Glasses or no glasses classification

Statistical methods for machine learning and Algorithms for massive datasets
Experimental project report

Alesia Sommariva
960824

University of Milan
Master's Degree in Computer Science
Academic year 2020/2021

Contents

1 Introduction

The project chosen for this report is image classification via convolutional neural network. We address the problem of the classification of images produced by a GAN network containing faces of people who wear / do not wear glasses, both prescription and sunglasses. So, the final objective is to classify wheter the faces in the dataset wear glasses or not.

This project is based on the dataset "Glasses or no Glasses" published on Kaggle and released under the Attribute-ShareAlike 4.0 Creative Commons license (CC BY-SA 4.0). The neural networks are implemented by using the TensorFlow framework (version 2.6.0), in particular by adopting the Keras library.

We investigate several convolutional neural networks by tuning different parameters in order to find the most simple network with the highest accuracy and lowest loss.
In section 2 we present the dataset and the applied preprocessing techniques. In section 3 we present the experiments performed for model selection and discuss the training details. In section 4 we address the problem of scalability and in section 5 we present the experiments results and observations. In section 6 we present our conclusions.

2 Dataset overview and preprocessing

In this section we describe the structure of the dataset and the preprocessing techniques applied. The dataset is available at the following link:

`https://www.kaggle.com/jeffheaton/glasses-or-no-glasses`

2.1 Dataset structure

The dataset is composed by two `.csv` files, `train.csv` and `test.csv`, conaining the latent vectors and a directory with the pictures generated by those vectors. The `train.csv` file has 4500 rows and 514 columns. The first column contains an unique identifier, the next 512 are latent numbers used by the GAN network and the last one contains the label. The `test.csv` file has 500 rows and 513 columns. The first column contains the id and the last 512 are the latent numbers. In total there are 5000 images. Each image comes in `.jpg` file format.

In this project we will use the 4500 labeled images for training, validation and test. The last 500 unlabeled images will be used for final visual testing. After checking the number of images per class we noticed an imbalance between the two classes, in particular there are more images labeled as "glasses" (corresponding to label 1) than "no-glasses" (corresponding to label 0). In fact, there are 1644 images labeled as "glasses" and 2856 labeled as "no-glasses". Moreover, some images did not form clearly, and are associated with the wrong label. This can cause poor performance in training, so this will be taken into account.

2.2 Preprocessing

We decided to import images in a shape of 100 x 100 x 3 (height, width, colour). Firstly we need to download and decompress the dataset from Kaggle. Then, from the `train.csv` file we take ids and labels and associate the with the labeled images, and from `test.csv` we take the ids to do the same with the not labeled ones.
All the pictures are RGB images. Therefore, the channels values are in the [0, 255] range. This range is not ideal for neural network, thus we have to standardize values to be in [0, 1] range.

We found out that 63.47% of the images are labelled as "glasses", and the other 36.53% are labeled as "no glasses". Therefore, we have some unbalanced classes. We decided to apply data augmentation by performing some moves, such as random horizontal flip, random rotation of 0.1, and random contrast adjustment of 0.2. Thus, we were able to produce the necessary number of images for the "no-glasses" class, in order to have the same number of images as the "glasses" class. At the end, we will have 5712 images. An example of augmented pictures is shown in figure 1, whereas in figure 2 we can see an example of 25 images from the original dataset.

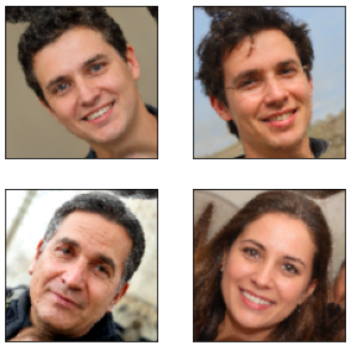

Figure 1: Augmented images

In this way we have two datasets: one unbalanced and one balanced over which we will perform model selection (see next section 3).

3 CNNs

In this section we describe the experiments performed in order to find the best CNN. We used a Stratified Cross Validation to tune parameters and also a

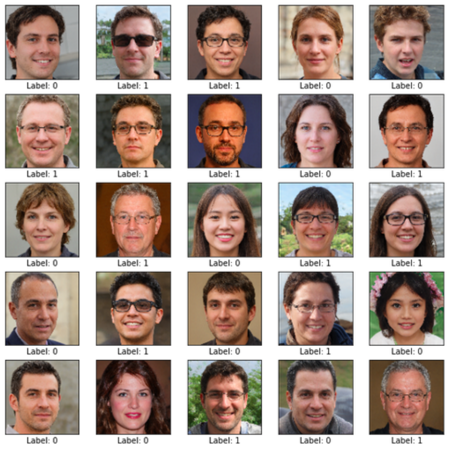

Figure 2: Images of the original dataset

manual Grid Search for finding out the number of filters in the convolutional layer and the number of blocks (this concept will be described later). We will compare those two approaches.

