THE JOINT PROJECT OF

ALGORITHMS FOR MASSIVE DATASETS

STATISTICAL METHODS FOR MACHINE LEARNING

DETECTING WEARING GLASSES BASED ON NEURAL NETWORK ARCHITECTURES

PROFESSORS

DARIO MALCHIODI

NICOLÒ CESA-BIANCHI

BY

ALI RAFIEI

Declaration

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

1- Which parts of the dataset have been considered? 4

2- How data have been organized? 4

3- The applied pre-processing techniques. 5

4- The considered regression algorithms and their implementations. 7

5- How the proposed solution scales up with data size? 8

6- Description of the experiments 9

7- Comment on the experimental results 9

1- Which parts of the dataset have been considered?

The structure of the dataset on the Kaggle website is:

- Train.csv (4.500 samples)
- Test.csv (500 samples)
- 5000 PNG images

Based on the definition of test/evaluation/validation dataset test.csv should have labels for each sample but they are missed.

So I have used all 4.500 samples of the train.csv file as train and test data and worked with the test.csv file as new samples for prediction/inference.

Due to working with the train.csv file which includes 4.500 samples therefore I used the first 4.500 PNG images for train and test data and used the last 500 images for prediction/inference.

2- How data have been organized?

Both, the CSV file and images directory are not divided into the classes of “not wearing glasses” and “wearing glasses”.

2.1. Downloading the dataset by using Kaggle API

2.2. I read train.csv to understand the label of each image.

2.3. Each row/image in the train.csv has an “id” and I need this id is the key to classify images in the image directory.

2.3.1. Create a Pandas data frame for “not wearing glasses images” and consider it as class 0.

2.3.2. Create another Pandas data frame for “wearing glasses” images and consider it as class 1.

2.3.3. These two created data frames for each class have two main columns. One for “id” and one for “label”. Based on the “id” name I create a new column and call it “name”. “name” column is the exact

name of each sample like “face-1.png” and helps me to find the image in the images directory and put it info the “{not} wearing glasses”.

2.3.4. When I divide images into wearing and not wearing glasses at the same time divide each of them in to train and test data like:
directory: train_test_split
→ directory: train
→ directory: not_wearing_glasses (1.315 images - 80%)
→ directory: wearing_glasses (2.284 images - 80%)
→ directory: test
→ directory: not_wearing_glasses (329 images - 20%)
→ directory: wearing_glasses (572 images - 20%)

3- The applied pre-processing techniques.

3.1. Data augmentation: this technique would be useful especially when the number of data is not enough.
How does it work? By using some arguments like zoom range, shear range, rescale, horizontal flip, vertical flip, and some other techniques create new images and add them to the model to feed the model with more and different samples to learn more efficiently. This technique is used both for train data and test data.

3.2. The size of raw images in the original dataset is 1024x1024 with PNG format, during loading the directory of train and test samples I reduce them to 128x128 with JPG format. This approach would be helpful during convolution tasks in the deep neural network training process.

3.3. Dropout: Fully connected architecture refers to the model that there is a connection/weight between all neurons. This kind of model leads to reaching high accuracy on training data but sometimes leads to overfitting. To avoid overfitting we can randomly remove some of these connections to make the learning process for the model a bit challenging.

3.4. Early Stopping: sometimes we witness significantly differences between the accuracy of the training dataset and the accuracy of the test dataset, for example during training after some epochs we see the model start to overfit. For avoiding this kind of problem we can use the early stopping technique. The argument of monitor could be validation loss so the value of mode should be minimum because we want to find the lowest value for losses. The patience argument is about the number of epochs to wait to be hopeful to see a positive effect.

3.5. Reduce Learning Rate On Plateau: when we do not see a huge amount of changes in the value of accuracy or loss it means we are close to the local minimum so we need to reduce the learning rate changes like weight updating.

3.6. Model Checkpoint: By using early stopping we can stop the model to overfit but still it is possible we stop the training process in the 17^{th} epoch of 20 while the best accuracy has been heppend in epoch15. In this situation, we use model checkpoint to save the last best model.

3.7. Manage input/output neurons: The shape of original images is (1024x1024x3) but I reshaped them to (128x128x3) so I need to fix the input shape of the first layer of the architecture to (28x28x3). On the other hand, the number of classes is two so I use only one neuron at the last layer which could be 0 or one to show all 2 classes.

