



A real-time system for facial expression recognition

Emanuele Alessi (1486470)

Nicole Colace (1646808)

Project of the Machine Learning course
Academic year 2018-2019

Summary

1. Introduction	3
2. Problem definition	4
3. Our objective	6
4. Literature study	7
5. FER2013 dataset	9
6. Project setup	11
6.1 TensorFlow	11
6.2 OpenCV	12
7. Data pre-processing	13
8. Data augmentation	15
9. Implementation of our model	17
10. Train and test results	23
11. Testing the model (instruction for the use)	27
12. Conclusions	28
13. References	29

1. Introduction

“2018 is the year when machines learn to grasp human emotions” (Andrew Moore, the dean of computer science at Carnegie Mellon).

With the advent of modern technology our desires went high and it binds no bounds. In the present era a huge research work is going on in the field of digital image and image processing. The way of progression has been exponential and it is ever increasing. Image Processing is a vast area of research in present day world and its applications are very widespread.

Image processing is the field of signal processing where both the input and output signals are images. One of the most important application of Image processing is Facial expression recognition. Our emotion is revealed by the expressions in our face. Facial Expressions plays an important role in interpersonal communication. Facial expression is a non-verbal scientific gesture which gets expressed in our face as per our emotions. Automatic recognition of facial expression plays an important role in artificial intelligence and robotics and thus it is a need of the generation. Some application related to this include Personal identification and Access control, Videophone and Teleconferencing, Forensic application, Human-Computer Interaction, Automated Surveillance, Cosmetology and so on.

The objective of our project is to develop a real-time facial expression recognition system which can take human facial images containing some expression as input and recognize and classify it into seven different expression class such as:



Figure 1 - Sample of images from FER2013 dataset

2. Problem definition

Human facial expressions can be easily classified into 7 basic emotions: happy, sad, surprise, fear, anger, disgust, and neutral. Our facial emotions are expressed through activation of specific sets of facial muscles. These sometimes subtle, yet complex, signals in an expression often contain an abundant amount of information about our state of mind.

Through facial emotion recognition, we are able to measure the effects that content and services have on the audience/users through an easy and low-cost procedure. For example, retailers may use these metrics to evaluate customer interest. Healthcare providers can provide better service by using additional information about patients' emotional state during treatment. Entertainment producers can monitor audience engagement in events to consistently create desired content.

Humans are well-trained in reading the emotions of others, in fact, at just 14 months old, babies can already tell the difference between happy and sad. But can computers do a better job than us in accessing emotional states? To answer the question, we designed a deep learning neural network that gives machines the ability to make inferences about our emotional states. In other words, we give them eyes to see what we can see.

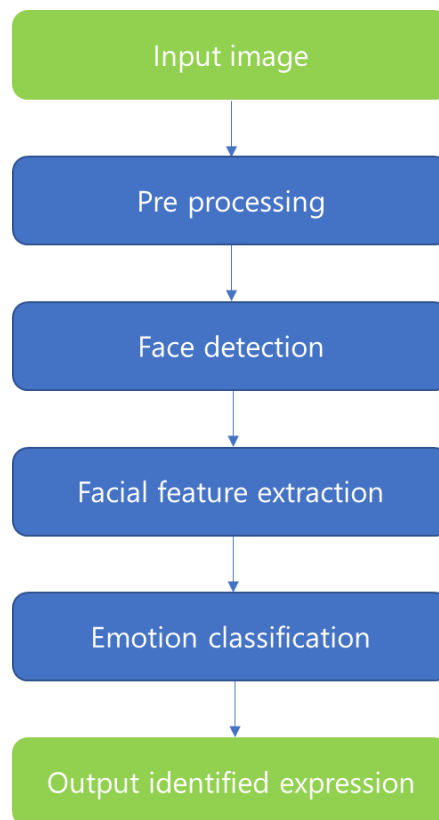


Figure 2 - Problem formulation of our project

Facial expression recognition is a process performed by humans or computers, which consists of:

1. Locating faces in the scene (e.g., in an image; this step is also referred to as face detection)
2. Extracting facial features from the detected face region (e.g., detecting the shape of facial components or describing the texture of the skin in a facial area; this step is referred to as facial feature extraction)
3. Analyzing the motion of facial features and/or the changes in the appearance of facial features and classifying this information into some facial-expression interpretative categories such as facial muscle activations like smile or frown, emotion (affect) categories like happiness or anger, attitude categories like (dis)liking or ambivalence, etc. (this step is also referred to as facial expression interpretation).

3. Our objective

After several searches on the web regarding the papers dealing with the topic of face detection and facial expression recognition, we have concluded that all the papers that we found encourage to use deep learning neural networks in order to classify as good as possible facial expressions.

Then our task will be focused on the design and development of a deep learning model that is able to compute in real-time:

1. Face detection
2. Facial expression recognition

and possibly make the project easily portable not only on desktop but also on smartphones and IoT.

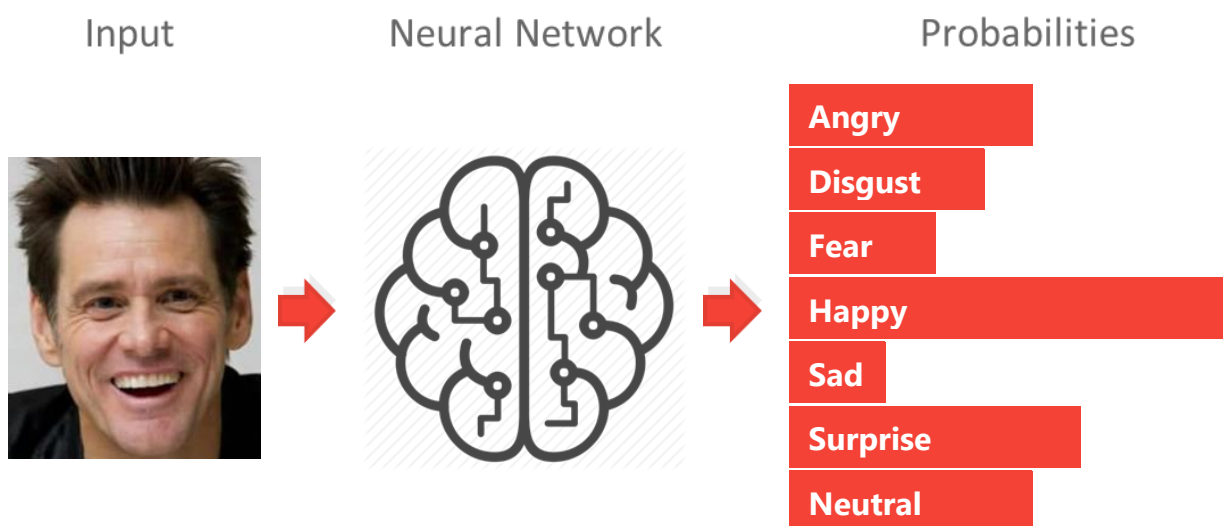


Figure 3 - A high-level idea of our deep learning model

4. Literature study

Before proceeding with the project description, we want to specify that our implementation is based on the paper "Real-time Convolutional Neural Networks for Emotion and Gender Classification" (2017) written by Octavio Arriaga, Paul G. Ploger and Matias Valdenegro.