We decided to do the entire model selection through a "cascade" mode, along with a grid search. However we focused the grid search only on two parameters, to avoid high computational cost. When we say "cascade" mode we mean that for each single parameter we perform a cross validation of 3 folds, for 3 possible values, and then take the values with the best results. We also know, however, that this approach gives a sub-optimal result compared to a complete grid search. However, the complete grid search is computationally much more expensive, and this is why we performed it only on two parameters.

3.1 Experiments

Firstly we perform some experiments with different architectures:

- First architecture (base): two blocks, each composed by a convolutional layer with ReLu as activation function and a 2D max pooling layer where in the first block we have 16 filters and kernel size of (3,3) in convolutional layer. In the second block we have 32 filters and kernel size of (5,5) with 2 strides. Then a flatten layer, followed by a dense layer of 64 neurons with ReLu as activation function and the last layer with one neuron and sigmoid as activation function. We will use the last layer to get the prediction.

- Second architecture (two convolutional layers per block): two blocks composed by two convolutional layers with ReLu as activation function and a 2D max pooling layer. In the first block we have 16 filters and kernel size of (3,3) in convolutional layer, in the second block we have 32 filter and kernel size of (5,5) with 2 strides in convolutional layer. Then a flatten layer, followed by a dense layer of 64 neurons with ReLu as activation function. The last layer is composed by one neuron and a sigmoid as activation function.

- Third architecture (addition of batch normalization and dropout): in this last architecture we decided to take the previous one and to add a batch normalization layer after each convolutional layer and a dropout of 0.5 between the two final dense layers.

All these architectures are trained for 20 epochs, and at the end of each one we tested the model over a validation set in order to observe how the model evolves and performs. In each architecture we have used binary cross entropy as loss function, since we handled a binary classification problem. We used `Adam` as optimizer to perform the gradient descent. In addition we used a learning rate scheduler callback; this function decreases the learning rate exponentially at each epoch.

Then, we are going to perform tuning over several parameters with Stratified Cross Validation with 3 folds on those two cases:

- with the unbalanced dataset with the network that obtained the best result depending on the value of the loss
- and with the balanced dataset with the network that obtained the best result depending on the value of the accuracy

The parameters and the respective values considered for tuning are presented below:

- number of blocks: 1, 2, or 3
- number of filters on the first convolutional layer: 8, 16 and 32. This number will be doubled or triplicates on an eventual second or third layer
- number of neurons in the fully connected layer: 32, 64 and 128
- dropout rate: 0.3, 0.5 and 0.7

Cross-validation or KFold Cross Validation systematically creates and evaluates multiple models on multiple subsets of the dataset. This, in turn, provides a population of performance measures. Then we can calculate the mean of these measures to get an idea of how well the procedure performs on average.

With the Cross-validation technique the entire dataset is partitioned in K subsets (also known as folds), in this case K=3, of size m/K (we assume for simplicity that K divides m), where m is the size of the dataset. Basically at each iteration the model is trained on two folds and then tested on the last one. This process runs until each fold has been the test set exactly once.

Thus, Stratified Cross Validation extends K-fold Cross Validation by ensuring an equal distribution of the target classes over the splits. This ensures that the classification problem is balanced. For each fold we then compute the accuracy and loss average, taking more into account the loss for the unbalanced dataset and the accuracy for the balanced one. We will look at the computational cost when the difference of the results is minimal.

For the number of blocks and numbers of filters on convolutional layers we perform also a manual grid search. With this grid search we combine all the possible configurations about the number of blocks and number of filters, performing a cross validation at each iteration. Then we will take the configuration with the best result and compare it with the result obtained from the cascade mode.

3.1.1 Unbalanced dataset

After having tested the three starting architectures, the best one turns out to be the last architecture with batch normalization and dropout.

