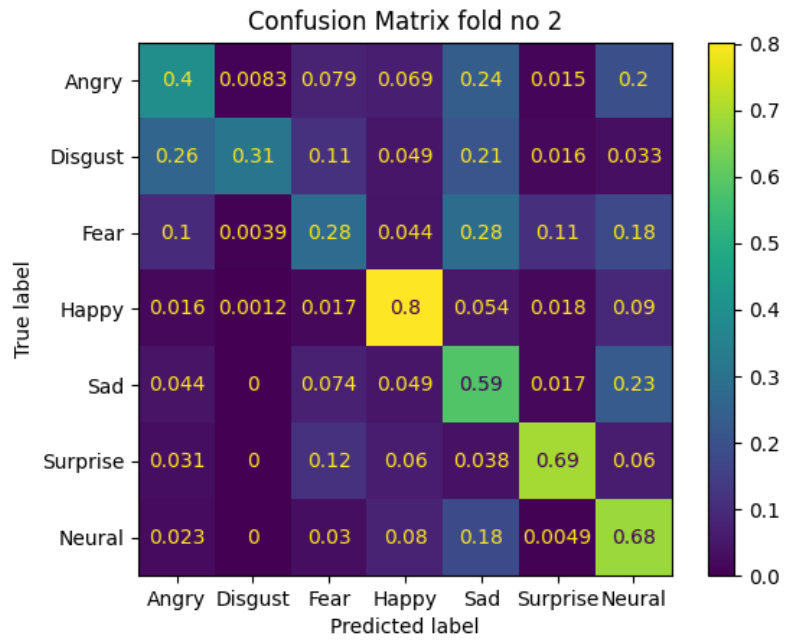


Figure A.68: Confusion matrix over 50 epochs with Sequential model fold 2

### Fold 3

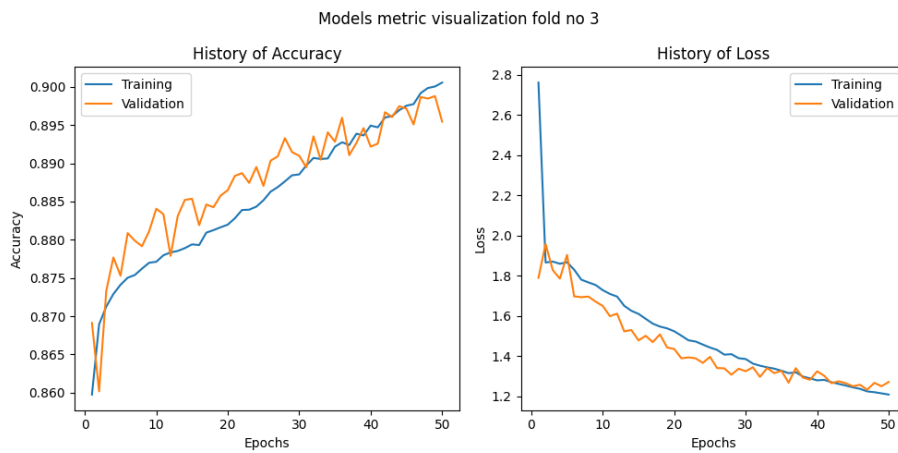


Figure A.69: Train and validation accuracy over 50 epochs with Sequential model fold 3



