Emotion recognition on Real-Time using Keras

Luis Enrique Meza. Tecnológico de Monterrey Campus Querétaro. ISDR. A01114143.

Abstract. - This document guides us to the implementation and explanation on the use of a convolutional neural network to detect emotions on real-time.

I. INTRODUCTION:

Convolutional neural networks have been implemented in a lot of things since they were discovered. One of their main appliances nowadays, is object and face detection or recognition. We can find these kinds of examples on our actual smartphones that identify our face and immediately unlocks the device, on door locks that scans our retina and opens in the matter of microseconds, etc.

With all the applications out there, a lot of ideas have emerged on how to use these particular machine learning approach, one of them being, the concern of a well-being, which in this case ended up as an emotion recognition analysis in real time in order to be used for different applications later on, such as psychological or medical, among others.

II. DATASET:

The dataset used for this implementation is called Fer2013, the model was found on a YouTube video [1] that helped also on the development of the structure to test it in real-time.

Even if the model can be found on a .csv format, I preferred to use the images by itself, just so it would be easier to separate them into different emotions and test the models varying on emotions.

Also, the original dataset has 7 emotions, but on this approach, we only used 5, and even 3 on one of the architectures to reach a certain percentage of accuracy.

This dataset includes the following images for training: 4,005 for angry, 7,255 for happy, 4,965 for a neutral state, 4,830 for sad and 3,201 pictures for surprise. For validation: 491 for angry, 879 for happy, 626 for neutral, 594 for sad and 416 for surprise.

The images from the dataset are on a gray scale and its image is 48 by 48 pixels.

Figure 1. Example of dataset images with the "Happy" label.

III. WORKING WITH THE DATA:

After a little bit of research, and experiences with the linear regression model I did before, to have a more effective model it is better to normalize all the data. To do this, and because this dataset consists of gray images, you just have to divide each picture by 255, which is the greatest value of a pixel. This was made possible by using the Image Data Generator section of keras.

In addition to this, the Image Data Generator was used for data augmentation, only for the train set. I applied rotation, shift, zoom and horizontal flip to the images.

IV. MODEL USED:

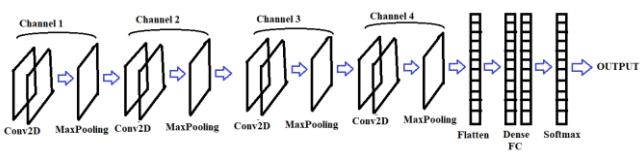
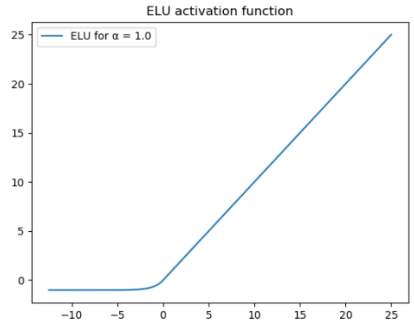


Figure 2. Scheme of the Network used.

On the first attempt of the model, I used an architecture based on four basic channels, each channel consists of two conv2D layers, with an 'elu' activation function, one batch Normalization layer after each conv2D layer and at the end a Max Pooling and a Dropout layer.

a) Conv2D: receives the number of filters we want, the strides in the form (3,3), a kernel initializer stablished on 'he_normal" which tells the layer to initialize the weights based on a truncated normal distribution centered on 0. [2]

b) Activation: For the activation (that was included in the parameters of the Conv2D layers) 'elu' was used.



$$f(x) = \begin{cases} x, & \text{if } x >= 0 \\ \alpha(exp(x) - 1), & \text{otherwise} \end{cases}$$

Figure 3. Elu activation function. [3]

c) Batch Normalization: Used for accelerating the learning of the neural network. What this does is change the mean to zero and the standard deviation to one. Even if the paper that introduced this function says it is better to add this layer before the activation function, a lot of reported experiments showed better results on adding it after the activation function, just as we did in this case. [4]

d) Max Pooling: Used to reduce dimensionality by selecting the maximum value of the regions size stride (2,2 in this case).

e) Dropout: Randomly drops (deactivates) some neurons. This is used to battle overfitting. Sometimes on the model 10% drop was used, but on the last layer 40% was used.

