Facial Expressions and CNN

Xavier Marrugat

***Abstract*—In this paper it will be discussed different Convolutional Neural Network models for facial expression recognition. Making use of Fer2013 data set we will be able to detect which expression is in an image.**

I. Introduction

Nowadays image recognition is being used in all kind of subjects like in automotive industry, health care, smart cities, etc. But could it be possible to detect which expression is expressing someone? What does it require?

Machine learning can makes easy this work using neural networks. Here we are going to build a Convolutional Neural Network that will learn how to classify 7 different expressions: anger, disgust, fear, happiness, sadness, surprise and neutral. Also it will be discussed what influences has each hyperparameter to the performance of the network and which could be the best model of the CNN for this purpose.

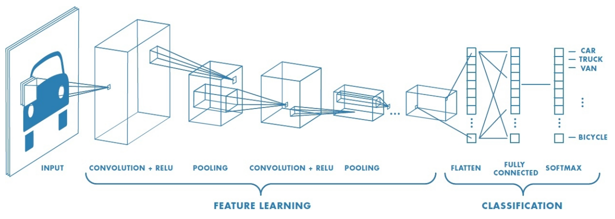


Fig. 1. Basic example of a CNN structure[2]

A. Related work

As related work it can be found a great paper called "Deep Facial Expression Recognition: A Survey"[7]. There it is shown and compared all the facial expressions data sets. Also it is compared different models and methods to implement CNNs.

II. Data

There are a lot of data sets of facial images, ones that have been made in a laboratory, those that have been created by spontaneous reactions and other ones that the face is not totally aligned and has some degree. In this paper it will be used the well known dataset Fer2013 provided by a kaggle competition[6]. This dataset has 28709 images of faces for training, 3590 as public test and 3590 more as private test. Also it has both spontaneous expression and posed ones.

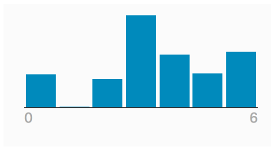


Fig. 2. This shows the number of sample in the whole dataset of each facial expression

The images are specified in a csv file where the first column is the expression number (from 0 to 6) as described before, the second one is the pixels of the image and the third column specifies whether the image is for training or public or private testing.

The images have a size of 48x48 and as they are provided as a list of numbers we will have to reshape them before we use them to feed the network. Also if we want to know how the images look like we could plot them, although it is not necessary.

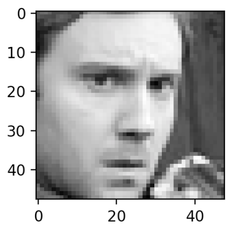


Fig. 3. Plotting a 48x48 image from a list of pixels

III. Convolutional Neural Networks

The reason this type of neural network has been chosen is that it works very well with image recognition as it extracts features in them that are used later to feed the network.

A basic convolutional network has 4 operations: convolution, non-linearity, pooling (or sub sampling) and classification (fully connected layer).

A. Understanding an image

Before we discuss each operation of the CNN we have to understand how an image is represented.

Every image can be seen as a matrix of pixel values, as we have seen before. Each pixel has a value from 0 to 255. Another image's feature are the channels. Those are used to represent a component of an image. For example a picture taken with a camera will have three channels: red, green, blue (RGB). Each channel can be understood as a two-dimension matrix with its pixels having values from 0 to 255.

Note: The data set we are using only has 1 channel as they are in gray-scale.

B. Convolution

This operation of the network is responsible of extracting features from an image. To make it understandable lets assume we have an image of size 5x5 pixels. Then it uses another two-dimension array (called *kernel, filter or feature detector*) to extract the features, which in our example would be of size 3x3. This second matrix will be slide over the image by one pixel (the number of pixels to slide are called *stride*) and for every position a multiplication of the two matrices will performed to finally make the sum of all elements to get a final integer which will be stored in another matrix called *convoluted feature, activation map or feature map*. So we could say that filters act as a feature detection.

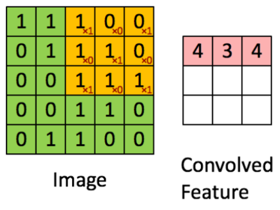


Fig. 4. Example of an 5x5 image being applied a 3x3 filter[5]

The values of the filters get learned during the training process but we still have to specify the number of filters, their size and the stride value. Note that if we have a bigger number of filter then we will get more features extracted, which means more feature maps. Also having a larger stride will produce smaller feature maps.

Also sometime we will need to pad the input image with zeros around so we can apply the filter without leaving the borders. This parameter is called *zero-padding* and it is also useful to to control the size of the feature maps.

C. Non-linearity

After every convolution operation we have to perform another one called ReLU (Rectified Linear Unit). ReLU is a non-linear operation which replaces non-positive values in the feature map by a zero. The purpose of this is to change the linear output from the convolution operation to a non-linear one as most real-world problems are of the second type.

