BABEŞ BOLYAI UNIVERSITY, CLUJ NAPOCA, ROMÂNIA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Face expression recognition for social good - road to smart acting -

– ITSG report –

Teacher
Laura Dioșan
Intelligent Tools for Social Good

Author
Bogdan-Daniel Bălănescu
Software Engineering, 258-1

2019

Abstract

This paper studies the problem of Face Expression Recognition (REC) in an attempt to build a tool that helps novice actors better analyze their performance and get real-time feedback to improve. Starting from the Convolutional Neural Network proposed in [1], and trained on the FER-2013 emotion database, this paper slightly improves the 66% learning accuracy of the proposed algorithm to 68%. An application, Acting Mirror, is proposed and presented in the last part of the paper, as well as all the results and comparisons with other state of the art research. In the conclusion, we see that the current study of FER is facing a few issues we raise from our study and we propose a few improvement action points that could be tackled in the future.

Contents

Chapter 1

Introduction

1.1 What? Why? How?

For novice actors, it is often hard to control their emotions and to express what is on their script well, so the need for an expert to be with them when they rehearse is needed. A tool that would give them feedback in real-time may just make their start easier. It could be there for them anytime and for free, rather than hiring an expert to watch them rehearse.
This paper addresses this problem by proposing a tool able to recognize emotions from real-time video footage or photos.

- ***What?*** The problem of Face Expression Recognition (FER) is a classification problem which has received an important amount of attention in the last decade with various approaches such as the feature-based Tree-Augmented-Naive Bayes (TAN) classifier, Local Binary Patterns (LBP) classifier, Support Vector Machines and numerous Neural Networks based approaches. [3]

- ***Why?*** The importance of FER is present in a variety of domains, such as: psychology, neuroscience and even philosophy. Science today still cannot explain for sure where do emotions come from or if emotions are the ones driving our decisions or not. With this in mind, an approach that involves artificial intelligence capable of classifying emotions from photos would prove useful until other studies provide a better or a more accessible solution.

- ***How?*** In this paper, we present an artificially intelligent solution to recognizing emotions from pictures. From a dataset of pictures labeled with emotions we will train a model able to classify emotions.

[The work of next laboratories will add here: A short discussion of how it fits into related work in the area. Summary of the basic results and conclusions.]

1.2 Paper structure and original contribution(s)

[Should present here this information when improvement of the algorithm is done: The research presented in this paper advances the theory, design, and implementation of the proposed alogrithm.]

The main contribution of this report is to present an intelligent algorithm for solving the problem of Face Expression Recognition.

The second contribution of this report consists of building an intuitive, easy-to-use and user friendly software application. Our aim is to build an application that will help novice actors to better analyze their facial expressions and get real-time feedback while rehearsing. Main feature present in the application (and other possible improvements):

- ***Main feature:*** The user shall be able to record themselves in real-time and see their emotions in the video.
- ***Future improvement:*** The user shall be able to provide an acting script with labeled emotions on certain sections of text.
- ***Future improvement:*** The user shall receive real-time feedback based on the labeled script when they make a mistake.

The third contribution of this thesis consists of providing a comparison between state of the art results in FER and the proposed algorithm.

[Should be present in the next laboratories: The present work contains xyz bibliographical references and is structured in five chapters as follows.]

In the second chapter we will take a look at a formal introduction of our FER problem and weight the advantages and disadvantages of using AI to solve it.

The third chapter will describe the state of the art in FER.

In the the fourth chapter we will further detail our proposed approach to solving the problem.

[Short placeholder here, until we reach detailing the third chapter: (dataset with pictures and labels (emotion) -> new dataset facial points (features) and labels (emotion) -> model to solve the problem)]

The fifth chapter will present a comparison for the chosen methodology, data and results between two different approaches to solving this problem (ours and one more).

The final chapter will present our conclusions and future work to be done regarding this problem and application.

Chapter 2

Scientific Problem

++

2.1 Problem definition

For people pursuing their hobby of acting, it might be hard to hire an expert while rehearsing or even coming to terms with the tight schedule of a hard working day and a few minutes of spare time to rehearse a play. The need for a tool that would assist people when rehearsing, giving them feedback about their facial expressions in real life, arises and we pursue to deliver such an application.
The solution to such a tool is required to use an intelligent algorithm, because as far as we know, there exist no other methodologies for approaching this problem and it also falls into the category of complex problems that may be more easily solved using a neural network or other intelligent algorithms.

Advantages of solving the problem with an intelligent algorithm:

- it is much faster to reach a solution than proceeding with finding a non-intelligent algorithm to solve it
- it may be impossible to solve it using non-intelligent algorithms due to the vast lack of knowledge in the area of human emotions

Disadvantages of solving the problem with an intelligent algorithm:

- it may require a lot of work in training and retraining models until we reach a suitable (efficient and accurate) model to solve the problem
- it is impossible to reach, using A.I., a solution which has a 100% accuracy

Short description of our initial approach:

- From an existing dataset, FER-2013, which contains 35,887 48x48 pixel grayscale labeled images, 80% of pictures were used for training, while the remaining 20% were used for validation of the algorithm.
- The model was trained locally using Keras on an i7-7500u and an NVIDIA Quadro M520 1GB. It took a little bit over 2 hours of training and the achieved validation accuracy was 65%. While doing the validation precision/accuracy test, it takes approximately 1 milliseconds for the model to analyze a 48x48 pixel grayscale image.

Chapter 3

State of art/Related work

3.1 State of the Art

In this section we present a few methods utilized in order to solve the Facial Expression Recognition (FER) problem.

First, we will take a look at [8], where the problem is solved using k-NN (Nearest Neighbors) and MLP (Multilayer Perceptron).

- ***What kind of data did they use?*** Coefficients describing elements of facial expressions (as features) and a range of seven emotional states (as labels). The emotional states detected are: neutral, joy, sadness, surprise, anger, fear, disgust. The images they used were from the KDEF database.

- ***How does their model(s) work?*** Using Microsoft Kinect 3D for face modeling, they were able to extract 3D models of the face as 3D points, but Kinect 3D also can extract Action Units (AC) based on those points, which basically represent certain features of the face. Choosing 6 of those Action Units (upper lip raising, jaw lowering, lip stretching, lowering eyebrows, lip corner depressing, outer brow raising) they were able to train a 3-NN and an MLP classifier in order to solve the FER problem.

- ***What were their results?*** They tested the models for two cases: a) subject-dependent and b) subject-independent. For the 3-NN classifier, they got around 95-96% accuracy and for the MLP algorithms the results were around 75-76%.

please indicate the dataset used for obtaining these

Second, let us look at [7], where three significant challenges in FER are discussed: illumination variation, head pose and subject-dependence. We shall focus on the ones that target visual-only databases.

- ***What about illumination variation?*** The paper compares different approaches to FER, like SVM (Support Vector Machines) with 31-50% accuracy, Deep Networks with 48-96% accuracy and KNN with 92-96% accuracy. The most utilized dataset was the CK+ dataset. The paper suggests using Fast Fourier Transform and Contrast Limited Adaptive Histogram Equalization (FFT+CLAHE) to overcome poor lighting conditions, among other techniques.

- ***What about subject-dependece?*** Subject-dependence means the model is only able to recognize the expressions of the faces it trained with. The paper proposes a solution, where geometric face features were extracted, and using a part-based hierarchical recurrent neural network (PHRNN) to model the facial morphological variations, a multi-signal convolutional neural network (MSCNN) to find the spatial features of face, an accuracy of around 98.5% can be achieved. The used dataset was CK+.

Last, but not least, let us take a look at [2], a practical approach using a Convolutional Neural Network (CNN).

- ***What kind of data did they use?*** The used datasets were MMI and CKP and they recognized emotions from the following list: anger, sadness, disgust, happiness, fear and surprise.

- ***How does their model(s) work?*** Their proposed model is independent from any other third-party feature extraction frameworks and it performs better than the previously proposed CNN models, with an accuracy of 93-99% on the above mentioned datasets. Their model begins with a convolutional layer over the input and then filtering max pooling layer, before entering two fully connected layers (comprised of convolution + pooling, then convolution + convolution, then concatenation and pooling), after which they classify the images with softmax layer.

- ***What were their results?*** Their accuracy was around 99%.

So far, we have seen three great state of the art examples. One which uses an third-party framework to manipulate the dataset so that the model can be a very simple one to train, like a 3-NN. Another set of examples where top state of the art models were presented and certain impediments were discussed (such as illumination variation and subject-dependency) and one other example which is independent on any other third-party framework in its learning, while still achieving good results.

3.2 Useful Tools

Now, let us give a list of useful tools to use when developing intelligent applications:

- Tensorflow, is a Python framework for building machine learning models. A javascript version, Tensorflow.js, also exists.
- Tensorflow Lite, a framework for building machine learning models compatible with portable devices (i.e. mobile phones).
- ML Kit for Firebase, a mobile SDK that empowers mobile applications with Google's machine learning packages. It is also possible to host one's own machine learning model in Firebase (have not experienced with this yet).
- YOLO: Real-Time Object Detection, is a state of the art object detection system, but which can also be trained for a different purpose. It is open source and written in C++.
- fastAI, is a Python framework for buildign machine learning models. Also comes with pre-trained models for certain problems.

Chapter 4

Proposed approach

4.1 Algorithm description

Starting from the Convolutional Neural Network proposed in [1] (and which is inspidred by the Xception [4] architecture), and presented in the following figure (see Figure 4.1), our approach is to use an already existing algorithm capable of classifying emotions and integrate it in an easy to use application in order to help novice actors get real-time feedback about their performance.

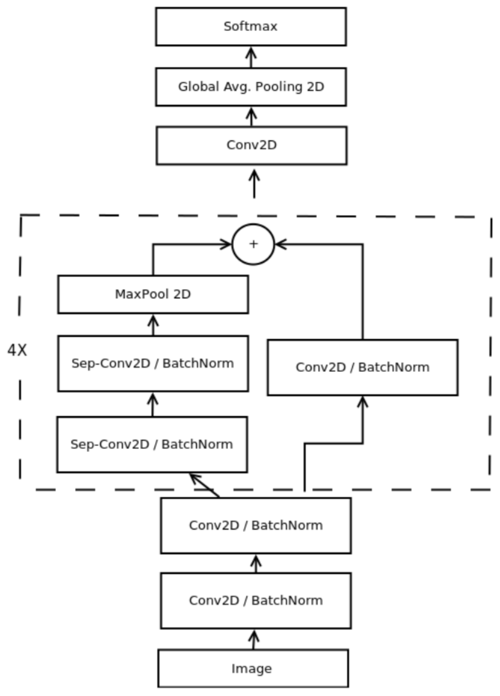

Figure 4.1: [1] proposed model for real-time classification of emotions

We used the FER-2013 emotion database, which contains 35.887 48x48 pixel grayscale labeled

images with the following emotions: (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). Initially, we performed 7 training trials, one using original approach proposed in [1], and then by changing the learning rate and the loss function during training.
For the original approach proposed in [1], we achieved a 65% validation accuracy over the FER-2013 database (using all the 35.887 images), and a 76% accuracy over the CK+ database using the same model trained on the FER-2013 database.
From our other approaches, the fifth one achieved a 64% validation accuracy over the FER-2013 dabase (using only 28.709 of the images), and a 79% accuracy over the CK+ database using the same model trained on the FER-2013 database.

Chapter 5

Application (numerical validation)

For the proposed algorithm, our approach was to perform several training sessions, with different inputs for the independent variables and see how they affect the learning. The reason for this, was to find a way to improve the accuracy of the model. As will be related in the following section, we have played around with the learning rate and with the loss function of the algorithm. Unfortunately, what we have found is that the algorithm reaches a plateou of learning from which it cannot escape, thus, the accuracy of the model being around 65%, almost like what was achieved in [1].

5.1 Methodology

In this chapter, we analyze the achieved results for the proposed algorithm using the following approach. We present the training dataset and independent variables, a plot of the achieved training results and how well it fairs against the CK+ dataset, presenting the precision for each emotion separately.
The independent variables we are experimenting with are the learning rate, and the loss function.
The dependent variables will be the weights of the model.
For our first attempt, we have used the whole FER-2013 dataset, and obtained a validation accuracy of 65%, and for the next attempts we shall only use the Training pictures from the FER-2013 dataset, and it will also enable us to compute the precision for the Public Test set and Private Test set of the FER-2013 dataset.
For comparison purposes between our models trained on the FER-2013 dataset, we also compute the precision for each model against the CK+ emotion dataset and present the results here.

here, the classifier was trained on FER and test on CK+ or

Side note: for the first five trials, we test the model on the CK+ dataset as it is, but after that, we realised we forgot to filter the CK+ dataset through the Haarscade Frontal Face Detector model provided by OpenCV. We came back, and redid the tests on the as such **processed** CK+ dataset.

Note: From here on, when we refer to testing our model against the CK+ dataset, we mean to test it against the processed CK+ dataset, unless stated otherwise.

5.2 Obtained results

The first trial:

- **Dataset**: FER-2013, 35.887 images, split into 80% training and 20% validation.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 7] tensor containing the percentages for each label.
- **Loss function**: Categorical Crossentropy.
- **Optimizer**: Adam.
- **Learning rate**: 0.1.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result was a training validation accuracy of 65%, and a 76% accuracy on the CK+ dataset.