The authors proposed a state-of-the-art deep learning model known as mini-Xception to perform facial expression recognition, the implementation have been validated in a real-time facial expression system that provides face-detection, gender classification and that achieves human-level performance when classifying emotions with an accuracy score of 66%.

More precisely, the proposed model is a Neural Network composed of a sequence of hidden layers, where convolution operations, batch normalization and pooling, are performed within them, while the output layer is represented by a softmax classifier.

Considering the complexity of our project we are forced to focus on a deep learning oriented approach, considering, in this case, not very effective the use of machine learning algorithms, as some surveys explained that they would not reach an acceptable accuracy value, because of the difficulty in correctly distinguishing the expressions that may seem similar to each other but that really are different (just to give an example, the disgusted expression is very similar to the angry expression, or the frightened expression is often mistaken for other expressions).

Regarding the choice of the dataset, nowadays we have a rather small number of datasets for emotion recognition, however we have found that many of them are not available online, therefore we have only taken into consideration the public datasets comparing them from the viewpoints of the number of samples, of the variance of the images, of the head poses and, above all, of the number of labels (emotions).

An overview of the facial expression datasets. P = posed; S = spontaneous; Condit. = Collection condition; Elicit. = Elicitation method.

Database	Samples	Subject	Condit.	Elicit.	Expression distribution	Access
CK+ [33]	593 image sequences	123	Lab	P & S	6 basic expressions plus contempt and neutral	http://www.consortium.ri.cmu.edu/ckagree/
MMI [34], [35]	740 images and 2,900 videos	25	Lab	P	6 basic expressions plus neutral	https://mmifacedb.eu/
JAFFE [36]	213 images	10	Lab	P	6 basic expressions plus neutral	http://www.kasrl.org/jaffe.html
TFD [37]	112,234 images	N/A	Lab	P	6 basic expressions plus neutral	josh@mplab.ucsd.edu
FER-2013 [21]	35,887 images	N/A	Web	P & S	6 basic expressions plus neutral	https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge
AFEW 7.0 [24]	1,809 videos	N/A	Movie	P & S	6 basic expressions plus neutral	https://sites.google.com/site/emotiwatchallenge/
SFEW 2.0 [22]	1,766 images	N/A	Movie	P & S	6 basic expressions plus neutral	https://cs.anu.edu.au/few/emotiwatchallenge.html
Multi-PIE [38]	755,370 images	337	Lab	P	Smile, surprised, squint, disgust, scream and neutral	http://www.flinthbox.com/public/project/4742/
BU-3DFE [39]	2,500 images	100	Lab	P	6 basic expressions plus neutral	http://www.cs.binghamton.edu/~lijun/Research/3DFE/3DFE_Analysis.html
Oulu-CASIA [40]	2,880 image sequences	80	Lab	P	6 basic expressions	http://www.cse.oulu.fi/CMV/Downloads/Oulu-CASIA
RaFD [41]	1,608 images	67	Lab	P	6 basic expressions plus contempt and neutral	http://www.socsci.ru.nl:8180/RaFD2/RaFD
KDEF [42]	4,900 images	70	Lab	P	6 basic expressions plus neutral	http://www.emotionlab.se/kdef/
EmotionNet [43]	1,000,000 images	N/A	Web	P & S	23 basic expressions or compound expressions	http://cbcs1.ece.ohio-state.edu/dbform_emotionet.html
RAF-DB [44], [45]	29672 images	N/A	Web	P & S	6 basic expressions plus neutral and 12 compound expressions	http://www.whdeng.cn/RAF/model1.html
AffectNet [46]	450,000 images (labeled)	N/A	Web	P & S	6 basic expressions plus neutral	http://mohammadmahoor.com/databases-codes/
ExpW [47]	91,793 images	N/A	Web	P & S	6 basic expressions plus neutral	http://mmlab.ie.cuhk.edu.hk/projects/socialrelation/index.html

Figure 4 - List of all available dataset for emotion detection taken from the paper "Deep Facial Expression Recognition: A Survey"

After a preliminary study we chose to use FER2013 as a dataset to train and test our model, because we considered it the best compromise between the canons of choice just mentioned.

Other details will be explained more accurately in the next paragraphs.

5. FER2013 dataset

The FER2013 is an open-source dataset which was introduced during the International Conference on Machine Learning (ICML 2013), Challenges in Representation Learning. FER2013 is a large-scale and unconstrained database collected automatically by the Google image search API. All images have been registered and resized to 48*48 pixels after rejecting wrongfully labeled frames and adjusting the cropped region.



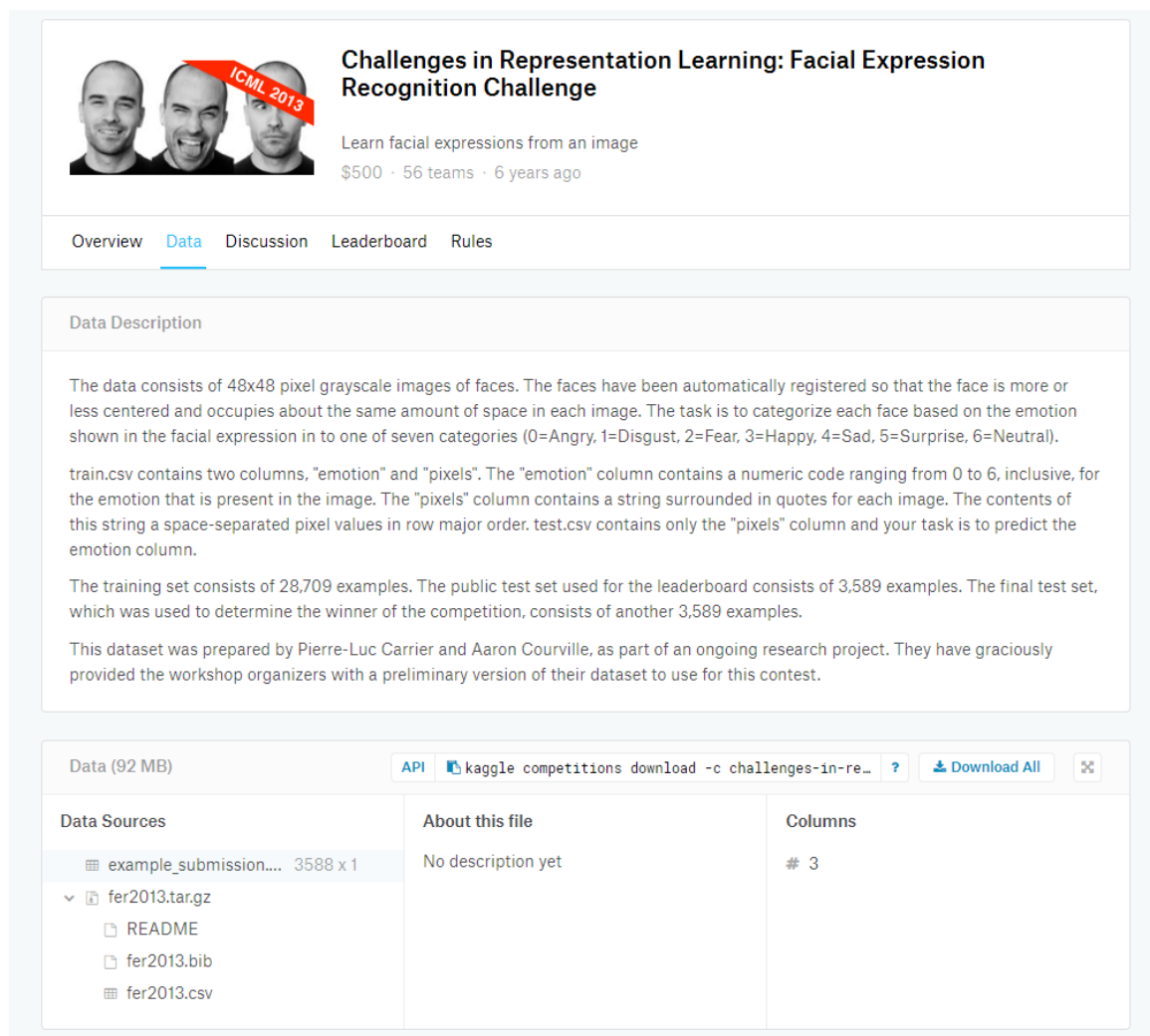
Figure 5 - Example images from FER2013 dataset