Although ReLU is not the only non-linear function that can be used, there exist *tanh or sigmoid* for example, it has been found to perform very well with images.

D. Pooling

Pooling or sub-sampling is used to reduce the dimensions of each feature map generated from the convolution operation. It is used for several reasons, the first one is to make the input more manageable as the size is reduced. Also at it reduces the number of parameters and computations in the network it allows to control the overfitting. Another good reason of using pooling is that makes the network invariant to small transformations, distortions and translations in the input image. The last reason is that helps us arrive to an scale invariant representation of an image meaning it would detect object no matter where they are located.

We have different types of pooling: max, average or sum of all elements. Max pooling can extract very well edges and that is why it is mostly used in image recognition that is why it will be used in this paper.

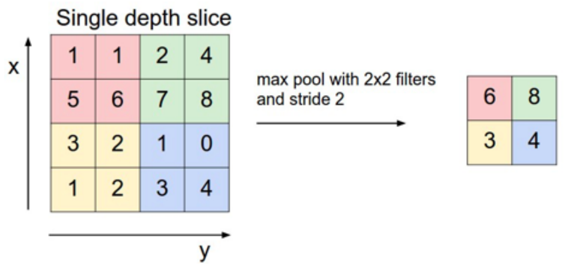


Fig. 5. This is an example of how max pooling works [8]

E. Classification

In this part of the network is done the classification which means that from an input image we will get the expression that the machine has detected. This can be referred as the *dense layer or fully connected layer* which in fact is a very basic neural network with weights. It usually has a large number of input nodes but the number of output nodes are exactly the same as the possible ways to classify an image. This means that if we are designing our CNN to classify between six different objects we will have a total number of six output nodes in the dense layer. Also instead of getting a clear result we will get a probabilistic number indicating how likely the input image is to be each different object. It is understood that a well designed and trained network will output a bigger value to the label that it is supposed to be the classified object.

IV. Method

There are a several libraries for machine learning purposes. The one used here is called Tensorflow. Its name comes from the behavior of tensors passing from one node to another in a network. This library developed by Google makes easy the task of implementing a convolutional neural network.

There are two main types of this library, the CPU and the GPU. Unfortunately the computer it is being used in this paper only supports the CPU tensorflow type. This means it will take longer to train the model.

The *main.py* file has mainly 3 parts. First it gets the data from the csv file and divides it in training and testing. Each one has an array with the images, which are arrays of pixels, and an array with the labels which indicates what expression represent each image.

The second part process each image to reshape them into a 48x48 array. Also they are normalized to 1 so instead of ranging from 0 to 255, pixels would have values from 0 to 1.

The third part of the code is the actual creation of the convolutional neural network. Following a model we create each operation seen before (convolution, non-linearity, pooling and dense). Also it is added a dropout generalization of 0.4 which means 40% of the elements will be randomly dropped out during training. This is done to avoid overfitting. In addition an estimator is added to to calculate loss, accuracy and evaluate the model. When training checkpoints will be stored in a folder in order to restore learned values or to follow with the training.

There are several CNN models and it is quite difficult to choose one that will benefits you in the first instance but it is known that the deeper the model is the better results you will get. As commented before, although the computer where the models are trained does not have a lot of computational power we will be testing 4 different models.

A. Hyperparameters

Hyperparameters are those parameters that will not be changed during the training of a neural network like weights and, in this case, the filter values would do. So they are quite important because as they can have a great impact in the results we get.

In a convolutional neural network we have 4 hyperparameters to take into account:

1) **Kernel size**: the dimensions of the filters we are going to use
2) **Number of filters**: number of filters to be used to extract features. Their values will be learned while training.
3) **Stride**: number of steps the filter will move.
4) **Pool size**: dimensions of the resulting pooling matrix.
5) **Padding**

In each convolution operation we have to define this parameters so that are some key aspects we will be changing from one model to another.

B. Model 1 - main.py

Firstly we are going to test a basic set up. It will do a convolution operation, then max-pooling, another convolution operation, max-pooling and finally the results will be passed to a fully connected network with 1024 nodes. The hyper parameters are set like this:

1) **Kernel size**: it has dimensions of 5x5 in both convolution operations.
2) **Number of filters**: In the first convolution it uses 32 filters and in the second one 64.
3) **Stride**: it is in both cases 2.

4) **Pool size**: it is in both cases 2 outputting the result with half of its dimensions.
5) **Padding**: it is always set so the output from the convoltional operation has the same size as the input image.
6) **First layer of fully connected nodes**: 1024

Also the non-linear function is in all cases the same, ReLU, which we have talked previously. Also we have set the training mode to make 10.000 iterations.