3.8. Balancing labels weight: sometimes the number of different classes is not balanced. For example, I have 10 train data for class A and 6 train data for class B. so in this situation I need to make my model aware that consider more weight to class B during training to cover the lack of enough data. Another way to overcome this problem is using data augmentation which is mentioned in section 3.1.

4- The considered regression algorithms and their implementations.

For the implementation of the model, the TensorFlow library is used.

The model is created based on deep neural network architectures.

4.1. Input layer:

4.1.1. Input shape: The image is inserted as a 128x128x3 3d-array which 128x128 is the image size and 3 is the number of the image's channels (RGB). So I need to fix the input shape to (128, 128, 3).

4.1.2. First convolution: I apply 32 3x3 filters to the image. We know these kernels (filters) have different weights which are updated during feed forwarding and back propagation scenario.

4.1.3. Activation function: by applying filters on images we should decide about the output value. RelU activation function puts zero for less than zero values and the same value for outputs that are more than zero.

4.1.4. Padding: due to using convolution techniques we will lose some information about some pixels of the image, especially corner pixels so I fix padding to "same" to stop losing more information about input data.

4.2. The I use dropout technique and remove about 20% of data about weights to avoid overfitting (this concept is described in more detail in section 3.3).

4.3. Max Pooling:

4.3.1. Filters: Next operation is max pooling, like the convolution step here I also have 32 filters but they are 2x2 and by applying them to the image we will choose the greatest number.

4.3.2. Stride: it means the movement of the filters on the image is not 1 pixel by 1 pixel but also 2 by 2.

4.4. Repeat similar operations like sections 4.1 and 4.3

4.5. Flatten: here I want to connect the CNN architecture to the fully connected or dense neural network, so I need to reshape the 3-dimensional matrix to a 1-dimensional vector. The output of the last

operation is 64 32x32 filters so I reshape it to a 64x32x32=65.536 structure.

4.6. The next layer includes 128 neurons. This number could increase or vice versa, next we can add another layer with some other neurons, also it is possible to add a dropout task between them, but for our model based on some learning experience and the number of data, it seems good enough to start.

4.7. Compiling:

4.7.1. Optimizer: for updating the weight during each epoch we need to use an optimizer. They are different ways to update weights and one of them is Adam.

4.7.2. Loss: to measure loss value we need to compare the real value with the evaluated value and here because of working with 2 classes I use binary cross entropy.

4.7.3. Metrics: available options are accuracy and loss.

5- How the proposed solution scales up with data size?

5.1. One technical solution is using GPU instead of working with CPU to increase the speed of learning or using TPU during working with tensors (multi-dimensional matrix)

5.2. Distributed computation: these days instead of using vertical scaling of hardware which means making stronger the limited number of computers we use horizontal scaling which means using a high number of computers with good enough resources. We can distribute computations by using map reduce techniques.

5.3. Collecting data in a smart way: the input of the model could be image. This input will be downloaded, unzipped, resized, augmented and maybe other operations to reach a suitable structure to feed the model. We will work on different architectures to find the best or at least the good enough

structure for our algorithms so we can save each result after different operations on data and use them for next training tasks.

5.4. Hyper parameters tuning: for example if we have more than one million images setting the batch size to 10 leads to a really slow training process, when the number of data increases we can tune this kind of hyperparameters to increase the speed of learning.

6- Description of the experiments

6.1. Loading the data: based on what is described in section 2 I read CSV files and connect the labels to the images directory to prepare train and test data.

6.2. Setting hyperparameters: some hyperparameters for preprocessing data and some for training the model in a smart way like what is mentioned in section 3.

6.3. Start training: training is done on the personal computer and Google Colab by using CPU for computation.

7- Comment on the experimental results

7.1. Batch size: the number of data for both classes as the training dataset is 3.599 and the value of batch size argument during loading data was 500, so it means in each epoch we have 3599/500=7.198 feed forward and backpropagation for updating the weights that we need to round it to 8.

7.2. Epochs: it refers to the number of iterations that all the data are seen for the training process.

7.3. Accuracy: after each epoch, the accuracy will calculate based on training data.

7.4. Val Accuracy: at the end of each epoch after training and calculating the accuracy for training data the Val Accuracy will be measured for test data.

7.5. Loss and Val Loss: the loss value will be measured after each epoch both for training data and test data.