This dataset was created for an ongoing project by Pierre-Luc Carrier and Aaron Courville, then shared publicly for a Kaggle competition, shortly before ICML 2013.

FER2013 consists of about 35000 greyscale images, with seven expression labels (anger, disgust, fear, happiness, sadness, surprise and neutral).

The emotions distribution is defined as following:

Labels	# of images
Angry	4593
Disgust	547
Fear	5121
Happy	8989
Sad	6077
Surprise	4002
Neutral	6198



The screenshot shows the Kaggle competition page for "Challenges in Representation Learning: Facial Expression Recognition Challenge". At the top, there are three small images of a man's face with different expressions, a red "ICML 2013" banner, the competition title, a subtitle "Learn facial expressions from an image", and a prize of "\$500 · 56 teams · 6 years ago". Below this is a navigation bar with "Overview", "Data", "Discussion", "Leaderboard", and "Rules". The "Data" tab is selected, showing a "Data Description" section. The description states that the data consists of 48x48 pixel grayscale images of faces, categorized into seven emotion classes (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). It details the training set (28,709 examples), the public test set (3,589 examples), and the final test set (3,589 examples). It also mentions the dataset was prepared by Pierre-Luc Carrier and Aaron Courville. At the bottom, there is a "Data (92 MB)" section with a "Data Sources" table, an "About this file" section, and a "Columns" section. The "Data Sources" table lists "example_submission...." (3588 x 1) and "fer2013.tar.gz" (which contains "README", "fer2013.bib", and "fer2013.csv"). The "About this file" section says "No description yet". The "Columns" section shows "# 3".

Challenges in Representation Learning: Facial Expression Recognition Challenge

Learn facial expressions from an image

\$500 · 56 teams · 6 years ago

Overview **Data** Discussion Leaderboard Rules

Data Description

The data consists of 48x48 pixel grayscale images of faces. 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. The task is to categorize each face based on the emotion shown in the facial expression in to one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

train.csv contains two columns, "emotion" and "pixels". The "emotion" column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image. The "pixels" column contains a string surrounded in quotes for each image. The contents of this string a space-separated pixel values in row major order. test.csv contains only the "pixels" column and your task is to predict the emotion column.

The training set consists of 28,709 examples. The public test set used for the leaderboard consists of 3,589 examples. The final test set, which was used to determine the winner of the competition, consists of another 3,589 examples.

This dataset was prepared by Pierre-Luc Carrier and Aaron Courville, as part of an ongoing research project. They have graciously provided the workshop organizers with a preliminary version of their dataset to use for this contest.

Data (92 MB) [API](#) [kaggle competitions download -c challenges-in-re...](#) [Download All](#)

Data Sources	About this file	Columns
<ul style="list-style-type: none"> example_submission.... 3588 x 1 fer2013.tar.gz <ul style="list-style-type: none"> README fer2013.bib fer2013.csv 	No description yet	# 3

Figure 6 - Kaggle page where we can download FER2013

6. Project setup

Let us know enlist the technologies that we used while working on this project (in order of importance):

- Python 3.6.x
- TensorFlow 1.13 (used to create, train and test our neural network)
- OpenCV (used to do face detection)
- NumPy (used for managing the dataset and for calculating metrics)
- Matplotlib (used to generate plots of training and test results)
- Scikit-learn (used only to divide the dataset into train set and test set)
- Tqdm (used for printing a progress bar of the neural network training process)
- PyCharm as our IDE

6.1 TensorFlow

TensorFlow is probably the most famous framework for working out any large-scale Machine Learning: originally created by the *Google Brain Team*, it is an open-source library which bundles mainly Deep Learning models and algorithms.

The library can train and run Deep Neural Networks for many tasks, ranging from digit classification to image recognition.

But how does it work?

TensorFlow allows the creation of so-called “dataflow graphs”; structures that describe how data moves through a graph. Here:

- A node represents a mathematical operation;
- An edge between two nodes symbolize a “Tensor” (short for multidimensional array).

The nodes, though, are not executed in Python: to ensure a higher speed of computation, in fact, the library executes these operations in C++, so that they can be worked out at low-level.



Figure 7 - TensorFlow logo

Another great advantage is that the developer can choose to execute calculations either on the CPU or the GPU, to ensure more computational power to the program.

As of 2019, TensorFlow is accredited as one of the most used libraries for Deep Learning and it keeps growing, even with a recent release for JavaScript.

As we felt that TensorFlow was what we needed for this task (since it is more powerful than other machine learning frameworks, such as Keras, PyTorch, Caffe etc...), we decided to abandon the advantage of having less and more concise code lines in favor of more computational power.

The main reason why we chose TensorFlow is, in addition to the one already mentioned, that the deep learning model is easily exportable for use on desktops, smartphones and IoT (using TensorFlow Lite).

For this reason, we will not list a code example here as our project was entirely made with TensorFlow.

6.2 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, etc...

OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection and it has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million.

The library is used extensively in companies, research groups and by governmental bodies.

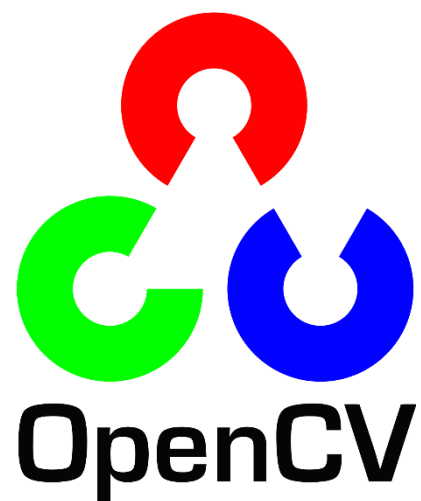


Figure 8 - OpenCV logo

7. Data pre-processing

Before we can work on the dataset, we must load it in memory and apply some pre-processing techniques.

Our procedure consists of initializing two empty lists (one used for training the model, and the other one will be used for testing purposes) that will contain all the images in the dataset.

We decided to use 80% of the images for training the model, while keeping the remaining 20% for the testing phase. Images used for training and testing are randomly picked from the available images of FER2013, using the *train_test_split* function of scikit-learn library.

