Robotic Practicals
Topic 13: Visual Navigation

Group 20, Vuk Pajovic, Darko Lukic, Leonardo Cencetti

Spring Semester 2018/2019

1 Introduction

In this practical the goal was to design and implement an image detection algorithm in Python. For this purpose PyTorch is utilized to model and train a Convolutional Neural Network. The model is later uploaded on the Segway Loomo robot in order to test how it works in a real-world application.

2 CNN model development

The first part of this practical concerned training and testing the Convolutional Neural Network(CNN) for detecting the particular object in the picture. The objective was to detect and find in a picture of a Minion, reported in figure 2. Intersection over Union was used as the performance evaluation criteria for the image recognition. Furthermore, for the optimal results, the algorithm should detect if there is a minion in the picture, and only if there is one, recognize it and find the borders. This means that the network objectives were both classification, to detect the presence of a minion, and regression, to find the borders of the minion. For this reason, the overall loss function included two different loss functions, one for each objective. The fist one, used for the classification, was Binary Cross Entropy loss function. For the regression part, the Mean Square Error loss function was used. In order combine both into one single loss function, a parameter γ was set to assign the weight of each loss function in the total. The resulting loss function was the following:

$$L = (1 - \gamma)L_{bce} + \gamma L_{mse} \tag{1}$$

where the L_{bce} is the Binary Cross Entropy loss function, and the L_{mse} is the Mean Square Error loss. The parameter γ is one of the several hyperparameter used in this CNN implementation. The other parameters are the Learning rate, number of epochs and batch size. The next section will address the choice of their values and their tuning.

3 Questions

Q1—*In your opinion, which of the three techniques is the most effective for hyperparameters tuning?*

In the assignment paper three methods for hyperparameters tuning are proposed:

- Babysitting the model (manual search),
- Grid Search and
- Random search.

The most effective technique depends on the use-case. Manual search requires manual intervention and good understanding of a model. It is useful if one posses a very good understanding of the problem and has limited computing resources, but often tends to incur in personal biasing of the results.

On the other hand, grid and random search are generally more efficient in finding the right value. Manual intervention is not required but the selection of a good initial ranges of hyperparameters might significantly improve the performance. The main drawbacks of grid and random search methods are the required computational power and time complexity of the optimization, since the possible combinations of parameters tend to grow exponentially.

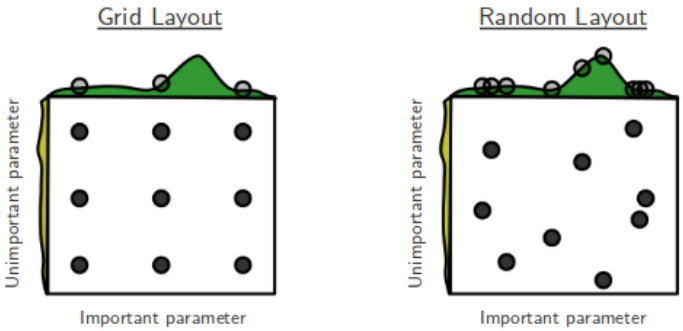

Figure 1: Grid search vs Random search [1]

Comparing those two techniques we can tell the random search is better for our use case. For example, in Fig. 1 using random search an improvement caused by "Important parameter" is detected. In contrast, the same improvement is not detected by grid search. This is the main problem of grid search, which might sometimes not find the optimal combination of parameters. This, of course, can be addressed by using an higher "sampling frequency", more adequate to the underlying performance surface, although this would significantly increase the number of combination to be tested and therefore the computation time.

Q2—*Why do you think is the reason?*

Random search gives better results in cases when hyperparameters are not equally important. In our case, after some testing with arbitrary values, we could see that the batch size hyperparameter had a very small impact on the classification and regression performance. Furthermore, it was found that the most important parameters were learning rate and γ.

