Facial Emotion Recognition using Convolutional Neural Networks

Charif Chaibainou
CentraleSupelec - Paris
charif.chaibainou@student.ecp.fr

Abstract

Emotion recognition is essential in human/machine interactions, whether it is in marketing, advertising, or in medical applications (like gauging the emotions of autistic children). This project, undertaken in the Deep Learning course at CentraleSupelec, aims at studying the performance of deep neural networks -convolutional ones in particular-, in the detection of basic emotions from frontal face images. This project is not about outperforming the state of the art results on emotion recognition (which is 75.2% of test accuracy on the FER2013 dataset [5]), but rather (due to time and material constraints) analyzing the impact of different variations on the model's performance. For this task, I used deep neural networks, centered around the VGG16 architecture, with the FER dataset (taken from a 2013 Kaggle contest). The different results are shown and interpreted. The influences of the hyperparameters, regularization, pretrained weights, etc. are analyzed.

1. Introduction

1.1. Objectives and Methodology

In this paper, I want to test the VGG architecture in the emotion recognition problem. Here is the methodology I applied :

- Firstly, I tried to reproduce the VGG architecture from scratch (same layers except the output softmax one, with the right number of classes), train it on the training set and tune the hyperparameters and regularize to increase the validation accuracy, and then test it on an unseen dataset.
- Then, I would compare this model with the pretrained model on ImageNet, to show the influence of pre-trained weights and fine-tuning. - Finally, I'll explore some methods to improve the model, such as using ensembles and propose other ways to improve my results.

1.2. The VGG16 structure

VGG is a CNN designed by Oxford's Visual Geometry Group. 3x3 convolution and 2x2 pooling layers are used in this network.

VGG released two different CNN models, specifically a 16-layer model and a 19-layer model. In this paper, I use the 16-layer one (arbitrary choice). This model has shown good results in image recognition, and its straightforwad architecture made me choose it.

The model architecture is shown in the figure 1.

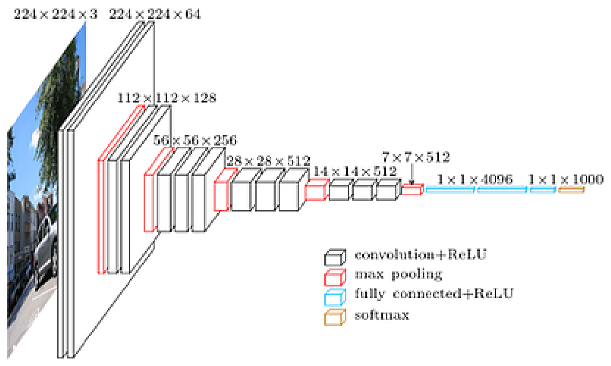


Figure 1. The VGG16 network architecture

1.3. ImageNet

ImageNet is a research project to develop a large database of images with annotations, e.g. images and their descriptions.

The images and their annotations have been the basis for an image classification challenge called the ImageNet Large Scale Visual Recognition Challenge or ILSVRC since 2010. The result is that research organizations battle it out on predefined datasets to see who has the best model for classifying the objects in images.

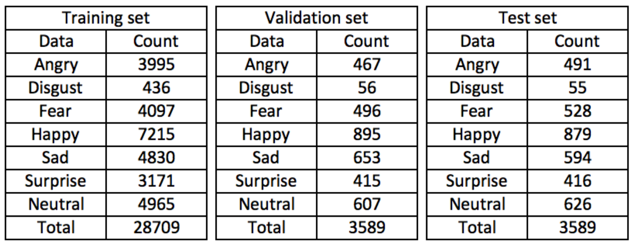
For the classification task, images must be classified into one of 1,000 different categories.

For the last few years very deep convolutional neural network models have been used to win these challenges and results on the tasks have exceeded human performance.

1.4. Data visualisation

For this paper, I used the FER dataset, taken from a 2013 Kaggle competetion. 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. I have a training, validation and test set, as shown in figure 2.

N.B : As we can see, the "disgust" class is largely under-represented. I chose to keep it as it is and not do any data augmentation, as it is not the main goal of this paper. However, we will see that this unbalance will not be a hindrance to our model performance.



