2022-2023 CAS ADS Project

Facial emotion recognition using deep learning

June 2023

Kim Lan Vu, Emilie Zucchinetti
kmln.vu@gmail.com, zucchinetti@outlook.com

Abstract

The growing interest for facial emotion recognition has led to a considerable amount of research in recent years, more specifically in the larger context of the development of artificial intelligence. In this perspective, this project takes a lean approach on the subject as we aim to recognize seven basic emotional states (anger, disgust, fear, happiness, neutral, sadness, surprise) based on images of human facial expressions. The classification is performed using a convolutional neural network (CNN) on the FER-2013 dataset. We then implement our model in a graphical user interface (GUI) application that will access the webcam to recognize the user's facial expression on the live feed before mapping it with a corresponding emoji that will finally be displayed on screen in real-time. The complexity of the dataset made it difficult to build a high-accuracy model, but we have achieved satisfactory results, in comparison with preliminary work on the same dataset. Once implemented in the GUI, the model recognizes the emotions displayed quite well, although it is sometimes surprisingly challenged by instances that would seem clear to the human eye. There are, however, several options to explore with the potential to improve the results.

Table of Contents

1 Introduction

Facial expressions play a central role in the recognition of emotions and are forms of non-verbal communication. For decades, decoding such emotion expressions has been a research interest. Recently, the high diffusion of cameras and the technological advances in machine learning have played a prominent role in the development of the Facial Emotion Recognition (FER) technology [1]. FER analyzes facial expressions from static images or videos in order to reveal information on a person's emotional state. The goal is to automate the process of determining emotions from human faces in real-time, by analyzing the various features of a face such as eyes, mouth, and other features, and mapping them to a set of emotions.

In informal electronic communication, people use emojis as functional equivalents of facial expressions. An emoji is a pictogram or a graphic representation of a face embedded in text. The primary function of an emoji is to fill in emotional cues otherwise missing from typed conversation[1]. Emojis are thus extremely popular and widely used on social media and other forms of digital communication.

In this project, we will create a graphical user interface (GUI) application that will access the webcam in order to recognize the user's facial expression on the video and then map it with a corresponding emoji that will be displayed on screen. To do so, we will build a deep learning model that classifies facial expressions in seven categories representing basic emotional states: anger, disgust, fear, happiness, neutral, sadness, surprise.

2 Method

Our main working environment is Google Colab, which provides an interactive environment that allows us to combine Markdown text and executable code in one canvas called a notebook[2]. Additionally, Google Colab grants us the chance to utilize GPU resources for executing the code, leading to faster model training. We are using the Chrome browser and Python programming language[3]. The deep learning model is written in a notebook, while the GUI application is written in a .py file. The data files are stored in Google Drive, with which Colab is integrated, and Github[4].

To support our analysis, we will use different Python libraries and packages, namely:

[1] https://en.wikipedia.org/wiki/Emoji
[2] https://colab.research.google.com
[3] https://www.google.com/chrome, https://www.python.org
[4] https://www.google.com/drive, https://github.com

- Keras: a deep learning API written in Python, running on top of the machine learning platform TensorFlow[5].
- Matplotlib: a comprehensive library for creating static, animated, and interactive visualizations[6].
- NumPy: a library that contains multidimensional array and matrix data structures[7].
- OpenCV: a module which is used for image and video processing[8].
- OS: a module that provides functions for interacting with the operating system[9].
- Pandas: a package providing fast, flexible, and expressive data structures that allows to work with "relational" or "labeled" data[10].
- Pickle: a module that implements binary protocols for serializing and de-serializing a Python object structure[11].
- Scikit-learn: an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities.[12]
- SciPy: a library that provides algorithms for optimization, integration, interpolation, algebraic equations, differential equations and statistics[13].
- Seaborn: a data visualization library based on matplotlib which provides a high-level interface for drawing attractive and informative statistical graphics.[14]
- TensorFlow: a library for fast numerical computing that can be used to create Deep Learning models[15].
- Tkinter: a GUI framework that is built into the Python standard library[16].

3 Data

This project work was performed using the FER-2013 dataset which is publicly available on Kaggle[17]. This dataset was chosen because it covers exactly the topic of interest of this project and its large size and consistent and high quality composition allow us a coherent analysis. The

[5] https://keras.io
[6] https://matplotlib.org
[7] https://numpy.org
[8] https://opencv.org
[9] https://docs.python.org/3/library/os.html
[10] https://pandas.pydata.org
[11] https://docs.python.org/3/library/pickle.html
[12] https://scikit-learn.org
[13] https://scipy.org
[14] https://seaborn.pydata.org
[15] https://www.tensorflow.org
[16] https://docs.python.org/3/library/tkinter.html
[17] https://www.kaggle.com/datasets/msambare/fer2013

dataset has been widely used in academic research, competitions, and industry applications related to facial expression analysis, emotion detection, and facial recognition. The data consists of 35’887 grayscale images, each with a resolution of 48x48 pixels. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. These images represent facial expressions of seven different emotions: anger, disgust, fear, happiness, neutral, sadness, surprise. The FER-2013 dataset was created by gathering the results of a Google image search of each emotion and synonyms of the emotions [2].

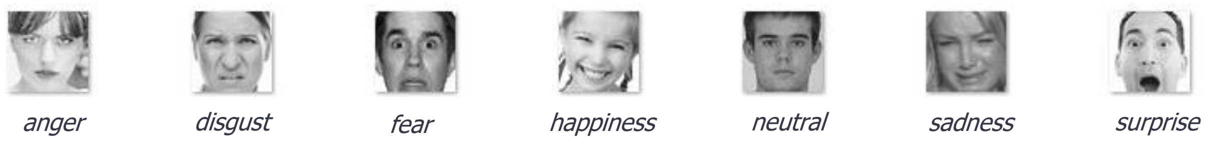


Fig. 1: Samples of the FER-2013 Dataset

The training set contains 28’709 samples and the test set contains 7’178 samples.