Figure A.70: All metrics over 50 epochs with Sequential model fold 3

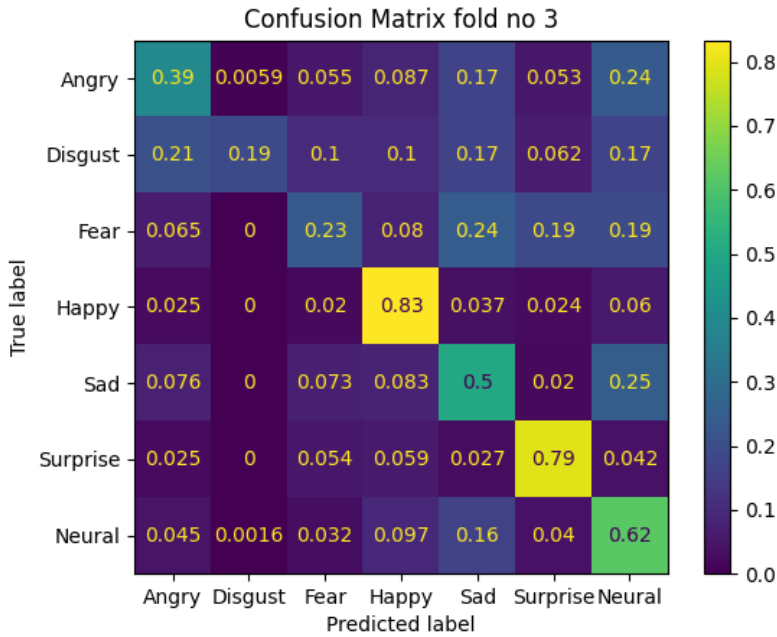


Figure A.71: Confusion matrix over 50 epochs with Sequential model fold 3

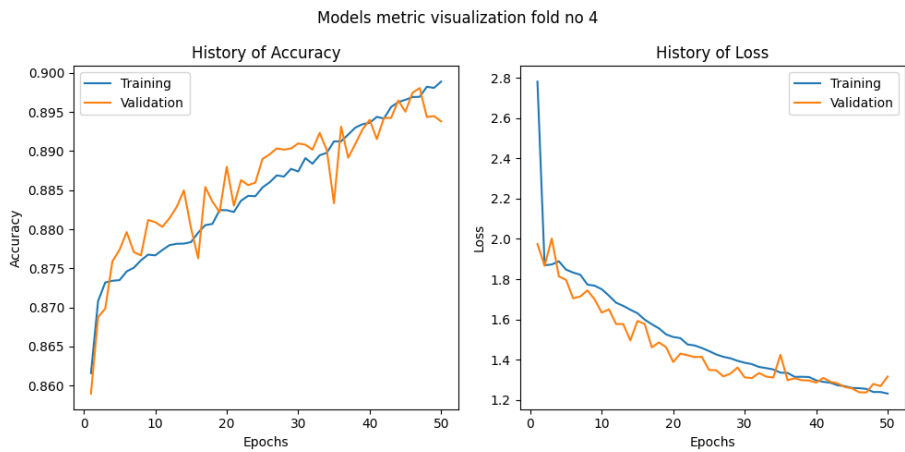


Figure A.72: Train and validation accuracy over 50 epochs with Sequential model fold 4