Table 5.1: The precisin results against the CK+ dataset for the first training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.16	45
Disgust	0.00	59
Fear	0.03	25
Happy	0.68	69
Sad	0.12	28
Surprise	0.00	83
Neutral	0.03	1
Weighted Average	0.19	310

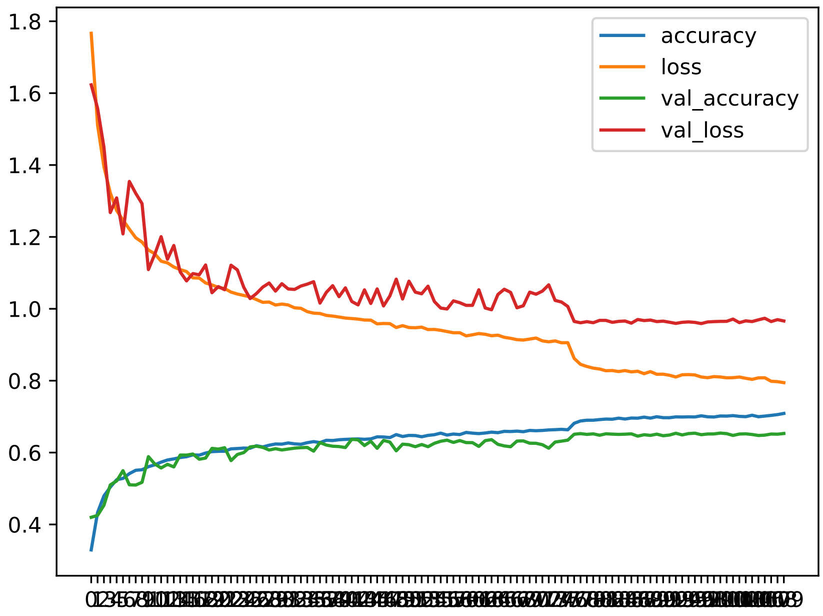

Figure 5.1: Training results using all the 35.887 images in the FER-2013 emotion dataset

Table 5.2: The precisin results against the **unprocessed** CK+ dataset for the first training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.30	45
Disgust	0.91	59
Fear	0.43	25
Happy	0.94	69
Sad	0.39	28
Surprise	0.97	83
Neutral	0.00	1
Weighted Average	0.76	310

The second trial:

- **Dataset**: FER-2013, 28.709 images, split into 80% training and 20% validation.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 7] tensor containing the percentages for each label.
- **Loss function**: Categorical Crossentropy.
- **Optimizer**: Adam.
- **Learning rate**: 0.1.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result was a training validation accuracy of 63%, and a 76% accuracy on the CK+ dataset.

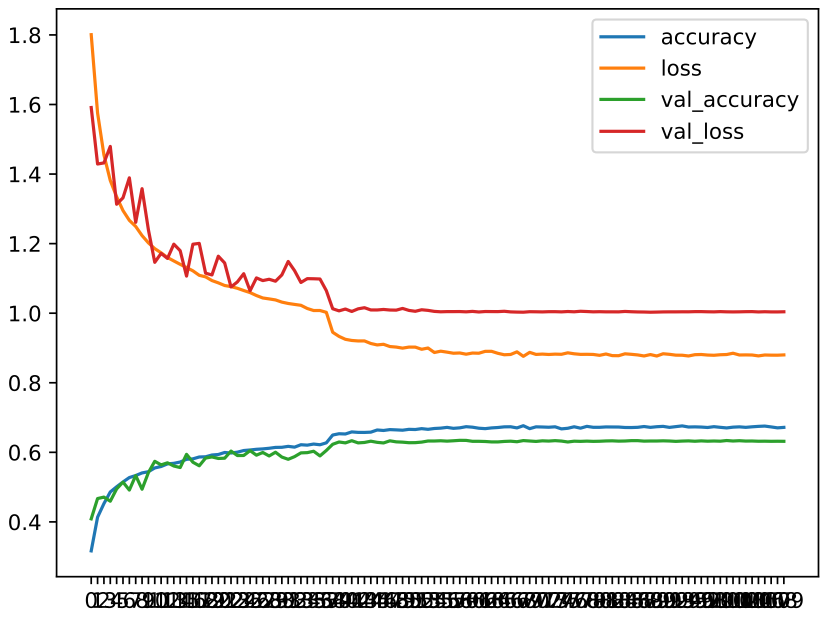

Figure 5.2: Training results using 28.709 images from the FER-2013 emotion dataset

Table 5.3: The precisin results against the CK+ dataset for the second training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.21	45
Disgust	1.00	59
Fear	0.53	25
Happy	0.91	69
Sad	0.52	28
Surprise	0.94	83
Neutral	0.00	1
Weighted Average	0.76	310

Table 5.4: The precisin results against the FER-2013 Public Test dataset for the second training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.51	467
Disgust	0.67	56
Fear	0.47	496
Happy	0.82	895
Sad	0.52	653
Surprise	0.74	415
Neutral	0.55	607
Weighted Average	0.62	3589

Table 5.5: The precisin results against the FER-2013 Private Test dataset for the second training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.53	467
Disgust	0.47	56
Fear	0.47	496
Happy	0.84	895
Sad	0.50	653
Surprise	0.74	415
Neutral	0.61	607
Weighted Average	0.63	3589

Table 5.6: The precisin results against the **unprocessed** CK+ dataset for the second training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.19	45
Disgust	0.00	59
Fear	0.05	25
Happy	0.52	69
Sad	0.15	28
Surprise	1.00	83
Weighted Average	0.43	310

The third trial:

- **Dataset**: FER-2013, 28.709 images, split into 80% training and 20% validation.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 7] tensor containing the percentages for each label.
- **Loss function**: Categorical Crossentropy.
- **Optimizer**: Adam.
- **Learning rate**: 0.01.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result was a validation accuracy of 63%, and a 60% accuracy on the CK+ dataset.

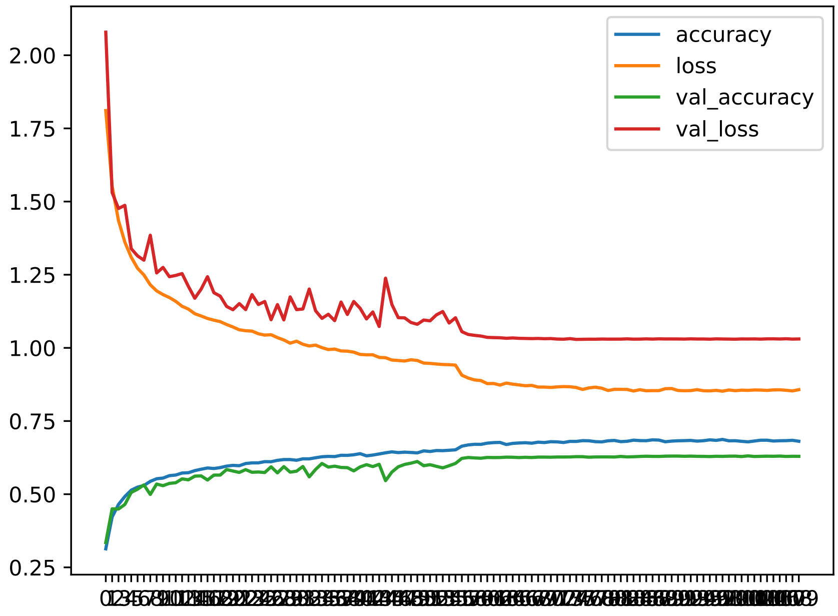

Figure 5.3: Training results using 28.709 images from the FER-2013 emotion dataset

Table 5.7: The precisin results against the CK+ dataset for the third training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.25	45
Disgust	0.00	59
Fear	0.80	25
Happy	0.89	69
Sad	0.54	28
Surprise	0.95	83
Neutral	0.00	1
Weighted Average	0.60	310

Table 5.8: The precisin results against the FER-2013 Public Test dataset for the third training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.52	467
Disgust	0.62	56
Fear	0.48	496
Happy	0.84	895
Sad	0.54	653
Surprise	0.74	415
Neutral	0.55	607
Weighted Average	0.63	3589

Table 5.9: The precisin results against the FER-2013 Private Test dataset for the third training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.57	467
Disgust	0.73	56
Fear	0.48	496
Happy	0.85	895
Sad	0.48	653
Surprise	0.76	415
Neutral	0.59	607
Weighted Average	0.64	3589

Table 5.10: The precisin results against the **unprocessed** CK+ dataset for the third training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.17	45
Disgust	0.00	59
Fear	0.02	25
Happy	0.62	69
Sad	0.18	28
Surprise	1.00	83
Neutral	0.02	1
Weighted Average	0.45	310

The fourth trial:

- **Dataset**: FER-2013, 28.709 images, split into 80% training and 20% validation.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 7] tensor containing the percentages for each label.
- **Loss function**: Mean Square Error.
- **Optimizer**: Adam.
- **Learning rate**: 0.01.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result were a validation accuracy of 63%, and a 69% accuracy on the CK+ dataset.

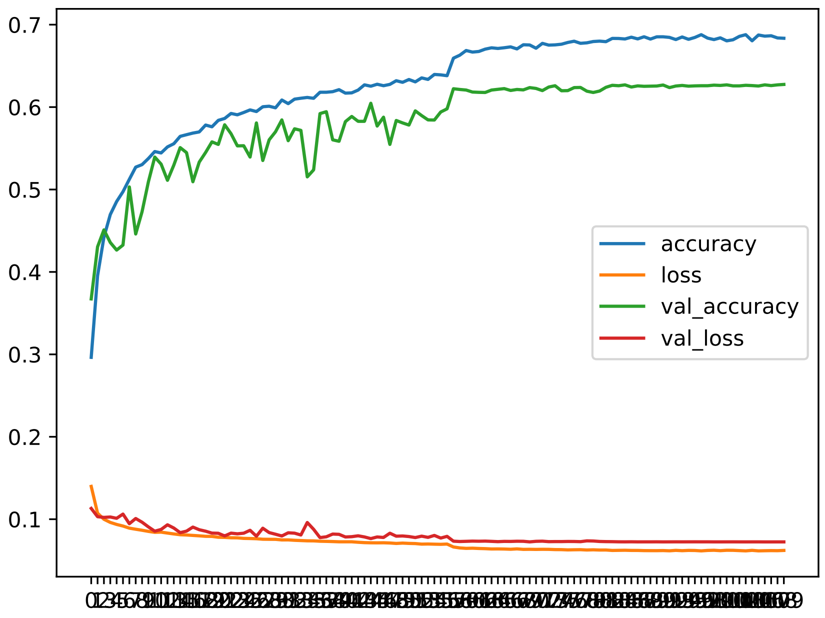

Figure 5.4: Training results using 28.709 images from the FER-2013 emotion dataset

Table 5.11: The precisin results against the CK+ dataset for the fourth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.23	45
Disgust	0.67	59
Fear	0.56	25
Happy	0.85	69
Sad	0.43	28
Surprise	0.96	83
Neutral	0.00	1
Weighted Average	0.69	310

Table 5.12: The precisin results against the FER-2013 Public Test dataset for the fourth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.53	467
Disgust	0.61	56
Fear	0.51	496
Happy	0.82	895
Sad	0.54	653
Surprise	0.76	415
Neutral	0.53	607
Weighted Average	0.63	3589

Table 5.13: The precisin results against the FER-2013 Private Test dataset for the fourth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.56	467
Disgust	0.52	56
Fear	0.50	496
Happy	0.85	895
Sad	0.49	653
Surprise	0.74	415
Neutral	0.58	607
Weighted Average	0.63	3589

Table 5.14: The precisin results against the **unprocessed** CK+ dataset for the fourth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.13	45
Disgust	0.00	59
Fear	0.12	25
Happy	0.49	69
Sad	0.12	28
Surprise	1.00	83
Neutral	0.00	1
Weighted Average	0.42	310

The fifth trial:

- **Dataset**: FER-2013, 28.709 images, split into 80% training and 20% validation.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 7] tensor containing the percentages for each label.
- **Loss function**: Mean Square Error.
- **Optimizer**: Adam.
- **Learning rate**: 0.1.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result were a validation accuracy of 64%, and a 79% accuracy on the CK+ dataset.

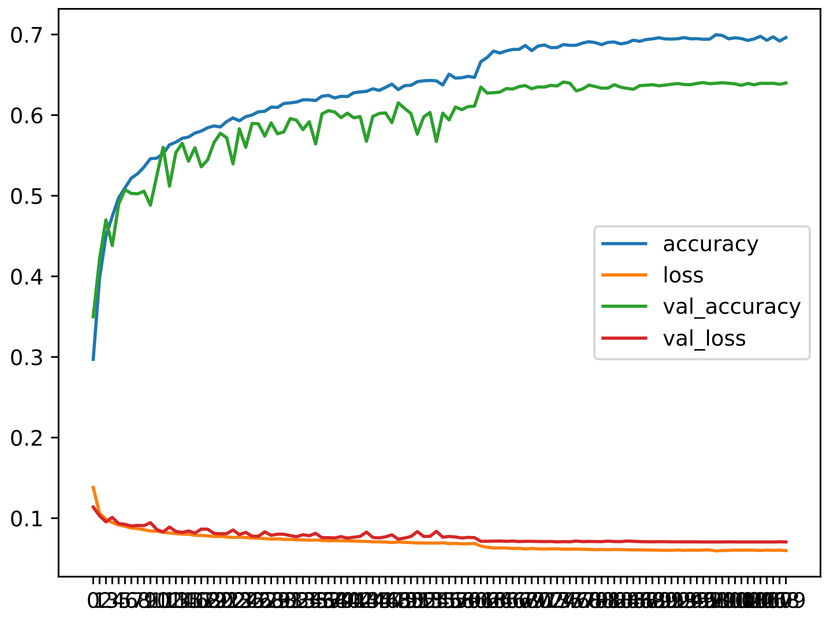

Figure 5.5: Training results using 28.709 images from the FER-2013 emotion dataset

Table 5.15: The precisin results against the CK+ dataset for the fifth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.21	45
Disgust	1.00	59
Fear	0.67	25
Happy	0.94	69
Sad	0.55	28
Surprise	0.94	83
Neutral	0.00	1
Weighted Average	0.79	310

Table 5.16: The precisin results against the FER-2013 Public Test dataset for the fifth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.53	467
Disgust	0.56	56
Fear	0.48	496
Happy	0.84	895
Sad	0.56	653
Surprise	0.78	415
Neutral	0.53	607
Weighted Average	0.63	3589

Table 5.17: The precisin results against the FER-2013 Private Test dataset for the fifth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.54	467
Disgust	0.62	56
Fear	0.47	496
Happy	0.87	895
Sad	0.49	653
Surprise	0.76	415
Neutral	0.58	607
Weighted Average	0.64	3589

Table 5.18: The precisin results against the **unprocessed** CK+ dataset for the fifth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.16	45
Disgust	0.00	59
Fear	0.05	25
Happy	0.52	69
Sad	0.18	28
Surprise	0.80	83
Neutral	0.80	1
Weighted Average	0.37	310

The sixth trial:

- **Dataset**: FER-2013, 28.709 images, split into 80% training and 20% validation.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 7] tensor containing the percentages for each label.
- **Loss function**: Squared Hinge Loss.
- **Optimizer**: Adam.
- **Learning rate**: 0.1.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result were a validation accuracy of 63%, and a 57% accuracy on the CK+ dataset.

