Artificial Intelligence

Emotion recognition with Convolutional neural networks

Leaded by M.Eng Kazimierz Kiełkowicz

Wojciech Chmura, Konrad Krukar

May 2020

Contents

1 Overview

This project will show the process of building a Convolutional neural network to recognize emotions on humans faces. This type of application can be use full and be applied in various system such as security cameras.
As we know nonverbal signals sometimes say more then words.

2 Introduction

3 Convolutional Neural Networks

4 Dataset description

Data that we got is in one *.csv file with two columns:

- number of emotion
  in rage 0 - 6:
  - 0 - angry
  - 1 - disgust
  - 2 - fear
  - 3 - happy
  - 4 - sad
  - 5 - surprise
  - 6 - neutral
- string of pixels
  Pixels are in one long string (2304 of them), they are in gray scale 0 - 255.
  We need to preprocess them into array of 48x48x1 that will could be reorganised as a picture also we will change to rage to 0 - 1 scale - by dividing via 255

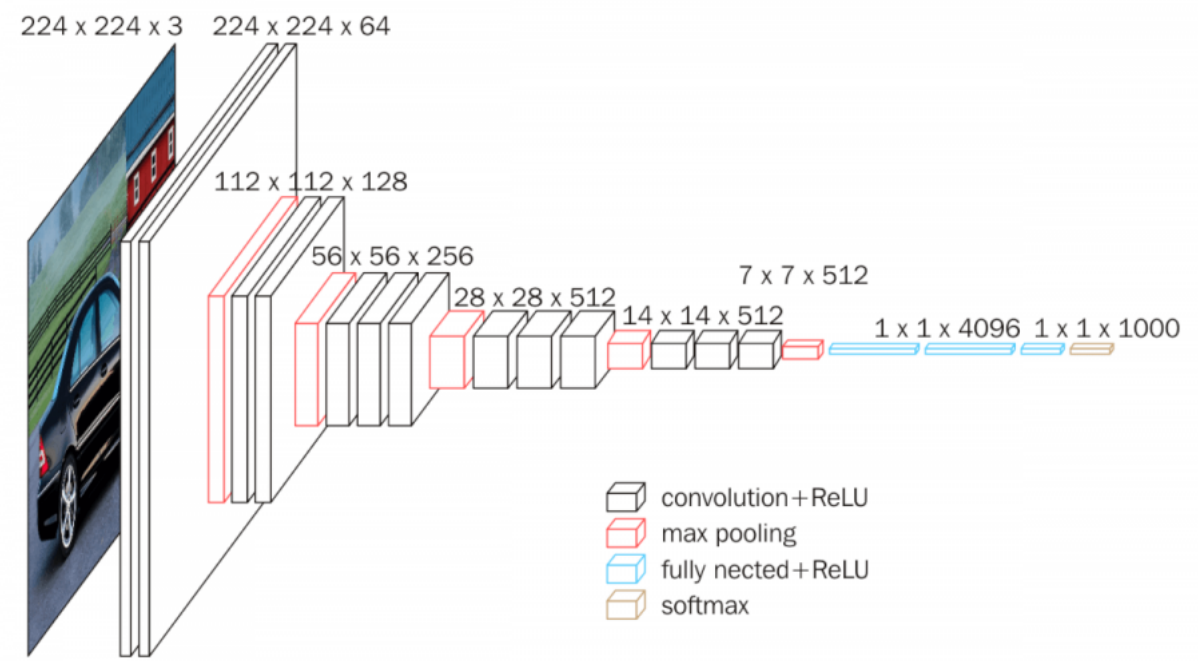
```
#CONST
imageSize, depth = 48, 3
numberOfClasses = 7
filePath = 'data.csv'

def processPixels(pixels):

    pixels_list = [item[0] for item in pixels.values.tolist()]

    score_array = []
    for index, item in enumerate(pixels_list):
        data = np.zeros((imageSize, imageSize), dtype=np.uint8)
        pixel_data = item.split()

        for i in range(0, imageSize):
            index = i * imageSize
            data[i] = pixel_data[index:index + imageSize]

        score_array.append(np.array(data))

    score_array = np.array(score_array)
    score_array = score_array.astype('float32') / 255.0
    return score_array
```

  After that we need to multiplay the layers into (48, 48, 3) becouse VGG16 CNN is prepared for RGB pictures. So basically recopy gray pixels 2 times more.

```
def makeVGG16input(array_input):
    array_input = np.expand_dims(array_input, axis=3)
    array_input = np.repeat(array_input, 3, axis=3)
    return array_input
```

5 Model builling

We decided to use pre-traind VGG16 model, without its top two layers.



VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.

In Keras it's really easy to implemnt this model and freeze layers to non-trainable (In code in "Implementaion" section) Our VGG model looks lika this:

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 48, 48, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 48, 48, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 48, 48, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 24, 24, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 24, 24, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 24, 24, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 12, 12, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 12, 12, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 12, 12, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 12, 12, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 6, 6, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 6, 6, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 3, 3, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
_________________________________________________________________
global_average_pooling2d_2 ( (None, 512)               0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

6 Implementation

7 Quantization

8 Results

9 Conclusion