Glasses or No Glasses

Experimental joint project for
"Algorithms for Massive Datasets" &
"Statistical Methods for Machine Learning"

Nicola Manca, 979311
Master in Computer Science, Università degli Studi di Milano

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.

CONTENTS

I. Introduction

In this report I present and analyze the experimental results of a *Convolutional Neural Network (CNN)* applied to *Jeff Heaton*'s dataset *"Glasses or No Glasses"* [1], which is composed by *5000* images of different faces, generated by a *Generative Adversarial Network (GAN)*. The dataset also includes, for each image, its label and the *512* number latent vector that was used by the GAN to generate it, however I decided not to use those for classification.

The goal was to train the CNN to classify if the faces in the dataset were wearing glasses or not. To achieve this, I started by analyzing the dataset and visualizing some of the images, but after a while I realized that a good amount of them had the wrong label. Also *500* of the images didn't have a label at all, so the first work to do was to label all the images myself.

Once that was done I had to balance the two classes in the dataset, because there were mostly *'glasses'* images, by removing some of them. After that I preprocessed the images before classification: reducing their size and normalizing pixel values. Finally I created my CNN model and tested various combinations of hyperparameters values to find the best one, reaching a satisfying level of accuracy in the test set.

The project has been developed entirely on *Google Colab* [2] and takes around *10 minutes* to run (from downloading the dataset, which takes approximately *5m 30s*, to the results of the CNN on the test set), while using a TPU as hardware accelerator, which is free of charge.

II. Dataset and preprocessing

The *"Glasses or No Glasses"* dataset [1], published on *Kaggle* in 2020 by *Jeff Heaton*, was created for a competition at the *Washington University in St. Louis*. It contains *5000 GAN-generated* images of people's faces, along with two *csv* files (one for the train set, the other for the test set) indicating their labels and the *512* numbers vector used by the GAN to generate each image. The images are in the *PNG* format, they are RGB and their size is *1024x1024*. The *csv* files divide the dataset into *train* and *test* set, the first *4500* images belong to the former and the remaining *500* to the latter.

When starting to visualize the dataset, I immediately discovered that a lot of labels were wrong *(around 10% of them)*, so before doing anything else I decided to create my own *csv* files and label the *5000* images myself. I did this with the help of a script I created that visualized 9 images at a time, and allowed me to quickly write all the labels. Doing that, I also realized that some of the images were distorted, due to some errors in the generation, sometimes they included features or parts of other faces, and in others it was difficult to state whether they had glasses or not (often only some parts of the glasses were present, *e.g.* only the temples, only the lenses), but fortunately there were only a few of them and I decided to remove *6*. With my new labels and after removing the distorted ones, there were *4994* images left, *2800* of the *'glasses'* class, and *2194* of the *'no glasses'* class. Seeing that the dataset wasn't balanced *(it also wasn't at the beginning)*, I had to remove the surplus of *'glasses'* images from my experiment, namely *606* of them, to finally have a total of *4388* images, *2194* of each class.

This was done by creating a script that randomly generates the *IDs* of the images to remove. These previous operations maintained the original ratio of about *1/10* between *train* and *test* set, so at this point the cardinality of the sets was respectively: *3986* and *402* images.

The last preprocessing steps were to reduce the size of the images from *1024x1024* to *50x50* using the '*resize*' function of the *cv2* library *(this was done to speed up the training of the model without compromising its accuracy)*, and to normalize the values of the pixels, from the range [0, 256] to [0, 1], this was done as it is good practice with deep neural networks *(smaller training time, avoid unstable weight configurations that cause worse generalization accuracy)*.

Fig. 1: An example of the GAN-generated images for each class.

Fig. 2: Some examples of distorted images.

III. Classification model and implementation

Once the data was ready, it was time to create the classification model.

Considering that I decided to work only with the images, excluding the *512* numbers vectors that were provided, and the fact that I had to use a neural network, the subsequent choice of a *Convolutional Neural Network* was rather easy.

To create, train, and test the model I used the *Tensorflow* [3] library for *Python 3*, which is open source and contains tools, libraries, and resources dedicated to Machine Learning; it also contains the high level API *Keras* [4], which is built on top of it.