	Train set	Test set	Total
angry	3995	958	4953
disgust	436	111	547
fear	4097	1024	5121
happy	7215	1774	8989
neutral	4965	1233	6198
sad	4830	1247	6077
surprise	3171	831	4002
Total	28709	7178	35887

Table 1: distribution of the classes in the FER-2013 dataset

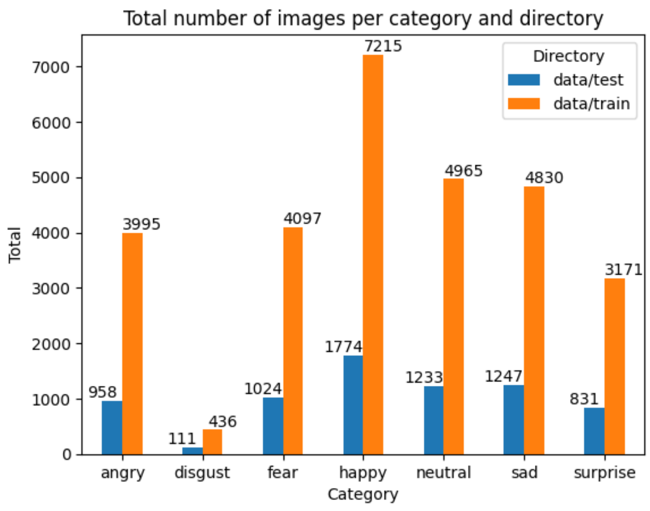


Fig. 2: distribution of the classes in the FER-2013 dataset

The classes are unevenly distributed (see table 1 and fig. 2). In particular, the class “disgust” is strongly underrepresented while the class “happiness” is largely overrepresented. Some data preprocessing will thus be needed on the train data so that we can train our deep learning model with a balanced dataset.

For the mapping of the facial emotion with an emoji, we will use a set of Google emoji images, which are available on the emojipedia website, a website that references the meaning and common usage of emojis in the Unicode Standard[18]. The set consists of seven color images of 400x400 pixels. The selected images are generic and representative of the most commonly used emojis.

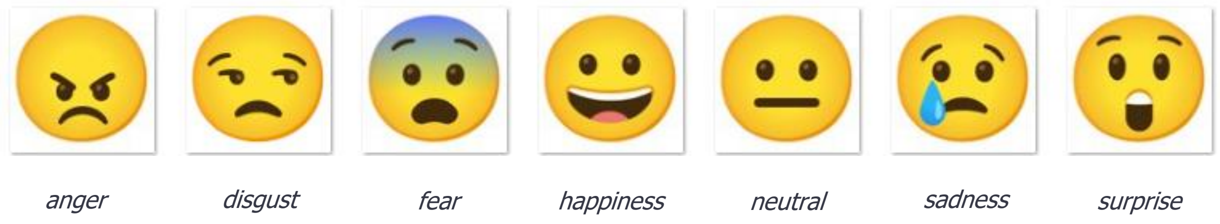


Fig. 3: Google emoji images used in the GUI application

4 Data preprocessing

As seen in the previous section, the different classes of emotions are not equally represented in the dataset. In a first attempt, we tried to use the original dataset, without applying much data preprocessing to it. Although the model generated with the original dataset performed fairly well (test accuracy 62%), an examination of the confusion matrix (see fig. 4) showed that the model performed highly when confronted with an image from the category "happy" (80%), which is by far the most represented class, whereas it achieved significantly lower results with the other classes. Because of the uneven distribution of the classes, it is hard to tell if the model actually learned patterns from the images, or if it just learned that the probability of having an image of type "happy" was so high, that it would score well by consistently predicting this category.

[18] https://emojipedia.org/google/

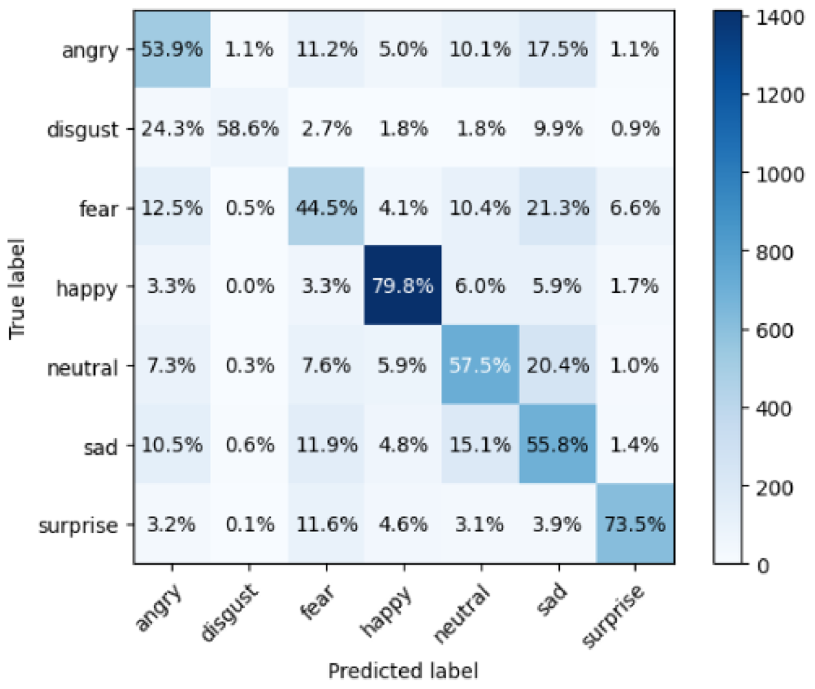


Fig. 4: confusion matrix with the model trained on the original dataset

To control this bias, we need to ensure that the model expects each class to have the same probability of appearing. To achieve this, we need to rebalance the training dataset and perform some data augmentation on the categories which are less represented than the others. We don’t need to rebalance the testing dataset as we want to be able to compare both the initial training dataset with the rebalanced dataset on a similar test set. As we wanted to maximize our use of the available data, we proceeded to oversample the underrepresented categories (instead of randomly sampling from overrepresented categories, which would have downscaled our original dataset). Ultimately, our goal was to create a new training dataset with 7’000 images in each category, as the category “happy” (the most represented category) contains 7’215 images in the training set.
In order to create artificially new images for the underrepresented categories, we applied some transformations to the original images in those categories. It may prove beneficial to transform the original images and feed the model with additional variety. Thanks to the transformations, the model is taught to recognize faces at different angles and distances. Of course, those transformations should not be too big. Otherwise the model might learn to classify images which are not relevant for the actual test set on which it will be finally evaluated. We tried two different sets of parameters and tested the results (with twice more variation flexibility in the second test). We finally chose the narrower range of transformations, as it yielded better results. Concretely, we rotated the images randomly between -15 and +15 degrees, we sheared the images by a maximum of 10% in either direction (creating a distortion effect), we zoomed