After several attempts we decided to apply a procedure by which we normalize the images (represented as matrices) so as to transform them into matrices whose values are in the range $[-1, 1]$. Once all the images are pre-processed, as stated above, many of them will be picked for the training phase, while the others will be used for testing the model.

The pre-processing algorithm can be summarized with the following pseudocode:

```
1. dataset = empty list
2. for each image i of FER2013 do:
3.     load i in memory as a matrix with shape 48*48
4.     normalize i dividing each value of the matrix by 255
5.     subtract each value of the normalized matrix i with 0.5
6.     multiply each value of the resulting matrix i by 2
7.     dataset.append(i)
8. train_set, test_set = train_test_split(dataset, test_size=20%)
```

this technique with which we transform images has allowed us to increase the accuracy of our final model by 4%.

After the splitting phase we generated the following plots for the samples distribution of both train set and test set.

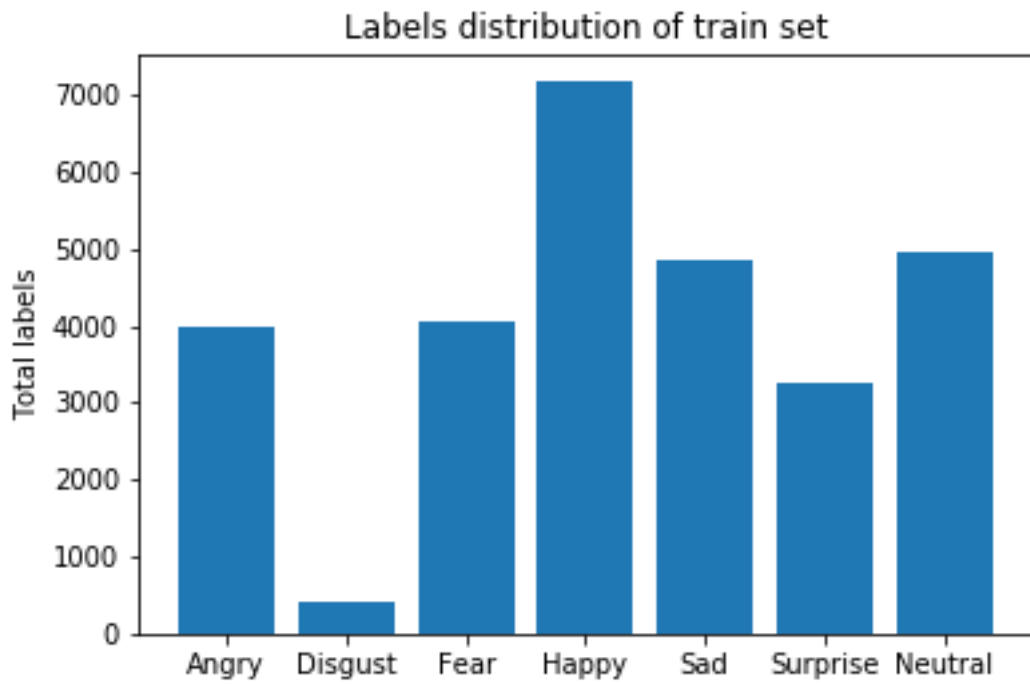


Figure 9 – Samples distribution of train set

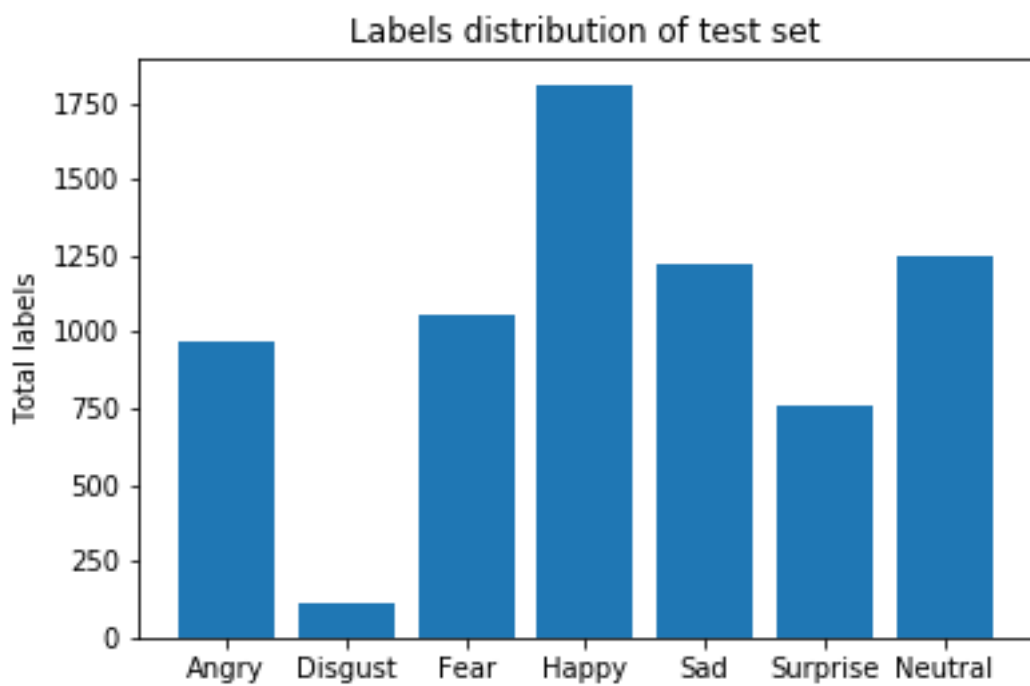


Figure 10 – Samples distribution of test set

8. Data augmentation

The performance of deep learning neural networks often improves with the amount of data available.

Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples.

Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image.

Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, and much more.

The intent is to expand the training dataset with new, plausible examples. This means, variations of the training set images that are likely to be seen by the model.

As such, it is clear that the choice of the specific data augmentation techniques used for a training dataset must be chosen carefully and within the context of the training dataset and knowledge of the problem domain. In addition, it can be useful to experiment with data augmentation methods in isolation and in concert to see if they result in a measurable improvement to model performance, perhaps with a small prototype dataset, model, and training run.

Modern deep learning algorithms, such as the convolutional neural network, or CNN, can learn features that are invariant to their location in the image. Nevertheless, augmentation can further aid in this transform invariant approach to learning and can aid the model in learning features that are also invariant to transforms such as left-to-right to top-to-bottom ordering, light levels in photographs, and more.

We applied data augmentation, through the TensorFlow function *ImageDataGenerator*, only to the training dataset, and not to the test dataset.

For our purposes we decided to augment the train set with:

- Rotation
- Width shift
- Height shift
- Zoom
- Horizontal flip



Figure 11 - An example of how data augmentation is applied on a given image

With the data augmentation technique, our final model improved its predictive ability gaining about 4%-5% more accuracy.

9. Implementation of our model

Deep learning is a popular technique used in computer vision. We chose Convolutional Neural Network (CNN) layers as building blocks to create our model architecture. CNNs are known to imitate how the human brain works when analyzing visuals.

For our purposes we propose a mini-Xception neural network, which is a state-of-the-art model for the real-time facial expression recognition.

