Computer Vision Report

Authors: Jorge Martín, Mateusz Klimas, Juan Luis Ruiz-Tagle

Introduction

This report aims to summarize all our work done on the computer vision part of the deep learning course. All the exercises convoluted around the same topic, namely identifying and classifying images of traffic signs. Two datasets were provided: the German Traffic

Sign Benchmark and the CIFAR100 dataset. Three problems were laid out, which shared very similar goals but were approached from different perspectives. In the following sections of the report, these three problems will be exposed to detail.

Our approach

To develop this assignment, we have followed several steps in order to achieve good results in our detection and classification model. A brief summary can be read in this section.

First of all, we have decided that any bounding box located either on the top or the bottom of each image are irrelevant (100 pixels of the top and 100 of the bottom), as the signs in the images are located in the central part of each image.

The next step is to get rid of the bounding boxes that don't belong to a sign. To achieve this, we have developed a specific classifier capable of distinguishing between if an image is an image of the background or it is an image of a sign. A dataset has been generated using the sign images (gotten from the bounding boxes) and getting random images from the background. The images of the background have been generated slicing each image in many small images that don't contain any signal (using the bounding boxes to ensure this statement).

Once we can remove the remaining bounding boxes that don't belong to a sign, we apply the technique called Non-Maximum Suppression (NMS) in order to deal with bounding boxes that are overlapping each other on one signal. This technique consists in taking a list of proposal boxes with a confidence score and a threshold, and it returns a list of filtered proposals with only one bounding box for each detection (no overlapping). Each of the proposed boxes are classified in order to get the one with the highest confidence.

Problems encountered

During the development of our assignment, we have faced different problems. One of them was that when we wanted to perform data augmentation over the signal images, if we set a high number of parameters (which results in many new images), the compute engine machine started to crash, which led us to restart the machine every team without any results. To solve this, we increased the machine's memory, which solved this problem.

Another problem that we have encountered was that the first classifier that we obtained using the default images obtained from the dataset, its accuracy was stuck around 80%. After using some data augmentation techniques, like rotating and changing the brightness of the image, we managed to improve this accuracy to 92%.

Also, when we were developing our detector for the signals (to know if an image belongs to the background or a signal), at first, when we trained our model we were getting the same accuracies for all the epochs, with no improvement. Thanks to the advice of Iago, we tweaked some parameters and we managed to make this model work, getting good results.

FNN results

In the first task, we were only allowed to use layers which had been explained in the lecture at that point in time. Convolutional layers were not available, so we had to solve the problem without their help.

We designed an architecture which first had a Flatten layer to flatten the input image, and then to sets of Dense layers with 250 units followed by Dropouts and BatchNormalization respectively. The dropout coefficient in these layers was set to 0.2, and they used "elu" as activation function. In the end, there was one more Dense layer which condensed all the information from previous layers and outputted a vector with the number of classes (43 in the case of the German signs, and 100 in the CIFAR dataset).

We manually tested different combinations of parameters. We realized that putting the BatchNormalization after the Dense layers was better than putting them before. Training the net more than 80 epochs did not help to get any improvements, so we stack to this number. Other activation functions did not perform as good as elu, so they were discarded. A regular SGD optimizer was used with the default configuration.

Here are the slides for this task.

The best results we obtained were:

- German Traffic Signs: Test Accuracy: 88.9%, Test Loss: 0.5952
- CIFAR 100 dataset: Test Accuracy: 25.6%, Test Loss: 3.20

CNN results

The second task consisted to solve task 1 using CNNs, to understand how powerful they are for computer vision tasks. We tested several values of dropout, kernel_regularization, activation functions, kernel initializers, kernel_size, strides, filters_initial_number and optimizers.

The model architecture we designed had 8 convolutional layers, starting with 2 filters in the first one and doubling the number of filters every second layer (up to 16 filters). After testing different options for the kernel size, we found out that starting with big kernels in the first layer (6,6) and then small ones in the rest (2,2).

