Argyrakis Yannis 16133
Kirstein Julien 14136

Artificial intelligence Project

Introduction

In our artificial intelligence project, we had to program a multiple real-time face and mood detection application. The mood detection had to perform emoticon placement in function of the mood detected on each face. In this document, you will find a technical overview of the project with the technology used and the different steps of the project.

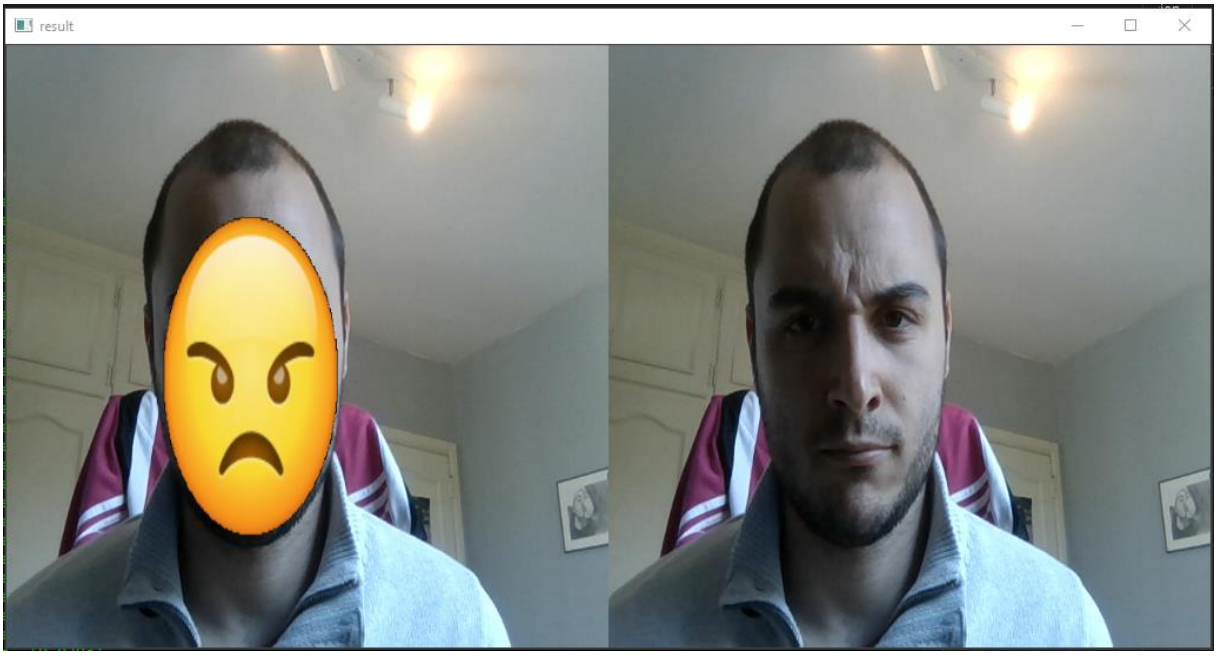
Choice of the programming language

Initially, we had to choose a programming language adapted to the project specifications. We therefore chose the very popular Python language because it is particularly suitable for artificial intelligence. Indeed, it has all the necessary tools, it is very intuitive and compatible with many OS such as Windows, Linux, MacOS and Unix. On the other hand, it is less suitable for mobile applications even if there exist some solutions.

Global structure of the project

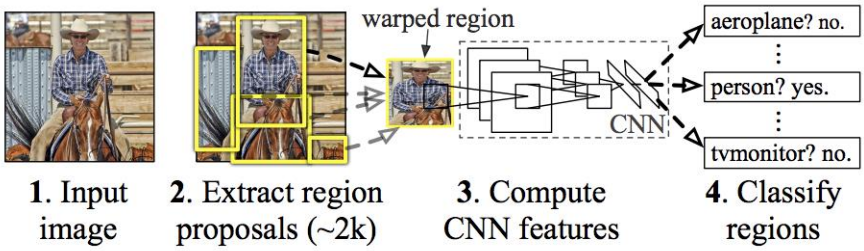
The application is a Kivy app and is running on windows, Linux and macOS. It can detect in real-time human faces using a custom trained object detection algorithm. Then, it classifies the mood of each detected faces using a custom trained model and place the corresponding mood emoticon onto each previously face detected.