### Fold 4

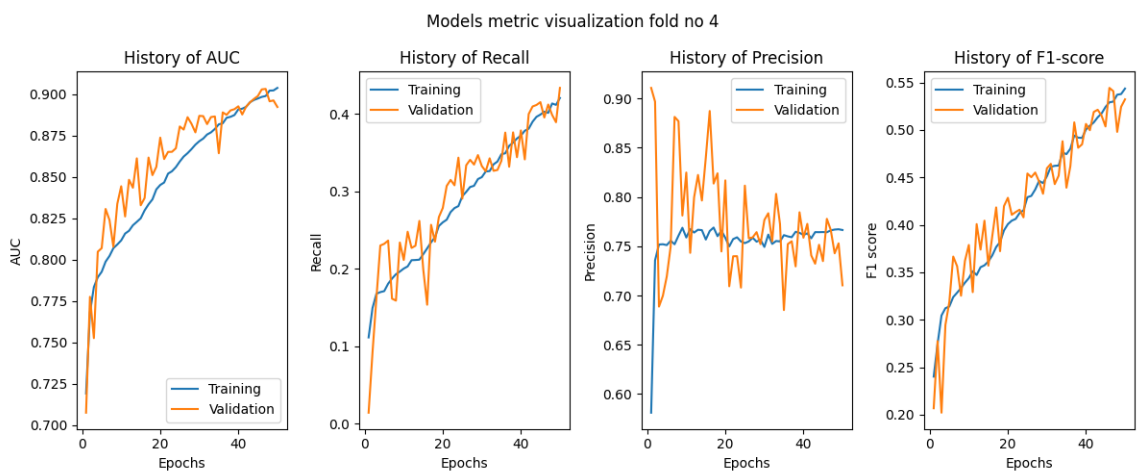


Figure A.73: All metrics over 50 epochs with Sequential model fold 4

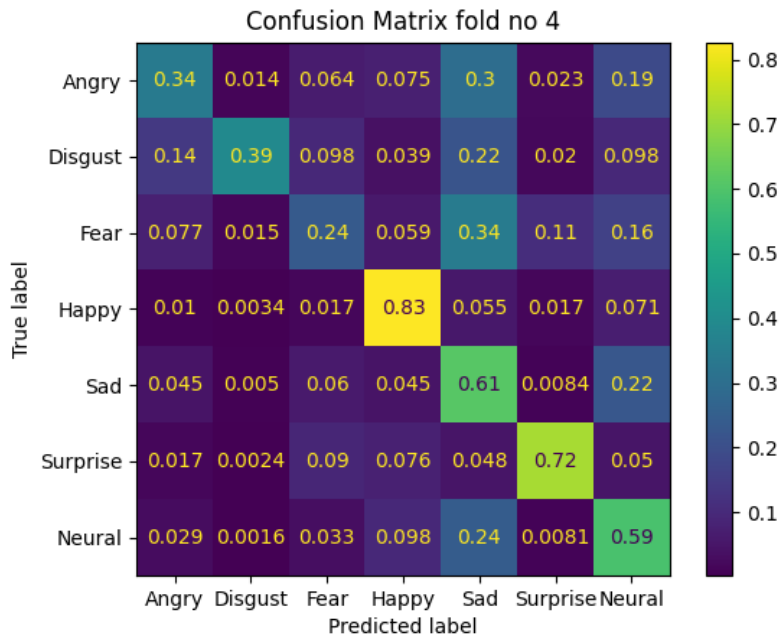


Figure A.74: Confusion matrix over 50 epochs with Sequential model fold 4

### Fold 5

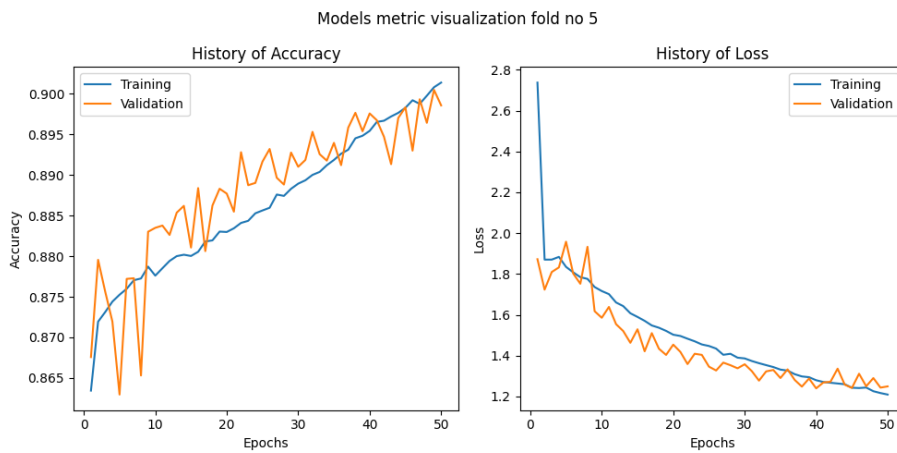


Figure A.75: Train and validation accuracy over 50 epochs with Sequential model fold 5