Adding Maxpooling2D every 2 layers supposed a significant improvement of the results. Probably this architecture could have been simplified, increasing the number of filters from the very beginning and decreasing the number of layers.

The evolution of the accuracy across the different epochs, as well as final training accuracy and the processing time, is no longer available so we cannot put it here.

Here are the slides for this task.

The best results we obtained were:

- German Traffic Signs: Test Accuracy: 91.41%, Test Loss: 0.4340
- CIFAR 100 dataset: Test Accuracy: 57.34 %, Test Loss: 1.66

Comparison

We have seen how CNNs obtain much better accuracy than FNN. CNNs advantage is their preservation of locality among pixels, and this is reflected in the final results. Another advantage is that they are quicker since they are suited for this specific family of problems.

Data augmentation

One of the decisions made after getting low accuracy training the signal classifier model was to do data augmentation. This is done applying two different techniques.

1. **Brightness** data augmentation: each image is augmented by either randomly darkening or brightening it. This is done with an *ImageDataGenerator* function given by Keras API. The result is the following below:

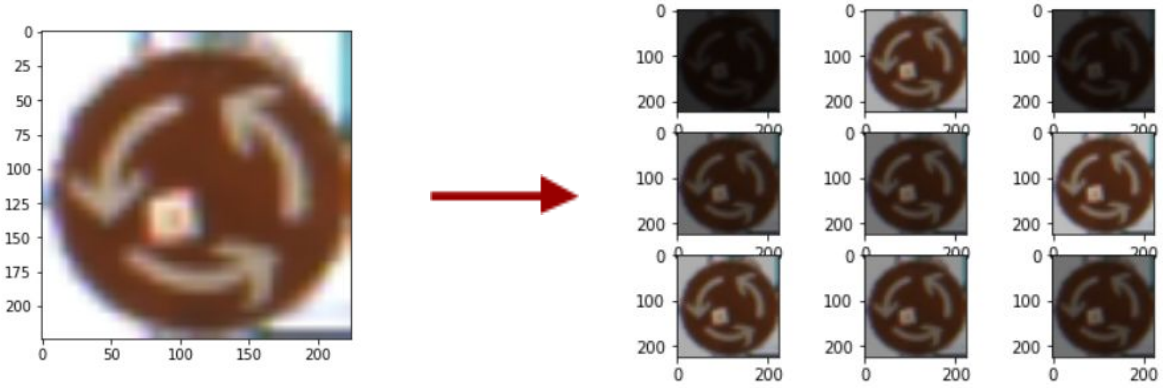

Figure . Result of applying brightness changes to a specific signal.

2. **Rotation** data augmentation: the image given is rotated clockwise and counterclockwise by 5 and 10 degrees. This is done with the OpenCV API and the result is the following one:

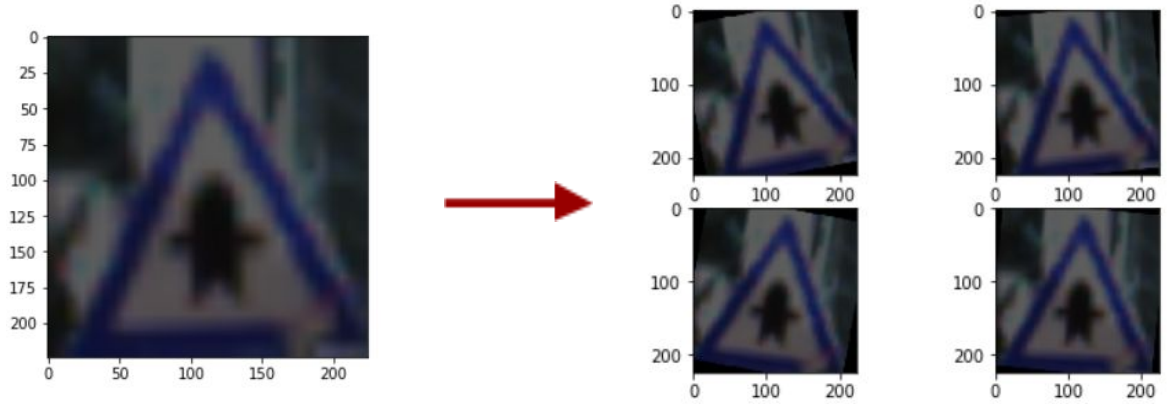

Figure . Result of applying rotations to a specific signal.