Training set		Validation set		Test set	
Data	Count	Data	Count	Data	Count
Angry	3995	Angry	467	Angry	491
Disgust	436	Disgust	56	Disgust	55
Fear	4097	Fear	496	Fear	528
Happy	7215	Happy	895	Happy	879
Sad	4830	Sad	653	Sad	594
Surprise	3171	Surprise	415	Surprise	416
Neutral	4965	Neutral	607	Neutral	626
Total	28709	Total	3589	Total	3589

Figure 2. Data count

Before getting to the first model, let's have a look at our data (figure 3). The images are all in gray-scale, more or less face-centered.

Figure 3. Data overview (128 first images)

2. The from-scratch model

At first, the idea was to reproduce the exact architecture of the VGG16 model, with replacing just the last layer (which was a softmax dense layer with 1000 classes) by a softmax layer with my 7 classes of emotions.

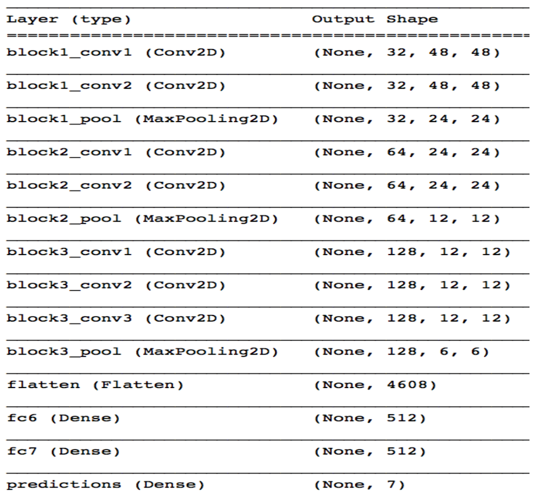
However, with this model, the validation accuracy remained constant epoch after epoch... I suspected either a dying RelU (which could have been solved by a leaky Relu activation function) or a vanishing gradient issue; the results were indeed barely better than a random choice. So I had to modify the model as follows (structure in figure 4) :

- I removed the two last convolutional/max-pooling blocks;

- I decreased the filters (starting with 32 filters instead of 64 in the first block);

- I reduced the number of neurons of the classification block (from 4096 to 512).

The corresponding file in the code is 'from_scratch.py', the model is stored in 'naive_v1.h5'



```
Layer (type)                   Output Shape
==================================================
block1_conv1 (Conv2D)          (None, 32, 48, 48)
block1_conv2 (Conv2D)          (None, 32, 48, 48)
block1_pool (MaxPooling2D)     (None, 32, 24, 24)
block2_conv1 (Conv2D)          (None, 64, 24, 24)
block2_conv2 (Conv2D)          (None, 64, 24, 24)
block2_pool (MaxPooling2D)     (None, 64, 12, 12)
block3_conv1 (Conv2D)          (None, 128, 12, 12)
block3_conv2 (Conv2D)          (None, 128, 12, 12)
block3_conv3 (Conv2D)          (None, 128, 12, 12)
block3_pool (MaxPooling2D)     (None, 128, 6, 6)
flatten (Flatten)              (None, 4608)
fc6 (Dense)                    (None, 512)
fc7 (Dense)                    (None, 512)
predictions (Dense)            (None, 7)
```

Figure 4. From-scratch model architecture

2.1. Results

With 100 epoch of training/validation, ran on Google Colaboratory tool, the test accuracy was equal to 56.9%. The confusion matrix is as follows (figure 5).

N.B : Surprisingly, the 'disgust' class, which is widely under-represented, is predicted better than some other classes.

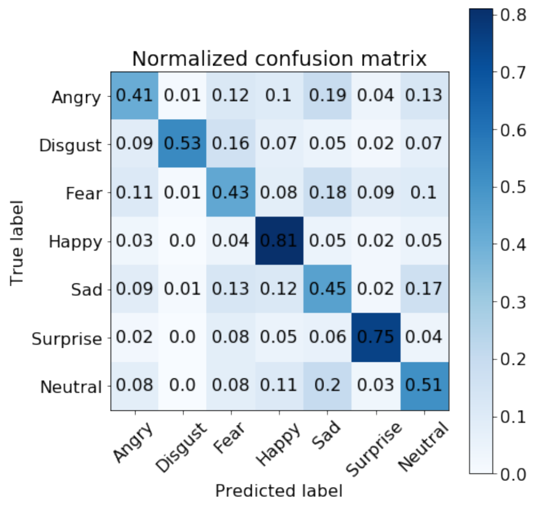
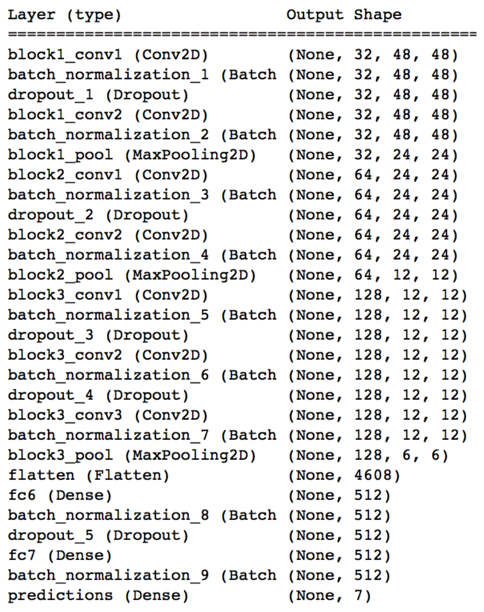