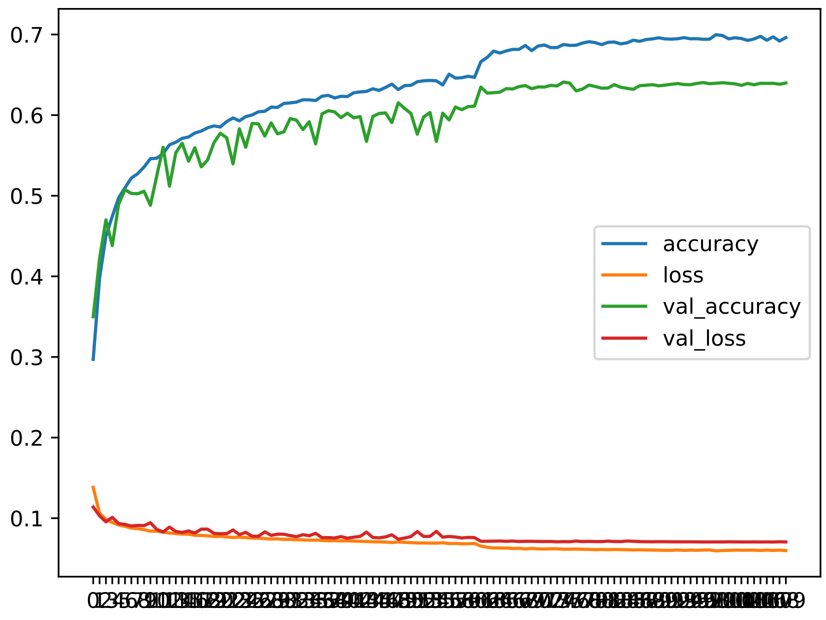

Figure 5.6: Training results using 28.709 images from the FER-2013 emotion dataset

Table 5.19: The precisin results against the CK+ dataset for the sixth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.23	45
Disgust	0.00	59
Fear	0.60	25
Happy	0.89	69
Sad	0.41	28
Surprise	0.95	83
Neutral	0.00	1
Weighted Average	0.57	

Table 5.20: The precisin results against the FER-2013 Public Test dataset for the sixth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.49	467
Disgust	0.00	56
Fear	0.51	496
Happy	0.85	895
Sad	0.54	653
Surprise	0.73	415
Neutral	0.53	607
Weighted Average	0.62	3589

Table 5.21: The precisin results against the FER-2013 Private Test dataset for the sixth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.52	467
Disgust	0.00	56
Fear	0.50	496
Happy	0.87	895
Sad	0.50	653
Surprise	0.72	415
Neutral	0.59	607
Weighted Average	0.63	3589

The seventh trial:

- **Dataset**: FER-2013, 28.709 images, split into 80% training and 20% validation.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 7] tensor containing the percentages for each label.
- **Loss function**: Squared Hinge Loss.
- **Optimizer**: Adam.
- **Learning rate**: 0.01.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result were a validation accuracy of 63%, and a 58% accuracy on the CK+ dataset.

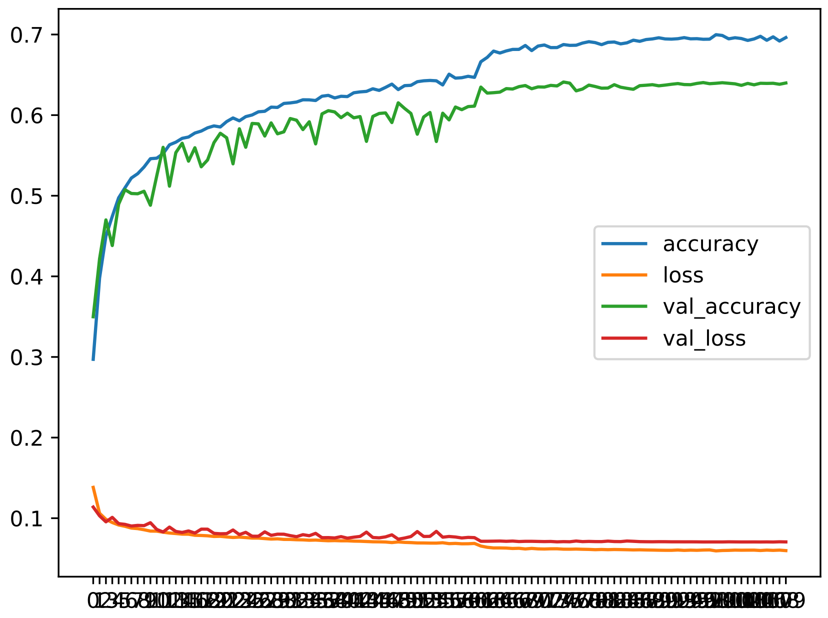

Figure 5.7: Training results using 28.709 images from the FER-2013 emotion dataset

Table 5.22: The precisin results against the CK+ dataset for the seventh training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.25	45
Disgust	0.00	59
Fear	0.50	25
Happy	0.92	69
Sad	0.46	28
Surprise	0.97	83
Neutral	0.00	1
Weighted Average	0.58	310

Table 5.23: The precisin results against the FER-2013 Public Test dataset for the seventh training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.50	467
Disgust	0.00	56
Fear	0.46	496
Happy	0.82	895
Sad	0.53	653
Surprise	0.71	415
Neutral	0.51	607
Weighted Average	0.60	3589

Table 5.24: The precisin results against the FER-2013 Private Test dataset for the seventh training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.53	467
Disgust	0.50	56
Fear	0.48	496
Happy	0.85	895
Sad	0.49	653
Surprise	0.71	415
Neutral	0.57	607
Weighted Average	0.62	3589

Analyzing the current situation:

While the achieved results were not a big improvements over the original approach, by changing the loss function from Categorical Crossentropy to Square Mean Error, we have managed to raise the accuracy of the model against the CK+ dataset from 76% (accuracy that is obtained by training either on just 28.709 of the images or on all 35.887 images from the FER-2013 dataset) to 79% (trained on just only 28.709 of the images from the FER-2013 datast). In the following attempts we aim to augment the dataset, so that it contains even more images and possibly achieve better results.

After these 7 trials we have noticed that all the models encounter problems at identifying the Angry emotion over the CK+ database. A point for improvement could be to find out whether the Angry emotion from the FER-2013 dataset gets confused with another emotion. We could do this either by training the model for just the Angry emotion and another one (Fear, Sad), or by eliminating the Angry emotion from the dataset and see if the precision for emotions Fear and Sad improves.

Another identified problem is that on certain trials, the emotion Disgust is not recognized at all within the CK+ dataset. Could it be that Disgust gets confused with another emotion, like Angry? Further investigation is needed.

Another observation could be that training with a high learning rate from the beginning may cause the model to reach a plateau much sooner. Lowering the learning rate seems to delay that.

Also, chaning the loss function from Categorial Crossentropy to Mean Square Error yielded improvements in the model later on (as in later epochs) as well, instead of stopping to learn since epoch 70-90.

Our best attempt was the fifth one: which uses Square Mean Error as the loss function and a learning rate of 0.1. Subsequent trials will use this setup.

5.3 Case for improvement

The eight trial - augmented dataset x6:

- **Dataset**: Augmented FER-2013 x6, 172.254 images, split into 80% training and 20% validation sets. Each of the below points describes an operation made for 28.701 images from the FER-2013 dataset using a python library called imagaug, [5]:
  - **Original**: 28.709 images from the original FER-2013 dataset.
  - **Additive Gaussian Noise**: with a scale of 0.07*255.
  - **Multiply**: with random values between 0.25 and 1.50.
  - **Salt and Pepper**: with a percentage value of 0.03.
  - **Gaussian Blur**: with a value of 0.50.
  - **Clouds**: for each image a cloud like mask was put over.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 7] tensor containing the percentages for each label.
- **Loss function**: Mean Square Error.
- **Optimizer**: Adam.
- **Learning rate**: 0.1.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result was a training validation accuracy of 68%, and a 71% accuracy on the CK+ dataset.

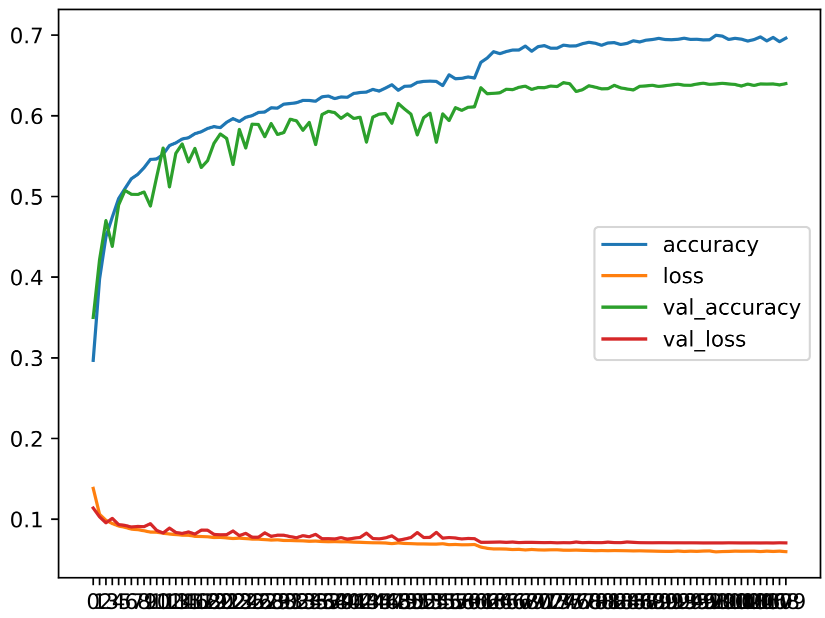

Figure 5.8: Training results using 172.254 augmented images from the FER-2013 emotion dataset

Table 5.25: The precisin results against the CK+ dataset for the eight training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.18	45
Disgust	0.86	59
Fear	0.33	25
Happy	0.89	69
Sad	0.38	28
Surprise	0.96	83
Neutral	0.00	1
Weighted Average	0.71	310

Table 5.26: The precisin results against the FER-2013 Public Test dataset for the eight training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.51	467
Disgust	0.62	56
Fear	0.51	496
Happy	0.83	895
Sad	0.56	653
Surprise	0.74	415
Neutral	0.57	607
Weighted Average	0.64	3589

Table 5.27: The precisin results against the FER-2013 Private Test dataset for the eight training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.55	467
Disgust	0.57	56
Fear	0.52	496
Happy	0.85	895
Sad	0.50	653
Surprise	0.75	415
Neutral	0.63	607
Weighted Average	0.65	3589

Analyzing the current situation:

There does not seem to be much improvement, apart from the increase on the validation accuracy against FER-2013 from 64% to 68%, compared to our fifth and most successful trial. But this improvement comes with a cost on the validation accuracy against the CK+ dataset from 79% to 71%. From the observation that on the CK+ dataset emotions Angry, Fear and Sad have an unreliable precision, we decide to build a Confusion Matrix for the dataset we trained on, FER-2013.

Table 5.28: The confusion matrix for FER-2013 Public Test dataset for the eigth training trial

Emotions	**Angry**	**Disgust**	**Fear**	**Happy**	**Sad**	**Surprise**	**Neutral**
Angry	**0.59**	0.01	0.09	0.06	0.10	0.03	0.10
Disgust	0.27	**0.41**	0.09	0.05	0.11	0.18	0.05
Fear	0.12	0.01	**0.43**	0.04	0.21	0.09	0.11
Happy	0.03	0.01	0.02	**0.85**	0.02	0.02	0.05
Sad	0.15	0.01	0.09	0.05	**0.51**	0.02	0.17
Surprise	0.03	0.01	0.08	0.06	0.02	**0.78**	0.03
Neutral	0.08	0.01	0.07	0.09	0.14	0.02	**0.61**

Table 5.29: The confusion matrix for FER-2013 Private Test dataset for the eigth training trial

Emotions	**Angry**	**Disgust**	**Fear**	**Happy**	**Sad**	**Surprise**	**Neutral**
Angry	**0.61**	0.02	0.09	0.03	0.15	0.01	0.09
Disgust	0.31	**0.45**	0.07	0.02	0.13	0.02	0.00
Fear	0.15	0.01	**0.43**	0.04	0.19	0.10	0.09
Happy	0.03	0.01	0.02	**0.88**	0.02	0.02	0.03
Sad	0.11	0.01	0.11	0.07	**0.52**	0.01	0.18
Surprise	0.05	0.00	0.10	0.04	0.03	**0.76**	0.02
Neutral	0.06	0.01	0.05	0.08	0.16	0.03	**0.63**

From the confusion matrixes we observe that there is high confusion between Agnry and Disgust. We decide to investigate further, and what we find is that the distribution of data across the 7 emotions proposed in FER-2013 is uniform. In 5.9 we observe that Disgust makes only 2% of the FER-2013 dataset and that the other emotions each make an average of 16-17% of the dataset. For this reason, we believe that excluding the Disgust emotion from our training dataset may improve the overall accuracy of the model. The following will be 2 trials training the model on the original, and respectively on the augmented version, of the FER-2013 dataset. The attempts will use the setup discussed in the fifth trial.