An example is shown below, where we can see the real time face of the person on the right and the prediction model that places the emotion on the left.

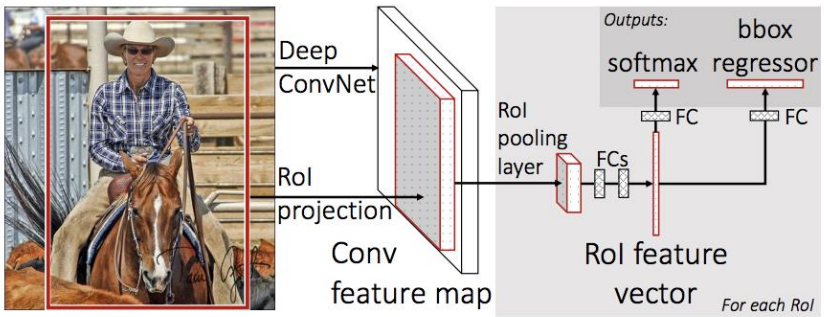


Choice of a face detection algorithm

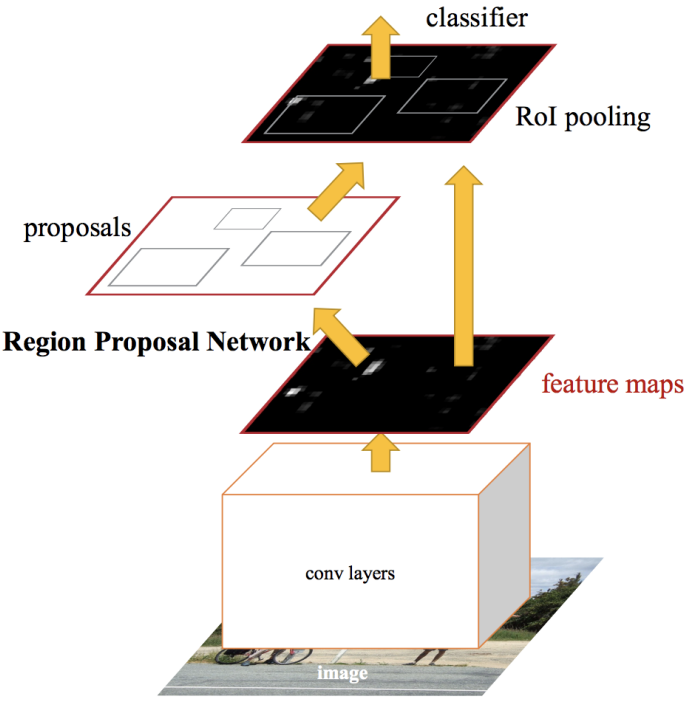
Many object detection algorithms exist. In this project, we compared the R-CNN, Fast R-CNN, Faster R-CNN, and the YOLO (You Only Look Once) algorithms. R-CNN is a method that use selective search to extract approximately 2000 regions proposals from an image also called ROI (regions of interest) and then fed each region into CNN to extract features and apply SVM (support vector machines) to classify them. Fast R-CNN is a faster method, in place of passing each region through CNN, it directly fed the whole image into CNN to generate conventional feature maps where we can identify the region of proposals using selective search. It then reshapes those regions into a fixed size and apply SoftMax classification to classify every proposed region. Faster R-CNN is an even faster method. As Fast R-CNN, the whole image is fed into CNN to generate conventional feature maps but in this method, this is not a selective search algorithm that is used to find the region of proposals, but a region proposal network also called RPN that is used. It then reshapes those regions into a fixed size and classify them. YOLO, in contrary to the previous object detection algorithms that uses regions to localize the object into the image, use a single convolutional network to predicts the bounding boxes and classify each of them. Therefore, it is the faster method and the most suited object detection algorithm for our purpose that require real-time performance. However, because of the spatial constraint of the YOLO algorithm, it struggles with small object. In our case it is not a problem but as a lot of other computer problems, it shows that an optimal solution never exists and that we always must choose a balanced solution between accuracy and speed.