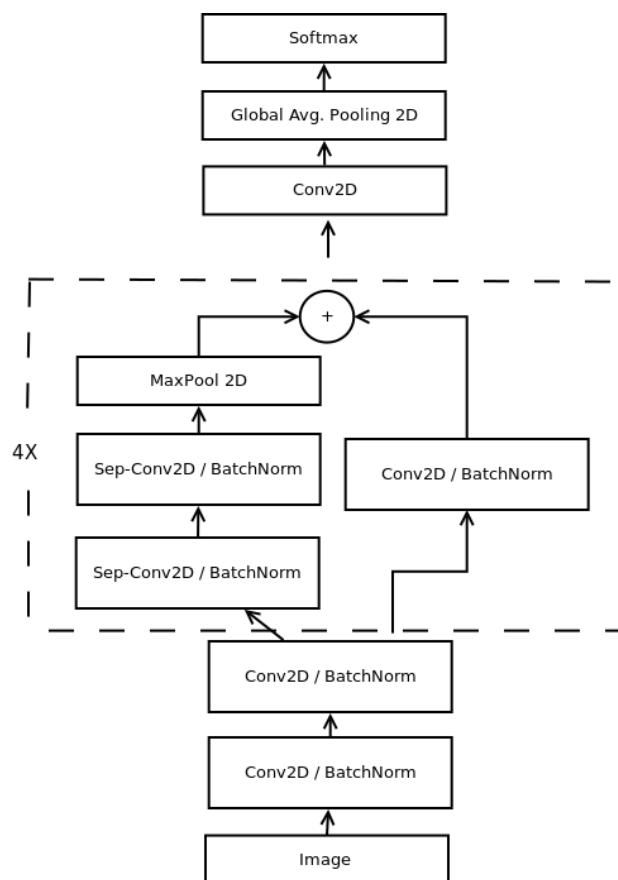


Figure 12 - mini-Xception architecture for emotion classification

Our model was implemented totally from scratch with TensorFlow library. This architecture is different from the most common CNN architectures, since they use fully connected layers at the end where most of parameters resides. Also, they use standard convolutions. Modern CNN architectures such as Xception (proposed by Francois Chollet in the paper "Xception: Deep Learning with Depthwise Separable Convolutions") leverage from the combination of two of the most successful experimental assumptions in CNNs: the use of residual modules and depth-wise separable convolutions.

There are various techniques that can be kept in mind while building a deep neural network and is applicable in most of the computer vision problems. Below are few of those techniques which are used while training the mini-Xception model below.

1. Data augmentation: as mentioned in [paragraph 8](#).
2. Kernel regularization: it allows to apply penalties on layer parameters during optimization. These penalties are incorporated in the loss function that the network optimizes. Argument in convolution layer is nothing but L2 regularization of the weights. This penalizes peaky weights and makes sure that all the inputs are considered.

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2 + \lambda \sum_{i=1}^n \theta_i^2$$

Figure 13 - Regularization term (in red box) with lambda fixed to 0.01

3. Batch normalization: it normalizes the activation of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. It addresses the problem of internal covariate shift. It also acts as a regularizer, in some cases eliminating the need for Dropout. It helps in speeding up the training process.
4. ReLU (Rectified Linear Unit) activation function: it is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning, thanks also to its derivative that is effective for the vanishing gradient problem

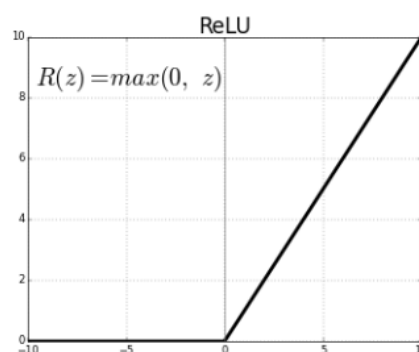


Figure 14 - Graphic representation of the ReLU activation function

- Global Average Pooling: it reduces each feature map into a scalar value by taking the average over all elements in the feature map. The average operation forces the network to extract global features from the input image.

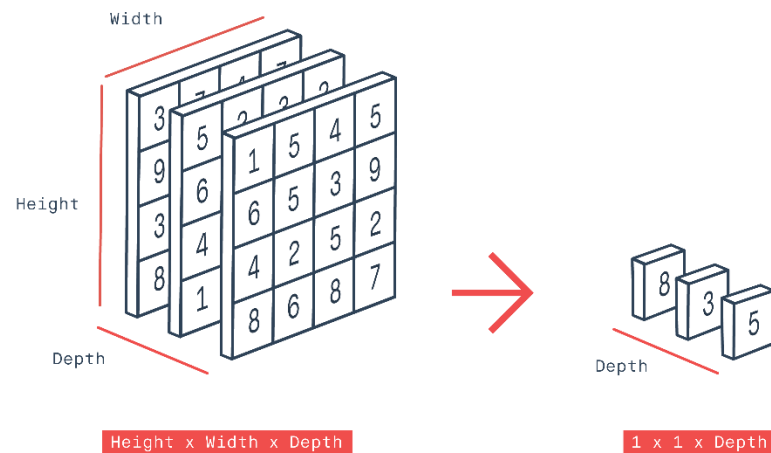
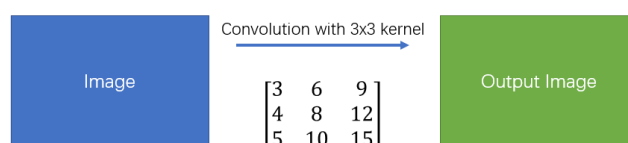


Figure 15 - An example of how global average pooling is performed

- Depthwise Separable Convolution: these convolutions are composed of two different layers: depth-wise convolutions and point-wise convolutions. Depth-wise separable convolutions reduce the computation with respect to the standard convolutions by reducing the number of parameters.

Simple Convolution



Spatial Separable Convolution

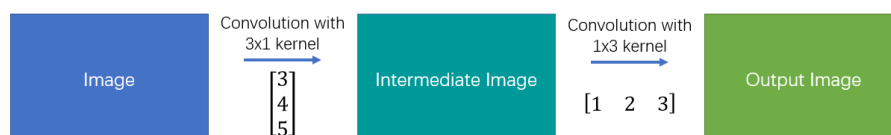


Figure 16 - Difference between simple convolution and separable convolution

7. Learning rate decay: the learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. The learning rate may be the most important hyperparameter when configuring the neural network. Therefore it is vital to know how to investigate the effects of the learning rate on model performance and to build an intuition about the dynamics of the learning rate on model behavior. During the training phase, we noticed that after some epochs, it was necessary to decrease the learning rate in proportion to the number of epochs. For this reason, considering that our learning rate is initially set to 0.001, we used the exponential decay technique, with which, after each epoch, the learning is updated with the following formula (given $\eta^{(t)}$ that is the learning rate at epoch t):

$$\eta^{(t+1)} = \eta^{(t)} * 0.95$$

Figure 17 - Exponential decay formula

8. Adam Optimizer: deep learning neural networks are trained using the stochastic gradient descent optimization algorithm. Adam is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks and a lot of research has been done to address the problems of Adam. Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum. In this optimization algorithm, running averages of both the gradients and the second moments of the gradients are used. Given parameters $w^{(t)}$ and a loss function $L^{(t)}$, where t (initially set to 0) is the current training iteration, Adam's parameter update is given by:

$$\begin{aligned} 1. \quad & m_w^{(t+1)} = \beta_1 m_w^{(t)} + (1 - \beta_1) \nabla_w L^{(t)} \\ 2. \quad & v_w^{(t+1)} = \beta_2 v_w^{(t)} + (1 - \beta_2) (\nabla_w L^{(t)})^2 \\ 3. \quad & \hat{m}_w = \frac{m_w^{(t+1)}}{1 - (\beta_1)^{t+1}} \\ 4. \quad & \hat{v}_w = \frac{v_w^{(t+1)}}{1 - (\beta_2)^{t+1}} \\ 5. \quad & w^{(t+1)} = w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \varepsilon} \end{aligned}$$

Figure 18 - Adam optimization algorithm

In our case, $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$.