Traffic sign detection

In order to perform the signs detection and posterior classification, we have followed the following steps in order to achieve this functionality.

Rule filter

The first thing that we have decided to do before applying any deep learning techniques was to ignore many bounding boxes proposed by the default model using a set of rules. The rules used for this purpose are described in the following list:

1. Eliminate proposal regions whose top-left corner and bottom-right corner are located between the first 100 top pixels.
2. Eliminate proposal regions whose top-left corner and bottom-right corner are located between the first 100 bottom pixels.
3. Eliminate proposal regions whose proportion is not between the 3:4 and 4:3 aspect ratio.
4. Eliminate proposal regions whose size is bigger than 90x90 pixels.

All these rules that we have defined come from the analysis that we have performed checking that, for example, signs in the given dataset don't appear neither on the top or bottom of the image, they are always located more less in the middle. Related to the aspect ratio, we can have variations in the shape of the rectangles obtained by the detector, but the tendency is to have squared rectangles, which means very wide or high rectangles doesn't make any sense, as most of the signals can be boxed in a square. Finally, regarding the size of the proposal regions, the signs that appear in the images are very small, which leads us to the conclusion that accepting big proposal regions doesn't make any sense, as there will not be any big signal in the images.

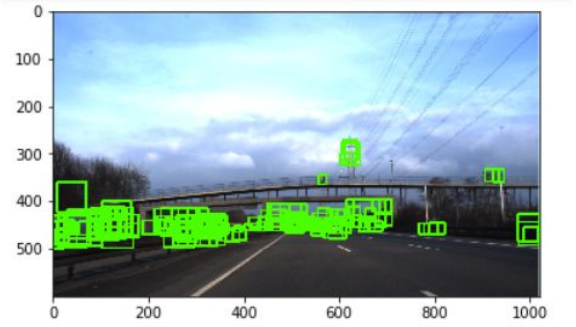

Figure . Result after applying the defined rules.

Background detector

Once we have applied the previous rules, now we have to deal with the remaining proposal regions that we have, but some of them still don't belong to a signal. To solve this issue, a model capable of distinguishing between background and signals has been designed. To design this model, first of all we need a dataset containing information about both classes (background and signal).

To obtain images of the signals, we use those which we obtained through the bounding boxes that are defined in the original dataset. These original signal images are augmented using the data augmentation techniques described previously. We obtain around 10.000 images after all this process.

For the background images, we created some functions that allowed us to slice each of the original images and keep the slices that don't contain any signal. The slicing size that we chose was 50x50 pixels. After the generation of each slice, we check if that slice is overlapping with some of the bounding boxes belonging to that image. If there is an intersection with a signal's bounding box, then that slice is not used and the algorithm keeps going for the next slice. After the generation of all the slices (around 210.000), we apply another function to this set in order to remove those slices whose "*whitness*" threshold exceeds the 85% (this "*whitness*" threshold is the average of all the pixels of the image, and if there are many 0 pixels, or close to 0, then the average "*whitness*" will be high). We apply this function to get rid of very white slices, as they mostly belong to the sky part, and with the previous rules we are already dealing with that proposal regions. After the filtering, we reduce the number of slices to 160.000.

Once we have generated enough images for each class, we take the 10.000 images belonging to the signals, and 20.000 images belonging to the background class, each of them correctly labeled. We put them together, shuffle and separate in train, validation and test sets. After 5 epochs, we have managed to get an accuracy of 99.15% on the test set.