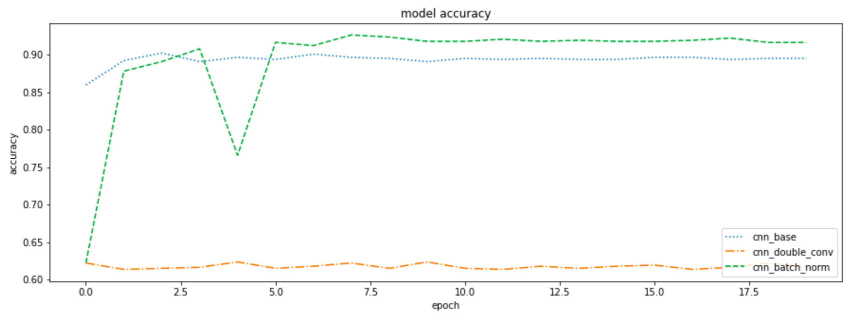

Figure 3: Accuracy

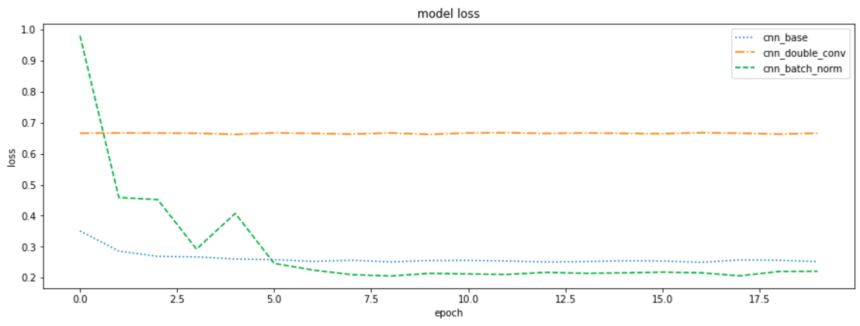

Figure 4: Loss

Then we performed the model selection with the cascade mode and grid search mode by varying the number of blocks and number of filters.

After this process we chose the architecture resulting from the cascade mode with one single block and 16 filters in the first convolutional layer. The other architecture with 32 filters only performs slightly better and it is not worth the additional computational cost.

Continuing the model selection with the cascade mode on the number of neurons and the dropout, the architecture of the final network will have one single block, 16 filters on convolutional layer, 32 neurons on dense layer and a dropout of 0.7, getting an accuracy of 89.5% and a loss of 26.1%.

3.1.2 Evaluation

At the end of the model selection we trained the best final model on the train/validation splits of the dataset. In figure 5 we can observe the resulting training and loss.

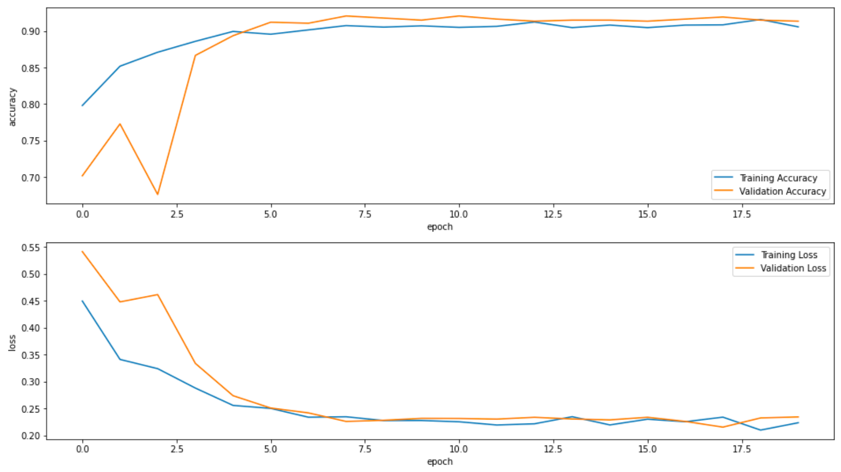

Figure 5: Accuracy and Loss

It is interesting to see how already after the tenth epoch both training accuracy and validation accuracy attest on extremely similar values and keep them until the end of the epochs, reaching an accuracy of over 0.91 and a loss of 0.23. The same holds for both validation loss and training loss. This is a sign that the neural network is performing well.

Even more remarkable is how well the model generalizes for this train/test split.

3.1.3 Balanced dataset

Also in this case the best one turns out to be the third architecture with dropout and batch normalization.