C. Model 2- main2.py

In this second model we have just changed one parameter, the number of filters used in the first convolution operation. With this changed we should extract more features from the initial image getting better results. So the settings are as follow:

1) **Kernel size**: it has dimensions of 5x5 in both convolution operations.
2) **Number of filters**: In both convolutions are being used 64 filters.
3) **Stride**: it is in both cases 2.
4) **Pool size**: it is in both cases 2 outputting the result with half of its dimensions.
5) **Padding**: it is always set so the output from the convolution operation has the same size as the input image.
6) **First layer of fully connected nodes**: 1024

As before the non-linear functionis in all operations the same, ReLU. Again we have set the training mode to make 10.000 iterations.

D. Model 3- main3.py

Here we have changed 3 hyperparameters: the number of filters in each convolution operation and the number of nodes of the dense network. As before we will get more features from the initial image but also in the second convolution more features will be extracted. This should impact the performance but also get better results.

1) **Kernel size**: it has dimensions of 5x5 in both convolution operations.
2) **Number of filters**: In first convolution we have 64 filters and 128 in the second one.
3) **Stride**: it is in both cases 2.
4) **Pool size**: it is in both cases 2 outputting the result with half of its dimensions.
5) **Padding**: it is always set so the output from the convolution operation has the same size as the input image.
6) **First layer of fully connected nodes**: 2048

With ReLU as non-linear function and the test is set to make 10.000 iterations.

E. Model 4 - main4.py

In this last model we have not only changed hyperparameters but the whole structure of the network. First we have two convolution operations, then the max-pooling, 2 more convolutions, again max-pooling to minimizing the dimensions of the sample and finally the dense layer.

1) **Kernel size**: it has dimensions of 5x5 in all convolution operations.
2) **Number of filters**: First we have 32, then 64, 128 and finally 256.
3) **Stride**: in all cases is equal to 2.
4) **Pool size**: it is equal to 2 in booth max-pooling operations outputting the result with half of its dimensions.
5) **Padding**: it is always set so the output from the convolution operation has the same size as the input image.
6) **First layer of fully connected nodes**: 1024

With ReLU as non-linear function and the test is set to make 10.000 iterations.

This model is supposed to be the best one really extracting features from the images that can be useful to feed into the classifier. So in this case we could expect to work better than the other 3 models.

V. Results and analysis

The winner of the kaggle competition got an accuracy of 71%, that was 6 years ago. Nowadays there are models that can get nearly a 94% of accuracy or even better using Deep Residual Learning[4].

It is clear then that with our models proposed initially, which are no as much computational difficult and with an easy structure, is difficult to get good results. Also taking into account the computer that is performing the tests.

A. Model Comparison

The time has been an extremely important factor. As you can see in the plot every time we have added a minimal change the time to train 10.000 iterations has increased a lot. In fact, for the fourth model it has nearly being impossible to finish all the predefined iterations.

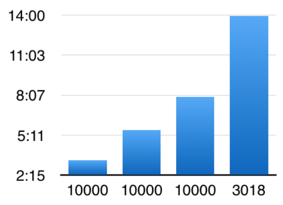
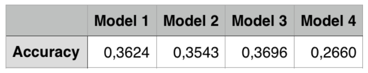


Fig. 6. Amount of time taken to train models

Note: the amount of time is approximated as the system could have done some unrelated background work while training the models.

In terms of accuracy we see the firsts three models has very similar scores. Although it is true that doubling the number of filters in the next layer of convolution works better than having the same number of filters, like in model 2. The last one has the lowest value but it has done nearly 3 times less iterations.



	Model 1	Model 2	Model 3	Model 4
Accuracy	0,3624	0,3543	0,3696	0,2660

Fig. 7. Accuracy of each model after the above iterations

VI. Conclusion

We have confirmed that going deeper when it comes learning features from an image is better but it impacts negatively the amount of time to train the model. So it is nearly mandatory to use and have a specific computer using a powerful GPU to actually be able to train a model whose accuracy is good enough for a real-world environment.

We have seen that hyperparameters has some importance in the results but the real key is which model is being used in the Convolutional Neural Network. Also the more iterations we perform the more accurate the results would be as the weights will be better balanced but as always time is a key factor here.

Although maybe we have not scored well in terms of accuracy we are not far from getting better results. The only thing we really need is time, time to train the fourth model or money to by a better computer.

References

[1] *An Intuitive Explanation of Convolutional Neural Networks*. URL: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[2] *Architecture of a CNN*. URL: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html.

[3] Daphne Cornelisse. *An intuitive guide to Convolutional Neural Networks*. URL: https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050.

[4] *Deep Residual Learning for Image Recognition*. URL: https://arxiv.org/pdf/1512.03385.pdf.

[5] *Feature extraction using convolution*. URL: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution.

[6] Ian Goodfellow et al. *Challenges in Representation Learning: A report on three machine learning contests*. 2013. URL: http://arxiv.org/abs/1307.0414.

[7] Shan Li and Weihong Deng. *Deep Facial Expression Recognition: A Survey*. URL: https://arxiv.org/pdf/1804.08348.pdf.

[8] *Max pooling takes the largest values*. URL: https://cs231n.github.io/convolutional-networks/.