For the training process we decided to choose a batch size parameter, fixed to 32, to determine how many images go through the neural network at a time and then to calculate the loss function that will be used to update the weights.

Given a batch of greyscale images, with shape 48*48, as shown in figure 12, our mini-Xception computes the forward step in the following way:

1. *input* = tensor of images with shape (32,48,48,1)
2. *forward_tensor*^(θ) = *input* -> [CONV.(8 filters) -> BATCH NORM. -> RELU]*2
3. for $t=1$ to 4 do:
 - 3.1 *residual_tensor* = *forward_tensor*^($t-1$) -> [CONV.(16*2 ^{$t-1$} filters) -> BATCH NORM. -> RELU]
 - 3.2 *forward_tensor*^(t) = *forward_tensor*^($t-1$) -> [SEP.CONV.(16*2 ^{$t-1$} filters) -> BATCH NORM. -> RELU]*2 -> MAX-POOL
 - 3.3 *forward_tensor*^(t) = *residual_tensor* + *forward_tensor*^(t)
4. *output_tensor* = *forward_tensor*⁽⁴⁾ -> CONV.(7 filters) -> GLOB.AVG.POOL -> SOFTMAX

Through the softmax layer, our neural network will give a probability tensor in output, in other words, for each image in the batch, will be returned a probability vector of length 7 (7 is the number of labels).

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_{j=1}^n e^j}$$

Figure 19 - Softmax formula

With the probability vectors, we can return, for each image in the batch, the class with the highest probability score using the argmax function.

$$\underset{x}{\operatorname{argmax}} f(x) = \{x \mid \forall y : f(y) \leq f(x)\}$$

Figure 20 - Argmax formula

For what regards the loss function, softmax cross entropy (also known as categorical cross entropy) was used to train the mini-Xception. We calculate the sum (for each image in the batch) of the cross entropies comparing the predicted classes with the effective classes; then the final result will be divided by the batch size and changed sign.

Supposing that each instance of the batch is x , $p(x)$ is the class predicted by the model, and $y(x)$ is the ground-truth label, the loss function is defined as following:

$$L = H(y, p) = -\frac{1}{|batch|} \sum_{x \in batch} y(x) * \log(p(x))$$

Figure 21 - Loss function used

10. Train and test results

In this paragraph we will show the results obtained after the training phase. We trained the model until its convergence (for 80 epochs) using, obviously, the train set (80% of the dataset size). The accuracy is computed at the end of each epoch, evaluating the neural network performance on the test set (20% of the dataset size), while the loss is calculated instead on the training set.