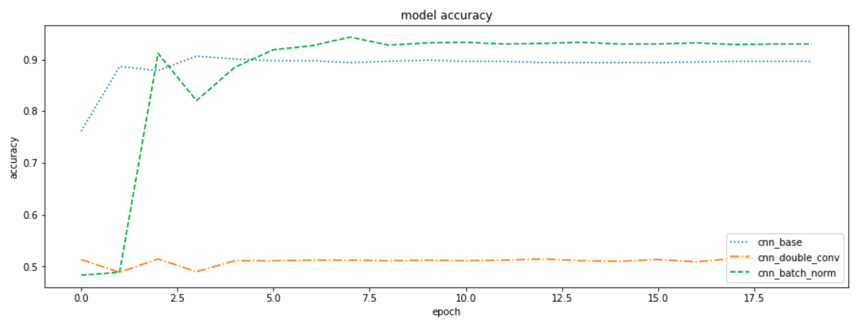

Figure 6: Accuracy

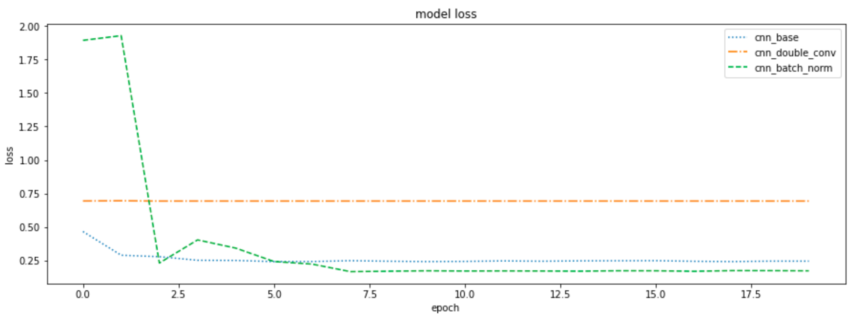

Figure 7: Loss

We executed the same experiments for the balanced dataset. So, after first tuning with the cascade mode, it appears that 32 filters in the first convolutional layer and one single block is the best architecture. Note that the accuracy in these first experiments already increased to 0.91 and the loss dropped to 0.20. After the manually grid search it appears that the best architecture is composed by one single block and 32 filter in the convolutional layer.

So at the end of the tuning phase with the cascade mode for the remaining parameter, i.e. number of neurons in the second-last layer and the dropout, the architecture of the final network will have one single block, 32 filters in the first convolutional layer and 64 neurons with a dropout of 0.7.

3.1.4 Evaluation

As before, we trained the best final model on the train/validation splits of the balanced dataset.

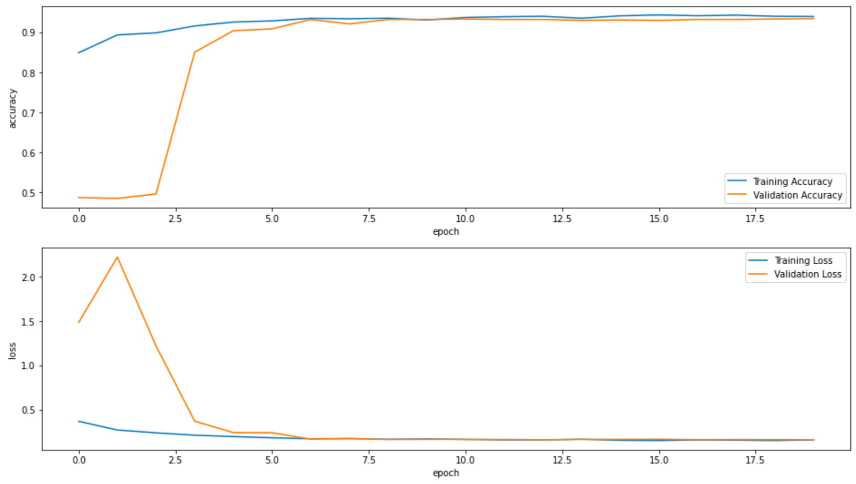

Figure 8: Accuracy

It is even more interesting to note how after even fewer epochs than before (about 5), train and loss settle on the same values and keep them until the end, in addition to the fact that the neural network generalizes well and even better than the previous one. We also have an accuracy of 0.92 and a loss of 0.16

4 Scalability

To cope with the problem of scalability first of all we thought of exploiting some methods made available by TensorFlow regarding how to treat the dataset during training: with `Dataset.cache` we keep the images in memory after having loaded them from the disk during the first epoch. This will ensure that the dataset does not become a bottleneck while training the model. Then, with `Dataset.shuffle` we perform a complete shuffle at each iteration, in which the size of the shuffle buffer is never less than the size of the set. We use `Dataset.batch` which allows us to divide the dataset into batches of 32 samples so that the network takes small amounts of data at a time. Finally we use `Dataset.prefetch`, which loads subsequent batches while it is pulling on the current one.