in or out of the images by a factor of up to 10%, we enabled random flipping of images horizontally and finally, we filled the newly created pixels that can appear after a given transformation (for instance after a rotation) with the value of the nearest pixel in the original image.
We then trained our model on this rebalanced dataset. We observed that although the model achieved a very slightly lower accuracy (61%) than when it was trained on the original dataset, it was however much better at predicting images from underrepresented categories, such as "disgust" or "surprise" (see section 6 and fig. 10), confirming the benefits of data augmentation.

5 Exploratory Data Analysis

Image split between categories and test/train set

The initial dataset is composed of seven categories and split between a training test and a testing set. As previously seen in section 3, the distribution of images in each category is very uneven (see fig. 2). However, the distribution between the test set and the training set seems to be proportionally equal in the different categories (about 80%-20%, see fig. 5).

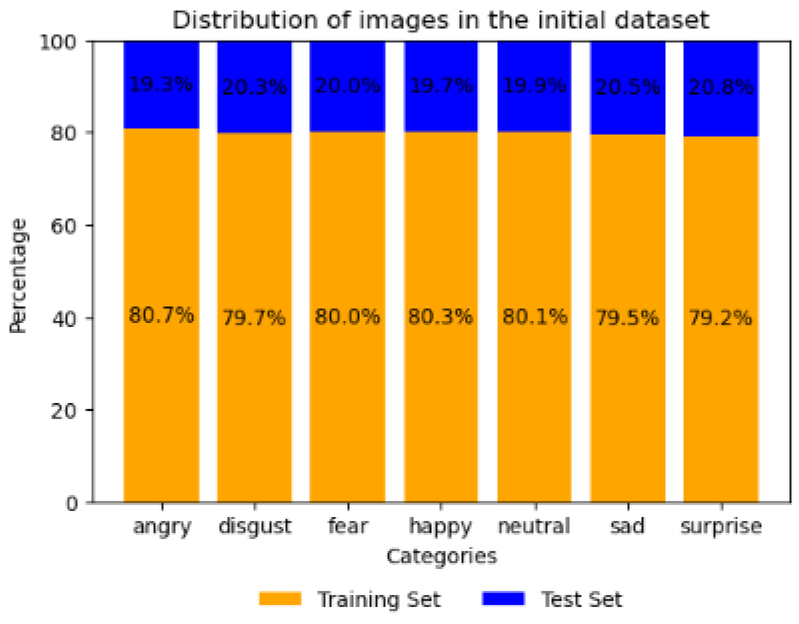


Fig. 5: distribution of the images between the training set and the test set in the original dataset

With the data augmentation on the training set we artificially created an even number of images in each category. Consequently, the distribution of the images between the training set and the test set became much less heterogeneous in the different classes, as we didn't augment the test set (see fig. 6). This is by no means an issue for the training of our model -

on the contrary, in reality, it is rare to be confronted with totally equal probabilities for each class.

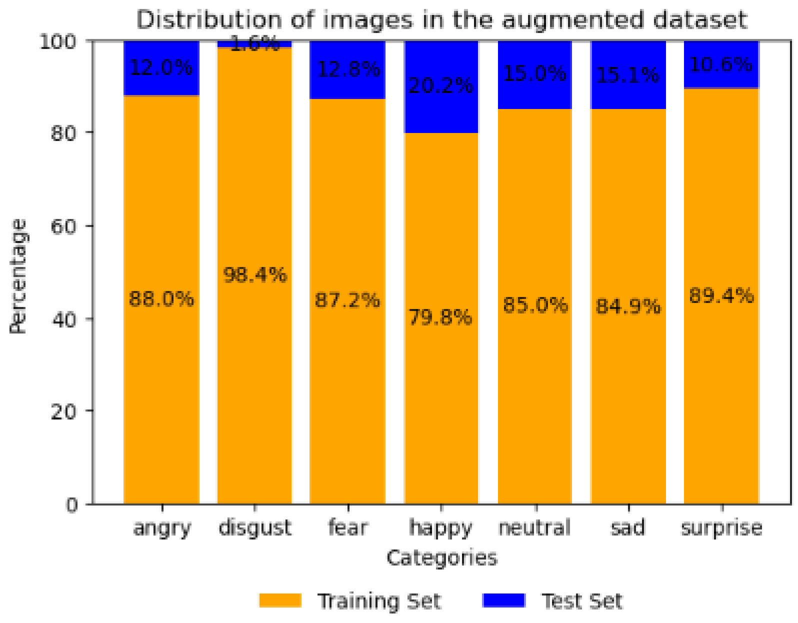


Fig. 6: distribution of the images between the training set and the test set in the augmented dataset

Heterogeneity in image quality due to data preprocessing

When performing data augmentation techniques such as zooming, rotation, and distortion on an original image to create additional images, we include heterogeneity in the quality of the images. For illustration purposes, we have applied different transformations to an original image, which is not part of our dataset and which has a higher resolution (see fig. 7).