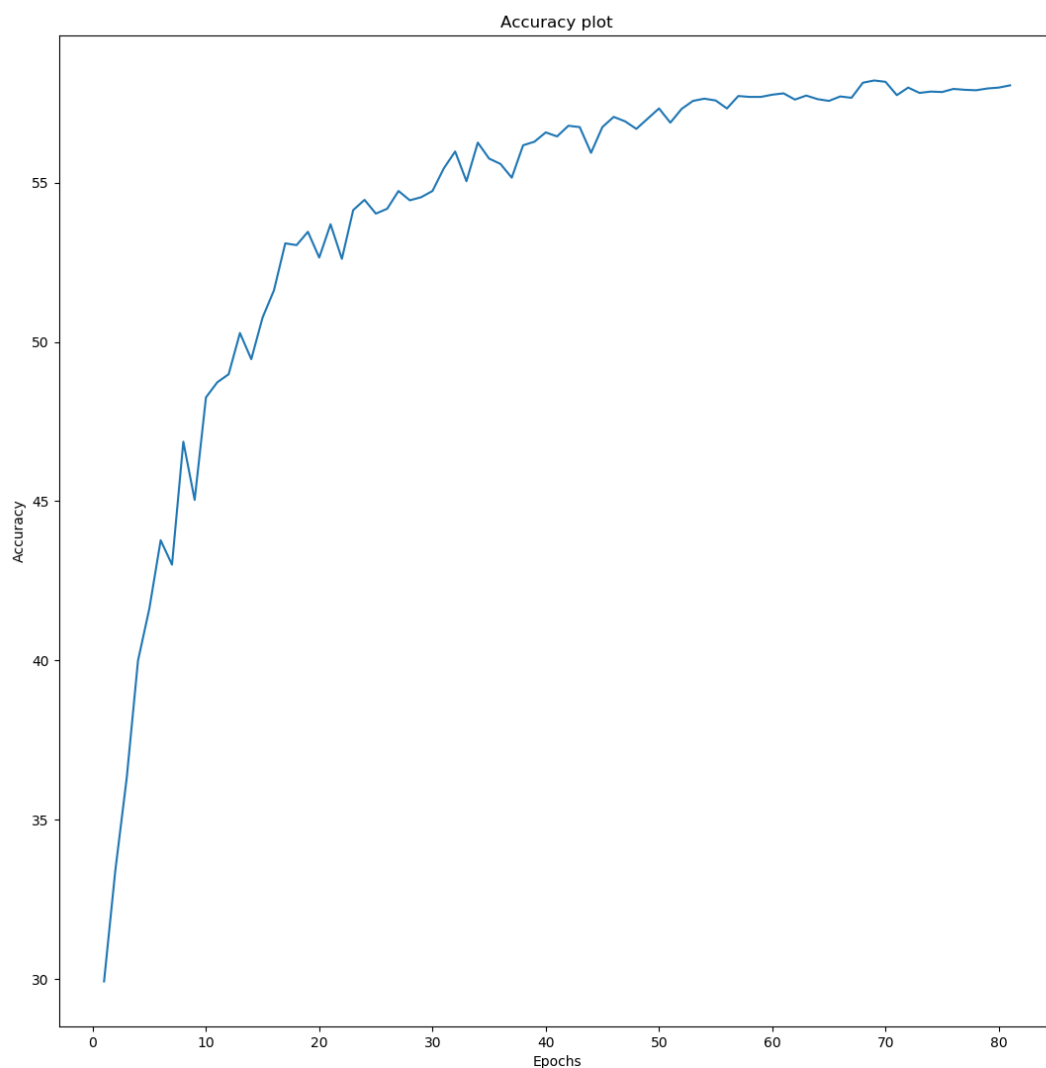


Figure 22 - Accuracy plot

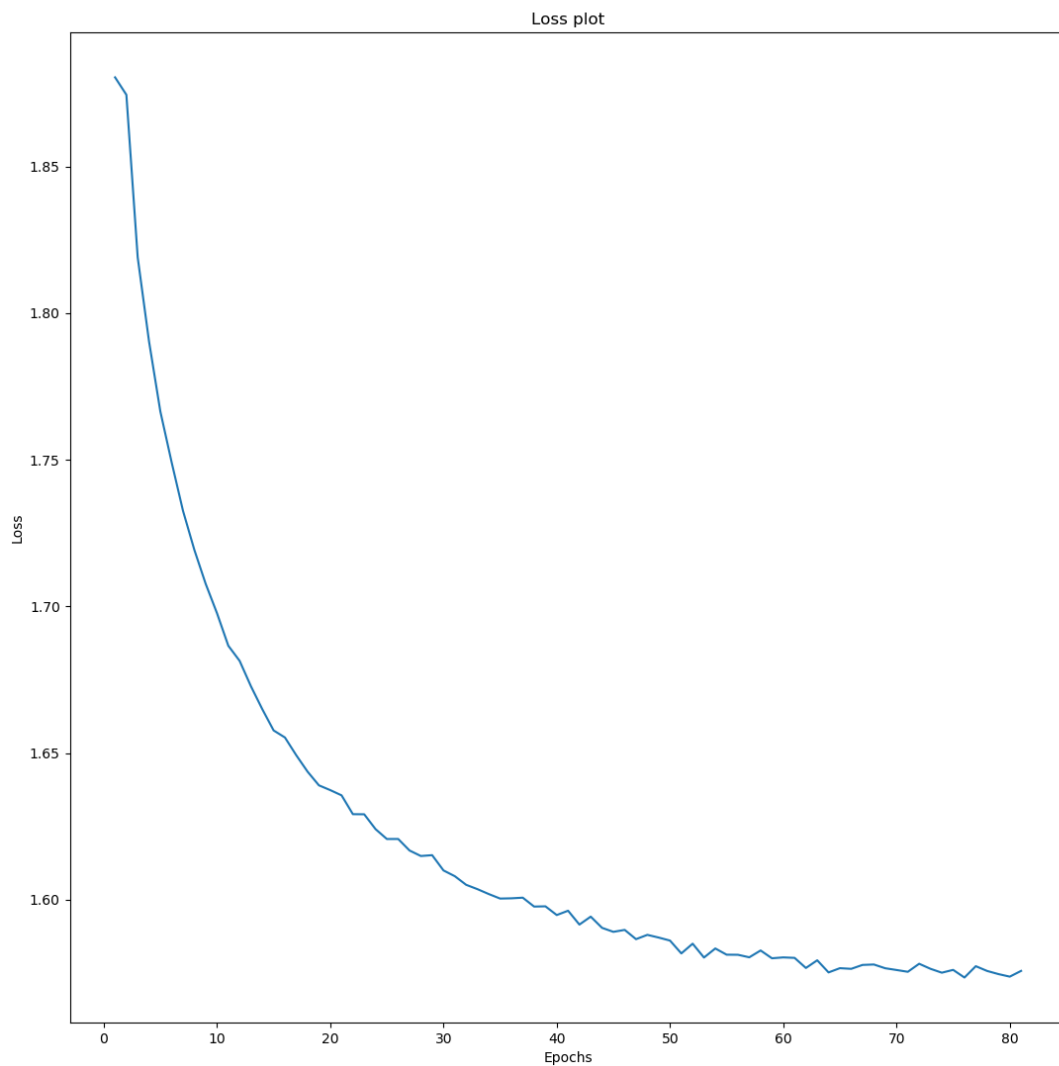


Figure 23 - Loss plot

After that, we wrote a script to print the confusion matrix in order to give the counts of emotion predictions and some insights to the performance of the multi-class classification model:

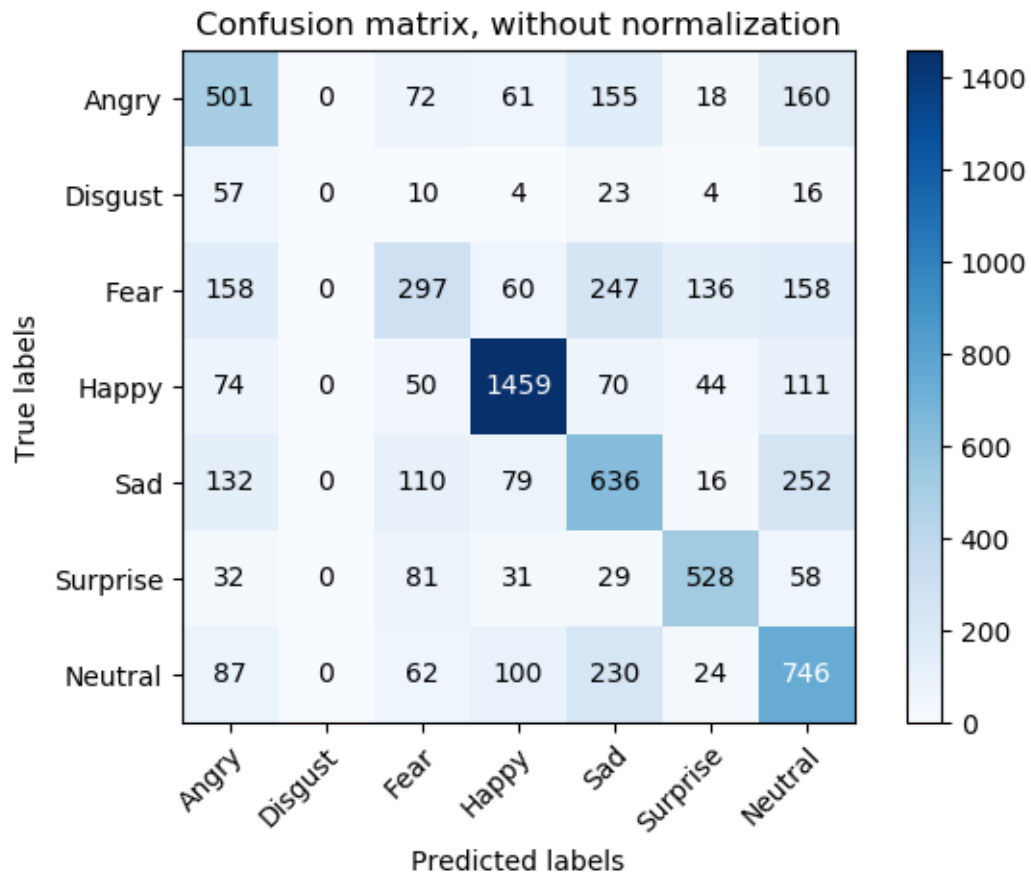


Figure 24 - Confusion matrix

The model performs really well on classifying positive emotions resulting in relatively high precision scores for happy and surprised. Happy has a precision of 81.3% which could be explained by having the most examples (~7000) in the training set. Interestingly, surprise has a precision of 68.6% having the least examples in the training set. There must be very strong signals in the surprise expressions.

For what regards negative emotions, model performance seems weaker on average. In particular, the emotion sad has a quite low precision of only 45.7%, same speech for the angry emotion, with a precision of 48.1%, and for fear emotion, whose precision is 43.5%. In addition, an interesting aspect, is that the model is not able to classify disgust expressions, in fact most of those samples are mistaken for angry, but considering the overall performances, however, we are satisfied with the results obtained.

Finally, also for the neutral expression the model performance is acceptable, obtaining a precision of 49.7%.

