Emotion Detection in Human Faces' Low-resolution Images Using Deep Neural Networks

Giacomo Russo
University of Lausanne
HEC Lausanne
MSc in Management
Business Analytics
giacomo.russo@unil.ch

Adrian van Kampen
University of Lausanne
HEC Lausanne
MSc in Management
Business Analytics
adrian.vankampen@unil.ch

Allan Mivelaz
University of Lausanne
HEC Lausanne
MSc in Management
Business Analytics
allan.mivelaz@unil.ch

***Abstract*—In this project, we use machine learning techniques, and more specifically deep learning models, to classify images of human faces into the respective emotion they portray. The images are gathered from the FER-2013 dataset as available on Kaggle. This allows us to focus on the Machine Learning aspect of the project, as the data is already clean, classified, and split into train and test sets.**

We describe our algorithmic approach in which we used different structures of Neural Networks. We start by configuring a hand-made neural network using convolutional layers, pooling operations and fully connected output layers. Yielding poor results, we move on to use transfer learning using the popular VGG-11 and VGG-16 pre-trained models, before unfreezing the latter and performing a full training of the model on our dataset.

Although early tries yield results indistinguishable from random classification, our last model's categorical accuracy reaches levels similar to what has been achieved in the literature, which given our limitations, is rather unexpected, but good. We finish by showing the resulting confusion matrices, and providing ideas for future work that could be done using the same framework and model.

1. Introduction

Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos and deep learning models, machines can accurately identify and classify objects — and then react to what they "see" [1]. Thanks to the fact that the input that can be analyzed are not strictly limited to static images nor to a specific shape, the number of applications of computer vision are plenty. Some of the many fields this research can be applied to are facial recognition, driver-less cars, medical diagnosis, digital security surveillance, and even human computer interaction.

The application on which we focus for this project is emotion recognition on human faces. As mentioned, recognizing images is an already pretty well-explored task, and efficient and powerful algorithms already exist that yield good results [2]. Human faces differ from objects in that identifying their emotions is often very subtle, and sometimes even difficult for humans.

However, thanks to a dataset gathered on Kaggle [3], we got the idea to try ourselves at that task, although it seemed complicated. Indeed, to expand our horizon over traditional machine learning models applied on traditional machine learning tasks, we decided to try and apply our knowledge of machine learning procedures and data analytics, as well as our curiosity and ability to learn from ourselves, to take on a computer vision problem. Furthermore, as mentioned, this computer vision problem seemed to us particularly interesting as it seemed more complicated than objects classification, but also because the applications seemed more human and concrete.

Indeed, after HEC Lausanne, few students will go and work in production and/or operations jobs where object recognition would be more useful. Those areas are usually better served by graduates from a more technical background, such as engineering graduates. On the contrary, human emotions are crucially important in a management context, and at least understanding some techniques that allow to recognize them seemed to be more important to us as future Management graduates.

2. Relevant literature and Research Question

Our main focus in this project was not frankly about the emotions themselves. Plenty of research in psychology and other fields have been conducted to understand and classify basic emotions (see [4] and [5]). This research has concluded in classifying human emotions into seven categories:

1) Anger
2) Disgust
3) Fear
4) Happiness
5) Neutral
6) Sadness
7) Surprise

To know whether this classification was complete, as in it could include every human facial image in existence without any leftovers was besides the point of our project. Indeed, our data, as discussed in chapter IV, already included this classification, such that it was not our choice to make.

In contrast with the research in psychology and medicine, the research of emotion recognition in a computer vision context is fairly recent (although early attempts using two-layer perceptrons were conducted as early as 1998 [6]). To be fair, this is mainly due to the whole field of computer vision being recent, and exploding nowadays with the rise of easily accessible powerful computing devices such as affordable GPUs and so-called ”Cloud” installations that allow for complex calculations that would otherwise be time-intensive to be conducted more efficiently and nearly for free. In that regard, this specific task has already been attempted using deep neural networks with fairly conclusive results. The best model used multi-level convolutional neural networks, achieving 73.03% accuracy to our knowledge [7], while others used the famous VGG-16 architecture pre-trained on the ImageNet data-set and fine-tuned to achieve higher task-specific categorical accuracy [8]. Some even used a mix of deep neural networks combined with a support-vector machine layer to achieve similarly high results [9].