Figure 5. Confusion matrix - From-scratch model

2.2. Tuning the from-scratch model

I tuned some parameters as follows :
- I included Dropout layers, of increasing size;
- I added a Kernel Constraint in the convolutional layers;
- I included an L2 regularizer;
- I added Batch Normalisation layers throughout the model.

The structure of the tuned model is shown in figure 6.



```
Layer (type)                 Output Shape
=================================================
block1_conv1 (Conv2D)        (None, 32, 48, 48)
batch_normalization_1 (Batch (None, 32, 48, 48)
dropout_1 (Dropout)          (None, 32, 48, 48)
block1_conv2 (Conv2D)        (None, 32, 48, 48)
batch_normalization_2 (Batch (None, 32, 48, 48)
block1_pool (MaxPooling2D)   (None, 32, 24, 24)
block2_conv1 (Conv2D)        (None, 64, 24, 24)
batch_normalization_3 (Batch (None, 64, 24, 24)
dropout_2 (Dropout)          (None, 64, 24, 24)
block2_conv2 (Conv2D)        (None, 64, 24, 24)
batch_normalization_4 (Batch (None, 64, 24, 24)
block2_pool (MaxPooling2D)   (None, 64, 12, 12)
block3_conv1 (Conv2D)        (None, 128, 12, 12)
batch_normalization_5 (Batch (None, 128, 12, 12)
dropout_3 (Dropout)          (None, 128, 12, 12)
block3_conv2 (Conv2D)        (None, 128, 12, 12)
batch_normalization_6 (Batch (None, 128, 12, 12)
dropout_4 (Dropout)          (None, 128, 12, 12)
block3_conv3 (Conv2D)        (None, 128, 12, 12)
batch_normalization_7 (Batch (None, 128, 12, 12)
block3_pool (MaxPooling2D)   (None, 128, 6, 6)
flatten (Flatten)            (None, 4608)
fc6 (Dense)                  (None, 512)
batch_normalization_8 (Batch (None, 512)
dropout_5 (Dropout)          (None, 512)
fc7 (Dense)                  (None, 512)
batch_normalization_9 (Batch (None, 512)
predictions (Dense)          (None, 7)
```

Figure 6. Tuned from-scratch model architecture

The corresponding file in the code is 'from_scratch_tuned.py', the model is stored in 'naive_tuned_v1.h5'
With these changes, the test accuracy increased to 61.5%, which shows the influence of regularization and batch normalisation mainly. The model is less prone to overfitting and hence able to generalise better on the test set. The corresponding confusion matrix is shown in figure 7. The improvement is for all the classes except "fear", which is confused with "sad" or "angry".

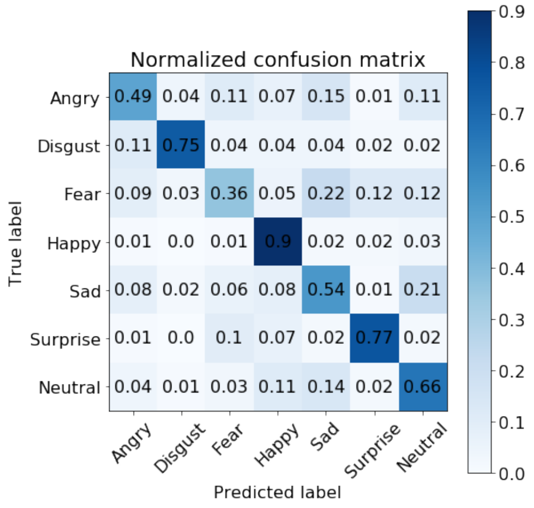


Figure 7. Confusion matrix - Tuned from-scratch model

Let us now see the performance of a pre-trained model.

