AMD-SM2L Project

Marco Riva

September 21, 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.

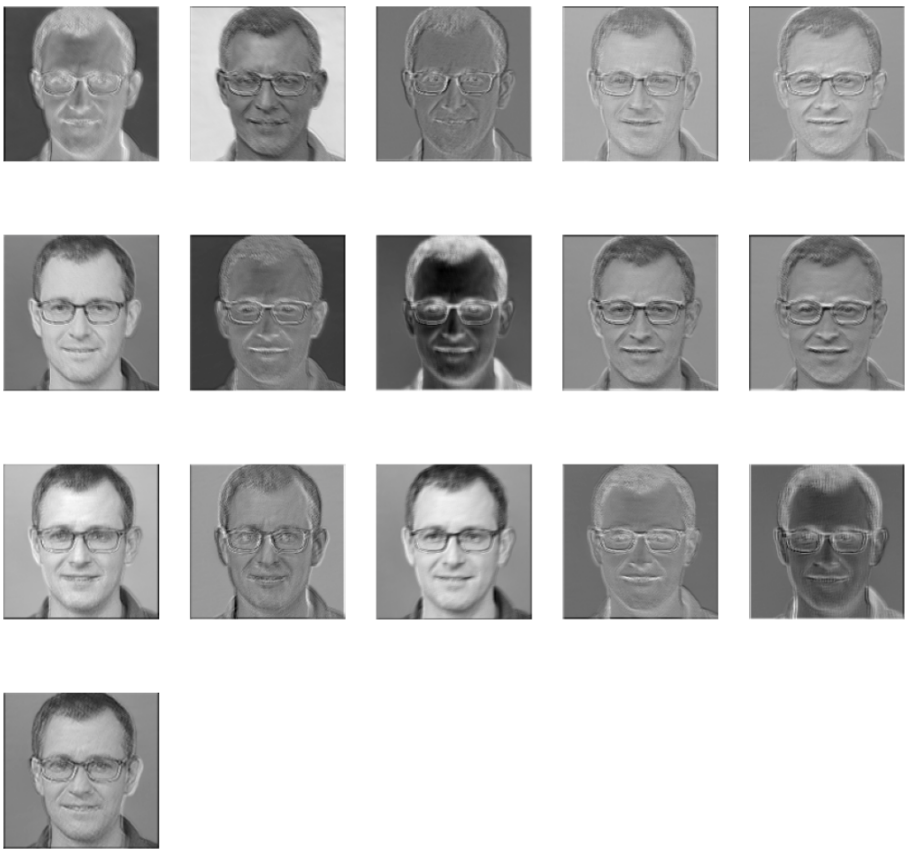
2 Data Processing

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 initially choose a batch size of 32 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 Selection

I chose 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.