In this project, we will try to expand on the research already conducted, that is we will test different model architectures in order to see whether we can achieve similar results, or even better ones, without too much fine-tuning. Indeed, our main constraints being time, limited resources – in terms of computing power – and knowledge, we will need our models to be fairly simple in their implementation.

3. Methodology

We started by looking into how we could implement deep learning algorithms to extract features from our dataset’s images. As stated previously, most deep learning algorithms are complex, and involve heavy computations, which requires a lot of computational power.

In particular we relied on the use of so-called ”convolutional” neural networks. These networks which rely on automated filtering are especially effective at pattern recognition, which make them perfectly suited for computer vision [10].

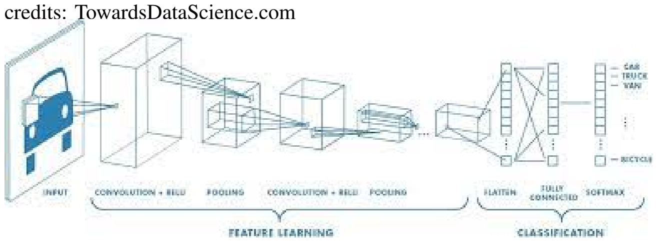
The CNN is a feedforward neural network, which can extract features from a two-dimensional image and optimize network parameters by using a back-propagation algorithm.

The structure of the convolutional neural network (see figure 1) is peculiar and unique since it is structured of two components.

The first component of a convolutional neural network is the eponymous convolutional layer. Each convolutional layer applies filters to an Nd-array as input which represents the images dataset. This results in an activation or not of each node in the layer according to automated filters.

This repeated application of filters over the same input ends up in what is called a feature map which indicates the locations and strength or weaknesses of a detected feature in

Figure 1. An example of a CNN architecture
credits: TowardsDataScience.com



the input, the presence of a smile for example. The output of a convolution layer depends on the dimension of the kernel, which is the size of the filter itself that does the operations with which the features are extracted from the images. For instance, using our 48x48 pixels images as input which pass through a convolutional layer with a 3x3 kernel size the output dimension is reduced to 46x46. Moreover the output depends on the number of input channels as well and thus, since our dataset is composed of gray scale images, the number of input channels is one – rather than three for RGB images. In the case the image had multiple channels the Kernel would have the same depth as that of the input image itself.

The operations that occur are multiplications of a set of weights with the input. Thus, the parameters that are learned during the training process are the weights associated with each filter. The more filters, the more features they can extract, but the higher the computation power needed to perform both the training and the validation process of the network.

Furthermore while working with such a small quality images we don’t want to reduce the dimension of the input while it’s being processed by the convolution layer and thus we add the padding which adds a given number of pixels to each image’s edges such that the output of the layer will have be of the same dimension as its input. In this case the padding is called ”same” padding, whereas the further reduction of the input size while it’s being processed by the convolution layer is done by applying a ”valid” padding. In our case we used the former.

After each convolutional layer, which extracts the features from each input by creating a given number of feature maps, we usually pass through a pooling layer. This type of layer helps avoid over-fitting the data, while having the helpful side-effect of reducing the computational needs of training the network.

The pooling layer has two main attributes, a filter size that can be seen as the kernel for the convolutional layer and the stride. The latter represents how many pixels are being processed during the pooling operation that can be – most commonly – either ”maximum” pooling (aka maxpooling) or ”average” pooling, which take respectively, given the filter size, the maximum or average value of the cells from the given dimension of filter size and moves by the given stride at each iteration. This will significantly reduce the dimensionality of the feature map without reducing the number of maps that are available from the output of the previous layer.

The pooling layers are very important for training since without the pooling, it would be almost impossible to train a CNN even with a very powerful computer.