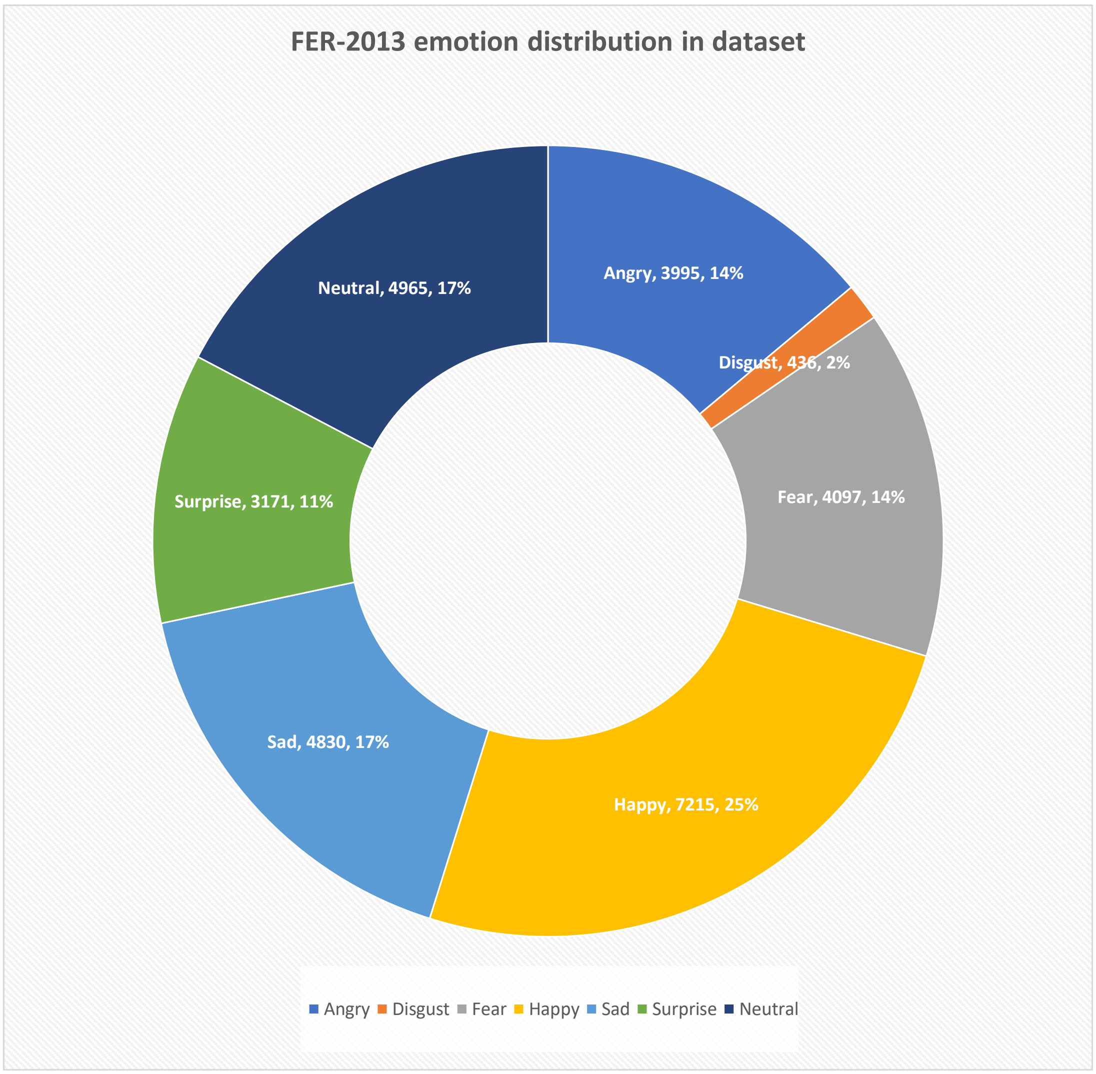

Figure 5.9: FER-2013 emotion distribution in dataset for the training images

The ninth trial - original dataset without the emotion "Disgust":

- **Dataset**: Original FER-2013 dataset, with the emotion "Disgust" removed, 28.273 images, split into 80% training and 20% validation.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 6] tensor containing the percentages for each label.
- **Loss function**: Mean Square Error.
- **Optimizer**: Adam.
- **Learning rate**: 0.1.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result was a training validation accuracy of 65%, and a 81% accuracy on the CK+ dataset.

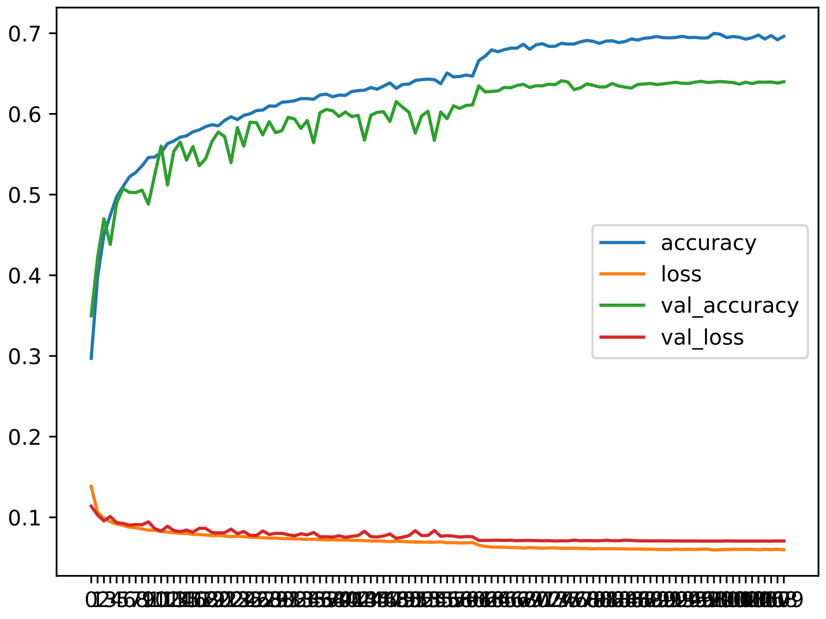

Figure 5.10: Training results using 28.273 images from the FER-2013 emotion dataset

Table 5.30: The precisin results against the CK+ dataset for the ninth training trial

Emotion	Precision	No. of pictures
Angry	0.84	45
Fear	0.39	25
Happy	0.93	69
Sad	0.45	28
Surprise	0.95	83
Neutral	0.00	1
Weighted Average	0.81	251

Table 5.31: The precisin results against the FER-2013 Public Test dataset for the ninth training trial

Emotion	Precision	No. of pictures
Angry	0.58	467
Fear	0.49	496
Happy	0.84	895
Sad	0.53	653
Surprise	0.76	415
Neutral	0.52	607
Weighted Average	0.64	3533

Table 5.32: The precisin results against the FER-2013 Private Test dataset for the ninth training trial

Emotion	Precision	No. of pictures
Angry	0.56	467
Fear	0.49	496
Happy	0.87	895
Sad	0.50	653
Surprise	0.74	415
Neutral	0.58	607
Weighted Average	0.64	3533

Analyzing the current situation:

There seems to be a slight improvement on the validation accuracy against FER-2013 from 64% to 65%, compared to our fifth and most successful trial. On the the CK+ dataset as well, from 79% to 81%, however, we decide to build the Confusion Matrix for the dataset we trained, on FER-2013, just to see if there are new confusions around.

Table 5.33: The confusion matrix for FER-2013 Public Test dataset for the ninth training trial

Emotions	**Angry**	**Fear**	**Happy**	**Sad**	**Surprise**	**Neutral**
Angry	**0.59**	0.09	0.04	0.11	0.03	0.13
Fear	0.11	**0.38**	0.04	0.24	0.08	0.15
Happy	0.02	0.02	**0.84**	0.02	0.02	0.07
Sad	0.10	0.11	0.04	**0.52**	0.02	0.21
Surprise	0.05	0.08	0.05	0.03	**0.77**	0.02
Neutral	0.06	0.05	0.09	0.16	0.02	**0.62**

Table 5.34: The confusion matrix for FER-2013 Private Test dataset for the ninth training trial

Emotions	**Angry**	**Fear**	**Happy**	**Sad**	**Surprise**	**Neutral**
Angry	**0.59**	0.10	0.02	0.15	0.02	0.12
Fear	0.15	**0.38**	0.03	0.20	0.11	0.13
Happy	0.04	0.02	**0.86**	0.03	0.02	0.04
Sad	0.12	0.09	0.04	**0.52**	0.02	0.22
Surprise	0.03	0.13	0.06	0.01	**0.73**	0.04
Neutral	0.04	0.05	0.05	0.16	0.02	**0.67**

From the confusion matrixes we observe that the only medium-high confusion left is between Fear and Sad. We continue with our tenth trial, and that means, training the on augmented FER-2013 dataset, but with the emotion Disgust removed, like here.

The tenth trial - augmented dataset without the emotion "Disgust":

- **Dataset**: Augmented FER-2013 x6, with the emotion "Disgust" removed, 169.638 images, split into 80% training and 20% validation sets. Each of the below points describes an operation made for 28.273 images from the FER-2013 dataset using a python library called imagaug, [5]:
  - **Original**: 28.273 images from the original FER-2013 dataset.
  - **Additive Gaussian Noise**: with a scale of 0.07*255.
  - **Multiply**: with random values between 0.25 and 1.50.
  - **Salt and Pepper**: with a percentage value of 0.03.
  - **Gaussian Blur**: with a value of 0.50.
  - **Clouds**: for each image a cloud like mask was put over.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 6] tensor containing the percentages for each label.
- **Loss function**: Mean Square Error.
- **Optimizer**: Adam.
- **Learning rate**: 0.1.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result was a training validation accuracy of 68%, and a 78% accuracy on the CK+ dataset.

Table 5.35: The precisin results against the CK+ dataset for the tenth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.67	45
Fear	0.35	25
Happy	0.93	69
Sad	0.44	28
Surprise	0.97	83
Neutral	0.00	1
Weighted Average	0.78	251

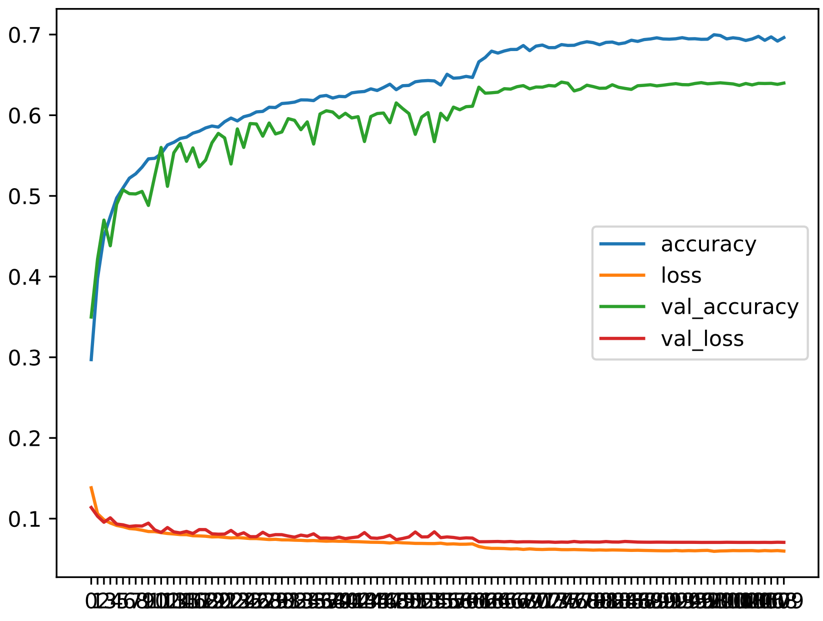

Figure 5.11: Training results using 169.638 images from the FER-2013 emotion dataset

Table 5.36: The precisin results against the FER-2013 Public Test dataset for the tenth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.58	467
Fear	0.52	496
Happy	0.85	895
Sad	0.57	653
Surprise	0.76	415
Neutral	0.58	607
Weighted Average	0.66	3533

Table 5.37: The precisin results against the FER-2013 Private Test dataset for the tenth training trial

Emotion	**Precision**	**No. of pictures**
Angry	0.59	467
Fear	0.51	496
Happy	0.85	895
Sad	0.49	653
Surprise	0.76	415
Neutral	0.63	607
Weighted Average	0.65	3533

Analyzing the current situation:

There seems to be a slight improvement on the validation accuracy against the ninth trial from 65% to 68%. On the the CK+ dataset as however, there is a slight decline from 81% to 78%. We decide to build the Confusion Matrix for the dataset we trained, on FER-2013 without the emotion Disgust, just to see the situation of confusions.