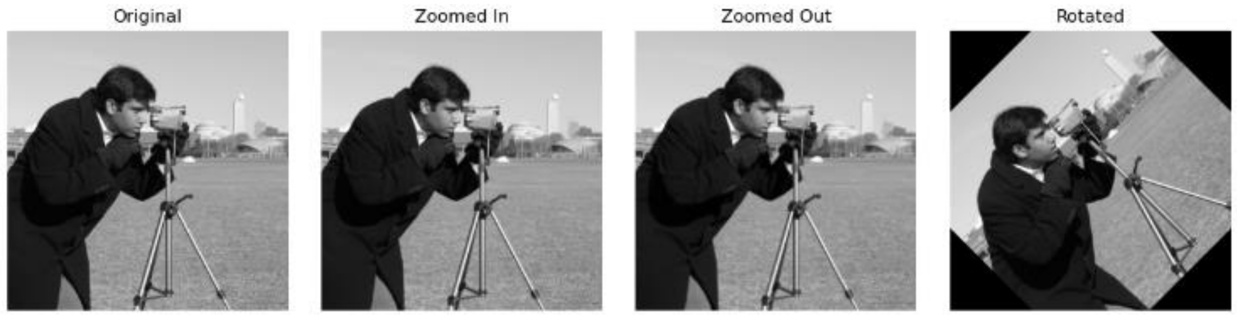


Fig. 7: data augmentation techniques applied on the images

Heterogeneity in the types of images in the FER-2013 dataset

The FER-2013 Dataset contains different types of images:

- *Cartoon images*, which have stylized, exaggerated or simplified visuals.
- *Photography images* in which lighting conditions, backgrounds, and types of people portrayed (gender, age, ethnicity) may vary.
- *Drawing images* which can vary from a range of artistic styles.

The heterogeneity of the different categories can have a decisive impact on the features our model will learn to recognize. A lack of heterogeneity in the data could lead the model to learn incorrect patterns. For example, if the class "happy" class contained only female faces and the class "sad" only male faces, the model would likely predict that most female faces are "happy". It is, however, not possible to have control over such potential biases in the FER-2013 dataset.

6 Deep learning model and analysis

As we are working with images, we built a convolutional neural network (CNN), a type of artificial neural network specifically designed for analyzing visual data as it is able to recognize patterns in images. A CNN has several layers that each learn to identify different features of an image. Filters are applied to each training image and the output of each layer is used as the input to the next layer.

We used 49’000 samples for training (preprocessed training set) and 7’178 samples for testing (original test set). Our CNN model consists of multiple convolutional layers with batch normalization, pooling layers, and dropout regularization. It aims to learn features of the input images and classify them into one of the 7 classes represented by the output layer.

Model architecture

1. The model starts with a Conv2D layer with 32 filters, each having a kernel size of 3x3. The activation function used is ReLU. The input shape of the layer is 48x48x1 as the input images are grayscale with dimensions 48x48 pixels.
2. Batch Normalization is applied after the first Conv2D layer to help normalize the activations of the previous layer.
3. Another Conv2D layer follows with 64 filters and a 3x3 kernel size, using ReLU as the activation function.
4. Batch Normalization is again applied after the second Conv2D layer.
5. MaxPooling2D is added with a pool size of 2x2 to reduce the spatial dimensions of the feature maps.
6. Dropout with a rate of 0.25 is used to randomly deactivate 25% of the neurons during training. This helps prevent overfitting by introducing some level of regularization.

7. The next block consists of two Conv2D layers with 128 filters and 3x3 kernel size each. They are followed by Batch Normalization and MaxPooling2D layers.
8. Another Dropout layer with a rate of 0.25 is added.
9. The subsequent block includes two Conv2D layers with 256 filters and 3x3 kernel size each. They are followed by Batch Normalization and MaxPooling2D layers.
10. Another Dropout layer with a rate of 0.25 is added.
11. The output of the convolutional layers is flattened using the Flatten layer
12. A fully connected Dense layer with 256 neurons and ReLU activation is added.
13. Batch Normalization is applied after the Dense layer.
14. Another Dropout layer with a rate of 0.5 is used.
15. Finally, a Dense layer with 7 neurons and softmax activation is added. The output layer produces probabilities for the 7 different classes in the dataset.

The model is compiled using categorical cross-entropy as the loss function, Adam optimizer with a learning rate of 0.0001, and accuracy as the metric to monitor during training.

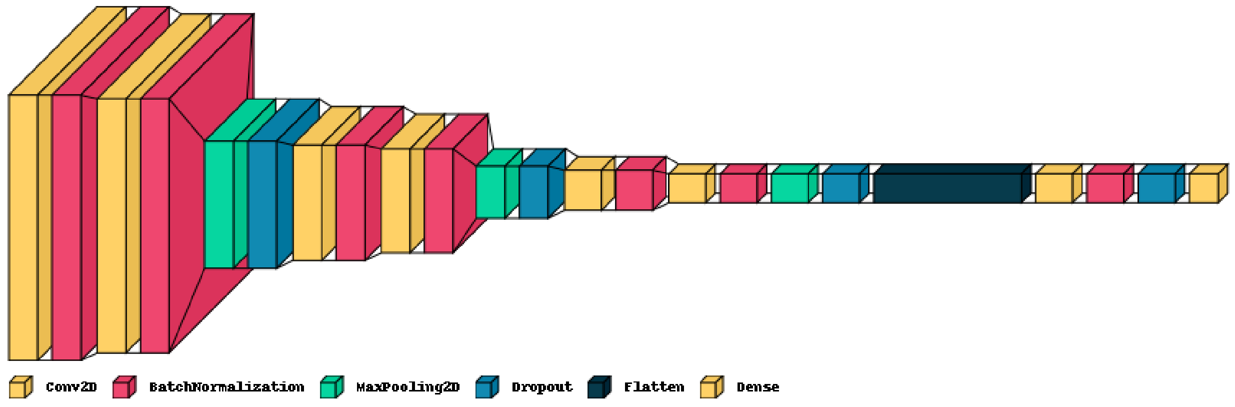
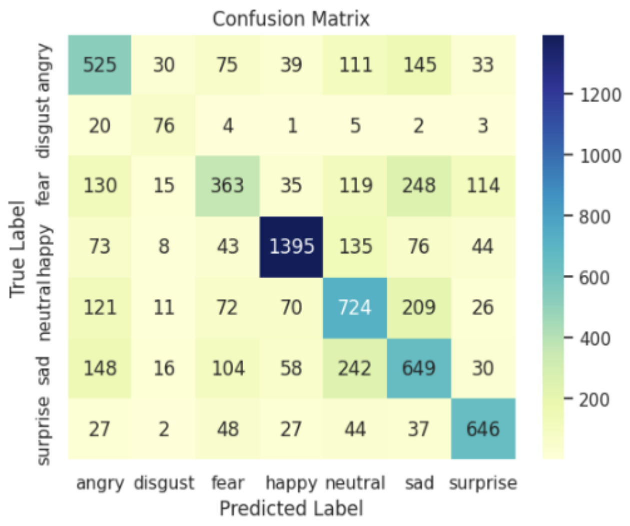