After the pooling process, the results are passed through a ReLu activation function that distributes the values of the features map as either 0, if the feature that has been extracted is very weak in that particular area of the map, or 1, if the feature extraction results to be effectively activated by the strong presence of a feature in that area.

Thus, we construct many Convolution layers that allow us to extract high level features in the first layer such as noses, smiles, eyes, teeth etc. The more convolutional layers the data passes through, the less interpretable the feature extracted become to human eyes.

As we reach the last Pooling operation the output of the latter will be flattened so as to make it processable by a fully connected layer. We need to flatten the input that reaches this point with the shape of [dim1,dim2,dim3,dim4], where dim1 stands for the number of batches dim2 is the number of input channels (colors) and dim3 and dim4 are the images dimension, to make it one dimensional. This is sometimes also referred to as reshaping.

When it reaches the fully connected layer, the input is processed as it was a normal operation that occurs within a common Neural Network. Usually we add a dropout function that allows to randomly deactivate some neurons within the layer to avoid overfitting.

After the path of the fully connected layers we interpret the output using the softmax function that takes as input the output of the last fully connected layer:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\text{for } i = 1, ..., K \text{and } \mathbf{z} = (z_1, ..., z_K) \in \mathbb{R}^K$$

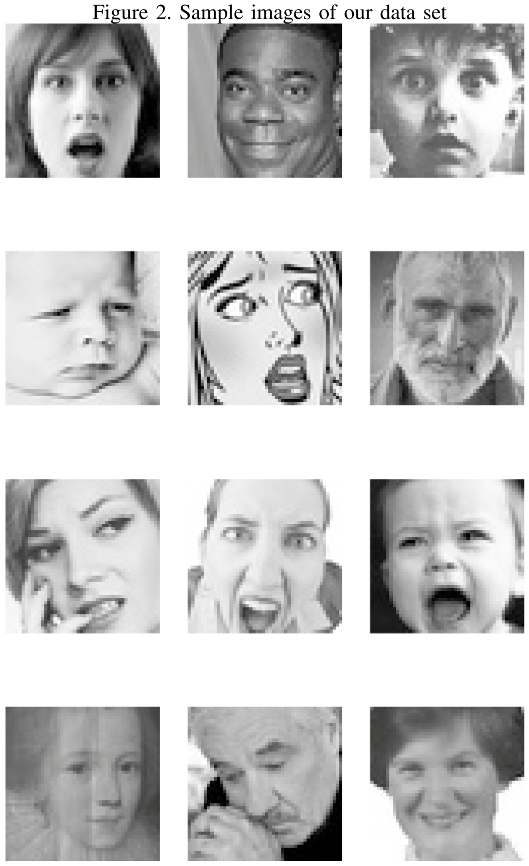
This function gives at the end the probabilities for all the classes that have been defined inside the first input layer which summed are equal to 1 so, we will have that the highest probability inside this vector will be the class predicted by our CNN.

4. Data

We used the FER-2013 data-set as available on the website Kaggle.com [3]:

> The data consists of 48x48 pixel gray-scale images of faces. The faces have been automatically registered so that the face is more or less centred 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 into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the public test set consists of 3,589 examples.

Figure 2. Sample images of our data set



Although the data-set is relatively large, one of its limitation is the images' quality. Indeed, each image is very small and gray-scale, which limits the amount of information that our models can exploit. However, this is also an essential characteristic of the task, since this task aims at providing usable models. In practical applications, one cannot expect the images to be of high quality. For instance, in surveillance applications, images taken from surveillance cameras are often gray-scale also, and subjects may be photographed from afar, which limits the quality of the resulting image. Thus it is actually a good test for our models to be relevant and potentially useful.

It is also noteworthy that the images contain photographs, but also drawings – such as paintings or cartoon drawings – as well as some odd images which potentially were misclassified as human faces. Therefore, it is to be expected that no model will have perfect accuracy.

The images were initially gathered from Google Images, pre-processed, and posted on Kaggle by Pierre-Luc Carrier and Aaron Courville. The main pre-processing steps were the reframing and cropping of the images to have the human faces occupy most of the frame, as well as the transformation into gray-scale for images which were originally colored.