Table 5.38: The confusion matrix for FER-2013 Public Test dataset for the tenth training trial

Emotions	**Angry**	**Fear**	**Happy**	**Sad**	**Surprise**	**Neutral**
Angry	**0.63**	0.11	0.03	0.11	0.03	0.09
Fear	0.12	**0.45**	0.03	0.21	0.08	0.11
Happy	0.02	0.02	**0.87**	0.01	0.02	0.05
Sad	0.11	0.11	0.05	**0.55**	0.02	0.16
Surprise	0.03	0.09	0.04	0.03	**0.78**	0.02
Neutral	0.08	0.06	0.10	0.15	0.02	**0.59**

Table 5.39: The confusion matrix for FER-2013 Private Test dataset for the tenth training trial

Emotions	**Angry**	**Fear**	**Happy**	**Sad**	**Surprise**	**Neutral**
Angry	**0.61**	0.09	0.03	0.16	0.02	0.10
Fear	0.14	**0.45**	0.04	0.18	0.09	0.10
Happy	0.04	0.02	**0.87**	0.03	0.02	0.03
Sad	0.10	0.13	0.06	**0.51**	0.02	0.17
Surprise	0.03	0.12	0.05	0.02	**0.76**	0.01
Neutral	0.05	0.06	0.07	0.17	0.02	**0.63**

From the confusion matrixes we observe that the only medium confusion left is between Fear and Sad. Further investigation on this matter would imply a train for just Fear and Sad to see if the neural network can learn the difference between them or not.

The eleventh trial - FER-2013 Sad and Fear:

- **Dataset**: Sad and Fear emotions from FER-2013 dataset, 9.927 images, split into 80% training and 20% validation.
- **Input size**: [1, 48, 48, 1] grayscale pixel images. Each pixel, a value between [0, 255].
- **Output size**: [1, 2] tensor containing the percentages for each label.
- **Loss function**: Mean Square Error.
- **Optimizer**: Adam.
- **Learning rate**: 0.1.
- **Batch size**: 32.
- **Epochs**: 110.
- **Training metric**: Accuracy.

The result was a training validation accuracy of 71%, and a 74% accuracy on the CK+ dataset.

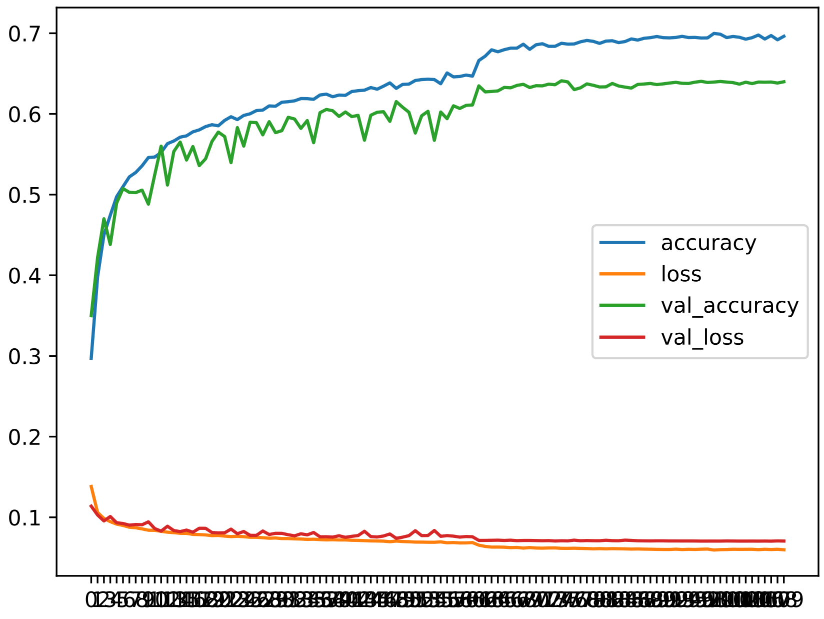

Figure 5.12: Training results using 9.927 sad and fear images from the FER-2013 emotion dataset

Table 5.40: The precisin results against the CK+ dataset for the ninth training trial

Emotion	Precision	No. of pictures
Fear	0.85	25
Sad	0.65	28
Weighted Average	0.74	53

Table 5.41: The precisin results against the FER-2013 Public Test dataset for the ninth eleventh trial

Emotion	Precision	No. of pictures
Fear	0.70	496
Sad	0.73	653
Weighted Average	0.72	1149

Table 5.42: The precisin results against the FER-2013 Private Test dataset for the eleventh training trial

Emotion	Precision	No. of pictures
Fear	0.73	496
Sad	0.72	653
Weighted Average	0.72	1149

Analyzing the current situation:

Considering we trained for just two emotions, the results indicate there is indeed quite a lot of confusion between these two emotions. We decide to build the Confusion Matrix to investigate this.

Table 5.43: The confusion matrix for FER-2013 Public Test dataset for the eleventh training trial

Emotions	**Fear**	**Sad**
Fear	**0.60**	0.40
Sad	0.20	**0.80**

Table 5.44: The confusion matrix for FER-2013 Private Test dataset for the eleventh training trial

Emotions	**Fear**	**Sad**
Fear	**0.65**	0.35
Sad	0.21	**0.79**

From the confusion matrixes the confusion between Fear and Sad is obvious. Since Sad is learned better than Fear, the next point of action could be removing the Fear emotion from the dataset. Other points for investigation could be training on different datasets and comparing the results with FER-2013.

because you state about a real-time ap-proach, some statis-tics about the average testing time (per image or per

Chapter 6

Acting Mirror

Coming back to our main purpose of studying the FER problem, we will take a look at the application we have developed. Acting Mirror takes a video stream from the camera, from a file on the machine it runs on, or from a youtube video, and processes the video stream drawing rectangles around the face of people in the stream and labeling them with one of 7 emotions: "angry", "disgust", "scared", "happy", "sad", "surprised", "neutral".

As illustrated in 6.1, the app works by reading a video stream, and passing that stream to a the haarscade frontal face detector. For each detected face, an intelligent alogirthm parses the pixels of the face in the format of a 48x48 grayscale pixel matrix, and labels the face with one of 7 emotions. In the end, the processed output video stream is shown to the user and also saved in a file called "output.avi".

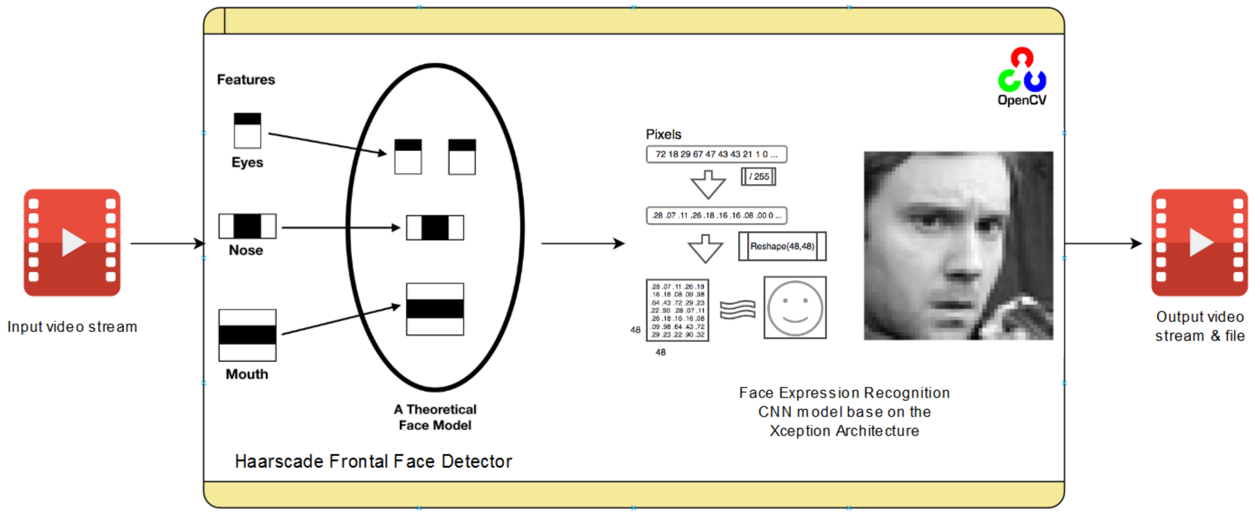

Figure 6.1: Acting Mirror Architecture

As illustrated in 6.2, Acting Mirror also presents a Graph with the percentage emotions over time of person 0 (the biggest person/face in the video stream) and a real-time percentage of each frame's emotion. All the details and options of running Acting Mirror can be found at our GitHub repository at Acting Mirror - Research and Development.

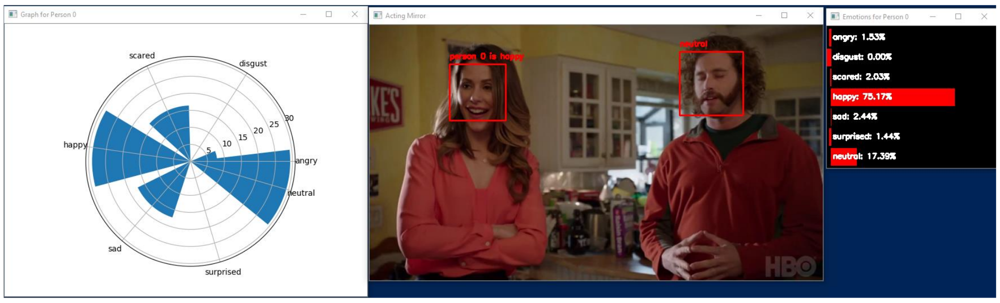

Figure 6.2: Streaming the "Hotdog, not Hotdog" scene from Sillicon Valley Season 4

<u>A few statistics about running Acting Mirror on our machine:</u>

- **Machine description**: CPU i7-7500u and an NVIDIA Quadro M520 1GB.
- **FPS** running the whole Acting Mirror either with Keras or TFLite Model:
  - **0 People**: 14.6
  - **1 Person**: 11
  - **20 People**: 3.5
- **Average time** to classify one face on **Keras** model: 0.00587 seconds / face
- **Average time** to classify one face on **TFLite** model: 0.00374 seconds / face

Chapter 7

Conclusion and future work

7.1 Analysis of Latest Results

While we were focused on finding a better set of independent variables that would help the model from [1] reach a better accuracy on the FER-2013 dataset, there were seven trials that utilized 80% of the FER-2013 dataset for training, leaving the rest to be used for computing the accuracy of the trained model. By using 10% of the FER-2013 dataset, the images intended for the Private Test, to compute each model's accuracy, we attempt at computing the Binomial proportion confidence interval.
The formula for the Binomial proportion confidence interval is as follows [9]:

$$A \in [acc - eps, acc + eps],$$

$$eps = z * (acc * (1 - acc)/n)^{\frac{1}{2}}$$

where *eps* is the value by which the accuracy of the model could vary if the training was done again, z is the number of standard deviations from the Gaussian distribution, *acc* is the classification accuracy of the model, and n is the size of the sample used to compute the classification accuracy of the model. The following computations are made for a confidence level of 95%, which means the z variable will have value 1.96, the size of the sample n will be 3.589, and the accuracy of each model *acc* will be as described in table 7.1.
The conclusion we can draw from the table 7.1 is that even though some trials achieved a slightly better accuracy than others, none of them are superior to the others, since the from the confidence interval of each trials we see that it contains all the other possible accuracies achieved by the other trials. If, however, we were to pick the 5th trial as our best model trained on the original FER-2013 dataset, then we could say that the accuracy of our model belongs to the interval [62, 66].

Side note: Even though the 7th trial's accuracy is not contained in the 8th trial, it may not be entirely fair to conclude that the 8th trial was better because it was trained on the augmented FER dataset, which contained 6 times more images than the original FER dataset.

Side note: The accuracy values used in the following table are in percentages.

Table 7.1: Confidence levels for trials 2 to 8

Trial no.	**Accuracy**	**Epsilon**	**Confidence interval**
2nd	63	2.04	[60.96, 65.04]
3rd	64	2.07	[61.93, 66.07]
4th	63	2.04	[60.96, 65.04]
5th	64	2.07	[61.93, 66.07]
6th	63	2.04	[60.96, 65.04]
7th	62	2.01	[59.99, 64.01]
8th	65	2.11	[62.89, 67.11]

7.2 State of the Art Comparison

During our research, we have focused a lot on finding a better set of independent variables that would help the model from [1] reach a better accuracy on the FER-2013 dataset. On each trial we have tested the model against the FER-2013 test sets (public and private) and against the CK+ dataset. The best results we got were on the fifth trial, with 64% accuracy on the FER-2013 dataset and 79% accuracy on the CK+ dataset. After augmenting the dataset, on the eight trial, we got an accuracy of 68% on the FER-2013 dataset, and just 71% accuracy on the CK+ dataset.
We decided to generate confusion matrixes in order to better analyze our situation, and what we found was that the FER-2013 dataset contains some confusions between emotions like Angry and Disgust, Sad and Fear, and between the aforementioned emotions and Neutral in smaller parts. We experimented with excluding the emotion Disgust from the dataset, but the improvements were not significant. We proposed 2 action points: either excluding the Fear emotion as well, which receives a big amount of confusion from Sad, or to train the model on a better dataset.
Comparing other state of the art results that trained on the CK+ dataset against our achievements looks like we need a lot of improvement to do in order to catch up.
Side note: while the attempts using Deep Networks [7] and CNN [2] extract their own features directly from the images, the SVM [7] and our model from [1] use features provided by another a different program. In our case, those features come from the Haarscade Frontal Face Detector provided by OpenCV [6] under the form of a 48x48 grayscale pixel roi. So while we show a comparison between these 4 state of the art models, we believe that this side note should also be kept in mind.