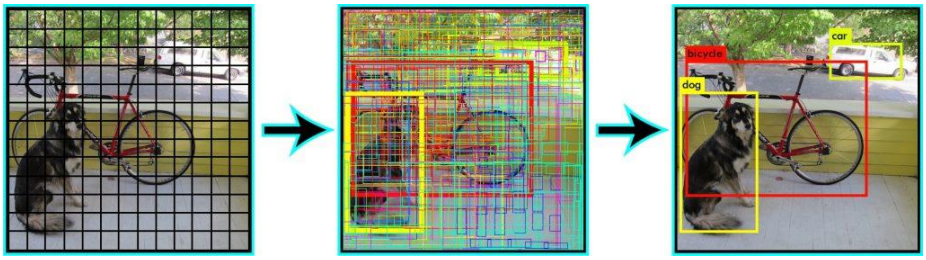
R-CNN



Fast R-CNN



Faster R-CNN



YOLO

Faces dataset

We used the famous wider face dataset. It is composed of more than 10000 images corresponding each to several faces' annotations. The initial wider face dataset was composed of multiple folders of images and annotations, so we merge all of them in only one directory so that it matches the YOLO format. In addition, each initial annotation has a specific structure that did not match the YOLO structure, so we converted each annotation to solve this incompatibility.

Faces dataset preparation

Each wider face dataset annotation was composed of a coordinate center, a width, a height in pixel and several information on the face that where not useful for us as shown below.

```
x1, y1, w, h, blur, expression, illumination, invalid, occlusion, pose
```

The YOLO annotation is composed of a class name, a coordinate center, a width, and a height in percentage corresponding to the max width and height of the image:

```
Class name x1, y1, w, h
```

http://shuoyang1213.me/WIDERFACE/index.html

Faces detection implementation

We build our own faces detection model using the YOLO algorithm. To achieve that, we used a pre-trained Yolo model and trained it with our own data. This way of training our model is called “transfer learning” and has the advantage to speed up the training process by reaching faster the desired model performance compared to using a random initial model.

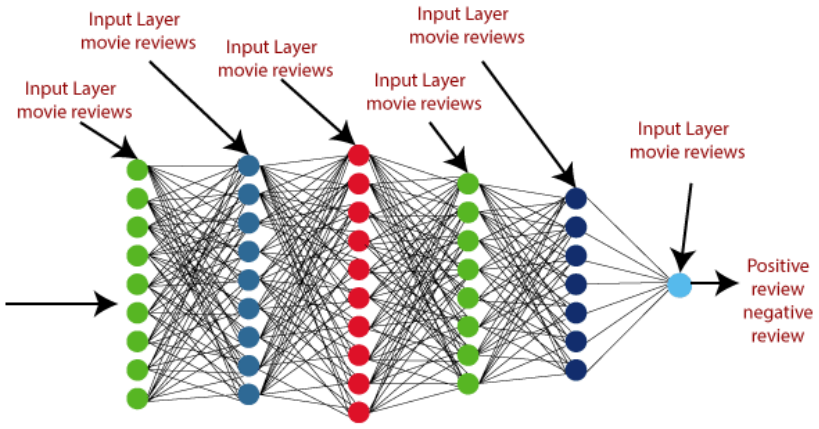
Mood classification

Once the faces have been detected, we had to classify the mood of each of them. To achieve that, we used the Keras sequential model. It classifies one faces at a time.