As stated, we chose this dataset in part because of its ease of use, being already very clean and standardized, as well as pre-organized into seven folders representing the seven emotions to be detected. It was also easy to access it, the original dataset being uploaded for a Kaggle competition which has since finished. The fact that the images were gray-scale also meants that only a single channel was necessary as input for our neural networks, which would reduce the complexity of the calculations, and so did the fact that the images were small.

Other noteworthy datasets that could have been used, but that we chose not to consider for the reasons stated above, were the AffectNet [11] dataset (which was difficult of access, due to OneDrive limitations, and much more heavy, having over one million color images), and the Google facial expression comparison [12] dataset. Other available datasets containing multi-modal information (such as electroencephalogram (EEG) and electrocardiogram (ECG) signals [13]) were also immediately discarded by us, considering the added complexity.

5. Implementation

Our project was developed using the PyTorch framework and thus, we tried to exploit all of its functionality as much as possible. We started by creating a Class that would transform our dataset into tensors, then used data loaders so as to already split the dataset into train and validation subsets, and into batches that could be processed by the Network. As we will explain in further details afterwards, this Class returned eventually images that have one single channel but, since we used also a pre-trained network through transfer learning, that was trained with RGB inputs (i.e. three channels) we had to implement another class that returned a three channels output image although our images were gray scale.

In a first step, we decided to build our own Neural Network from scratch. We began with a Network with only two convolutional layers followed by one Max Pooling layer each and the ReLu function and two fully connected layers in the classification part of the network. The structure of our first network is summarized in figure 3.

As shown in figure 3, we have a max pooling operation that is following each convolution layer. This was made to reduce the computational power needed to train the network as well as reduce overfitting. Moreover, in each convolutional layer we tried to add same padding and stride, but noted that they weren't necessary for the sake of the accuracy of the model so we decided to remove them eventually.

After a lot of tries we ended up building a second network that added more complexity to the previous mentioned and it is represented in figure 4.

With this CNN we added one convolution operation followed by another Max Pooling layer and an additional fully connected layer within the classification section of our network.

The reason behind the choice of adding more complexity to our model was due to the fact that our accuracy, even in the training set, was not growing at all.

We tried a lot of combination with the two aforementioned network, in terms of both optimizers, learning rates, and momentum when it came to choose the Stochastic Gradient Descent as optimizer. We didn't use the weight decay since our models were not performing very well already. We applied the same reasoning behind the choice of not adding a dropout layer since we could not even reach a high accuracy on the training set.

In a second step, and in view of the poor results on our "hand-made" CNNs, we decided to turn to transfer learning. Indeed, since the results of our models' output were indistinguishable from random classification, we wanted to see how a hundred million parameters network such as VGG-11 and then VGG-16, could perform with our dataset.

In order to do so we had to modify the return argument of our EmotionData class since it was returning 1 channel images such as the shape of our input data and the labels. We added a repeat methods for the images so as to have a three channels image as input for the new model. This was necessary since the VGGs were trained on the ImageNet dataset that has a three channels input and a resolution of 224x224.

As shown in the figure 5, the VGG-16 is a very deep neural network which, in summary, is composed of five convolutional blocks which each have two convolutional operations in the first two layers and three in the last three ones followed by Max Pooling operations layers. The second section of the network has three fully connected layer that we did not include, but replaced with a fully-connected output layer to have our output dimension, 7, rather than the standard, 1000. Not that the VGG-11 architecture is roughly the same, the only difference being that each convolutional block contains one fewer convolutional layer, bringing the total number of layers to 11 instead of 16 – hence the naming.

Starting with the VGG-11 we tried different optimizers as well as learning rate but we decided to keep frozen the convolutional layers so as to train only the three last dense layers. The results weren't satisfactory so we decided to move to another network with a different approach.

When implementing the VGG16 we wanted to get the best accuracy possible and thus, we decided to use the network's pre-trained weights on ImageNet as initialization, before making it learn without freezing any layer.