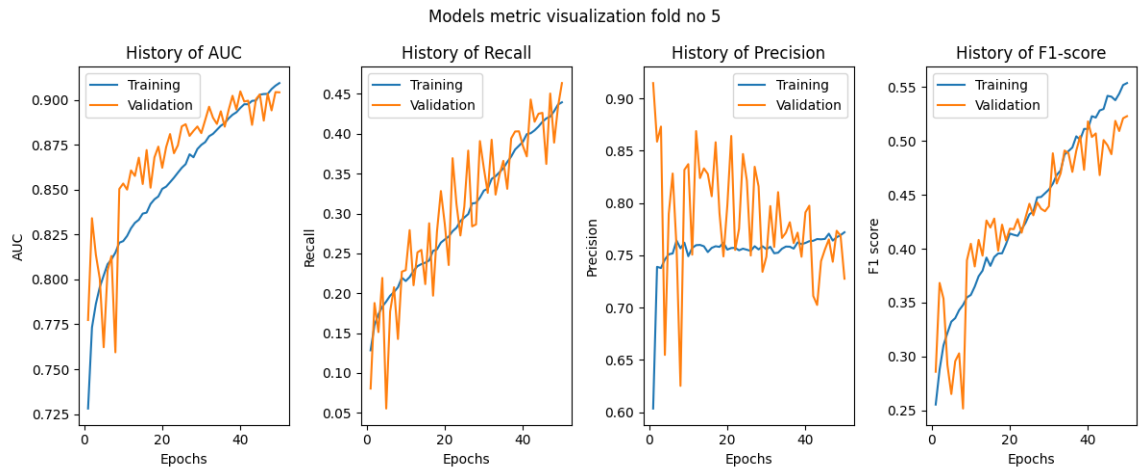


Figure A.76: All metrics over 50 epochs with Sequential model fold 5

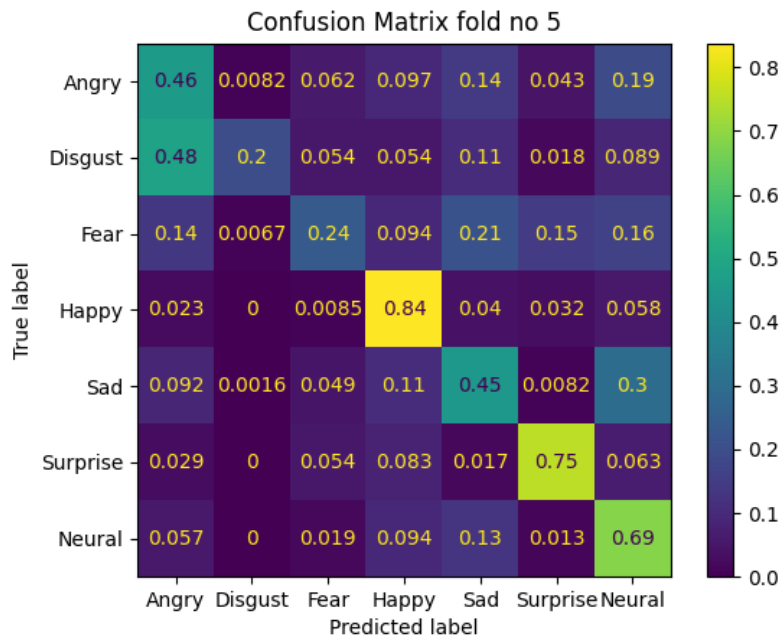


Figure A.77: Confusion matrix over 50 epochs with Sequential model fold 5

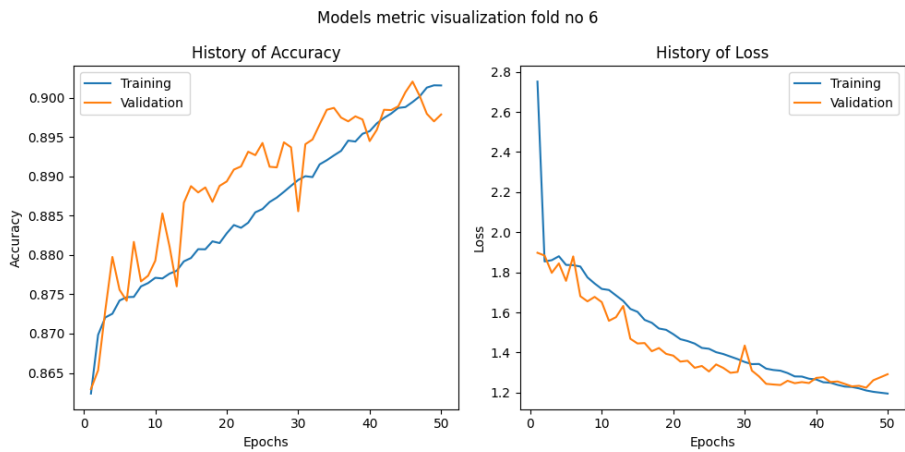


Figure A.78: Train and validation accuracy over 50 epochs with Sequential model fold 6