Fig. 8: graphic visualization of our CNN model

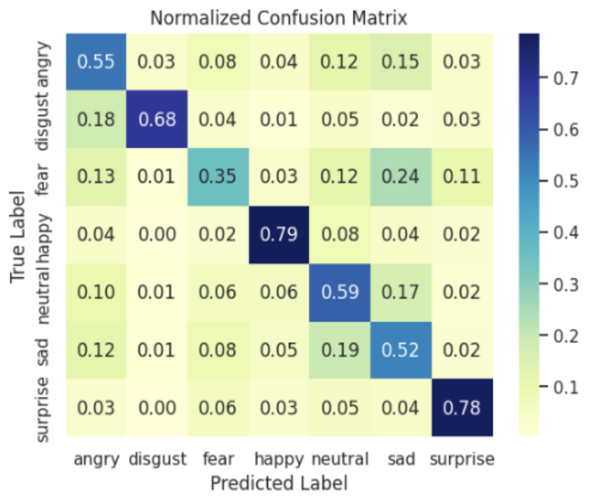
Training and results

We trained our model with a batch size of 32 and 50 epochs. Monitoring the validation accuracy, we used callbacks with a patience of 10, meaning the training would automatically stop if the validation accuracy didn't improve after 10 epochs, and only saved the best model. The training stopped after 38 epochs, as the best model was achieved at epoch 28 with an accuracy of 61%. The classification results of the model are shown with a confusion matrix (see fig. 9 and 10).



True Label \ Predicted Label	angry	disgust	fear	happy	neutral	sad	surprise
angry	525	30	75	39	111	145	33
disgust	20	76	4	1	5	2	3
fear	130	15	363	35	119	248	114
happy	73	8	43	1395	135	76	44
neutral	121	11	72	70	724	209	26
sad	148	16	104	58	242	649	30
surprise	27	2	48	27	44	37	646

Fig. 9: confusion matrix



True Label \ Predicted Label	angry	disgust	fear	happy	neutral	sad	surprise
angry	0.55	0.03	0.08	0.04	0.12	0.15	0.03
disgust	0.18	0.68	0.04	0.01	0.05	0.02	0.03
fear	0.13	0.01	0.35	0.03	0.12	0.24	0.11
happy	0.04	0.00	0.02	0.79	0.08	0.04	0.02
neutral	0.10	0.01	0.06	0.06	0.59	0.17	0.02
sad	0.12	0.01	0.08	0.05	0.19	0.52	0.02
surprise	0.03	0.00	0.06	0.03	0.05	0.04	0.78

Fig. 10: normalized confusion matrix

We can observe that the model achieved good results with the classes “happy” and “surprise” (almost 80% accuracy on both), and to a lesser extent the class “disgust” (68%) while it struggled with the class “fear” (35%) which is misinterpreted as “sad” (24%) or “neutral” (12%) in a significant number of cases. The results are mixed with the classes “angry”, “neutral” and “sad” (respectively 55%, 59% and 52%) as they tend to be confused with one another. We can assume that these results can be explained by the fact that the classes “happy” and “surprise” are more easily identifiable as they are characterized by strong distinctive features, whereas the other classes present more subtle characteristics that require finer interpretation.

After epoch 15, the validation loss starts to increase while the validation accuracy stabilizes at about 60%, suggesting that our model is becoming less confident or more ambiguous in its predictions. The model is visibly overfitting the data, as the much higher training accuracy (90%) confirms it (see fig. 11).

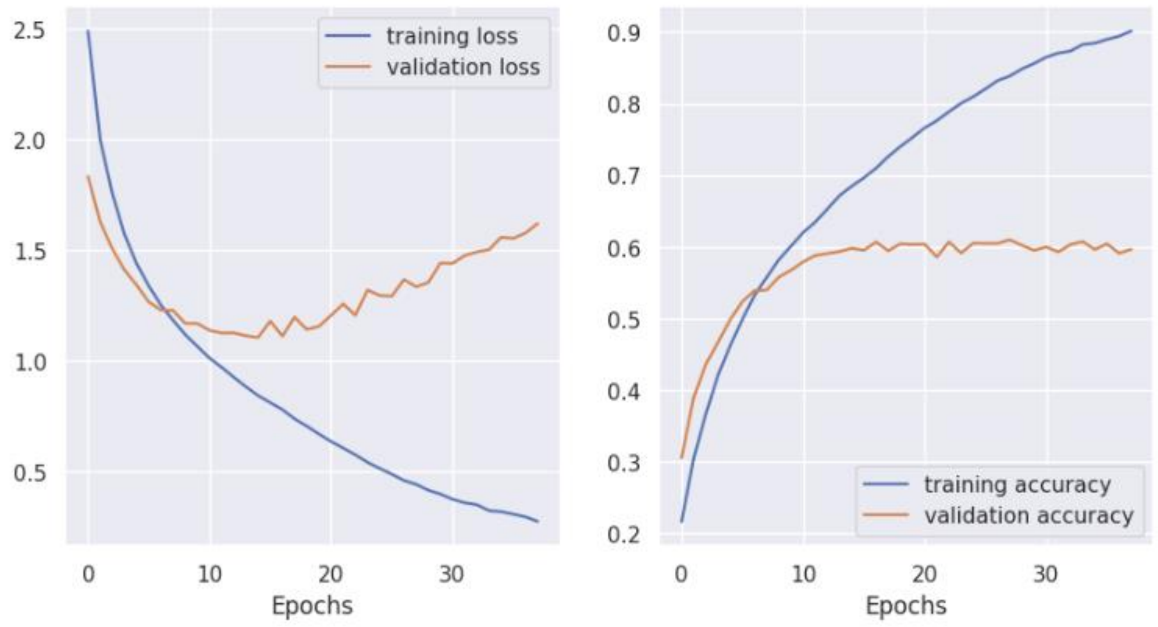


Fig. 11: Model loss and accuracy

Despite these limitations and after several less successful attempts, we consider this model sufficiently accurate to proceed to the next part of the project: the GUI application.