After doing all the adjustments such as learning rates or optimizers we derived that this was, even though the most computationally heavy, the best choice for the sake of our accuracy.

6. Results

In this section we discuss the results of the models previously described and we compare the results in terms of accuracy.

Before explaining the results of each model we implemented we wanted to tell you the common factor they all have which is a train batch size of 32 and a test batch size of 64.

To begin with we talk about the first model we built from scratch since (that we nick-named "CNN1"), even though it

Figure 3. Our first CNN

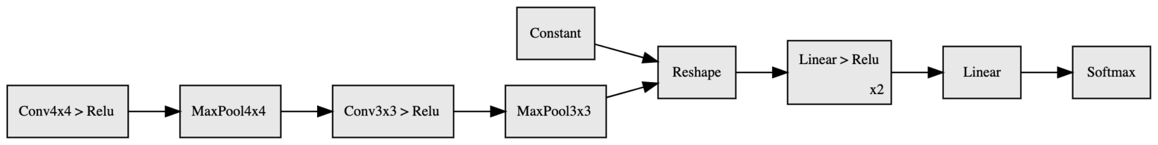


Figure 4. Our second CNN

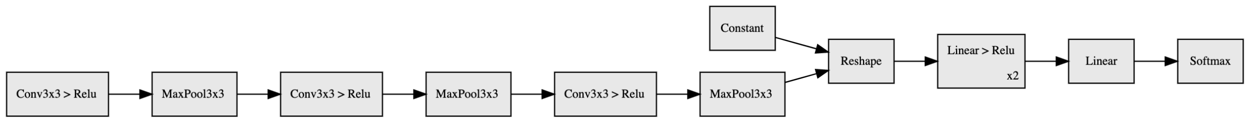


Figure 5. Visual representation of the VGG-16 neural net architecture

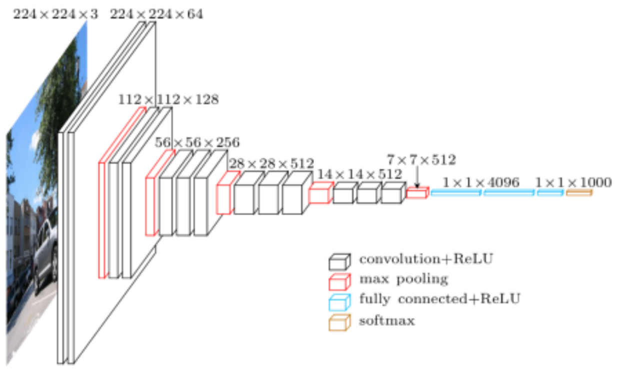
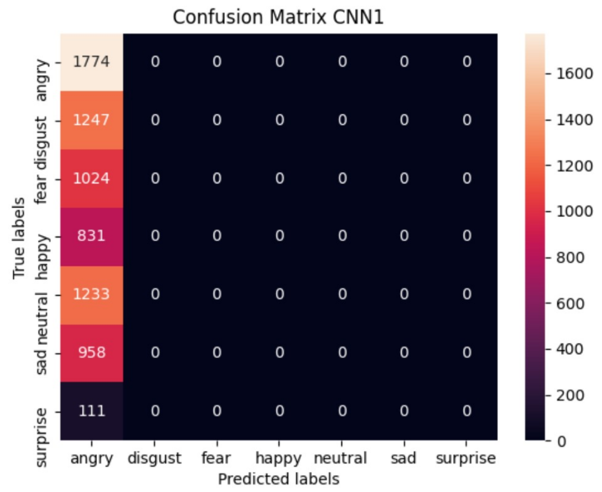


Figure 6. Confusion Matrix CNN1



	angry	disgust	fear	happy	neutral	sad	surprise
angry	1774	0	0	0	0	0	0
disgust	1247	0	0	0	0	0	0
fear	1024	0	0	0	0	0	0
happy	831	0	0	0	0	0	0
neutral	1233	0	0	0	0	0	0
sad	958	0	0	0	0	0	0
surprise	111	0	0	0	0	0	0

was the simplest one, it gave us the best accuracy in the validation set out of our two "hand-made" models.