Non-maximum suppression

To prevent overlapping proposed boxes from giving the same result multiple times, a non-maximum suppression algorithm is used. Given a list of proposal boxes B (red, green and blue) with its confidence scores and an overlap threshold N.

The steps to follow are

1. Select the proposal with the highest confidence score, remove it from B and add it to the final proposal list D. (Initially D is empty).
2. Now compare this proposal with all the proposals — calculate the IOU (Intersection over Union) of this proposal with every other proposal. If the IOU is greater than the threshold N, remove that proposal from B.
3. Again take the proposal with the highest confidence from the remaining proposals in B and remove it from B and add it to D.
4. Once again calculate the IOU of this proposal with all the proposals in B and eliminate the boxes which have a higher IOU than threshold.
5. This process is repeated until there are no more proposals left in B.

Finally, if the overlap threshold of our example is smaller than the IoU between the green and blue boxes and the green and red boxes, the result would be this one:

Figure. Overlapping problem before and after non-maximum suppression

Grid search

The last step of the pipeline consisted in setting up a grid search over some of the parameters that could be tweaked, in order to obtain the combination that maximizes the test accuracy. In the image below the parameters that were tested are displayed and ordered from highest to lowest accuracy (result column). A deeper grid search was performed prior to the one that uttered the results in the table, with up to 48 different combinations of the parameters. After realizing that high values of **SCORE_THRESHOLD** were not working properly and that the optimal box_size in pixels was 90x90 (8100), we launched a second grid search exploring different values for **box_ratio** and **clf_confidence.**

	IOU_THRESHOLD	SCORE_THRESHOLD	background_prob	box_ratio	box_size	clf_confidence	folder_name	pixels_borders	result
4	0.5	0.1	0.7	0.75	8100	0.1	final_config_4	100	25.35
6	0.5	0.2	0.7	0.75	8100	0.2	final_config_6	100	25.35
5	0.5	0.1	0.8	0.75	8100	0.1	final_config_5	100	25.20
7	0.5	0.2	0.8	0.75	8100	0.2	final_config_7	100	25.20
1	0.5	0.1	0.8	0.65	8100	0.1	final_config_1	100	24.96
3	0.5	0.2	0.8	0.65	8100	0.2	final_config_3	100	24.96
0	0.5	0.1	0.7	0.65	8100	0.1	final_config_0	100	24.92
2	0.5	0.2	0.7	0.65	8100	0.2	final_config_2	100	24.92

Conclusions and feedback

The highest accuracy we have obtained after applying all the aforementioned techniques and ideas, our highest accuracy is 25.35%, using the set of hyperparameters listed on the first row of the grid search image. We can conclude without a doubt that the traffic sign detection and classification problem is very hard. We have surely learnt a lot during the process. The slides of this last part of the work can be checked out here.

Regarding the feedback, we have to say that the course was well structured and we appreciate the motivation of the teacher and TA to teach us everything in detail. Using Google Cloud and learning about its potential has been very valuable for us. However, there are some things that could be better. Some thoughts:

- Throughout the whole project our notebook had struggled with memory problems. The RAM could not handle the amount of data that was necessary and this is something that we realise at the end of the project. Probably an early recommendation or advice about the machine configuration possibilities and limitations would solve these problems. We think that the cheapest GPU was not sufficient for our needs and caused some problems.
- In our opinion, the accuracy competition should be tested in a different way. A result upload shouldn't be enough. Probably the way in which platforms like Kaggle test the results is better, checking the labels that every group gets, instead of the accuracy result itself. You could provide a test set without labels and set up a script which automatically tests the accuracy from each group.
- In order to get the most out of these labs, we would appreciate that the code of whomever obtained the best solutions was shared with the rest of the students. We admit that after testing so many different techniques and approaches we are a bit disappointed with our own results. We are really curious about what we could have done to improve our results.

References

Keras documentation: https://keras.io/
Non-Maximum Suppression: Py Images Search: NMS
Rotate images for data augmentation: https://docs.opencv.org/
Data augmentation: ML mastery: How to configure image data augmentation