7 GUI application and mapping with emoji

Once the CNN model was completed, we built a GUI application using Tkinter, which provides the GUI framework, and OpenCV, which allows to perform image processing and computer vision tasks[19]. The purpose of the application is to detect faces in a video stream from the webcam, predict the emotions displayed by the detected faces using our CNN model, and map the predicted emotions to corresponding emoji images.

First, we define the paths for the emoji images and the corresponding emotion labels. We set up the video capture from the webcam using OpenCV's VideoCapture function. The video stream is resized to 600x500 pixels. To detect the face in the video, we use the Haar Cascade classifier, which proposes an algorithm that is capable of detecting objects in images, regardless of their location and scale in an image. Furthermore, this algorithm can run in real-time, making it possible to detect objects in video streams. The grayscale version of each frame is passed to the classifier, which returns the bounding boxes of the detected faces. For each detected face,

[19] For the development of the GUI application, we have been guided by a tutorial on https://data-flair.training/

a rectangle is drawn around it on the frame, and the region of interest containing the face is extracted and preprocessed. The region of interest is resized to 48x48 pixels, converted to grayscale, and normalized.
The preprocessed image is fed into our loaded CNN model to predict the emotion. The predicted emotion label is obtained by finding the index of the maximum value in the prediction array and is displayed on the frame. The predicted emotion is then mapped to the corresponding image, which is displayed in the frame next to the video feed.
Finally, when the GUI application is closed, the video capture is released.

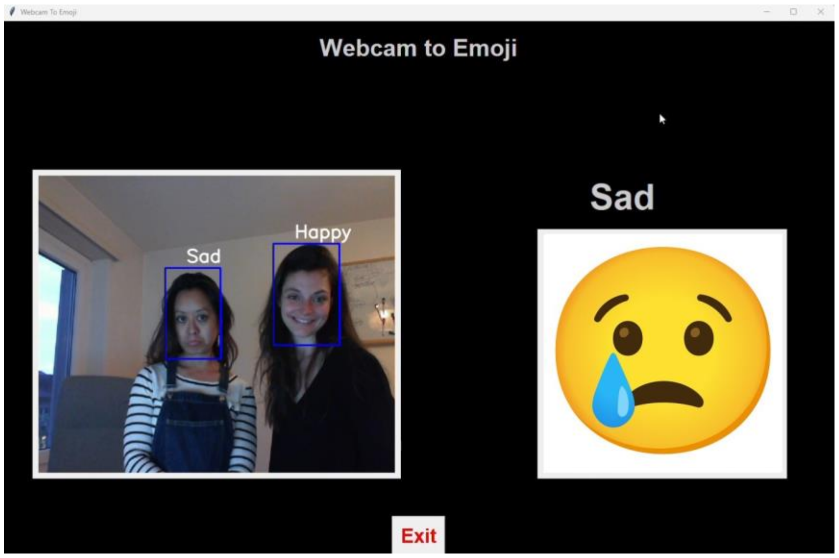


Fig. 12: screenshot of the GUI application, correct prediction of "sad" and "happy"

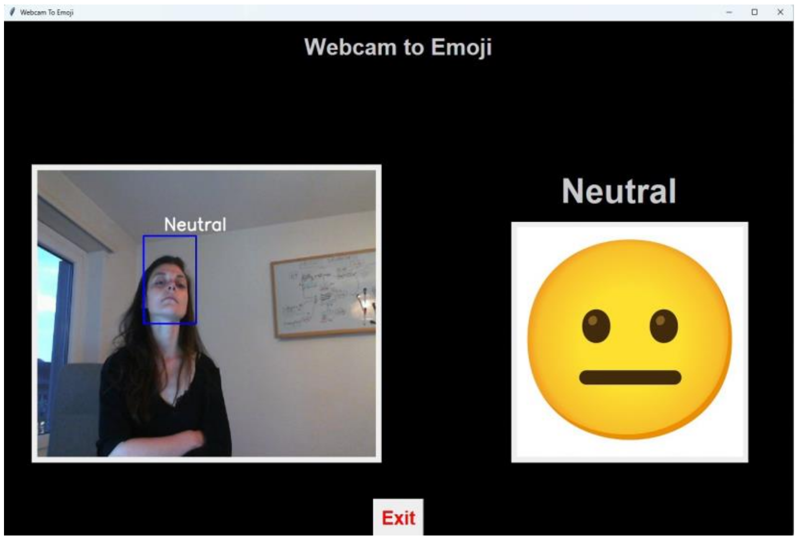


Fig. 13: screenshot of the GUI application, correct prediction of "neutral"

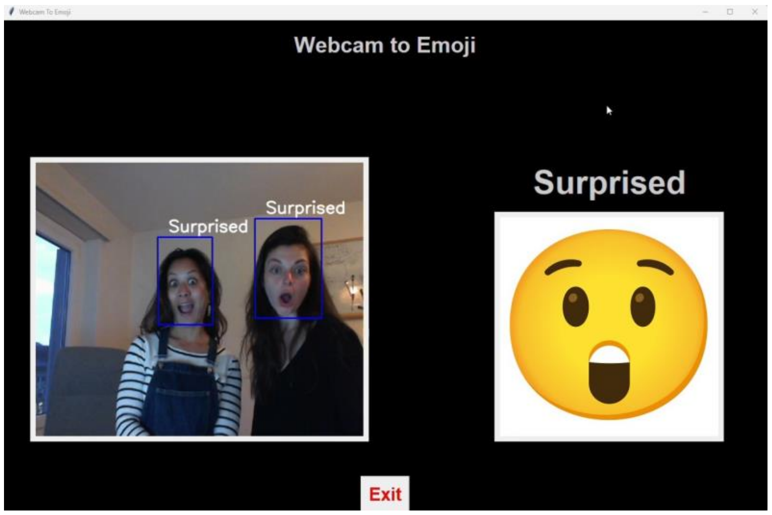


Fig. 14: screenshot of the GUI application, correct prediction of "surprised"