The sequential API provided by Keras allows us to define our model layer-by-layer. This model means that each new layer is connected to the previous layer.

On this model, the Input layer takes a tuple argument indicating the dimensionality of the input data. In our case we passed 48x48 images. When going through layer, the model takes the input layer and connects it with the following layer (In the example, the “dense_1” layer).

As the sequential model uses deep learning neural network, we show a typical scheme of how this neural network can be represented.

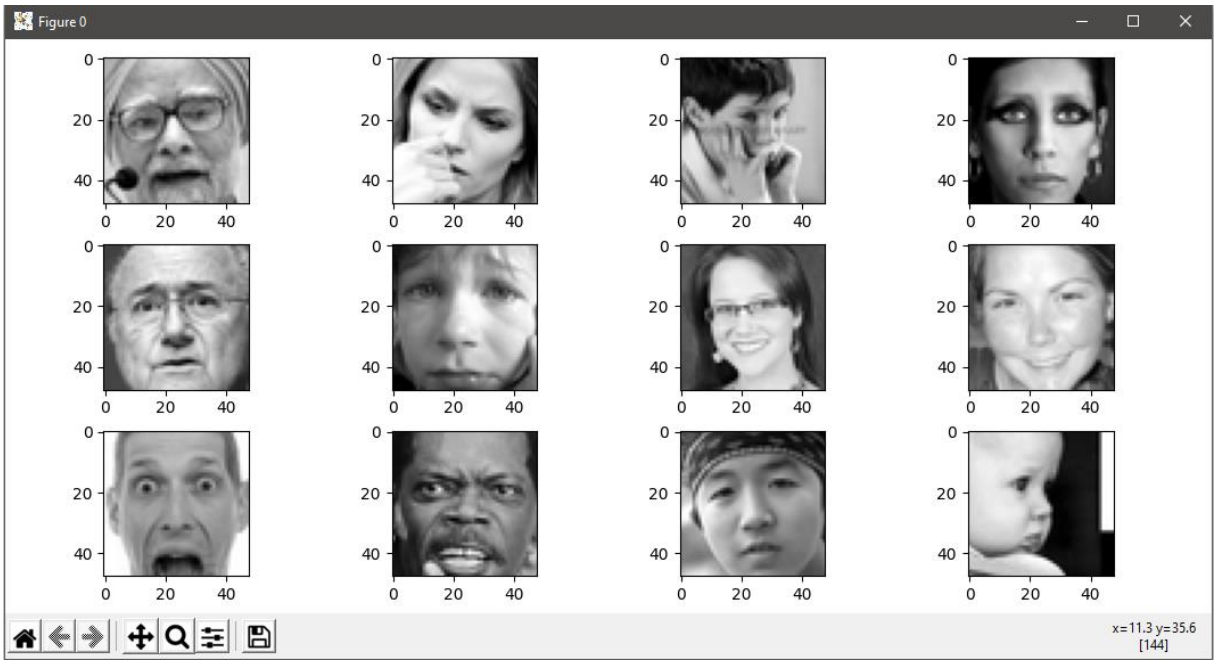


Keras Neural Network Sequential Model

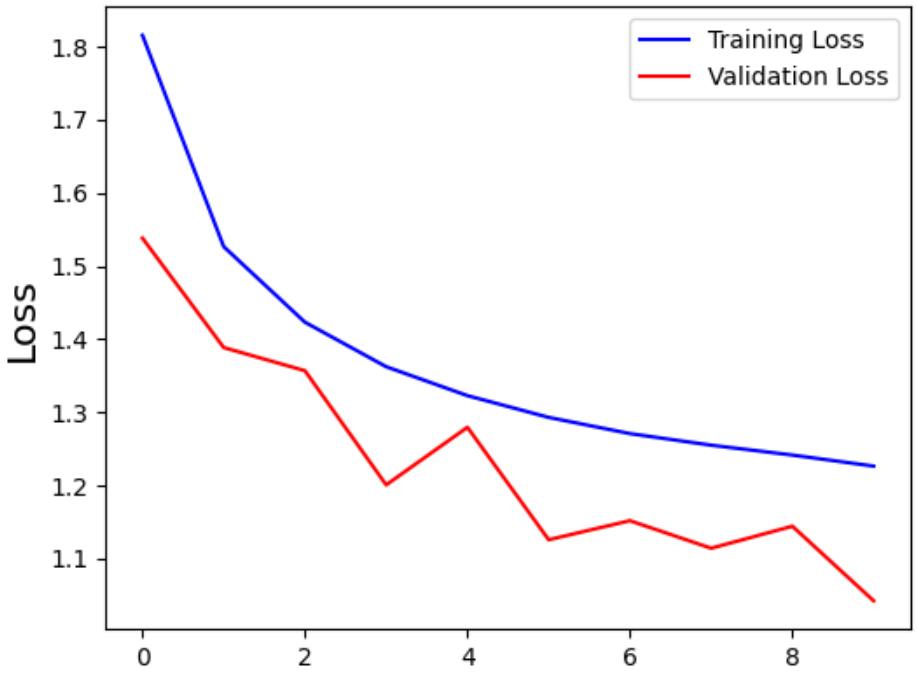
By training this kind of model, we obtain the weight vector. This weight vector transforms the input data inside the hidden layers. On each hidden layers, the input enters the node