### Fold 6

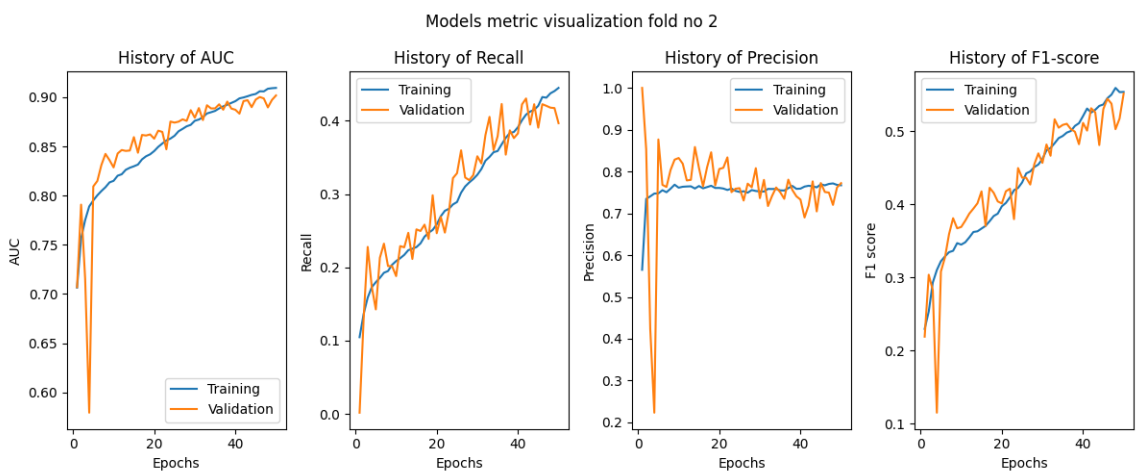


Figure A.79: All metrics over 50 epochs with Sequential model fold 6

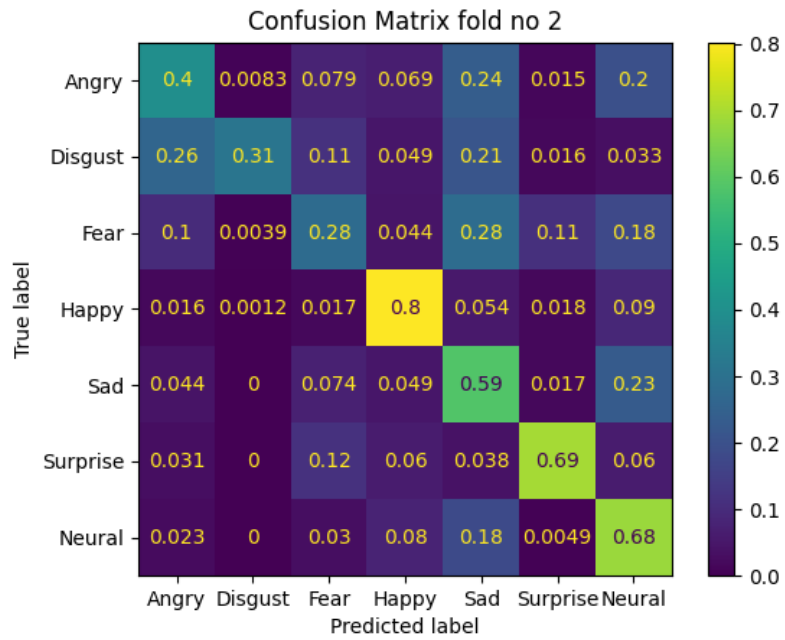


Figure A.80: Confusion matrix over 50 epochs with Sequential model fold 6

**Fold 7**

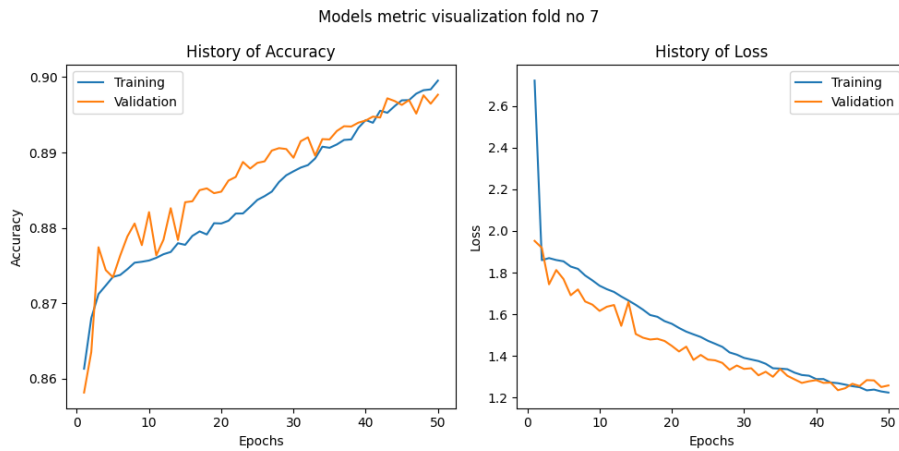


Figure A.81: Train and validation accuracy over 50 epochs with Sequential model fold 7



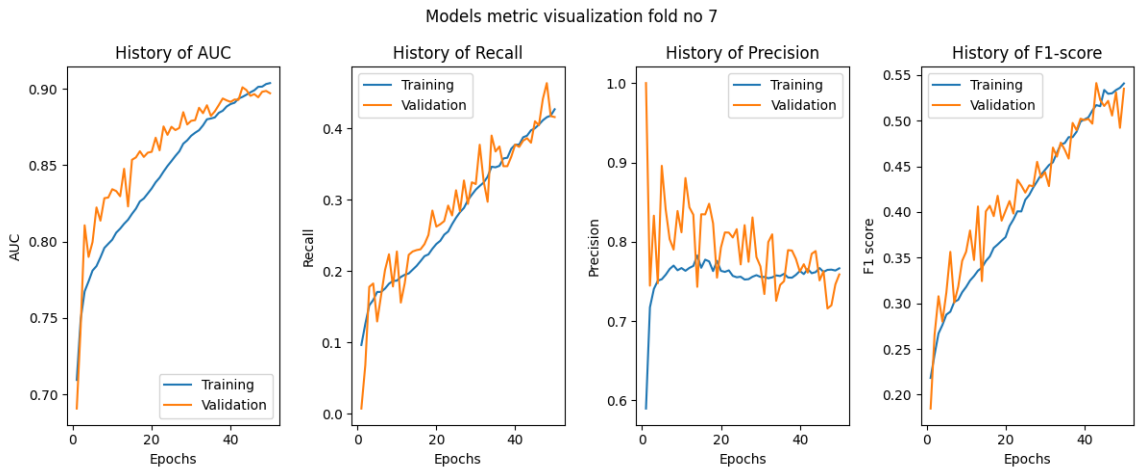


Figure A.82: All metrics over 50 epochs with Sequential model fold 7

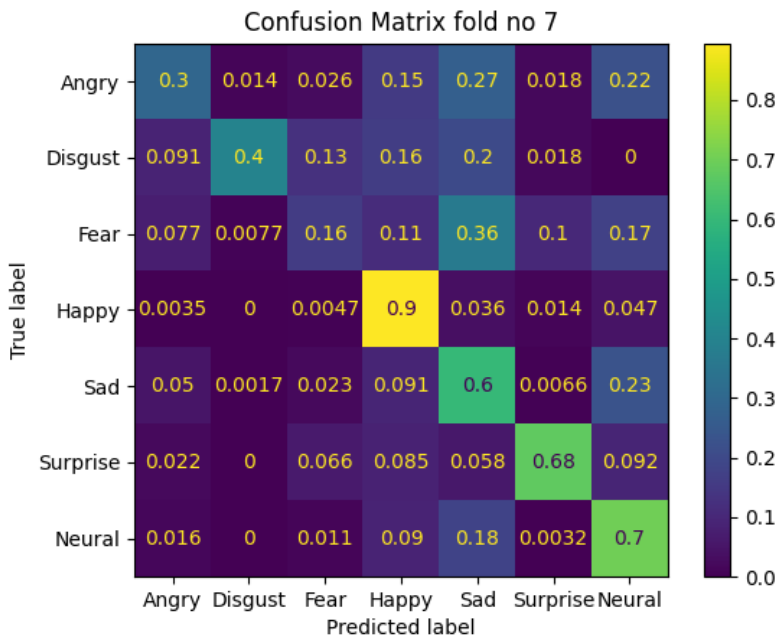


Figure A.83: Confusion matrix over 50 epochs with Sequential model fold 7

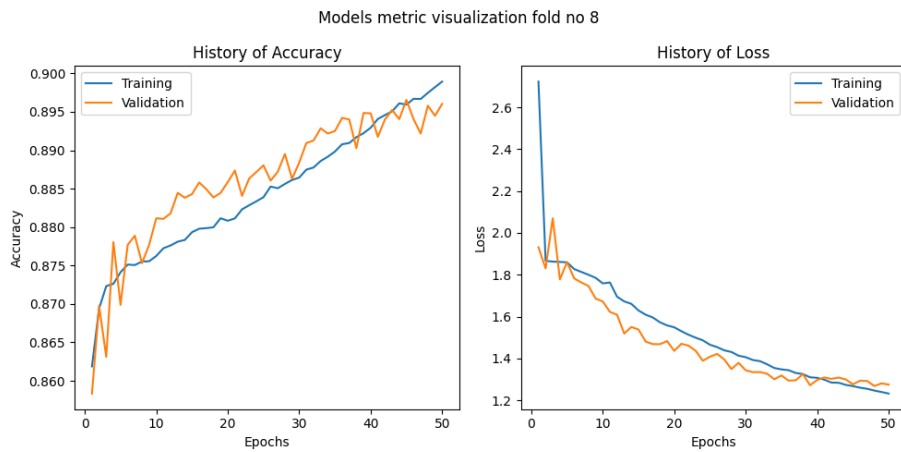


Figure A.84: Train and validation accuracy over 50 epochs with Sequential model fold 8