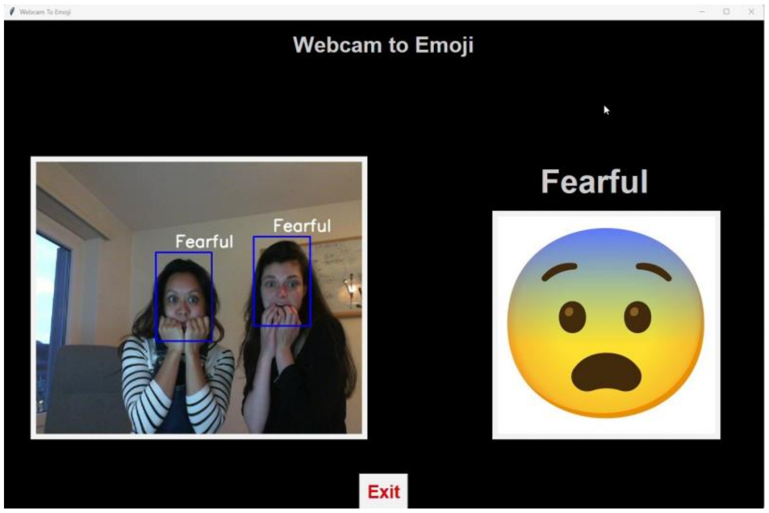


Fig. 15: screenshot of the GUI application, correct prediction of "fearful"

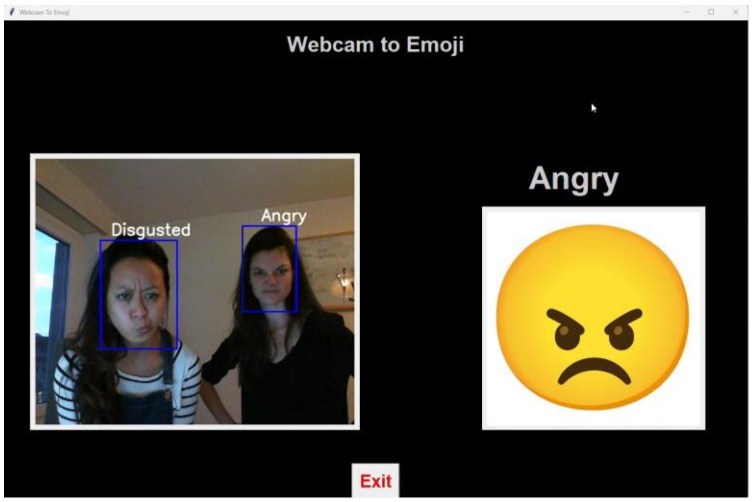


Fig. 16: screenshot of the GUI application, correct prediction of "angry" and "disgusted"

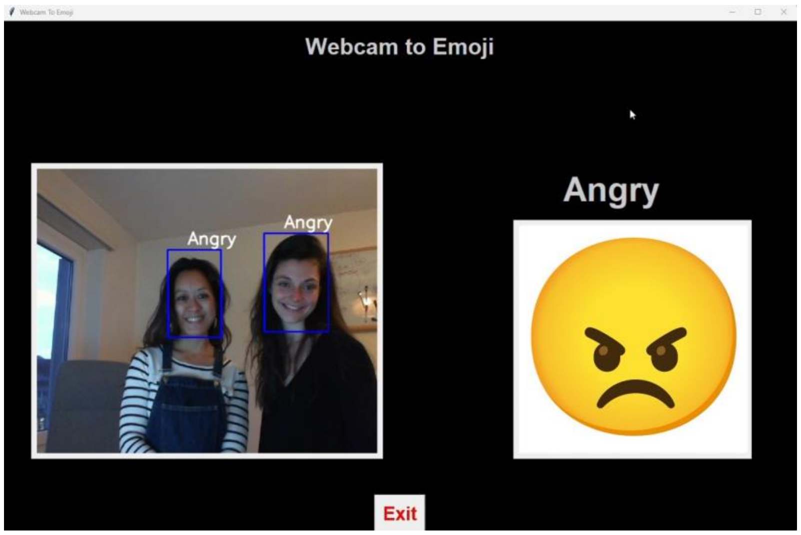


Fig. 17: screenshot of the GUI application, wrong prediction of "angry" (true label: "happy")

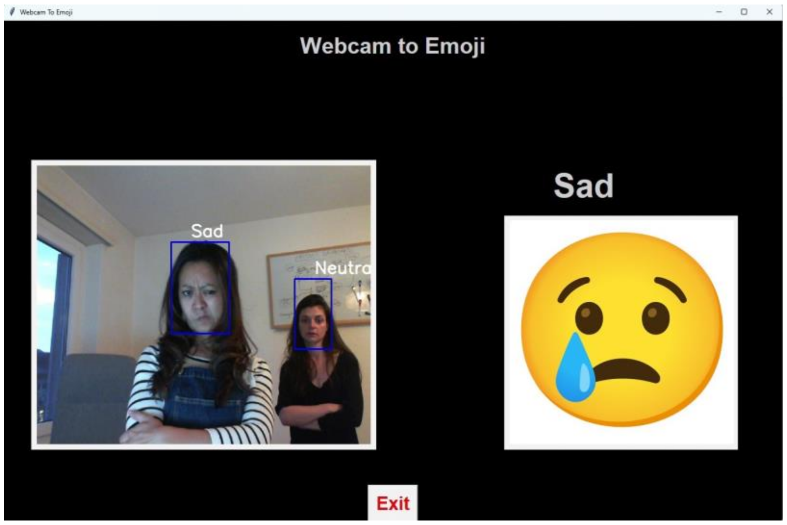


Fig. 18: screenshot of the GUI application, wrong prediction of "sad" (true label: "angry")

8 Results discussion

After several trials, we settled on a model that achieved 61% accuracy, which we consider acceptable, in light of previous research on the same dataset. A paper published in 2020 indeed claimed that "[the FER 2013 dataset] is one of the more challenging datasets with human-level accuracy only at 65±5% and the highest performing published works achieving 75.2% test accuracy" [3].

