Faciale emotion recognition with CNN

Adrien Guezennec *IEEE Member*,

***Abstract*—Going to the moon is not a problem anymore, but understand your emotions and faced them as never been so troublesome nowadays. Due to the enormous technological advance, the digitization of the image is now for granted, and everybody used it. I ask myself if it would be possible to identify the emotion on the image from someone's faces. In this paper, I will implement a facial emotion recognizer using a convolutional neural network. I based the problem on only seven emotions [neutral, happy, angry, fear, disgust, sad, surprise] and avoided the emotion in between, like confusion, concentration, or others. This purpose is to lower the different possible outputs and focus on the behavior of the neural network and understand the general process. Since, as I said, the primary purpose was the investigation and analyses, I used a library named Tensorflow. Famous library, developed by Google, in machine learning and especially in neural networks to help me with my task. In the end, I demonstrate the feasibility of my hypothesis but with some issues that I will explain in detail.**

I. Introduction

In machine learning, computer visualization as been and is still is an intriguing topic. Computer visualization is everything related to image processing, which makes facial emotion recognition part of computer visualization. This field has seen much improvement in the last ten years with self-driving cars or even with facial recognition to unlock our phones. It's part of our society now, and it does not seem very easy to go back. Analyzing and identify emotion has already be done, and you can find different models in different libraries. But the goal was not to use something existing but instead build my model for prediction and understand the design choices and their impact. For that, I needed to create my neural network and decide which architecture to implement. Since I had no detailed knowledge of neural networks, I had to do much research to improve my experience as well as my skills in deep learning. I tried to structure this paper the same way as I go through my experiment.

The first step was research. As it was an essential part of the project, I will go through the different learning steps and articles that I read. After the study, I will explain the architecture that I decided and the reason behind it. Subsequently, this choice conducts me to a phase of planning to prioritize the first and most important things I wanted to prove and show in this experiment — lastly, the implementation in Python with the issues that I faced and the result of my test.

Finally, I'll conclude with the lesson learns and my planning for the future work because the more I dug in, the more possibility I saw.

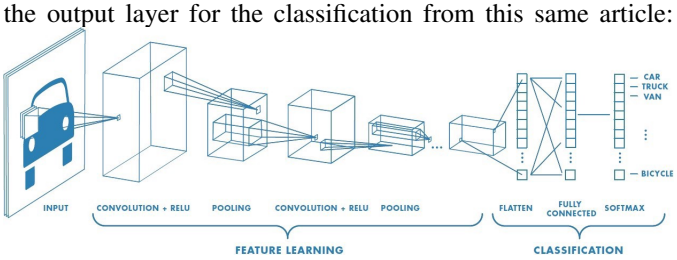
II. Methodology

A. Research

The research is a severe part as it will build the foundation for the project and influence the choices made. So, I focus my research on two main parts.

- Neural networks for images classification
- Optimizing neural networks

1) Neural networks for image classification: My investigation in neural networks for image classification lead me to a type of neural network called convolutional neural networks (also called CNN). CNN is a very famous algorithm capable of finding an essential aspect in an image and differentiate one from the others. In addition to being efficient and highly used, other architectures appear based on CNN architecture, like GoogleLeNet, VGGNet, or ResNet. I discovered and based my work on an article [1] found in towardsdatascience.com, and he explains in detail how the CNN algorithm works and how the different hidden layers act on an image. Here is an excellent scheme of CNN algorithm with the various hidden layers and the output layer for the classification from this same article:



On this image, we can see that there are two main layers in the CNN which is the convolutional layer and the pooling layer. The convolutional layer consists of filter the current image with a kernel filter, which goes through the image and extracts essential information in the picture, such as border or edges. During the process, it is crucial to add padding to the image because otherwise, it wouldn't extract information about the border of the image since he only iterates once. There are two types of padding, valid and same. Valid refers to no padding, and the same means that the output matrix will be the same as the input image size. After that, the image input goes through a pooling layer. The pooling layer can either take the average of the kernel filter or the max value of it.

We can quickly start making architectural decisions on the given information. I will explain my decision in the next chapter.

2) Optimizing neural networks: Now that we have a better understanding of CNN, let's talk about optimizer for CNN. The research about optimizer and optimizing a deep neural

network was the most time consuming for me. When you start to dig in, there are many types of optimizer, and it is not very easy when you start learning deep learning. But I focus my search on the gradient descent, backpropagation, and the Adam optimizer algorithm, which is what I need for my CNN. To do so, I based my founding on this 3 article written by the same author and well explained:

- Gentle introduction mini batch batch gradient desscend[2]
- 8 tricks for configuring backpropagation in neural networks[3]
- Gentle introduction to the adam optimization[4]

These 3 article leads us to the architecture decision that I made for my CNN on emotion recognition.

B. Architecture choices

After summarizing the previous step, I finally came up with a simple architecture for the beginning. Creating a neural network has to be planned and reflect but can also grow with time. So, I decided to implement three convolutional layers with max-pooling and a drop-out of 1% for the first pooling and 15% for the second pooling. The drop-out will exclude uninteresting features from an amount of respectively 10 and 15% of the output. The last group of layers is for the classification, where I have one layer for flattening the images, one dense layer, and the last one is a dense layer with seven output since our classification is for seven emotions. I didn't come up directly with this final architecture; this was an iteration process with testing, implementing, and compare the result.