### Fold 8

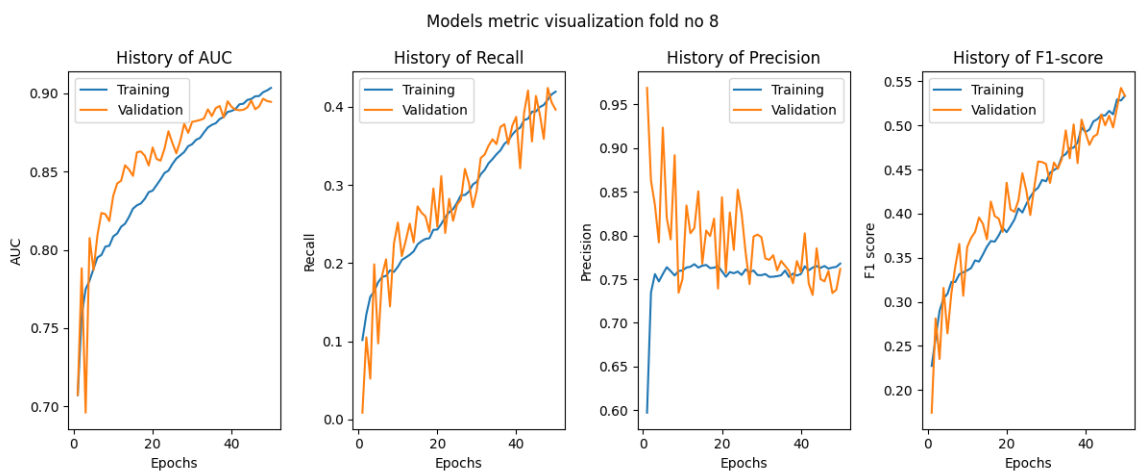


Figure A.85: All metrics over 50 epochs with Sequential model fold 8

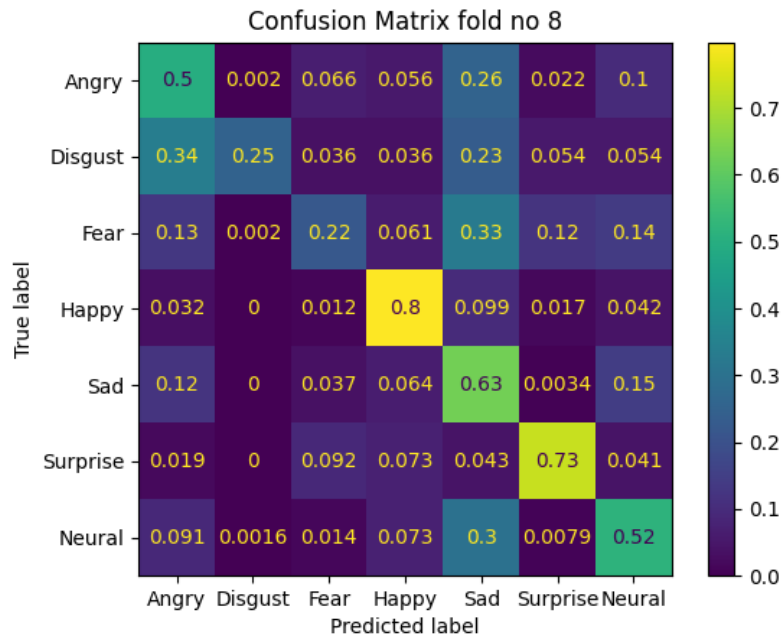


Figure A.86: Confusion matrix over 50 epochs with Sequential model fold 8

### Fold 9

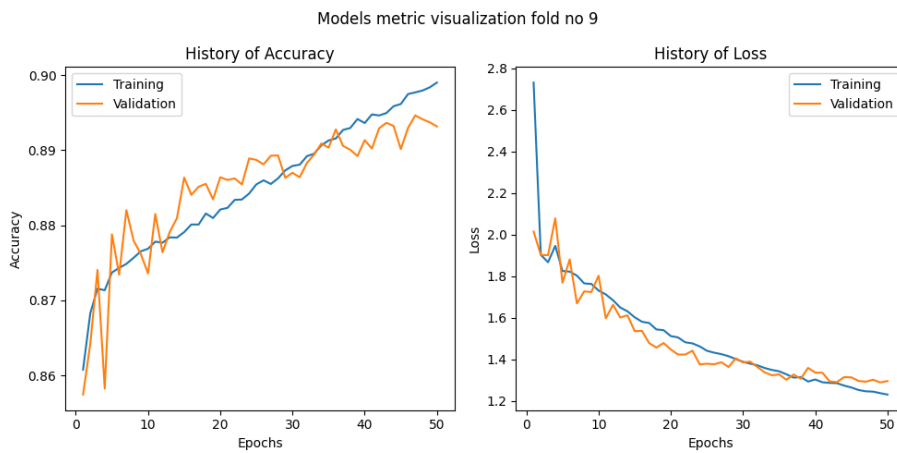


Figure A.87: Train and validation accuracy over 50 epochs with Sequential model fold 9