3. The pre-trained model

I used the Keras VGG16 model, with pretrained weights on ImageNet. I tried to replace only the last layer of the original VGG16 by my own 7-class softmax output layer, but it did not give better results than the from-scratch model.

First of all, the ImageNet dataset is made of RGB images... so to cope with my grayscale dataset, I copied the data of my images (1 channel) over 3 channels (with the Numpy function 'stack').

So I decided to test multiple versions of the original VGG16, by removing some layers and including my own classifier on top. Indeed, the last layers of the VGG16 model learned some very specific features, which do not correspond to face images, so they are not needed for the model.

The best performing model has the following structure (figure 8) : I removed the last convolutional block and the classifier, and replaced it by the fully-connected layer of the from-scratch tuned model. I also froze the 3 first convolutional blocks weights, so that they are not modified. Only the fourth block and the fully connected block are trained.

```
Layer (type)                 Output Shape
input_2 (InputLayer)         (None, 48, 48, 3)
block1_conv1 (Conv2D)        (None, 48, 48, 64)
block1_conv2 (Conv2D)        (None, 48, 48, 64)
block1_pool (MaxPooling2D)   (None, 24, 24, 64)
block2_conv1 (Conv2D)        (None, 24, 24, 128)
block2_conv2 (Conv2D)        (None, 24, 24, 128)
block2_pool (MaxPooling2D)   (None, 12, 12, 128)
block3_conv1 (Conv2D)        (None, 12, 12, 256)
block3_conv2 (Conv2D)        (None, 12, 12, 256)
block3_conv3 (Conv2D)        (None, 12, 12, 256)
block3_pool (MaxPooling2D)   (None, 6, 6, 256)
block4_conv1 (Conv2D)        (None, 6, 6, 512)
block4_conv2 (Conv2D)        (None, 6, 6, 512)
block4_conv3 (Conv2D)        (None, 6, 6, 512)
block4_pool (MaxPooling2D)   (None, 3, 3, 512)
flatten (Flatten)            (None, 4608)
fc6 (Dense)                  (None, 512)
batch_normalization_1 (Batch (None, 512)
dropout_1 (Dropout)          (None, 512)
fc7 (Dense)                  (None, 512)
batch_normalization_2 (Batch (None, 512)
predictions (Dense)          (None, 7)
```

Figure 8. Pretrained model architecture

3.1. Results

With 100 epoch of training/validation, ran on Google Colaboratory tool, the test accuracy rose to 65.9%. The confusion matrix is as follows (figure 8).

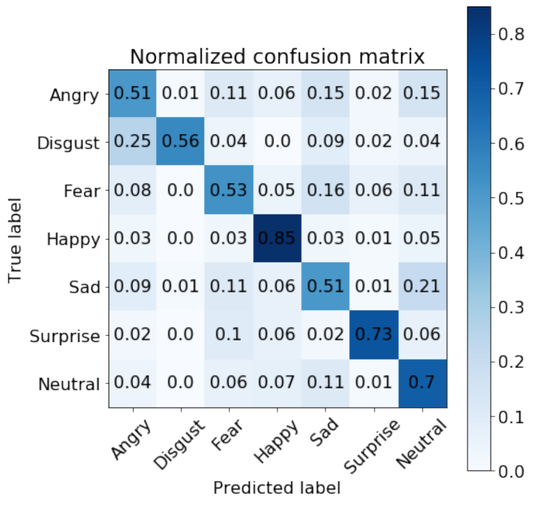


Figure 9. Confusion matrix - Pretrained model

The corresponding file in the code is 'model_pretrained_v6.py', the model is stored in 'pretrainedv6.h5'

The results are more homogeneous among the classes, with excellent prediction for the 'happy' class, followed by surprise and neutral.

These results show that the first convolutional blocks trained on ImageNet learned some features (edges, outlines, etc.) that are useful for any kind of image recognition problem.

3.2. Finetuning the pretrained model

I tried to go further by fine-tuning this model, since the ImageNet data set is different from the FER dataset. So I loaded the previous model weights, set a very small learning rate (0.000001), and trained the whole network.

With 100 epoch of training/validation, the test accuracy reached 66.5%. The confusion matrix is shown in the figure 9.