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 48, 48, 32)	320
batch_normalization_11 (Batc	(None, 48, 48, 32)	128
conv2d_10 (Conv2D)	(None, 48, 48, 32)	9248
batch_normalization_12 (Batc	(None, 48, 48, 32)	128
max_pooling2d_5 (MaxPooling2	(None, 24, 24, 32)	0
dropout_7 (Dropout)	(None, 24, 24, 32)	0
conv2d_11 (Conv2D)	(None, 24, 24, 64)	18496
batch_normalization_13 (Batc	(None, 24, 24, 64)	256
conv2d_12 (Conv2D)	(None, 24, 24, 64)	36928
batch_normalization_14 (Batc	(None, 24, 24, 64)	256
max_pooling2d_6 (MaxPooling2	(None, 12, 12, 64)	0
dropout_8 (Dropout)	(None, 12, 12, 64)	0
conv2d_13 (Conv2D)	(None, 12, 12, 128)	73856
batch_normalization_15 (Batc	(None, 12, 12, 128)	512
conv2d_14 (Conv2D)	(None, 12, 12, 128)	147584
batch_normalization_16 (Batc	(None, 12, 12, 128)	512
max_pooling2d_7 (MaxPooling2	(None, 6, 6, 128)	0
dropout_9 (Dropout)	(None, 6, 6, 128)	0

Figure 4. Part of the model summary.

After the four basic channels, one flatten layer is added, and two dense layers before the final dense layer which is activated by 'softmax' and gives the output of the neurons.

a) Flatten: Just flattens the input, like accommodating everything on an array.

b) Dense: Fully connected layers, adding a non-linearity property by using the elu activation as well.

c) Softmax: Generalization of a logistic regression, works for classifying at the end of the model.

flatten_2 (Flatten)	(None, 2304)	0
dense_4 (Dense)	(None, 64)	147520
batch_normalization_19 (Batc	(None, 64)	256
dropout_11 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 64)	4160
batch_normalization_20 (Batc	(None, 64)	256
dropout_12 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 3)	195

Figure 5. Final part of the model summary.

Last but never the least, in order to train I used two callbacks, the first one being a checkpoint to be saving the model according to the higher validation accuracy, and reducer on the learning rate, so if the validation loss is not going down, this will adjust the learning rate and make it smaller for it to try to learn a little bit more.

V. FACE RECOGNITION

To do the testing, I did a program based on an example I saw on YouTube [1]. In this program the hardest part was the face detection, which was possible due to a library called OpenCV which was originally developed by Intel.

Making use of this library made it a lot easier the search for a face in the capture (the detection of it).

After using the cascade classifier provided by the library, all that was left to do was adapting the capture to the specifications of the model. As I said earlier on this document, the sizes of the images on the dataset are 48 by 48 pixels and are on a gray scale. So, I had to change the color of the capture, resize it and normalize its values (since I made the same thing with the values on the dataset). After all that, any model was ready to be tested.

VI. RESULTS

At first, I tried the model with 5 emotions, but the results were kind of unsatisfying. The model was very good at recognizing happy, surprise and neutral, but did not show any recognition of other emotions whatsoever even if it had reached an accuracy close to 50% on validation.

So, I decided to test with different combinations of emotions. The first being the one that showed me the best results at first, happy, neutral and surprise.

VII. NEUTRAL-HAPPY-SURPRISE

Even if I ran the same model just changing the dataset (removing the other emotions) and adapting things such as the number of classes, the number of train and validation samples (that were used on the fit_generator function), it had a 70% accuracy on validation but in real-time testing performed excellent. It does not miss at all and works perfectly fine.

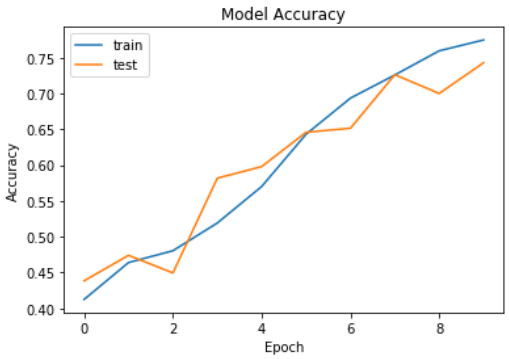


Figure 7. Model Accuracy for 'NHS'.