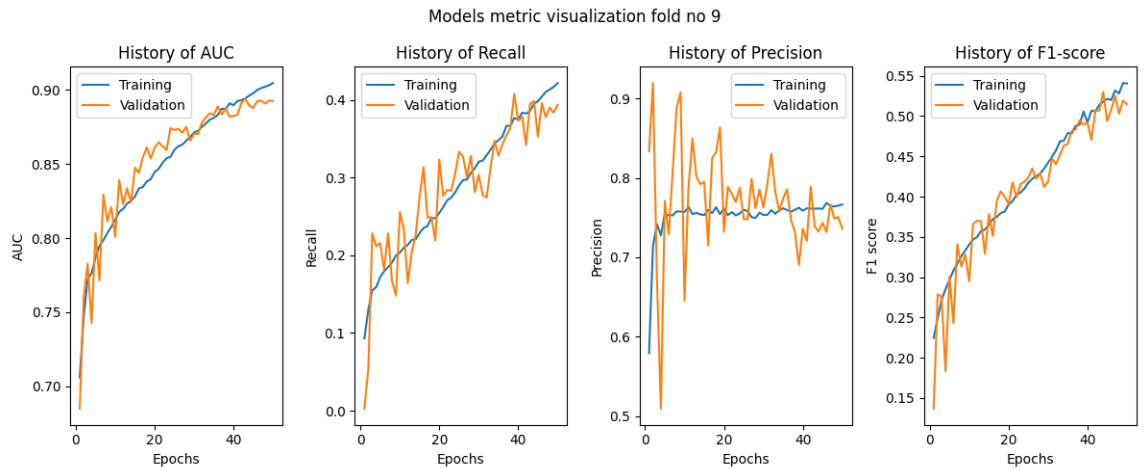


Figure A.88: All metrics over 50 epochs with Sequential model fold 9

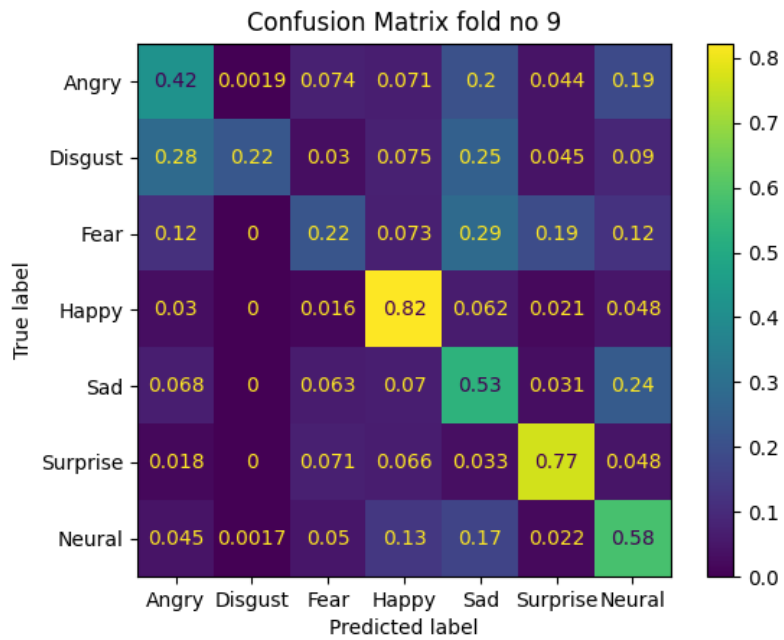


Figure A.89: Confusion matrix over 50 epochs with Sequential model fold 9

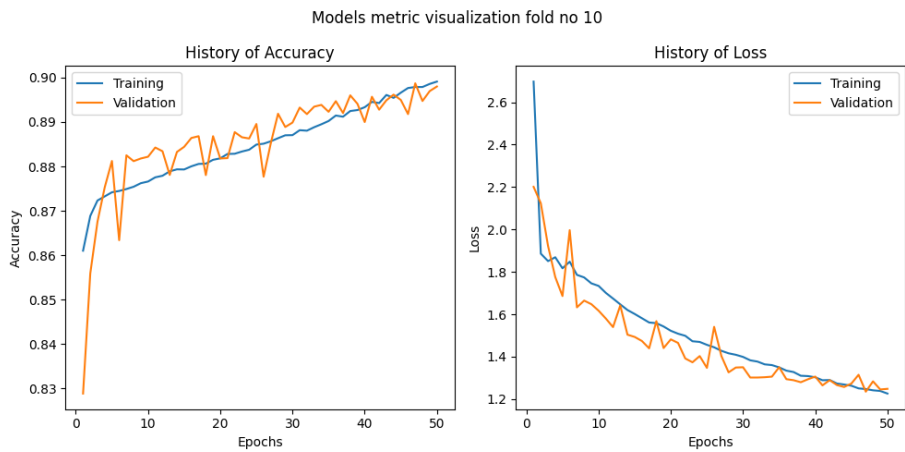


Figure A.90: Train and validation accuracy over 50 epochs with Sequential model fold 10