Every convolutional layer has a padding type of the same because I didn't want to lose information about the image. For the activation function, I used rectified linear units, also called ReLU, because in contrary to sigmoid ReLU, significantly accelerates the convergence of stochastic gradient descent. That's is not the only reason, but for a more detailed explanation I recommend to read this paper: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

C. Planning

Unfortunately, I didn't have months for this project, so I had to plan and prioritize the result that I wanted to have. Apart from the research which is mandatory before any coding/implementing part, I focus this project on two axes. The first one was to have at least 50% accuracy for the predictions, and the second one was to understand why I couldn't get more and how to improve it.

My working environment also came in planning, due to my lack of computing calculation I had first to focus on implementing my architecture so I could run my CNN during the night and analyze the result, make the modification during the day.

```
Epoch 50/50
28709/28709 [==============================] - 4s 143us/sample - loss: 0.1497 - acc: 0.9476 - val_loss: 2.7393 - val_acc: 0.5631
57.95486207857342
```

Accuracy

III. Results

A. Implementation

For the implementation, I'll first present the library that I used and explain why. I implement the CNN with the Tensorflow library in Python. I wanted to add this library to my toolbox and also because the library provides all the necessary tools for creating, modeling a deep neural network, and also allows the user to save or load the existing model. As I explained earlier above, my lack of computation power leads me to quickly implement a saving model method to reuse the model and not retrained the model each time I run my script. Before starting to implement my wanted architecture, I first start with some prototype to be familiar with the library. I also code the base around the CNN and the classes that I will need to run and load the dataset.

For the dataset, I choose one found on Kaggle [5], which provides a dataset with the training (28,709 samples), testing (3,589 samples), and validation (3,589 samples) data. The sample images in it were a 48x48 pixel image of a face-centered in grayscale. When dealing with images, having a grayscale to process is much easier and less computational needed because there is only one depth. A colored image is a mix between red, blue, and green, so instead of 48x48x1 for the picture, I would have had 48x48x3. All the images are labeled with emotion from 0 to 6. I found other dataset with many photos but asked much preprocessing, and this dataset allowed me to start working very quickly, so I decided to go with this one.

At first, I implement only one convolutional group layer (convolutional layer with a kernel size of 3x3+ max-pooling + drop-out) and one classification layers (flatten layer, dense with 128 output neurons layer + final dense with seven output for each emotion). I run it with a batch size of 64 and 10 epochs. It gives me an accuracy output of less than 50% and a validation loss of around 4.3. The first result was pretty bad, and I expected more than that, but with only one group of the convolutional layers, the neural networks couldn't learn enough from the images. So, for my second iteration, I trained my CNN with another convolutional group layer. The result was better, but not sufficient for my goals. It appears that my neural network didn't get enough information from the image. In reaction to that, I implement 5 group layers of convolutional layers, but it was worst, so I found out that my initial configuration during the architecture process was not that bad and showed better results.

I also realize that compute images in a neural network is a vast task speaking computationally. It's almost impossible to work with a slow laptop without a GPU.

B. Issues and tricks

As I said in the implementation part, my laptop hadn't enough computer power to perform many training or too large neural network with many layers and neurons by each layer, so I had to adapt my work. One solution to that was to rent a small EC2 on AWS and run the script on it. Thanks to the GPU provided by the EC2 server, the computation time was divided by one hundred, but since it wasn't free, I couldn't afford to run it all the time. I run it with 500 epochs on the server, but it didn't change that much the accuracy or the validation loss. I found out that after around 50 epochs, there was no real difference with the same architecture.

IV. Conclusion and future work

To conclude, I run different CNN architecture types with and without drop-out, many or fewer group layers, but having too many group layers doesn't help at all for the classification. It diluted the picture too much. In the end, I got 58% accuracy on the prediction with a loss of 0.15.

There is a way to improve and maybe adding batch normalization layers in between the convolutional layer, and the max-pooling layer could help with a higher drop-out, but in the future, I need a GPU to retrained my model and modify parameters faster.

The next step would be to extend my CNN to handle color images and to implement the facial emotion recognition on a website using my model loaded in Tensorflow.js.

References

[1] M. Peixeiro, "Introduction to convolutional neural networks." [Online]. Available: https://towardsdatascience.com/introduction-to-convolutional-neural-networks-cnn-with-tensorflow-57e2f4837e18.

[2] J. B. PhD, "Gentle introduction mini-batch gradient descent." [Online]. Available: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.

[3] J. B. PhD, "8 tricks for configuring backpropagation in neural networks." [Online]. Available: https://machinelearningmastery.com/best-advice-for-configuring-backpropagation-for-deep-learning-neural-networks/.

[4] J. B. PhD, "Gentle introduction to the adam optimization." [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[5] Kaggle, "Challenges in representation learning: Facial expression recognition challenge." [Online]. Available: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/overview.