The configuration of hyperparameters that allowed us to reach the accuracy of 26% (i.e. 12% more than random classification) was using Stochastic Gradient descent as optimizer with 0.8 momentum, a learning rate of 10^{-4} and cross entropy as loss. The confusion matrix of this results is shown in figure 6.

We can clearly see how much our model can only recognize the first class which is the most numerous and could not reach a very high accuracy even in the training set that was almost as the one of the validation.

The second model we show the results of is the one that has been trained with the base of the VGG-11 with the convolutional layers frozen and only the last dense layers free to update the weights.

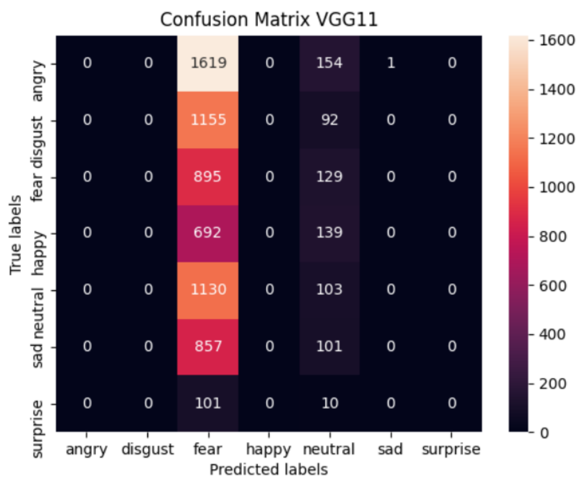
The optimizer we chose for this model was Adam with a learning rate of 10^{-4}. The accuracy we ended up with was not much higher than the showed in the previous model. The confusion matrix of this model is showed in figure 7.

This time the most recognized class is the emotion fear followed by the neutral one. But still, we are not even close to reach a good model that can gives us a reliable classification.

Lastly, we finally implemented the model that gave us the best accuracy, a fine tuned VGG-16 in which we allowed all the weights to be updated during the learning process and thus, we let the feature extraction section of the Network adapt to our dataset, instead of it being trained on a totally different images set.

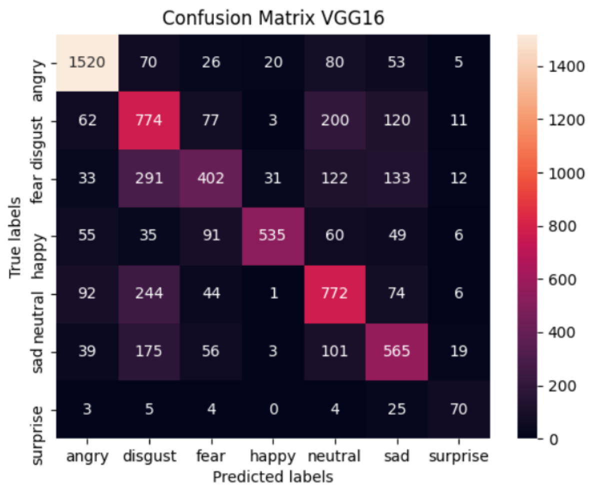
As we did for the models we presented before, we tried a lot of possible combinations for the optimizers as well as learning rates and momentum. The configuration that gave us the best accuracy on the test set was eventually the one trained with Stochastic Gradient Descent as optimizer, 10^{-2}

Figure 7. Confusion Matrix VGG11



as learning rate and a momentum of 0.8. In the figure 8, we show the confusion matrix for this model.

Figure 8. Confusion Matrix VGG16



As we could expect from a model that doesn't predict – almost – randomly as the ones showed before, we can see that the prediction is more accurate when it comes to predicting the most numerous labels and vice versa.

Even though we reach a satisfactory accuracy on the validation set we could notice that our model was overfitting, since we let over hundred millions parameters to learn our very small dataset. This can be seen in the learning process' visualization shown in figure 9 (in blue).