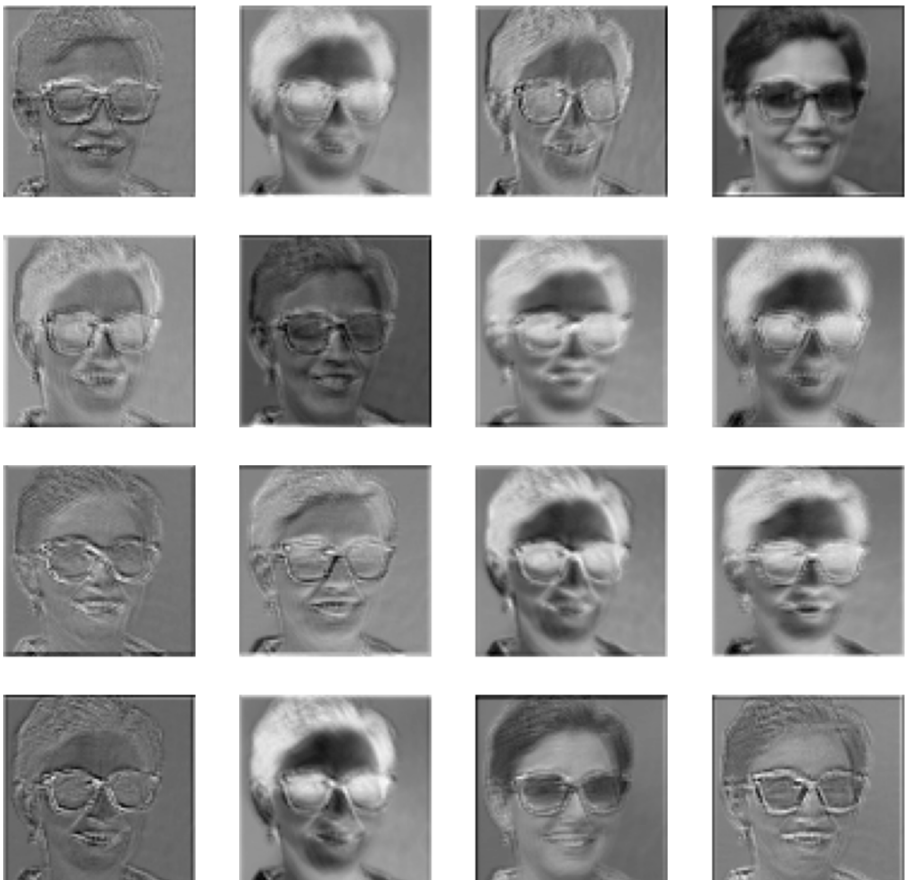
For my CNN the input is, obviously, the shape of the input image, so (160, 160, 3) and the first thing that the CNN does is rescaling the input from the [0,255] values of the RGB image, to a range of [0, 1] values.
I started with a CNN with one convolutional layer, with a kernel size of 5, because I needed a kernel size wide enough to include a portion of glasses from the image. The padding of the layer is set to "*same*", so it pads with zeros, to produce an output with the same shape of the input. I chose also 16 filters, to be sure to intercept all the nuances necessary to make the network understand which face had glasses or not, focusing on various elements of the image.
These are the outputs of the filters:



The activation is the RELU function and after the convolutional layer, I put one dense layer composed of 128 neurons and the output layer with one neuron, whithin the SIGMOID activation function because the problem is a binary classification. To improve the network performances, I added batch normalization to the convolutional layer. Batch normalization is a technique used to make neural networks more stable and faster, using normalization (subtracts the average and divides by the standard deviation) of the input values and rescaling and recentering them. In this way, the input values follow the trend of the activation function and the machine learning tunes the parameters of the normalization for each neuron. Since the activation function is the RELU, that is scale-invariant, the only parameter used by batch normalization is "*center*".

I also tried a deeper network adding a second convolutional layer, with kernel size of 3 and 16 filters. This choice is made because a deep network is more precise and can discover more details, but it can also lead to overfitting. The first layer ends with a "*MaxPooling2D*" operation, to reduce the size of the input by a (2,2) square.
Training this CNN produce a very good results and also the prediction values for the test set are pretty good, so even if I add a new layer the CNN doesn't overfit.
These are the outputs of the second layer filters: the images are more condensed

and the network can understand better the presence or not of the glasses.



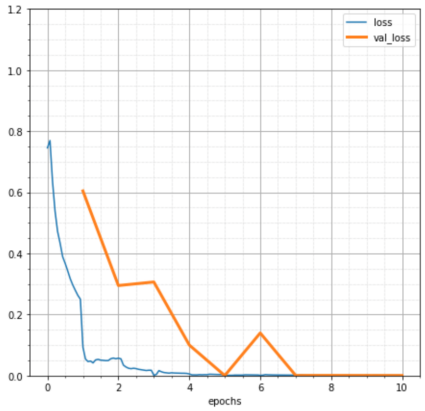
And this is the summary of the model:

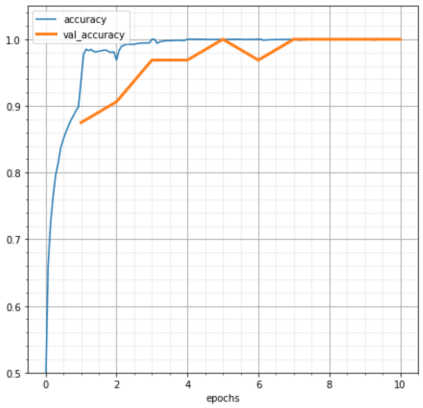
Layer (type)	Output Shape	Param #
rescaling_2 (Rescaling)	(None, 160, 160, 3)	0
conv2d_4 (Conv2D)	(None, 160, 160, 16)	1216
batch_normalization_6 (Batch	(None, 160, 160, 16)	48
activation_6 (Activation)	(None, 160, 160, 16)	0
max_pooling2d_2 (MaxPooling2	(None, 80, 80, 16)	0
conv2d_5 (Conv2D)	(None, 80, 80, 16)	2320
batch_normalization_7 (Batch	(None, 80, 80, 16)	48
activation_7 (Activation)	(None, 80, 80, 16)	0
flatten_2 (Flatten)	(None, 102400)	0
dense_4 (Dense)	(None, 128)	13107328
batch_normalization_8 (Batch	(None, 128)	384
activation_8 (Activation)	(None, 128)	0
dense_5 (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

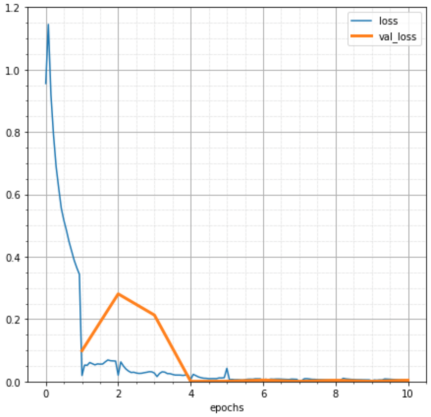
With the settings and the hyperparameters previously chosen, after only four epochs approximately the training accuracy converges and reaches values really close to 100% and the training loss is almost zero. For the test set, the results are very similar: the validation accuracy is around 98% after about seven epochs and the loss is less than 0.1.
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.

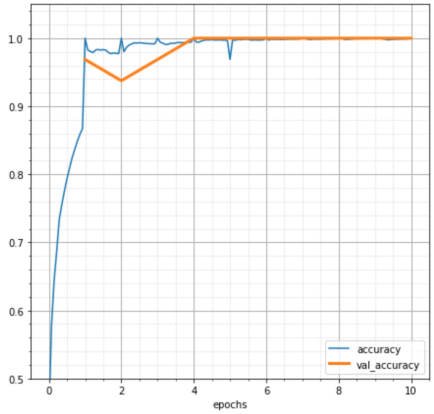




Since the test accuracy and loss are often very similar to those of training, the network doesn't overfit. Indeed, overfitting happens when a learning algorithm fits itself too much into the training set and can't predict the test data, so the result of overfitting is low training loss and high test loss. A method to prevent this is using dropout: this technique consist of "turning off" a random number of the neuron in the network, according to a predetermined probability, at each training iteration and in the next iterations the neurons dropped will be different. So the dropped neurons don't partecipate to the computation of the loss and their weights will not get updated. The reason of dropout is that it is possible for a layer of the network to evolve in a bad prediction model and other layers can fix this problem. With dropout, there is a probability that the neurons capable of this fixing are turned off, so the weights will be updated with better values, to prevent this bad evolution of the layer.

I tried to add dropout in my CNN, in the first dense layer, but just as I expected, the result were very similar to those of the previous model, because the network doesn't overfit.





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 in different ways: some of them (like the size of the kernel or the number of filters) are chosen following the structure of the problem and the theory on which the CNNs are based, others are chosen in an empirical way using a grid search. This is the case of the batch size and the learning rate, that are been chosen testing the network with a list of values that cover a varied range and that can train the network in different ways. Particularly, I noticed that the batch size that works better is 32 and the best learning rate starts from 0.1. Finally, 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.