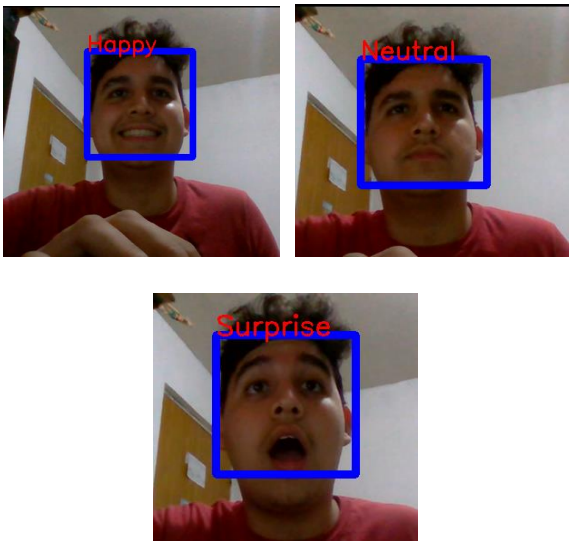


Figure 6.'NHS' Results.

VIII. NEUTRAL-ANGRY-SAD

Another approach I took was to try to train it for recognizing neutral, angry, and sad this time. But the results were not as great as the last one. On validation, the most accurate it got was around 40%, but in real-time testing, it did not predict nothing but neutral.

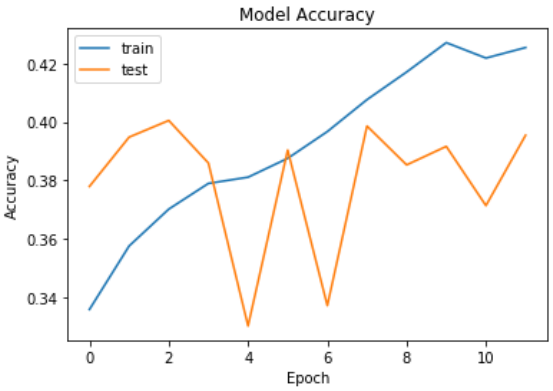


Figure 8. Model Accuracy for 'NAS'.

This only showed us why a complete model (with all 5 emotions) may not be working, so I took a glance on the sad and angry dataset.

Figure 9. Examples labeled as 'angry'

Figure 10. Examples labeled as 'sad'

As we can observe on the figures above, there could be a lot of discrepancy among the samples on the dataset. Some of the images tagged as angry seem neutral, sad and even happy, as well as images labeled as sad may be confused with other emotions.

This produces nothing but noise to the model, due to the simplicity of the dataset. Maybe on larger images with color on it would not be a problem to add this kind of noise, but on smaller pictures in a gray scale is hard even for a human to distinguish.

IX. CONCLUSIONS:

Emotion recognition can be useful in a lot of ways. As humans, we use it everyday trying to 'read' our closest ones to comfort them in case they are having a bad day, or to share the joy with them if they are feeling happy.

Even if that is a human task we are so used to, teaching a machine that is not that easy to do, and this proves how different we are when compared to artificial intelligence.

With all the inconveniences, it was proved that a model with higher accuracy can be achieved by placing more effort into the dataset, maybe bigger images, or another filter to decide which picture is which emotion.

X. REFERENCES:

[1]. DT TECH UPDATES. (2019). Emotion Detection Python Deep Learning – Keras. Recovered from: https://www.youtube.com/watch?v=PGH8uSzTTzc&t=299s

[2]. TensorFlow. (2015). He_normal initializer. TensorFlow API. Recovered from: https://www.tensorflow.org/api_docs/python/tf/keras/initializers/he_normal

[3]. Chris. (2019). How to use ELU with Keras? MACHINECURVE. Recovered from: https://www.machinecurve.com/index.php/2019/12/09/how-to-use-elu-with-keras/

[4]. Brownlee, J. (2019). How to accelerate Learning of Deep Neural Networks with Batch Normalization. Machine Learning Mastery. Recovered from: https://machinelearningmastery.com/how-to-accelerate-learning-of-deep-neural-networks-with-batch-normalization/

Keras. (2019). Conv2D layer. Convolution layers. Recovered from: https://keras.io/api/layers/convolution_layers/convolution2d/