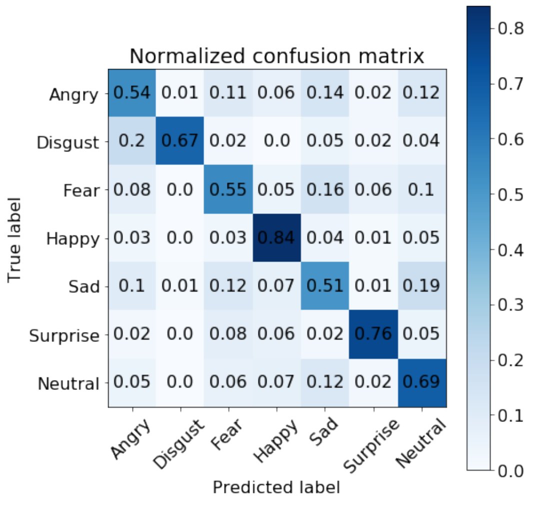


Figure 10. Confusion matrix - Finetuned pretrained model

The corresponding file in the code is 'model_pretrained_finetuned_v3.py', the model is stored in 'finetunedv3.h5'

This shows that even with pre-trained weights, some adjustments are to be made in order to fit the new dataset better.

4. The Ensembles method

As of now, the best performing model is the pretrained finetuned one. I could have tried to fine-tune it even further, but with the limited time and material at hand, I preferred exploring other possibilities:

4.1. Principle and results

In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Unlike a statistical ensemble in statistical mechanics, which is usually infinite, a machine learning ensemble consists of only a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives. The basic principle is shown in the figure 11.

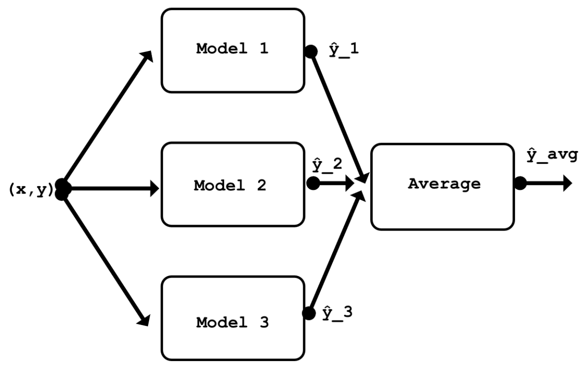


Figure 11. The ensembles principle

In this project, ensembles can be used for two reasons:

- Decrease the effect of random initialisation, by averaging the outputs of the same model initialised differently;
- Combine two well-performing models to increase the accuracy further.

I tried to implement the first point (*The corresponding file in the code is 'model_ensemble.py', the model is stored in 'ensemble.h5'*).
I combined two finetuned models, and got 66.3% accuracy on the test set. Not better than the single finetuned model, but I believe that with more models and more epoch, the results could have been better.

N.B : I tried concatenating the outputs of intermediate layers, but it performed poorly (like the model from scratch).

4.2. Options to improve the model

Here are a few propositions to go further :

- Preprocess the FER images, since some of them have different crop/angle/scale (eventhough they are already quite centered). This includes side shots, partial face shots, etc. Also, some pictures have 'shutterstocks.com' marked on them...

- Use data augmentation to get a more robust model to noise and small changes;

- Preprocess the images to fit the ones of ImageNet;

- Find pretrained weights on another dataset (not ImageNet), with images that are more similar to FER2013 (FaceNet for example);

- Use other activation function, such as LeakyReLu, to be able to make deeper models;

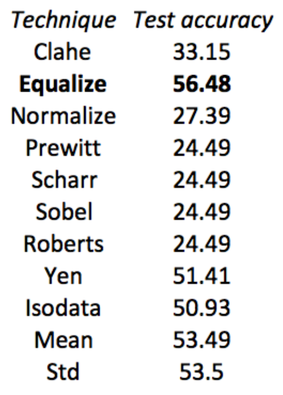
- Use a classifier such as SVM on top of the network;

- Use Ensembles to combine different models.

5. Images pre-processing

Instead of feeding my model with the raw images (reshaped), I tried to implement various Computer Vision techniques and compare their impact on the performance of my model. First, I wanted to use SIFT detectors but the low resolution of the images (48*48) gave very poor results. All the face detectors I tried failed to accurately compute descriptors for the FER2013 dataset.

So I implemented other techniques and compared them as shown in the following table, for the "from scratch" model (the lighter to train test) :



Technique	*Test accuracy*
Clahe	33.15
Equalize	**56.48**
Normalize	27.39
Prewitt	24.49
Scharr	24.49
Sobel	24.49
Roberts	24.49
Yen	51.41
Isodata	50.93
Mean	53.49
Std	53.5

Figure 12. Prepocessing results summary

The outputs of some of these filters are shown in the file "sample.ipynb".