Table 7.2: Comparison between State of the Art results and our results for the CK+ dataset

State of the Art	**Deep Networks [7]**	**SVM [7]**	**CNN [2]**	**Our results - model [1]**
CK+ Dataset	[48, 96]	[31, 50]	[93, 99]	[71, 79]

7.3 Brief Ethical Discussion

There are a few problems we see with regards to the FER problem. We believe it may be ok to use it, but just as long as it is not intrusive and misused.
Understanding that the current approach to the problem is not perfect is also important with regards to using these tools. For instance, when a dataset labels a picture as "Happy", is that person really happy? Or are they just smiling, laughing or having an uncontrollable hysterical laughther? Not all

laughter or smile should be associated with happiness, which is a state that can be manifested through a neutral, poker face, as well.

The context where FER would be applied should be an accepted context by all its users and only manifested for as long as accepted by the people and only for as long as it brings a positive value to the people. In figure 7.1 we propose a mind-map which describes 5 such applications.

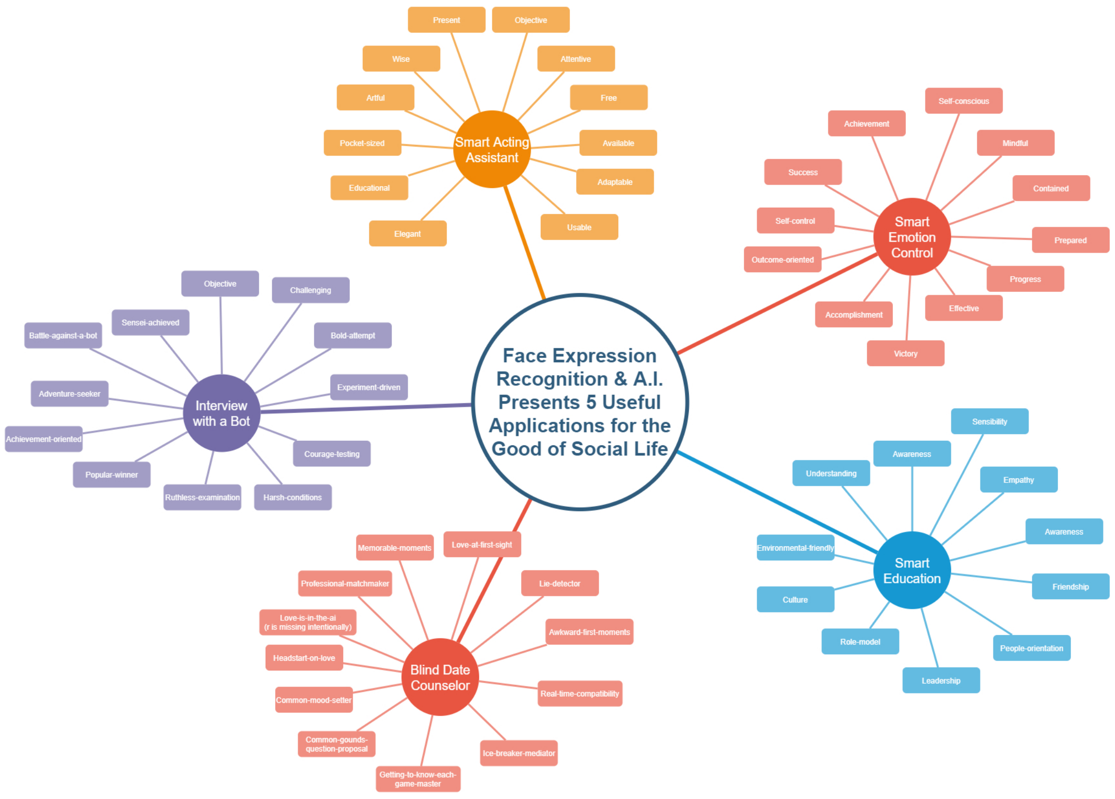

Figure 7.1: Applications for FER - MindMap

7.4 Acting Mirror and Future Work

Our practical contribution, Acting Mirorr is an applicaiton which takes a video stream from the camera, from a file on the machine it runs on, or from a youtube video, and processes the video stream drawing rectangles around the face of people in the stream and labeling them with one of 7 emotions: "angry", "disgust", "scared", "happy", "sad", "surprised", "neutral".
Further development of Acting Mirror could mean bringing it to mobile devices, optimizing the stream of data in the application, or adding new features to it, such as the ability to parse a script and give real-time audio feedback to actors about their performance or tips that may help them further increase their acting skills.

Other future work involving the FER problem could be creating and validating good face expression emotion datasets. One thing we learned was to search for confusion matrixes of the dataset we intend to use and study on before actually starting the work. It would have probably saved us a lot of trial and error time or enable us to achieve greater results.

Studying the ethical issues related to the FER problem is something we need now more than ever. With tools that are already developed, or even better ones in development, is only a matter of times before we, as a human race, are in a situation of having tools we do not properly understand their intended use or value. Making sure we are prepared and education on how and when to use such tools is of utmost importance.

In conclusion, we believe we are on the right track when it comes to developing tools that make use of intelligent algorithms, such as those capable of learning the FER problem, its just very important that we also become prepared and educated on how to use them. Experimenting, however, is part of the journey, which is why we propose the 5 applications from the MindMap at 7.1 as a next stepping stone in applied FER.

Bibliography

[1] Octavio Arriaga, Paul G. Ploger, and Matias Valdenegro. Real-time convolutional neural networks for emotion and gender classification. 2017.

[2] Peter Burkert, Felix Trier, Muhammad Zeshan Afzal, Andreas Dengel, and Marcus Liwicki. Dexpression: Deep convolutional neural network for expression recognition. *German Research Center for Artificial Intelligence (DFKI)*, 2016.

[3] Catalin Caleanu. Face expression recognition: A brief overview of the last decade. *SACI 2013 - 8th IEEE International Symposium on Applied Computational Intelligence and Informatics, Proceedings*, pages 157–161, 2013.

[4] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, 2016.

[5] Alexander Jung. imgaug - a python library.

[6] OpenCV. Opencv: Open source computer vision library.

[7] Najmeh Samadiani, Guangyan Huang, Borui Cai, Wei Luo, Chi-Hung Chi, Yong Xiang, and Jing He. A review on automatic facial expression recognition systems assisted by multimodal sensor data. *Multidisciplinary Digital Publishing Institute (MDPI)*, 2019.

[8] Pawel Tarnowski, Marcin Kolodziej, Andrzej Majkowski, and Remigiusz J. Rak. Emotion recognition using facial expressions. 2017.

[9] Wikipedia. Binomial proportion confidence interval.

Appendix A

Acting Mirror

A.1 About code refactoring and style

In this appendix, we describe our journey to refactoring the code in Acting Mirror according to a common style in Python, using Pylint. In here, you will find the state of the acting mirror module before and after refactoring, as well as a summary of some of the suggestions Pylint enlisted for the module. Without further ado, we will start with the source file for acting mirror: **acting_mirror.py**

A.1.1 acting_mirror.py before refactoring

```
from keras.preprocessing.image import img_to_array
import imutils
import cv2
from keras.models import load_model
import numpy as np
import tensorflow as tf
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import pafy
import time

import argparse
parser = argparse.ArgumentParser(description='Acting mirror arguments.')
```

```
parser.add_argument('--camera', action="store", dest="camera", type=int, help='camera from
    ↪ which to stream, referenced by its identifier')
parser.add_argument('--video', action='store', dest="video", type=str, help='offline video
    ↪ source from which to stream and to display the emotions')
parser.add_argument('--youtube', action='store', dest="url", type=str, help='youtube video
    ↪ source from which to stream and to display the emotions')
parser.add_argument('--details', action='store_true', dest='details', default=False, help='
    ↪ flag to show the emotion details window, if missing, the emotion detials window will
    ↪ not show')
parser.add_argument('--graph', action='store_true', dest='graph', default=False, help='flag
    ↪ to show a graph with the emotions percentage over time, if missing, the graph will
    ↪ not show')
parser.add_argument('--disable_rectangles', action='store_false', dest='disable_rectangles',
    ↪  default=True, help='flag to disable showing the face rectangles on the acting mirror
    ↪  window, if missing, the face rectangles will show')
parser.add_argument('--tflite', action='store_true', dest='tflite', default=False, help='
    ↪ floag to choose between keras and tflite. When this is missing, keras model will be
    ↪ used, otherwise tflite model will be used.')
args = parser.parse_args()

app_state = type('obj', (object,), {
    'show_graph': args.graph,
    'show_details': args.details,
    'show_rectangles': args.disable_rectangles
})
#print(args.graph)
#print(args.details)
#print(args.disable_rectangles)

acting_mirror_window_name = 'Acting Mirror'
graph_window_name = 'Graph for Person 0'
details_window_name = 'Emotions for Person 0'

# parameters for loading data and images
detection_model_path = 'haarcascade_files/haarcascade_frontalface_default.xml'
emotion_model_path = 'models/_mini_XCEPTION.110-0.68.hdf5'
```

```
# hyper-parameters for bounding boxes shape
# loading models
face_detection = cv2.CascadeClassifier(detection_model_path)
# loading emotion models
emotion_classifier = load_model(emotion_model_path, compile=False)
def classify_with_keras(roi):
    return emotion_classifier.predict(roi)[0]

interpreter = tf.lite.Interpreter(model_path="models/model.tflite")
interpreter.allocate_tensors()
def classify_with_tflite(roi):
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    interpreter.set_tensor(input_details[0]['index'], roi)
    interpreter.invoke()
    return interpreter.get_tensor(output_details[0]['index'])[0]

#EMOTIONS = ["angry", "disgust", "scared", "happy", "sad", "surprised", "neutral"]
EMOTIONS = ["angry", "scared", "happy", "sad", "surprised", "neutral"]

def initial_emotions_count():
    return {
        "angry": 0 ,
        #"disgust": 0,
        "scared": 0,
        "happy": 0,
        "sad": 0,
        "surprised": 0,
        "neutral": 1
    }

def update_graph_data():
    all_emotions_count = list(emotions_count.values())
    total_emotions = sum(all_emotions_count)
    percentage_emotions = [x / total_emotions * 100 for x in all_emotions_count]

    ax1.clear()
    ax1.bar(list(emotions_count.keys()), percentage_emotions)

emotions_count = initial_emotions_count()

fig = plt.figure()
ax1 = fig.add_subplot(111, projection='polar')
update_graph_data()

# starting video streaming
cv2.namedWindow(acting_mirror_window_name)
out = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10, (800,600))

if args.camera is not None:
    camera = cv2.VideoCapture(args.camera)
elif args.video is not None:
    camera = cv2.VideoCapture(args.video)
elif args.url is not None:
    url = args.url #'https://www.youtube.com/watch?v=GykoMAWwy6Y'
    vPafy = pafy.new(url)
    play = vPafy.getbest(preftype="webm")
    camera = cv2.VideoCapture(play.url)
else:
    camera = cv2.VideoCapture(0)

startTime = time.time()
frameNumber = 0
totalPredictionTime = 0
predictedFrames = 0
framesHeadstart = 10
while camera.isOpened():
    frame = camera.read()[1]
    #reading the frame
    frame = imutils.resize(frame,width=800)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize
        ↪ =(30,30),flags=cv2.CASCADE_SCALE_IMAGE)
    faces = sorted(faces, reverse=True,
        key=lambda x: (x[2] - x[0]) * (x[3] - x[1]))

    canvas = np.zeros((250, 300, 3), dtype="uint8")
    frameClone = frame.copy()

    for index in range(len(faces)):
        (fX, fY, fW, fH) = faces[index]
        # Extract the ROI of the face from the grayscale image, resize it to a fixed 48x48
            ↪ pixels, and then prepare
        # the ROI for classification via the CNN
        roi = gray[fY:fY + fH, fX:fX + fW]
        roi = cv2.resize(roi, (48, 48))
        roi = roi.astype("float") / 255.0
        roi = img_to_array(roi)
        roi = np.expand_dims(roi, axis=0)



        predictStartTime = time.time()
        if args.tflite:
            preds = classify_with_tflite(roi)
        else:
            preds = classify_with_keras(roi)
        predictEndTime = time.time()
        predictedFrames += 1
        if predictedFrames > framesHeadstart:
            totalPredictionTime += (predictEndTime - predictStartTime)
            averagePredictionTime = totalPredictionTime / (predictedFrames - framesHeadstart)
            print("Predicted frames: {} -> Average prediction time: {}".format(
                ↪ predictedFrames, averagePredictionTime))

        emotion_probability = np.max(preds)
        label = EMOTIONS[preds.argmax()]

        for (i, (emotion, prob)) in enumerate(zip(EMOTIONS, preds)):
            if index == 0:
                emotions_count[label] += 1
            if index == 0:
                # construct the label text
                text = "{}: {:.2f}%".format(emotion, prob * 100)
                w = int(prob * 300)
                cv2.rectangle(canvas, (7, (i * 35) + 5),
                (w, (i * 35) + 35), (0, 0, 255), -1)
                cv2.putText(canvas, text, (10, (i * 35) + 23),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45,
                (255, 255, 255), 2)

            if app_state.show_rectangles:
                # construct the rectangle frames for persons
                if index == 0 and (app_state.show_details or app_state.show_graph):
                    cv2.putText(frameClone, 'person {} is {}'.format(index, label), (fX, fY -
                        ↪ 10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
                else:
                    cv2.putText(frameClone, label, (fX, fY - 10), cv2.FONT_HERSHEY_SIMPLEX,
                        ↪ 0.45, (0, 0, 255), 2)
                cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH),
                              (0, 0, 255), 2)

    # frames per second
    frameNumber += 1
    elapsedTime = time.time() - startTime
    fps = frameNumber / elapsedTime
    if elapsedTime > 15.0:
        startTime = time.time()
        frameNumber = 0

    cv2.putText(frameClone, 'FPS: {}'.format(fps), (25, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.25,
        ↪  (255, 0, 0), 1)

    cv2.imshow(acting_mirror_window_name, frameClone)
    outputFrame = cv2.resize(frameClone, (800,600), interpolation = cv2.INTER_AREA)
    out.write(outputFrame)

    if app_state.show_details:
        cv2.imshow(details_window_name, canvas)

    if app_state.show_graph == True:
        update_graph_data()

        fig.canvas.draw()
        img = np.fromstring(fig.canvas.tostring_rgb(), dtype=np.uint8,sep='')
        img = img.reshape(fig.canvas.get_width_height()[::-1] + (3,))
        # img is rgb, convert to opencv's default bgr
        img = cv2.cvtColor(img,cv2.COLOR_RGB2BGR)
        # display image with opencv or any operation you like
        cv2.imshow(graph_window_name,img)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

camera.release()
out.release()
cv2.destroyAllWindows()
```