### Fold 10

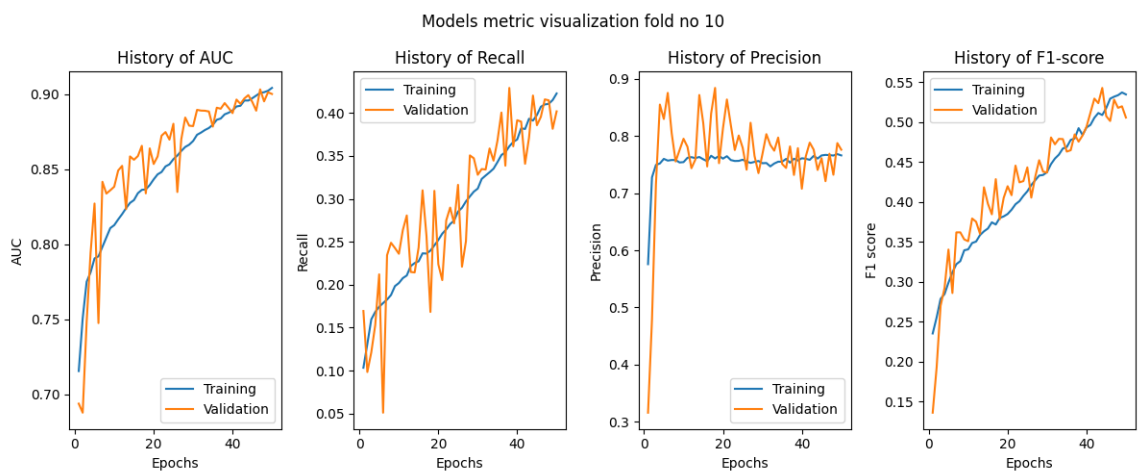


Figure A.91: All metrics over 50 epochs with Sequential model fold 10

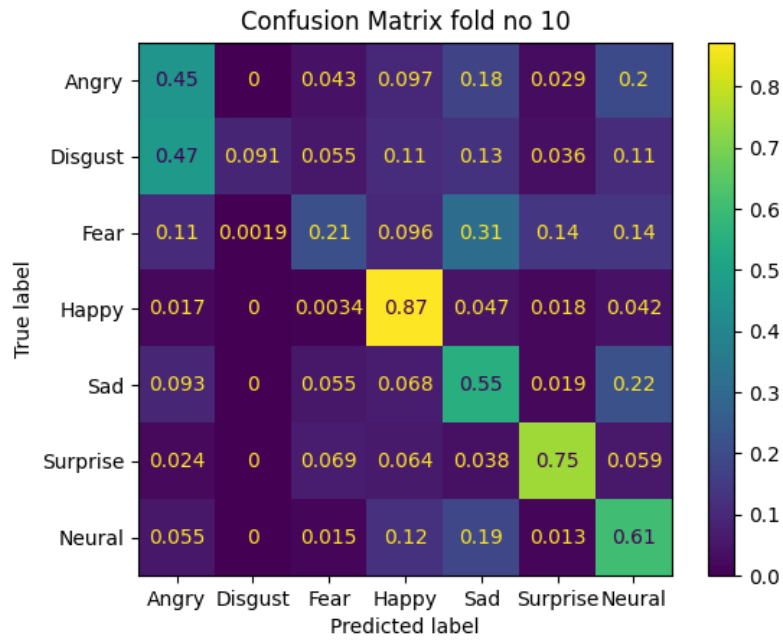


Figure A.92: Confusion matrix over 50 epochs with Sequential model fold 10

### Average scores all folds

```
Average scores for all folds:
> Accuracy: 89.66955065727234 (+- 0.1929681689823669)
> Loss: 1.2685469150543214
```

Figure A.93: Average scores for accuracy and loss with Sequential for all 10 folds on train dataset during 50 epochs

```
> FoLd 1 - Loss: 1.2354899644851685 - Accuracy: 89.92955684661865%
-----
> FoLd 2 - Loss: 1.245564341545105 - Accuracy: 89.71061706542969%
-----
> FoLd 3 - Loss: 1.2708773612976074 - Accuracy: 89.54344987869263%
-----
> FoLd 4 - Loss: 1.3160463571548462 - Accuracy: 89.38025236129761%
-----
> FoLd 5 - Loss: 1.249358892440796 - Accuracy: 89.85790014266968%
-----
> FoLd 6 - Loss: 1.2917723655700684 - Accuracy: 89.7862434387207%
-----
> FoLd 7 - Loss: 1.258326530456543 - Accuracy: 89.76636528968811%
-----
> FoLd 8 - Loss: 1.275283932685852 - Accuracy: 89.60423469543457%
-----
> FoLd 9 - Loss: 1.2946546077728271 - Accuracy: 89.3175482749939%
-----
> FoLd 10 - Loss: 1.2480947971343994 - Accuracy: 89.79933857917786%
-----
```

Figure A.94: Calculated accuracy and loss with Sequential model on train dataset per fold