Through a snapshot captured from the Weight and Biases framework that helped in the process of training and validating the models as well as visualizing the results of accuracy and losses, how our best model is clearly overfitting reaching a 64% accuracy on the validation set whereas on the training set, it is almost reaching 98%.

Within the same figure we show different configurations of learning rates and optimizers for the same model and for comparison we added the accuracy of the first model that we implemented. Since we added early stopping while training the VGG16 with a patience of 10 epochs, the corresponding curve is shorter than for our earlier models.

7. Conclusion

Using our modest knowledge of deep neural networks, we aimed to compare how different architectures of neural networks would achieve at the task of classifying images of human faces into seven different emotions. Although the results of our earlier tries proved unsuccessful, we managed to achieve a respectable validation accuracy using transfer learning, as well as a more powerful computing device.

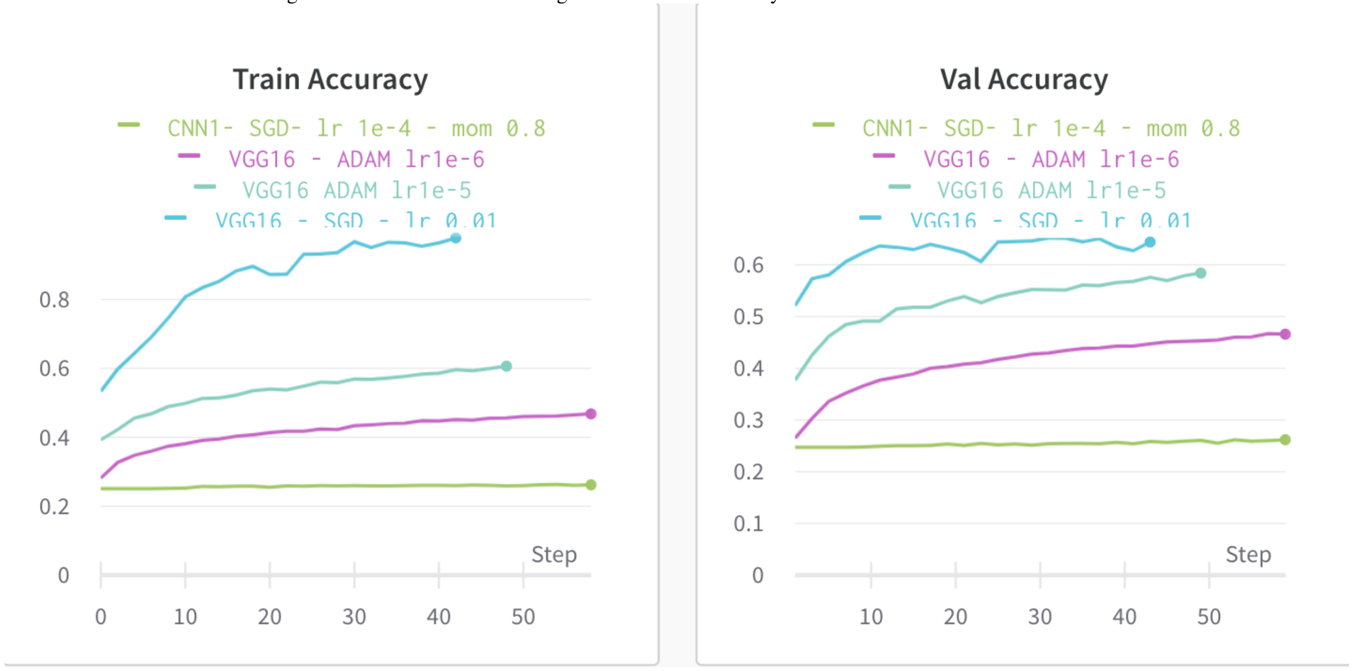
As we expected, the task of implementing the algorithms and performing the computations proved to be difficult. During the course of this project, we not only used our knowledge of data analytics in order to assess the results and estimate which method would be most fruitful to achieve our goal, but we also had to learn to use the PyTorch framework for deep learning. This framework proved to be more difficult than expected though, as it is relatively low-level compared to other frameworks like Keras, which might have been easier to implement and perform Cloud Computing on. Indeed, although we managed to use Google's collab tool to train early networks, we were quickly limited by the free trial version of it. Google Cloud Platform services would have been a good option to train the networks, but required knowledge that we did not have to configure it using torch. For future deep learning tasks, we will consider the keras framework as it is more easily implementable on Google Cloud Platform, which allows access to remote GPUs for very little investment (the equivalent of the first $300 being offered for every new account).