Regarding data storage, we could think of using HDFS, that is a distributed file system that allows you to store data on a cluster of nodes. It allows us to manage large amounts of data that obviously we would never be able to store on a single machine. Thanks to the batch operation on the data we can then load one or more chunks (in a limited number) at a time to form the next batch during the training phase.

5 Visual Tests

At the end of the experiments for both the unbalanced and the balanced dataset, we performed a visual test phase over the not labeled images of the dataset. We obtained consistent results with respect to what we saw in the training phase.

5.1 Test over the unbalanced dataset

As we can observe from figure 9, we have reached a coherent result with nearly 90.66% accuracy.

5.2 Test over the balanced dataset

Strangely we cannot say the same thing here (figure 10), because although we have achieved better results than before with an accuracy of nearly 91.33%, we have one wrong prediction.

6 Conclusions

In this project we developed and tested several architectures in order to find the one that brings the best results. For this purpose, after noticing a slight imbalance in the classes, we decided to create a second dataset by adding images for the smaller class using data augmentation. First of all, we tested three different architectures. Then with the best performing network we tried to optimize loss and accuracy by tuning some parameters.
With the evaluation of the final network in both cases we managed to obtain an improvement, even if it was small.

It is interesting to note that despite the slightly worse performance in the unbalanced dataset, in the visual test we do not even have a wrong prediction, unlike the case of the balanced dataset where in the same test we have a wrong prediction.

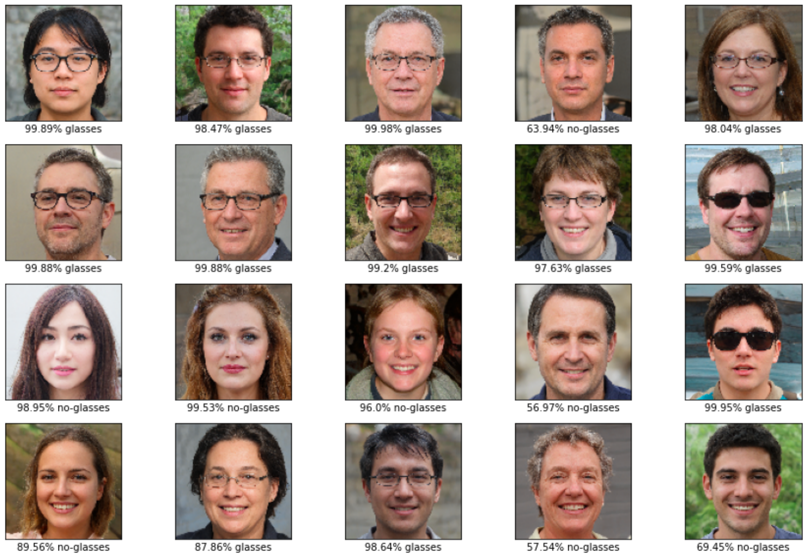

Figure 9: Visual test over balanced dataset with the respective network

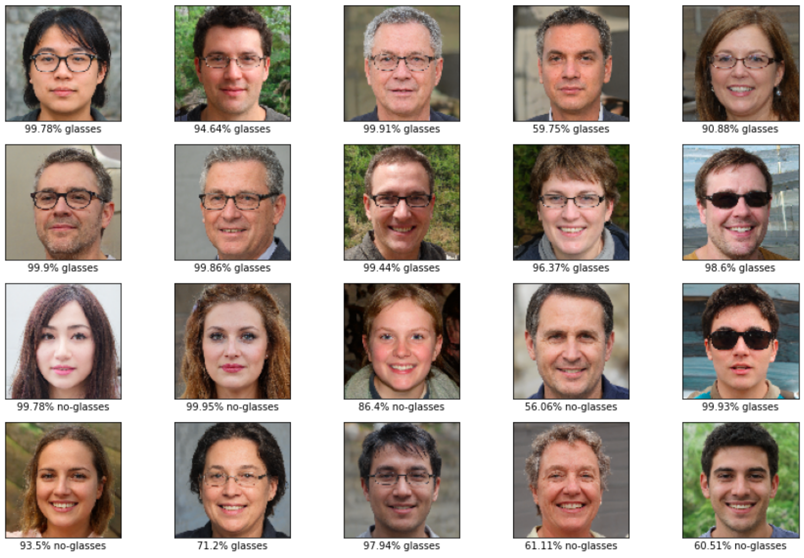

Figure 10: Visual test over unbalanced dataset with the respective network