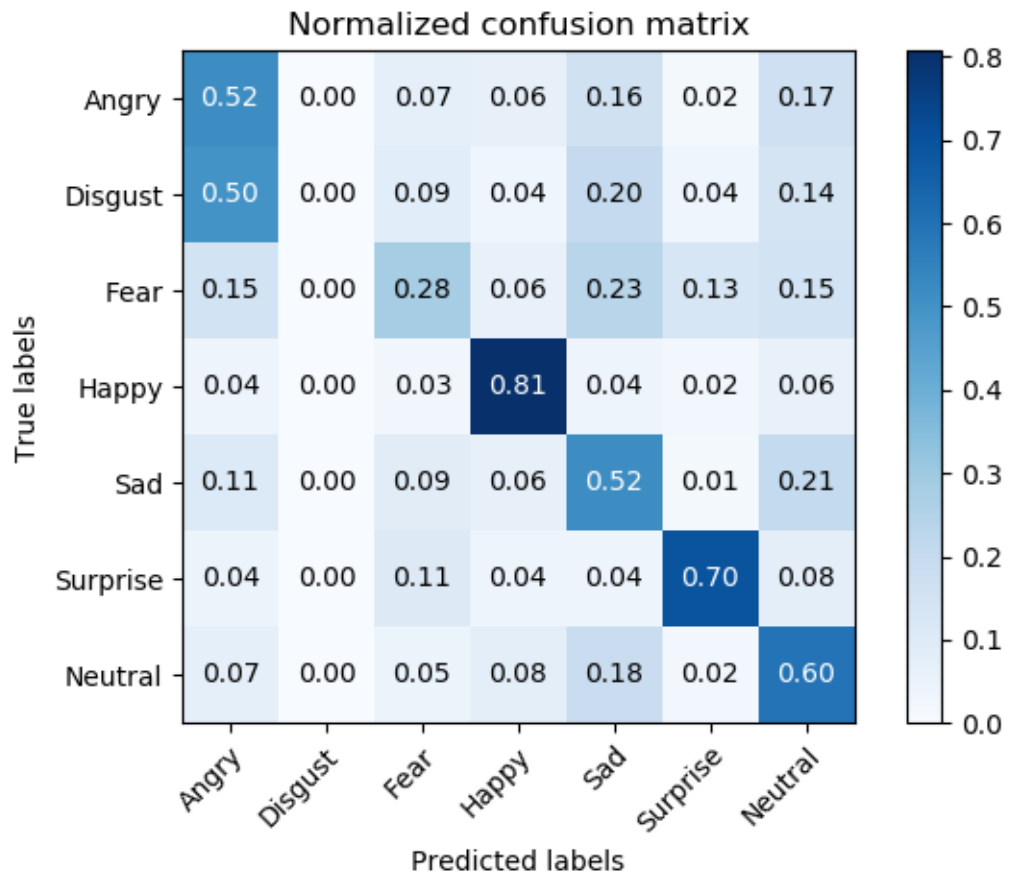


Figure 25 - Normalized confusion matrix

11. Testing the model (instruction for the use)

After completing the model train and test phase, we implemented also the face detection algorithm, with OpenCV library, which exploits the well-known algorithm proposed by Paul Viola and Michael Jones to detecting faces in the wild.

We tested our project on some image downloaded from google doing both face detection and emotion recognition together and these are the results:

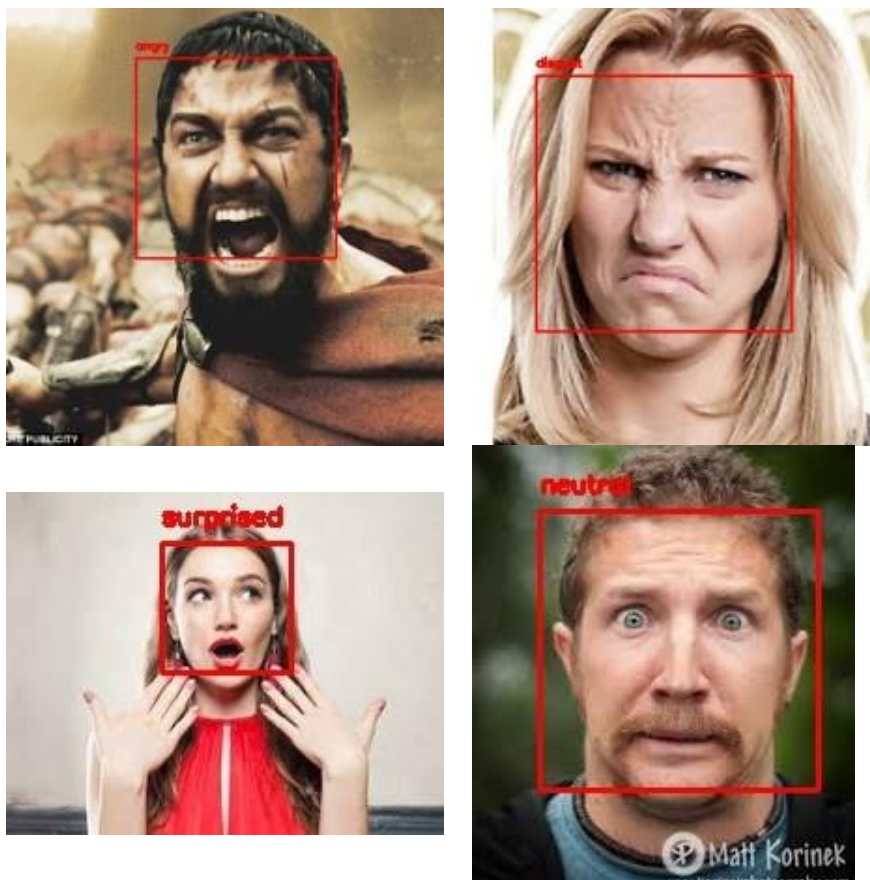


Figure 26 - Face detection + Emotion recognition

As a last thing, our final implementation performs face detection and emotion recognition in real time exploiting the computer webcam.

To run the real-time test, you need to open the command line, go to the project folder and launch the demo file, main.py.

The following are commands to be executed:

1. `cd <project directory>`
2. `python main.py`

12. Conclusions

With our work, we managed to create from scratch a deep learning model that is capable of performing a real-time classification task.

As for future work, it might be interesting to try to tweak the network in order to reach an accuracy value of 60%-65% or more.

An interesting method that we can apply to our project, in order to achieve better results, is to retrain the mini-Xception merging two or more dataset with the already used FER2013. Another solution that could allow us to increase the accuracy of a few points is to do more hyperparameter tuning, such as increasing or decreasing the convolutive layers of the network and evaluating any changes.

13. References

- [1] *Real-time Convolutional Neural Networks for Emotion and Gender Classification* by Octavio Arriaga, Paul G. Ploger and Matias Valdenegro
<https://arxiv.org/pdf/1710.07557.pdf>
- [2] *Xception: Deep Learning with Depthwise Separable Convolutions* by Francois Chollet
<https://arxiv.org/pdf/1610.02357.pdf>
- [3] *Deep Facial Expression Recognition: A Survey* by Shan Li and Weihong Deng
<https://arxiv.org/pdf/1804.08348.pdf>
- [4] *ADAM: A method for stochastic optimization* by Diederik P. Kingma and Jimmy Lei Ba
<https://arxiv.org/pdf/1412.6980.pdf>
- [5] *FER2013 dataset* by Pierre-Luc Carrier and Aaron Courville
<https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data>