The implementation of the model in the GUI application gives rather satisfactory results, with the emotions demonstrated being fairly well recognized (see fig. 12-16). However, the model shows its limitations when emotions that are very clear to the human eye are sometimes incorrectly predicted (see fig. 17-18), reflecting the fallouts shown in the confusion matrix. We can imagine several reasons for this. On one hand, an accuracy of 61%, while much better than pure chance, is still limited and cannot give reliable results. On the other hand, it is not always obvious either for the human eye to detect and classify emotions. Some images are ambiguous and can be easily classified wrongly by a human. The classification of the images in the dataset itself can be questioned. A closer look shows that it is for example not that easy to distinguish a "sad" expression from an "angry" one or a "surprised" one from a "fearful" one (see fig. 19).

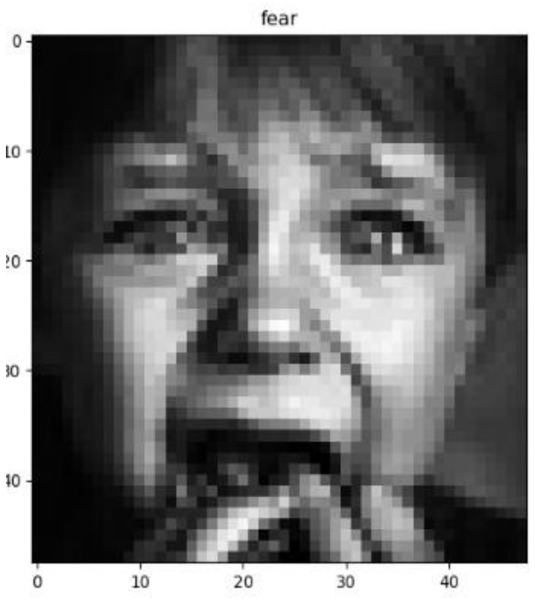


Fig. 19: a potentially ambiguous sample labeled as "fear"

This leads to another question: how can we ensure that we actually have a correctly labeled dataset ? We can't and we need to trust that the primary contributors and authors of the FER-2013 dataset, Pierre-Luc Carrier and Aaron Courville, did their best in labeling the dataset while keeping in mind that there might be some debatable or inaccurate samples [4].

9 Conclusion

In this project, we tried to build a CNN model using an open image dataset to predict emotions and implement this model in a GUI application to recognize a user's facial expression on a webcam feed and map it with an emoji. We obtained satisfying results although there is much room for development and improvement.

Limitations imposed by our dataset

The FER-2013 dataset is built with single categorical labels which reflect a limited scope of basic emotions. However, it seems a little simplistic to assume that only one single emotion is felt or displayed at a time. As humans, we are prone to experiment mixed feelings. Maybe it would be more appropriate to feed the model with labels indicating a combination of different emotions displayed instead of a unique emotion. Of course the FER-2013 dataset does not provide that level of detail.

Limitations of our CNN model

Considering the limited time resources we had to build and train our model, there are still many avenues left open to explore in order to improve the model. We attempted to work on a subset to speed up the training and reduce computation time but this resulted in lower accuracy and we quickly dropped that option. To address the overfitting issue of our model, we could consider reducing the complexity of the model by reducing the number of layers or the number of filters in each layer, or reducing the learning rate.
Transfer learning, the use of pretrained models, would be another interesting option to consider as it allows for significant time and resource savings since only a portion of the model needs to be retrained or fine-tuned. There are several pretrained models available that have been trained on large-scale datasets for emotion recognition from images. These models can be used as a starting point and then fine-tuned on a specific dataset to improve their performance. Instead of training a model from scratch, transfer learning leverages the knowledge learned by a pre-trained model on a different but related task. Numerous studies have used transfer learning on facial emotion recognition [5].

General and philosophical considerations

What level of performance can be expected from a machine learning model when asked to predict an emotion from an image, which implies having the ability to feel, express and interpret emotions ? Some images are ambiguous and could be easily classified wrongly by a human. Can we actually expect a model to outperform human in deciphering emotions?

Thanks to the GUI application, we actually had the opportunity of testing our model on ourselves. It appeared that when confronted with some very distinctive signs such as a round open mouth, the model tends to predict a "surprised" emotion. Are we just teaching the model to recognize some very obvious characteristics such as "a smile", "wide open eyes", "frowned eyebrows", etc. Even though those characteristics are not completely uncorrelated with emotions, they have little in common with true emotions.

One way to potentially obtain better results would be to label the dataset with clearly identifiable facial characteristics instead of emotions. We could then predict characteristics facial traits and expressions and map them to the corresponding emotions based on some defined rules (for instance: a smile = "happy").

10 Acknowledgements

This report serves as the final project of the CAS in Applied Data Science of the University of Bern. We would like to acknowledge and thank the program management and the lecturers of the CAS for their time, their interest in our project and their support.

11 References and Bibliography

[1] European Data Protection Supervisor, K. Vemou, A. Horvath, and T. Zerdick. 2021. EDPS TechDispatch : facial emotion recognition. Issue 1, 2021.

[2] M. Quinn, G. Sivesind, G. Reis, "Real-time Emotion Recognition From Facial Expressions", CS 229 - Stanford University

[3] A. Khanzada, C. Bai, F. Celepcikay, "Facial Expression Recognition with Deep Learning", arXiv: 2004.11823, 2020.
https://arxiv.org/ftp/arxiv/papers/2004/2004.11823.pdf

[4] Wolfram Research, "FER-2013" from the Wolfram Data Repository (2018)
https://datarepository.wolframcloud.com/resources/FER-2013

[5] J.C. Hung et al., "Multi-level transfer learning for improving the performance of deep neural networks: theory and practice from the tasks of facial emotion recognition and named entity recognition", Applied Soft Computing, Volume 109, September 2021.