A.1.2 acting_mirror.py refactoring suggestions

Running the command `pylint acting_mirror.py --disable=E1101,C0103` yielded the following suggestions:

- Some lines were too long, so they should be made shorter
- Some lines contained trailing whitespaces, which should be removed
- Some commas had none or too many whitespaces after them
- There was wrong indentation in some places
- The imports should have been written in a certain order or grouped

The full list of suggestions is this as follows:

```
************* Module acting_mirror
acting_mirror.py:15:0: C0301: Line too long (138/100) (line-too-long)
acting_mirror.py:16:0: C0301: Line too long (148/100) (line-too-long)
acting_mirror.py:17:0: C0301: Line too long (148/100) (line-too-long)
acting_mirror.py:18:0: C0301: Line too long (186/100) (line-too-long)
acting_mirror.py:19:0: C0301: Line too long (185/100) (line-too-long)
acting_mirror.py:20:0: C0301: Line too long (229/100) (line-too-long)
acting_mirror.py:21:0: C0301: Line too long (217/100) (line-too-long)
acting_mirror.py:63:19: C0326: No space allowed before comma
       "angry": 0 ,
                 ^ (bad-whitespace)
acting_mirror.py:65:20: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:66:19: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:67:17: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:88:34: C0326: Exactly one space required after comma
out = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10, (800,600))
                              ^ (bad-whitespace)
acting_mirror.py:88:61: C0326: Exactly one space required after comma
out = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10, (800,600))
                                                     ^ (bad-whitespace)
acting_mirror.py:88:65: C0326: Exactly one space required after comma
out = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10, (800,600))
                                                         ^ (bad-whitespace)
acting_mirror.py:88:69: C0326: Exactly one space required after comma
out = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10, (800,600))
                                                             ^ (bad-whitespace)
acting_mirror.py:88:84: C0326: Exactly one space required after comma
out = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10, (800,600))
                                                                            ^ (bad-
                                                                                ↪ whitespace
                                                                                ↪ )
acting_mirror.py:110:32: C0326: Exactly one space required after comma
   frame = imutils.resize(frame,width=800)
                            ^ (bad-whitespace)
acting_mirror.py:112:0: C0301: Line too long (126/100) (line-too-long)
acting_mirror.py:112:48: C0326: Exactly one space required after comma
   faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize
       ↪ =(30,30),flags=cv2.CASCADE_SCALE_IMAGE)
                                            ^ (bad-whitespace)
acting_mirror.py:112:64: C0326: Exactly one space required after comma
```

```
    faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize
        ↪ =(30,30),flags=cv2.CASCADE_SCALE_IMAGE)
                                                            ^ (bad-whitespace)
acting_mirror.py:112:79: C0326: Exactly one space required after comma
    faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize
        ↪ =(30,30),flags=cv2.CASCADE_SCALE_IMAGE)
                                                                        ^ (bad-whitespace)
acting_mirror.py:112:91: C0326: Exactly one space required after comma
    faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize
        ↪ =(30,30),flags=cv2.CASCADE_SCALE_IMAGE)
                                                                                    ^ (bad-
                                                                                    ↪ whitespace
                                                                                    ↪ )
acting_mirror.py:112:95: C0326: Exactly one space required after comma
    faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize
        ↪ =(30,30),flags=cv2.CASCADE_SCALE_IMAGE)
                                                                                        ^ (bad
                                                                                        ↪ -
                                                                                        ↪ whitespace
                                                                                        ↪ )
                                                                                        ↪
acting_mirror.py:114:0: C0330: Wrong continued indentation (add 11 spaces).
       key=lambda x: (x[2] - x[0]) * (x[3] - x[1]))
       ^ | (bad-continuation)
acting_mirror.py:121:0: C0301: Line too long (115/100) (line-too-long)
acting_mirror.py:128:0: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:129:0: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:140:0: C0301: Line too long (119/100) (line-too-long)
acting_mirror.py:146:26: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:148:0: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:154:0: C0330: Wrong continued indentation (add 14 spaces).
              (w, (i * 35) + 35), (0, 0, 255), -1)
              ^ | (bad-continuation)
acting_mirror.py:156:0: C0330: Wrong continued indentation (add 12 spaces).
              cv2.FONT_HERSHEY_SIMPLEX, 0.45,
              ^ | (bad-continuation)
acting_mirror.py:157:0: C0330: Wrong continued indentation (add 12 spaces).
```

```
               (255, 255, 255), 2)
               ^ | (bad-continuation)
acting_mirror.py:158:0: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:162:0: C0301: Line too long (146/100) (line-too-long)
acting_mirror.py:164:0: C0301: Line too long (113/100) (line-too-long)
acting_mirror.py:166:0: C0330: Wrong continued indentation (remove 2 spaces).
                             (0, 0, 255), 2)
                           | ^ (bad-continuation)
acting_mirror.py:167:0: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:175:0: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:176:0: C0301: Line too long (108/100) (line-too-long)
acting_mirror.py:179:45: C0326: Exactly one space required after comma
    outputFrame = cv2.resize(frameClone, (800,600), interpolation = cv2.INTER_AREA)
                                          ^ (bad-whitespace)
acting_mirror.py:179:66: C0326: No space allowed around keyword argument assignment
    outputFrame = cv2.resize(frameClone, (800,600), interpolation = cv2.INTER_AREA)
                                                               ^ (bad-whitespace)
acting_mirror.py:181:0: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:184:0: C0303: Trailing whitespace (trailing-whitespace)
acting_mirror.py:189:69: C0326: Exactly one space required after comma
       img = np.fromstring(fig.canvas.tostring_rgb(), dtype=np.uint8,sep='')
                                                                    ^ (bad-whitespace)
acting_mirror.py:190:13: C0326: Exactly one space required before assignment
       img = img.reshape(fig.canvas.get_width_height()[::-1] + (3,))
           ^ (bad-whitespace)
acting_mirror.py:192:30: C0326: Exactly one space required after comma
       img = cv2.cvtColor(img,cv2.COLOR_RGB2BGR)
                          ^ (bad-whitespace)
acting_mirror.py:194:36: C0326: Exactly one space required after comma
       cv2.imshow(graph_window_name,img)
                                ^ (bad-whitespace)
acting_mirror.py:1:0: C0114: Missing module docstring (missing-module-docstring)
acting_mirror.py:9:0: C0413: Import "import matplotlib.pyplot as plt" should be placed at
    ↪ the top of the module (wrong-import-position)
acting_mirror.py:10:0: C0413: Import "import pafy" should be placed at the top of the module
    ↪  (wrong-import-position)
acting_mirror.py:11:0: C0413: Import "import time" should be placed at the top of the module
```

```
    ↪  (wrong-import-position)
acting_mirror.py:13:0: C0413: Import "import argparse" should be placed at the top of the
    ↪ module (wrong-import-position)
acting_mirror.py:46:0: C0116: Missing function or method docstring (missing-function-
    ↪ docstring)
acting_mirror.py:46:24: W0621: Redefining name 'roi' from outer scope (line 123) (redefined-
    ↪ outer-name)
acting_mirror.py:51:0: C0116: Missing function or method docstring (missing-function-
    ↪ docstring)
acting_mirror.py:51:25: W0621: Redefining name 'roi' from outer scope (line 123) (redefined-
    ↪ outer-name)
acting_mirror.py:61:0: C0116: Missing function or method docstring (missing-function-
    ↪ docstring)
acting_mirror.py:72:0: C0116: Missing function or method docstring (missing-function-
    ↪ docstring)
acting_mirror.py:119:4: C0200: Consider using enumerate instead of iterating with range and
    ↪ len (consider-using-enumerate)
acting_mirror.py:185:7: C0121: Comparison to True should be just 'expr' (singleton-
    ↪ comparison)
acting_mirror.py:11:0: C0411: standard import "import time" should be placed before "from
    ↪ keras.preprocessing.image import img_to_array" (wrong-import-order)
acting_mirror.py:13:0: C0411: standard import "import argparse" should be placed before "
    ↪ from keras.preprocessing.image import img_to_array" (wrong-import-order)
acting_mirror.py:4:0: C0412: Imports from package keras are not grouped (ungrouped-imports)


------------------------------------------------------------------
Your code has been rated at 5.19/10 (previous run: 5.19/10, +0.00)
```

A.1.3 acting_mirror.py after refactoring

After taking into consideration the suggestions provided by Pylint, there are still a few remaining, but we choose not to fix them, because it would not serve any good purpose. Disabling the suggestions about lines being too long and trailining whitespaces (left intentionally for readability purposes), the command `pylint acting_mirror.py --disable=E1101,C0103,C0301,C0303` yields the following result:

```
************* Module acting_mirror
```

```
acting_mirror.py:61:0: C0116: Missing function or method docstring (missing-function-
    ↪ docstring)
acting_mirror.py:61:24: W0621: Redefining name 'roi' from outer scope (line 140) (redefined-
    ↪ outer-name)
acting_mirror.py:67:0: C0116: Missing function or method docstring (missing-function-
    ↪ docstring)
acting_mirror.py:67:25: W0621: Redefining name 'roi' from outer scope (line 140) (redefined-
    ↪ outer-name)
acting_mirror.py:76:0: C0116: Missing function or method docstring (missing-function-
    ↪ docstring)
acting_mirror.py:87:0: C0116: Missing function or method docstring (missing-function-
    ↪ docstring)
acting_mirror.py:135:4: C0200: Consider using enumerate instead of iterating with range and
    ↪ len (consider-using-enumerate)


------------------------------------------------------------------
Your code has been rated at 9.47/10 (previous run: 9.47/10, +4.28)
```

