AMD-SM2L Project

Marco Riva

September 7, 2021

Contents

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Dataset Organization

For the implementation of my neural network, I used all the dataset from Kaggle, which is organized in two different parts: the train set and the test set, composed of the images and the vectors that have created them. Every image in the train set has a label that tells us if the person in the image is wearing glasses or not. I also added the label to the test images and I checked the correctness of the train labels. After that, I moved the labeled data in different directories, used to create the classes of the Tensorflow dataset.
While checking the pictures, I noticed that some of them are very ambiguous, so I decided to remove these images from the dataset. I used Pandas for reading the CSV and modifying the labels, which were taken from a text file previously

made.

Now, with the data divided into train and test, both divided into two classes ("glasses" and "no_glasses"), I created the dataset using Tensorflow and Keras.

2 Data Processing

To create the dataset I decided to use the Keras method "*image_ dataset_ from_ directory*" from the preprocessing module. This function can create a Tensorflow dataset starting from the images, specifying the size of the batch and the size of the image. I choose a batch size of 64 and an image size of 160 pixels and I also set the shuffle argument to true, to sort the data randomly. Another method is applied to the dataset: the prefetch method. It overlaps data preprocessing and model execution during training. The dataset is also cached in memory, to speed up the training process.
The dataset has a shape of (160, 160, 3) because the images are colored and the RGB channels are 3.

3 The Model

I choose to build a convolutional neural network (CNN) for the binary classification. A CNN is a neural network with multiple layers, each of them applies a filter to the image in input, focusing only on a small portion of the input at a time. The input and the output of the convolutional layers have three dimension: the width and the height of the filter and the number of channels.

My CNN is composed of three convolutional layers, one dense layer with 128 neurons and the output layer with one neuron. For my CNN the input is, obviously, the shape of the input image, so (160, 160, 3).
The first thing that the CNN does is rescaling the input from the [0,255] values of the RGB image, to a range of [0, 1] values.
The convolutional layers have a kernel size of (5,5) and the filters are 16, 32, 64 respectively for the first, second, third layer, the padding of every layer is set to "*same*", so they pad with zeros, to produce an output with the same shape of the input. Each layer ends with a "*MaxPooling2D*" operation, to reduce the size of the input by a (2,2) square. In this way the shape changes from (160,160, 3), to (80, 80, 16), to (40, 40, 32), at the end of the convolutional layer it becomes (20, 20, 64). I add a batch normalization operation using only the "*center*" parameter because the internal layers have a RELU activation function and this function is scale-invariant. The output layer has a SIGMOID activation function because the problem is a binary classification.

This is the summary of the model:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling (Rescaling)        (None, 160, 160, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 160, 160, 16)      1216
_________________________________________________________________
batch_normalization (BatchNo (None, 160, 160, 16)      48
_________________________________________________________________
activation (Activation)      (None, 160, 160, 16)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 80, 80, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 80, 80, 32)        12832
_________________________________________________________________
batch_normalization_1 (Batch (None, 80, 80, 32)        96
_________________________________________________________________
activation_1 (Activation)    (None, 80, 80, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 40, 40, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 40, 40, 64)        51264
_________________________________________________________________
batch_normalization_2 (Batch (None, 40, 40, 64)        192
_________________________________________________________________
activation_2 (Activation)    (None, 40, 40, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 20, 20, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 25600)             0
_________________________________________________________________
dense (Dense)                (None, 128)               3276928
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               384
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
```

The model is optimized with the ADAM optimizer, with a learning rate that starts from 0.1 and decreases gradually to 0.00101 in ten epochs. The loss function is the "*binary_ crossentropy*": a typical loss used for binary classification, it is the categorical cross-entropy version for problems with two classes.

4 Comments on the model

With the settings and the hyperparameters previously chosen, after only four epochs approximately the training accuracy converges and reaches values really close to 100% and the training loss is almost zero. For the test set, the results are very similar: the validation accuracy is around 99% after about seven epochs and the loss is 0.05.
The training and test values are quite similar, so I think that the phenomenon of overfitting doesn't occur for this CNN. Adding the dropout of the 0.32% of

the neuron before the dense layers don't improve the learning in a consistent manner, the validation metrics converge in less epochs than before and the values are very similar.
The time necessary to train the CNN is reasonable: it takes about three minutes to start the training. After that, every epoch are completed in 20 seconds more or less. In Google Colaboratory, the total computation time for training the network using the entire dataset is always less than ten minutes.

5 Larger Dataset

I've used the entire dataset from Kaggle, so the CNN works efficiently with a data size of 5000 items, but I think that with small fixes it can be used with a larger amount of data.
Incrementing the number of the images, especially in the train set, can cause overfitting, so the CNN adapts itself too much with the training images and it can't predict well the test set. A solution to adopt can be reducing the percentage of the training data or simplifying the network.
Another possible backward of using a larger dataset is that it can make it impossible to cache the data: that slows down the entire training process but doesn't affect the correctness of the network.
If we have a large number of images, we can split the data not only in training and test set but also in the validation set. In this way, the CNN can use the validation images to test the accuracy of the network and we can use the test set to evaluate the predictions.

6 Conclusions

In conclusion, I think that the convolutional neural network built can predict with good accuracy if the people photographed are wearing glasses or not. The choice of the hyperparameters is been made testing the network with different values and evaluating it, focusing on the correctness of the prediction and the execution time. With a batch size of 32 images, the validation metrics fluctuated too much every epoch, showing overfitting, using a batch size larger than 64 the validation accuracy converged after more epochs (approximately after the ninth or tenth epoch). I choose 160 pixels for the image's size because it can be seen well if visualized and it isn't too big for the CNN. The filters are of size (5,5) to speed up the training process, decreasing the number of parameters to bet set by the network, without losing precision.