and is multiplied by the weight value of this layer. At the end, the resulting output give us a number. In our case, the number correspond to the emotion number of the image.

When we trained our model, our programs did the following things:

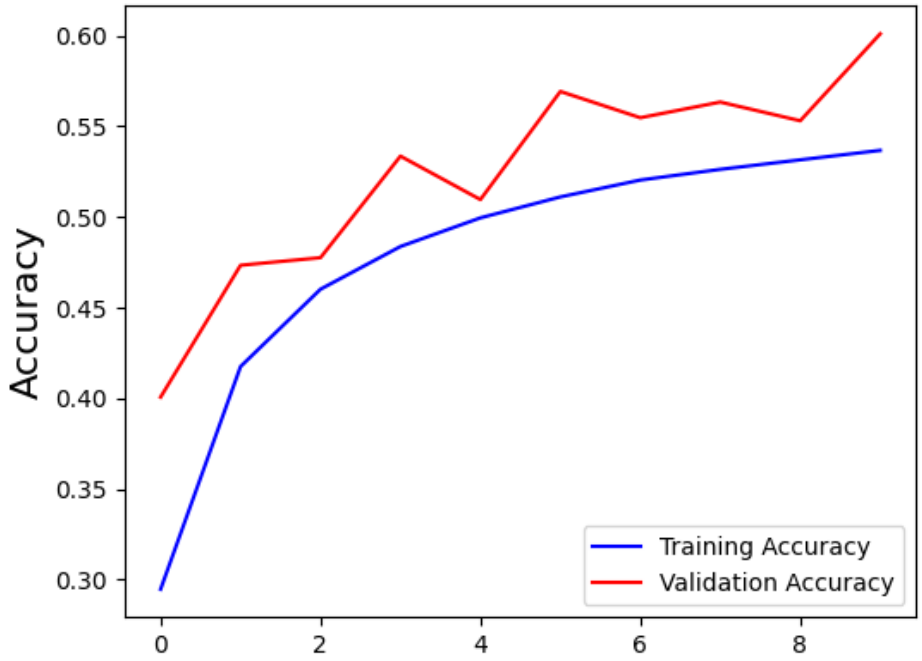
1. Plot some images chosen randomly to check that the dataset is operational (Result shown below)



2. Define the sequential model with all the layers and create a weight vector file
3. Define the number of epochs (how many times the model will go through the data.
   In our mode, we used 10 epochs because we noticed that the accuracy of our model would tend to a certain value at this point.
4. Run the model with training data & verifying with the test data.
5. Save the model with its configuration to a json file.
6. Evaluate and plot statistics of the model: Loss & Accuracy.