Despite these limitations, we managed to find one configuration of neural network that yielded acceptable results. Considering the little amount of testing that we could perform, this result was actually pretty impressive, and we are confident that given more time, resources, and focus, we could achieve even better results.

Regarding the overall picture, it is true that we limited our project to classification of gray-scale images. However, given the fact that these gray-scale images had to be split into 3 channels regardless, we are confident that given color images, we could achieve similar results using the same model architecture. In addition, since our images were small, it can be assumed that feeding larger images would only provide better accuracy, as more information would be available to our network. Hence, we are satisfied with our model as we think it can be used for more general inputs, our inputs being the main bottleneck of this project.

As closing thoughts, this project allowed us to explore more deeply the concepts learned in the last part of the lectures, getting our hands dirty with deep neural networks,

Figure 9. Visualization of training and validation accuracy evolution for VGG-16 and CNN1



using a brand new framework in the form of PyTorch. Through our preliminary research, as well as in trying to figure out the solutions to the problems encountered in our work, we learned a lot more than could have been learned from a less ambitious project. Although we did not manage to implement the last part of our initial project idea – which was to link the networks with a graphical interface showing corresponding emojis – , we feel that we achieved the largest part of the task that we had defined with relative success.

8. Acknowledgments

We want to thank Alberto Bellini for letting us use via remote control his Nvidia GeForce RTX 3090 GPU without which we would not have been able to train the model that eventually gave us the best accuracy.

References

[1] Computer vision: What it is and why it matters.

[2] Imagenet large scale visual recognition challenge (ilsvrc).

[3] Fer-2013.

[4] P Ekman and WV Friesen. Constants across cultures in the face and emotion. *Journal of personality and social psychology*, 17(2):124—129, February 1971.

[5] David Matsumoto. More evidence for the universality of a contempt expression. *Motivation and Emotion*, 16(4):363—368, December 1992.

[6] Zhengyou Zhang, Michael Lyons, M. Schuster, and Shigeru Akamatsu. Comparison between geometry-based and gabor wavelets-based facial expression recognition using multi-layer perceptron. pages 454 – 459, 05 1998.

[7] Oh I.-S. Kim K.-M. Nguyen H.-D., Yeom S. and Kim. Facial expression recognition using a multi-level convolutional neural network. *Proceedings from the International Conference on Pattern Recognition and Artificial Intelligence*, pages 217–221, 2018.

[8] I Gede Putra Kusuma Negara, Jonathan Jonathan, and Andreas Lim. Emotion recognition on fer-2013 face images using fine-tuned vgg-16. *Advances in Science, Technology and Engineering Systems Journal*, 5:315–322, 01 2020.

[9] Yichuan Tang. Deep learning using linear support vector machines. 2015.

[10] Wikipedia contributors. Convolutional neural network — Wikipedia, the free encyclopedia, 2021. [Online; accessed 15-June-2021].

[11] Ali Mollahosseini, Behzad Hasani, and Mohammad H. Mahoor. Affectnet: A database for facial expression, valence, and arousal computing in the wild. *IEEE Transactions on Affective Computing*, 10(1):18–31, Jan 2019.

[12] Aseem Agarwala and Raviteja Vemulapalli. A compact embedding for facial expression similarity. 2018.

[13] Stamos Katsigiannis and Naeem Ramzan. DREAMER: A Database for Emotion Recognition through EEG and ECG Signals from Wireless Low-cost Off-the-Shelf Devices, April 2017.