5.1. Normalizing the images

First of all, I tried to bring the images values (that were between 0 and 255) to a [0,1] range, by using OpenCV "normalize" method.

However, there is a problem with this approach because it fits the entire face in a square with both axes ranging from 0 to 1. Some distinguishing features lie at the edges of the face, so normalizing will push them back into a very similar shape. The faces will end up looking very similar. Basically this method gets rid of many valuable data for distinguishing emotions.

5.2. Histogram equalization

Then, I tried to improve the contrast by stretching the histogram to improve the image contrast. I used two methods: - The tradional histogram equalization;
- Contrast Limited Adaptive Histogram Equalization (CLAHE): the image is divided into small blocks, which should provide better results than a global equalization.
Surprisingly, it does not, and the global equalization outperforms the CLAHE. I suspect the low quality of the images to be the cause of this.

5.3. Edge operators

Edge operators are used in image processing within edge detection algorithms. They are discrete differentiation operators, computing an approximation of the gradient of the image intensity function.

I used the following filters : Sobel, Roberts, Scharr and Prewitt. However, the results were very poor as they predicted all test data as ”happy”.

5.4. Binary masks

I used thresholding to create a binary image from my grayscale images. I tried three thresholding algorithms :
- Yen : [6].
- Isodata : the procedure divides the image into object and background by taking an initial threshold, then the averages of the pixels at or below the threshold and pixels above are computed. The averages of those two values are computed, the threshold is incremented and the process is repeated until the threshold is larger than the composite average.
- Mean : uses the mean of grey levels as the threshold. It is used by some other methods as a first guess threshold.

They all gave consistent results, although they are not as good as the original model (without thresholding).

5.5. Scaling and Centering

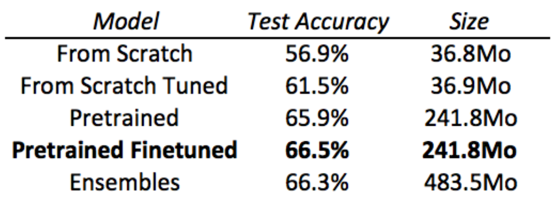
As a last resort to normalize my dataset, I also tried mean image substraction and scaling (by dividing by the standard deviation). The objective was to get a balanced dataset in order to improve convergence speed and prevent vanishing gradient issues.
It did perform well, but again, not as well as the original model.

5.6. Take away from image processing

It appears that some of the traditional techniques of computer vision remove some features that are used by the neural network to determine emotions, which explains why the performance is worse than the original model (without preprocessing). A more thorough study of the convolutional layers output might help in choosing the right computer vision approach to process the dataset.

6. Conclusion

All in all, this project allowed to test different variations around the VGG16 architecture and see the impact of modifying some parameters of the model. The overall results are shown in figure 12.



Model	*Test Accuracy*	*Size*
From Scratch	56.9%	36.8Mo
From Scratch Tuned	61.5%	36.9Mo
Pretrained	65.9%	241.8Mo
Pretrained Finetuned	**66.5%**	**241.8Mo**
Ensembles	66.3%	483.5Mo

Figure 13. Results summary

Given the size of the models, there might be other possibilities to explore in order to improve the model without complexifying it more. However, we can already see that with the right pretraining and some finetuning, we can already improve quite well the performance of a deep neural network model.

The code can be found at ’https://github.com/Charif-C/emotion_recognition_keras’

References

[1] K. Simonyan, A. Zisserman- Very Deep Convolutional Networks for Large-Scale Image Recognition- 2014

[2] O. M. Parkhi, A. Vedaldi, A. Zisserman- Deep Face Recognition- British Machine Vision Conference, 2015

[3] Matthew D. Zeiler, Rob Fergus- Visualizing and Understanding Convolutional Networks- 2013

[4] Ali Mollahosseini, David Chan, Mohammad H. Mahoor- Going Deeper in Facial Expression Recognition using Deep Neural Networks- 2015

[5] Christopher Pramerdorfer, Martin Kampel- Facial Expression Recognition using Convolutional Neural Networks: State of the Art- 2016

[6] Yen JC, Chang FJ, Chang S (1995), ”A New Criterion for Automatic Multilevel Thresholding”, IEEE Trans. on Image Processing 4 (3): 370-378, ISSN 1057-7149, doi:10.1109/83.366472