The Loss graphic shown on the image above represents the error of our model. As we can see, the curve is logarithmic and when we reach the 10th period, the function Loss value tend to be between 1.2 and 1.3.



The same principle applies to the accuracy curve. We can see on the graphic that the accuracy of our model is hardly more than 52-53%.

Mood dataset

We started with the fer2013.csv dataset (https://www.kaggle.com/deadskull7/fer2013). This dataset contains 35,888 images on each line which are classified into six emotions. An example of a line of the file containing an image is shown below.

	A	B	C	D	E	F	G	H	I
1	emotion,pixels,Usage								
2	0,70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98 91 84 84 90 99 110 126 143 153 158 171 169 172 169 165 129								

In this example, we see that the 2nd line has the label which is the emotion (0 is for “Angry”) followed by a list of 2,304 numbers that are the pixel intensity values of the image.

We had not a lot of images and they were not equitably distributed, so we combined multiple mood dataset. By testing the display of an image, we noticed that the resolution was 48x48. Therefore, we found one additional dataset on Kaggle (https://www.kaggle.com/sudarshanvaidya/random-images-for-face-emotion-recognition). This dataset added 5558 more files into our first 35,888 images.

We also added another dataset containing files that were not 48x48.

To be able to use those new datasets with the fer2013.csv dataset, we had to use a program to normalize the data of the images that labilized them in the fer2013 format.

At this point, the resulted dataset was not balanced between the different mood classes, so we did data augmentation via a program using the Keras library allowing us to change the scale of each image, the contrast, the lateral and longitudinal position, the rotation, etc. In our program, we fixed the number of 20,000 images by emotions.

At the end on the six emotions, we used a statistics program to count the repartition between our first dataset, the intermediate one (when we normalized two more datasets) that increased the number of images and the augmented dataset. At the end, the repartition was the following:

Initial fer2013 dataset = {0: 3995, 1: 436, 2: 4097, 3: 7215, 4: 4830, 5: 3171, 6: 4965}

Intermediate fer2013 dataset = {0: 9838, 1: 1422, 2: 9788, 3: 17610, 4: 11653, 5: 7948, 6: 11687}

Final fer2013 dataset = {0: 20000, 1: 20000, 2: 20000, 3: 20000, 4: 20000, 5: 20000, 6: 20000}

We can clearly see that the classes are much more balanced and with more data for each emotion. That will help to increase our Mood classification model accuracy.

Interface

Our app is not working on smartphone in this version, but we created a Kivy app so that in future versions, we could put our app on Android and IOS.

Improvements

Because we must run the mood classification for each face detected, if there are too many faces detected, the application is slow. Therefore, as an improvement, we should train our YOLO model to detect each face and mood in only one step. The difficulty of doing that is that there is not a dataset for that so we had to create our own dataset by using our classify mood on the wider face dataset, but it would take a long time to run and make a lot of mistakes because the wider faces are sometimes too small and cannot therefore been detected. The wider face is of course composed of unbalanced mood classes too so it would be needed to use data augmentation once again which is not easy for that type

of annotations. Our app work on windows, macOS and Linux. Because we used Python, this is not easy to run it on a phone. However, we could create a Kivy app and compiled it so that it run on smartphones or create a client-side website, but Python is not made for that, so it is quite difficult.

Conclusion

At the end of the project, our application can detect in real-time multiple faces and classify each one to obtain the corresponding mood and place the related emotion emoticon onto each one. The app is not running on smartphone in this version, but it could be possible by compiling our Kivy app for Android and IOS. If there are too many faces, because the mood classification is made one face at a time, it will be slow. To solve this problem, we should create a YOLO model that detect each face and mood in only one step.

Bibliography:

https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46

https://www.freecodecamp.org/news/facial-emotion-recognition-develop-a-c-n-n-and-break-into-kaggle-top-10-f618c024faa7/