A typical *CNN* starts with the input layer which is also the first convolution layer, followed by a max pooling layer which reduces the dimensionality of the convolutions's output, and this sequence of two layers can be repeated arbitrarily, possibly with different parameters (generally using *'relu'* as activation function). After all the convolution layers we typically find a flattening layer, which reduces the output to a single dimension, allowing it to be processed by a dense layer (*i.e.* a layer where each neuron has connections coming from every neuron of the previous layer). Ultimately we have the output layer which is also a dense layer and takes inputs from the previous one to determine the predicted class.

So this was the general structure of my model, but I had to determine the best parameters to achieve a good result, this phase is called hyperparameter tuning. An important choice is also the number of convolution layers to use, although it is always better to start with simple architectures and with just one convolution layer I was able to obtain a great result; I tried adding a second convolution layer but the results didn't get better.

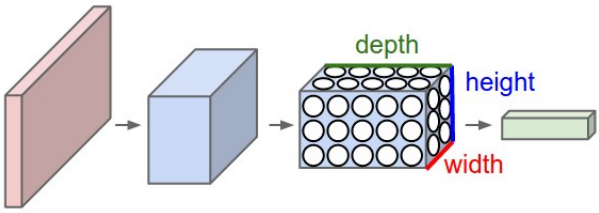

Fig. 3: General structure of a CNN for RGB images (3D input).

Also, considering that this is a classification problem with two classes, I already knew that my output layer would have one neuron with *'sigmoid'* as activation function. To tune the hyperparameters I used the function *GridSearchCV* from the *sklearn* [5] library, which performs an exhaustive search over the specified parameters values (provided as an argument) and returns the best configuration of values between all the possible combinations (using cross-validation over the data points, in my case the training samples). I chose the hyperparameters to tune and provided the function with some common values for this kind of models (I looked up other people's solutions to similar problems), and then added greater and lower values (which I previously also tested manually). I selected the following hyperparameters: *number of filters in the convolution layer*, *size of the filters in the convolution layer*, *size of the windows in the max pooling layer*, *number of neurons in the dense layer*, *batch size*, *epochs of training*. The *GridSearchCV* function ran for nearly 10 hours and then presented me with the best configuration of values (which coincided with the one I found while testing manually), so the final structure of my model was the following:

- a **convolution** layer, with **64 *3×3* filters** and *'relu'* as activation function
- a **max pooling** layer with ***2×2* windows**
- a **flattening** layer (which transforms the input into a 1D array)
- a **dense** layer, with **128 neurons** and *'relu'* as activation function
- an **output** layer, with **1 neuron** and *'sigmoid'* as activation function

With the following settings for compiling the model:

- loss function: **binary cross entropy** (trivial choice for a 2 class problem)
- optimizer: **Adam** [6] (optimization algorithm that can handle sparse gradients on noisy problems)
- metrics: **accuracy**

Finally I could start training my model, using a batch size of **64** (as found by the grid search) and the number of epochs equal to **3**, however the grid search found 4 because it used accuracy as metric and didn't take overfitting under consideration. Training the CNN took less than two minutes.

IV. Experimental Results

Here I present the results of my experiment, which I think are quite satisfying.

Train accuracy	**98.90%**
Train loss	0.0490
Test accuracy	**98.01%**
Test loss	0.0693

To further understand the performance on the two classes, I created a script in which the model makes a prediction for each of the test images, and at the end shows how many examples of each class were wrongly predicted.

The result was the following:

No Glasses	0
Glasses	8
Total examples	402

This shows that there is more uncertainty when predicting images of the *'glasses'* class.

This model, or some slightly modified version, could be also applied to larger datasets (with the images reduced at the same size), maintaining its time efficiency. Also, this problem can be approached in different ways, other than the one I chose, for example utilizing the *512* numbers vector instead of the images (or even using it along with the images); in these other cases there are different models that can be applied, such as a simpler Deep Neural Network with dense layers and *512* inputs for the latent vectors.

REFERENCES

[1] Jeff Heaton's "Glasses or No Glasses", kaggle.com/jeffheaton/glasses-or-no-glasses

[2] Google Colab, colab.research.google.com

[3] Tensorflow, tensorflow.org

[4] Keras, keras.io

[5] Scikit-learn, scikit-learn.org

[6] Kingma, Ba - Adam: a method for stochastic optimization. *arXiv:1412.6980*, 2014.