The improved code is as follows:

```
"""
Acting mirror is an application that takes a video stream
(local file, youtube video or camera) and labels faces with emotions.
For full documentation at repo: https://github.com/bogdanbalanescu/Intelligent-Tools-for-
    ↪ Social-Good
"""
import argparse
import time
import pafy
import matplotlib
import matplotlib.pyplot as plt
import cv2
import imutils
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import img_to_array
from keras.models import load_model
matplotlib.use('TkAgg')
```

```
parser = argparse.ArgumentParser(description='Acting mirror arguments.')
parser.add_argument(
    '--camera', action="store", dest="camera", type=int,
    help='camera from which to stream, referenced by its identifier')
parser.add_argument(
    '--video', action='store', dest="video", type=str,
    help='offline video source from which to stream and to display the emotions')
parser.add_argument(
    '--youtube', action='store', dest="url", type=str,
    help='youtube video source from which to stream and to display the emotions')
parser.add_argument(
    '--details', action='store_true', dest='details', default=False,
    help='flag to show the emotion details window, if missing, the emotion detials window
        ↪ will not show')
parser.add_argument(
    '--graph', action='store_true', dest='graph', default=False,
    help='flag to show a graph with the emotions percentage over time, if missing, the graph
        ↪  will not show')
parser.add_argument(
    '--disable_rectangles', action='store_false', dest='disable_rectangles', default=True,
    help='flag to disable showing the face rectangles on the acting mirror window, if
        ↪ missing, the face rectangles will show')
parser.add_argument(
    '--tflite', action='store_true', dest='tflite', default=False,
    help='floag to choose between keras and tflite. When this is missing, keras model will
        ↪ be used, otherwise tflite model will be used.')
args = parser.parse_args()

app_state = type('obj', (object,), {
    'show_graph': args.graph,
    'show_details': args.details,
    'show_rectangles': args.disable_rectangles
})

acting_mirror_window_name = 'Acting Mirror'
graph_window_name = 'Graph for Person 0'
```

```
details_window_name = 'Emotions for Person 0'

# parameters for loading data and images
detection_model_path = 'haarcascade_files/haarcascade_frontalface_default.xml'
emotion_model_path = 'models/_mini_XCEPTION.110-0.68.hdf5'

# loading models
face_detection = cv2.CascadeClassifier(detection_model_path)
# loading emotion model for keras
emotion_classifier = load_model(emotion_model_path, compile=False)
def classify_with_keras(roi):
    return emotion_classifier.predict(roi)[0]

# loading emotion model for tflite
interpreter = tf.lite.Interpreter(model_path="models/model.tflite")
interpreter.allocate_tensors()
def classify_with_tflite(roi):
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    interpreter.set_tensor(input_details[0]['index'], roi)
    interpreter.invoke()
    return interpreter.get_tensor(output_details[0]['index'])[0]

EMOTIONS = ["angry", "scared", "happy", "sad", "surprised", "neutral"]

def initial_emotions_count():
    return {
        "angry": 0,
        #"disgust": 0,
        "scared": 0,
        "happy": 0,
        "sad": 0,
        "surprised": 0,
        "neutral": 1
    }

def update_graph_data():
    all_emotions_count = list(emotions_count.values())
    total_emotions = sum(all_emotions_count)
    percentage_emotions = [x / total_emotions * 100 for x in all_emotions_count]

    ax1.clear()
    ax1.bar(list(emotions_count.keys()), percentage_emotions)

emotions_count = initial_emotions_count()

fig = plt.figure()
ax1 = fig.add_subplot(111, projection='polar')
update_graph_data()

# starting video streaming
cv2.namedWindow(acting_mirror_window_name)
out = cv2.VideoWriter('output.avi', cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'), 10, (800,
    ↪ 600))

if args.camera is not None:
    camera = cv2.VideoCapture(args.camera)
elif args.video is not None:
    camera = cv2.VideoCapture(args.video)
elif args.url is not None:
    url = args.url #'https://www.youtube.com/watch?v=GykoMAWwy6Y'
    vPafy = pafy.new(url)
    play = vPafy.getbest(preftype="webm")
    camera = cv2.VideoCapture(play.url)
else:
    camera = cv2.VideoCapture(0)

startTime = time.time()
frameNumber = 0
totalPredictionTime = 0
predictedFrames = 0
framesHeadstart = 10
while camera.isOpened():
    frame = camera.read()[1]
```

```
#reading the frame
frame = imutils.resize(frame, width=800)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_detection.detectMultiScale(
    gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30), flags=cv2.
        ↪ CASCADE_SCALE_IMAGE)
faces = sorted(
    faces, reverse=True, key=lambda x: (x[2] - x[0]) * (x[3] - x[1]))

canvas = np.zeros((250, 300, 3), dtype="uint8")
frameClone = frame.copy()

for index in range(len(faces)):
    (fX, fY, fW, fH) = faces[index]
    # Extract the ROI of the face from the grayscale image,
    # resize it to a fixed 48x48 pixels, and then prepare
    # the ROI for classification via the CNN
    roi = gray[fY:fY + fH, fX:fX + fW]
    roi = cv2.resize(roi, (48, 48))
    roi = roi.astype("float") / 255.0
    roi = img_to_array(roi)
    roi = np.expand_dims(roi, axis=0)

    predictStartTime = time.time()
    if args.tflite:
        preds = classify_with_tflite(roi)
    else:
        preds = classify_with_keras(roi)
    predictEndTime = time.time()
    predictedFrames += 1
    if predictedFrames > framesHeadstart:
        totalPredictionTime += (predictEndTime - predictStartTime)
        averagePredictionTime = totalPredictionTime / (predictedFrames - framesHeadstart)
        print("Predicted frames: {} -> Average prediction time: {}"
              .format(predictedFrames, averagePredictionTime))

    emotion_probability = np.max(preds)
```

```
        label = EMOTIONS[preds.argmax()]

        for (i, (emotion, prob)) in enumerate(zip(EMOTIONS, preds)):
            if index == 0:
                emotions_count[label] += 1

            if index == 0:
                # construct the label text
                text = "{}: {:.2f}%".format(emotion, prob * 100)
                w = int(prob * 300)
                cv2.rectangle(canvas, (7, (i * 35) + 5),
                              (w, (i * 35) + 35), (0, 0, 255), -1)
                cv2.putText(canvas, text, (10, (i * 35) + 23),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.45,
                            (255, 255, 255), 2)

            if app_state.show_rectangles:
                # construct the rectangle frames for persons
                if index == 0 and (app_state.show_details or app_state.show_graph):
                    cv2.putText(frameClone, 'person {} is {}'.format(index, label),
                                (fX, fY - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
                else:
                    cv2.putText(frameClone, label, (fX, fY - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
                cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH), (0, 0, 255), 2)

    # frames per second
    frameNumber += 1
    elapsedTime = time.time() - startTime
    fps = frameNumber / elapsedTime
    if elapsedTime > 15.0:
        startTime = time.time()
        frameNumber = 0

    cv2.putText(frameClone, 'FPS: {}'.format(fps), (25, 25),
                cv2.FONT_HERSHEY_SIMPLEX, 0.25, (255, 0, 0), 1)
    cv2.imshow(acting_mirror_window_name, frameClone)
    outputFrame = cv2.resize(frameClone, (800, 600), interpolation=cv2.INTER_AREA)
    out.write(outputFrame)

    if app_state.show_details:
        cv2.imshow(details_window_name, canvas)

    if app_state.show_graph is True:
        update_graph_data()

        fig.canvas.draw()
        img = np.fromstring(fig.canvas.tostring_rgb(), dtype=np.uint8, sep='')
        img = img.reshape(fig.canvas.get_width_height()[::-1] + (3,))
        # img is rgb, convert to opencv's default bgr
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        # display image with opencv or any operation you like
        cv2.imshow(graph_window_name, img)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

camera.release()
out.release()
cv2.destroyAllWindows()
```

Appendix B

Model training and validation

B.1 A summary of code refactoring and style

In this appendix, we briefly state the improvements we made to each file related to the process of training, testing and validating the model and also provide the final and improved version of the files. The approach taken to make the improvements will be similary to what we have seen in Appendix A.

B.1.1 data_loader.py

After the code rating has been improved from 5.33/10 to 9.35/10, using the command `pylint data_loader.py --disable=E1101,C0103,C0301,C0303,W0621`, the following suggestions were fixed:

- Some lines contained trailing whitespaces, which should be removed
- Some commas had none or too many whitespaces after them
- There was wrong indentation in some places

```
"""
Data loader reads a .csv file and creates a train and test dataset for our model to be
    ↪ trained with.
Requirements for the .csv file:
- it should have lines with label, followed by an array of 2304 (48x48) grayscale pixels,
    ↪ separated by spaces.
- e.g.: 1, 201 103 78 ... 101
The train images will be in variable: xtrain
The train labels will be in variable: ytrain
The test images will be in variable: xtest
The test labels will be in variable: ytest
"""
import pandas as pd
import cv2
import numpy as np
from sklearn.model_selection import train_test_split

dataset_path = 'datasets/fer2013/train_fer2013.csv'

image_size = (48, 48)

"""
Reads .csv file and returns the tuple (faces, emotions).
"""
def load_fer2013():
    data = pd.read_csv(dataset_path)
    pixels = data['pixels'].tolist()
    width, height = 48, 48
    faces = []
    for pixel_sequence in pixels:
        face = [int(pixel) for pixel in pixel_sequence.split(' ')]
        face = np.asarray(face).reshape(width, height)
        face = cv2.resize(face.astype('uint8'), image_size)
        faces.append(face.astype('float32'))
    faces = np.asarray(faces)
    faces = np.expand_dims(faces, -1)
    emotions = pd.get_dummies(data['emotion']).as_matrix()
    return faces, emotions

def preprocess_input(x, v2=True):
    x = x.astype('float32')
    x = x / 255.0
    if v2:
        x = x - 0.5
        x = x * 2.0
    return x
```

```
faces, emotions = load_fer2013()
faces = preprocess_input(faces)
xtrain, xtest, ytrain, ytest = train_test_split(faces, emotions, test_size=0.2, shuffle=True
    ↪ )
```

B.1.2 tf_model.py

After the code rating has been improved from 3.07/10 to 10/10, using the command `pylint data_loader.py --disable=E1101,C0103,C0301,C0303,W0611,W0614` the following suggestions were fixed:

- Some lines were too long, so they should be made shorter
- Some lines contained trailing whitespaces, which should be removed
- Some commas had none or too many whitespaces after them
- There was wrong indentation in some places
- The imports should have been written in a certain order or grouped

```
"""
Creates and trains a model to label faces with emotions from the FER-2013 dataset,
read through the data_loader module.
"""
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping
from keras.callbacks import ReduceLROnPlateau
from keras.layers import Activation, Convolution2D, Dropout, Conv2D
from keras.layers import AveragePooling2D, BatchNormalization
from keras.layers import GlobalAveragePooling2D
from keras.models import Sequential
from keras.layers import Flatten
from keras.models import Model
from keras.layers import Input
from keras.layers import MaxPooling2D
from keras.layers import SeparableConv2D
from keras import layers
from keras import losses
from keras.regularizers import l2
import pandas as pd
import cv2
import numpy as np
from data_loader import xtrain, ytrain, xtest, ytest
```

```
# parameters
batch_size = 32
num_epochs = 110
input_shape = (48, 48, 1)
verbose = 1
num_classes = 7
patience = 50
base_path = 'models/'
l2_regularization = 0.01

# data generator
data_generator = ImageDataGenerator(
    featurewise_center=False,
    featurewise_std_normalization=False,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=.1,
    horizontal_flip=True)

# model parameters
regularization = l2(l2_regularization)

# base
img_input = Input(input_shape)
x = Conv2D(8, (3, 3), strides=(1, 1), kernel_regularizer=regularization, use_bias=False)(
    ↪ img_input)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Conv2D(8, (3, 3), strides=(1, 1), kernel_regularizer=regularization, use_bias=False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

# module 1
residual = Conv2D(16, (1, 1), strides=(2, 2), padding='same', use_bias=False)(x)
residual = BatchNormalization()(residual)
```

```
x = SeparableConv2D(16, (3, 3), padding='same', kernel_regularizer=regularization, use_bias=
    ↪ False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = SeparableConv2D(16, (3, 3), padding='same', kernel_regularizer=regularization, use_bias=
    ↪ False)(x)
x = BatchNormalization()(x)
x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
x = layers.add([x, residual])

# module 2
residual = Conv2D(32, (1, 1), strides=(2, 2), padding='same', use_bias=False)(x)
residual = BatchNormalization()(residual)
x = SeparableConv2D(32, (3, 3), padding='same', kernel_regularizer=regularization, use_bias=
    ↪ False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = SeparableConv2D(32, (3, 3), padding='same', kernel_regularizer=regularization, use_bias=
    ↪ False)(x)
x = BatchNormalization()(x)
x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
x = layers.add([x, residual])

# module 3
residual = Conv2D(64, (1, 1), strides=(2, 2), padding='same', use_bias=False)(x)
residual = BatchNormalization()(residual)
x = SeparableConv2D(64, (3, 3), padding='same', kernel_regularizer=regularization, use_bias=
    ↪ False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = SeparableConv2D(64, (3, 3), padding='same', kernel_regularizer=regularization, use_bias=
    ↪ False)(x)
x = BatchNormalization()(x)
x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
x = layers.add([x, residual])

# module 4
```

```
residual = Conv2D(128, (1, 1), strides=(2, 2), padding='same', use_bias=False)(x)
residual = BatchNormalization()(residual)
x = SeparableConv2D(128, (3, 3), padding='same', kernel_regularizer=regularization, use_bias
    ↪ =False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = SeparableConv2D(128, (3, 3), padding='same', kernel_regularizer=regularization, use_bias
    ↪ =False)(x)
x = BatchNormalization()(x)
x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
x = layers.add([x, residual])
x = Conv2D(num_classes, (3, 3), padding='same')(x)
x = GlobalAveragePooling2D()(x)
output = Activation('softmax', name='predictions')(x)

model = Model(img_input, output)
#model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.compile(optimizer='adam', loss=losses.MeanSquaredError(), metrics=['accuracy'])
#model.compile(optimizer='adam', loss=losses.squared_hinge, metrics=['accuracy'])
model.summary()

# callbacks
log_file_path = base_path + '_emotion_training.log'
csv_logger = CSVLogger(log_file_path, append=False)
early_stop = EarlyStopping('val_loss', patience=patience)
reduce_lr = ReduceLROnPlateau('val_loss', factor=0.1, patience=int(patience/4), verbose=1)
trained_models_path = base_path + '_mini_XCEPTION'
model_names = trained_models_path + '.{epoch:02d}-{val_accuracy:.2f}.hdf5'
model_checkpoint = ModelCheckpoint(model_names, 'val_loss', verbose=1, save_best_only=True)
callbacks = [model_checkpoint, csv_logger, early_stop, reduce_lr]

model.fit_generator(data_generator.flow(xtrain, ytrain, batch_size),
                  steps_per_epoch=len(xtrain) / batch_size,
                  epochs=num_epochs, verbose=1, callbacks=callbacks,
                  validation_data=(xtest, ytest))
```

B.1.3 tf_model_validation.py

After the code rating has been improved from 2.94/10 to 10/10, using the command
`pylint data_loader.py --disable=E1101,C0103,W0614` the following suggestions were fixed:

- Some lines contained trailing whitespaces, which should be removed
- Some commas had none or too many whitespaces after them
- Some imports were not used

```
"""
Validates the model against the (faces, emotions) from a dataset,
read through the data_loader module, printing the precision for each label,
as well as the confusion matrix for the model.
"""
from sklearn.metrics import classification_report, confusion_matrix
from keras.models import load_model
import numpy as np
from data_loader import faces, emotions

emotion_model_path = 'models/_mini_XCEPTION.110-0.68.hdf5'

model = load_model(emotion_model_path, compile=True)
EMOTIONS = ["angry", "disgust", "scared", "happy", "sad", "surprised", "neutral"]

y_pred = model.predict(faces, batch_size=64, verbose=1)
y_pred_bool = np.argmax(y_pred, axis=1)
y_rounded_train_labels = np.argmax(emotions, axis=1)

print(classification_report(y_rounded_train_labels, y_pred_bool))
cm = confusion_matrix(y_rounded_train_labels, y_pred_bool)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print(cm)
```