Q3—*Which are the hyperparameters the model is most sensible of?*

As mentioned in the previous question, our model was the most sensible to γ and to the learning rate hyperparameters. In order to select the best values for our model, we first defined the performance criteria. The mean of the Intersection over Union was used both for the training set and for the testing set. Additionally, the binary classification accuracy was used to check how good was the classification. Since the binary classification accuracy could easily reach 99 percents, our main goal was to obtain the highest possible mean IoU. The highest values obtained for the mean IOU was the 65.1 percents, with the binary classification accuracy of 99.38 percents for the training set. The hyperparameter's values used for the mentioned result are reported in the table **??** in the column "Best Value".

Hyperparameter	Best Value	Min Value	Max Value	Number of combinations (different)
Learning rate	0.005	0.005	0.01	4
Number of epochs	10	10.0	30.0	6
Batch Size	50	50.0	1000.0	10
Gamma	0.99	0.5	0.99	12

Table 1: Investigation of different hyperparameters

Q4—*In your opinion, why the object detector is less effective on images taken from a real-world camera?*

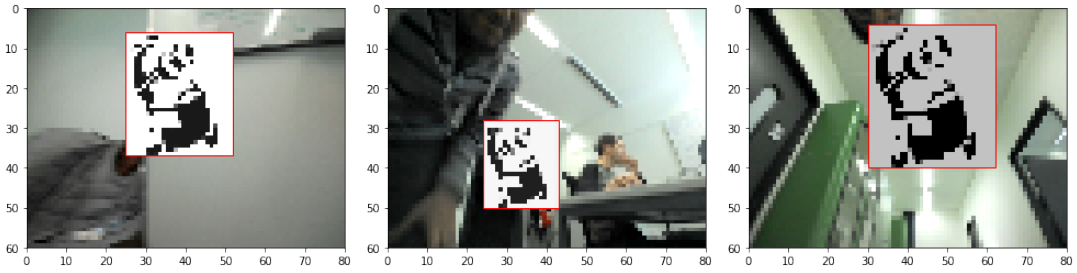

Figure 2: Examples of images generated for training

Firstly, the dataset includes a limited number of images, which characterize a wide range of backgrounds, but not all off them. This meant that while using the setup in the real-world, the pictures obtained for the object detection could have different backgrounds, and therefore make object detector less effective at understanding what is background and what is not. Another problem is how the training set was generated: if we check Fig. 2 we can see that the minion is superimposed onto the background, without any significant change. In most real-world tests, the pictures of the minion look very different from the dataset, mainly due to the different lighting conditions, white balance, geometrical deformations and different sharpness.

Q5—*Which are the main differences with respect to the settings the model has been trained on?*

The original training dataset includes images of many different backgrounds and a single image of a black and white minion (which is our target). The images are later augmented separately and merged together to create a new dataset for our neural net.

The following transformations and effects are applied to transform the minion and background images:

- Translation and scaling of minion,
- Brightness, contrast, saturation and hue to both, minion and background images and
- Horizontal flips to both, minion and background images.

Those transformations do not reflect a real-world situations very well. In order to improve the performances in a real-world we could applied additional transformations and effects to the initial images:

- Rotation, skew, shear, perspective changes,
- Heterogeneous brightness, contrast, saturation and hue,
- Crop (our fingers were hiding parts of the picture),
- Fold effect (the picture in a real-world is printed on a paper which folds easily)

Implementing the additional methods for image transformation would increase the variety of out input dataset, covering most real world cases. We would achieve much better performance in the real-world tests at the expense of increasing significantly the training period. This would probably limit the possibility to finely tune hyperparameters, since the time to test all combinations would be much larger. Another possible improvement of the dataset is to include pictures of the minion pattern, and with even more variety of the background colors and setup.

4 Conclusion

During the practical, the CNN was successfully trained to achieve a mean IoU of 65% and a classification accuracy of 99%. The model was then exported and deployed on the Segway Loomo. The robot was successfully tracking the target image but with sub-optimal performances due to insufficiently high quality of the training data.

In this exercise we also evaluated the advantages and disadvantages of different loss functions for regression and classification. We investigated all the different methods for the fine tuning of hyperparameters and learned how to achieve good results with the minimal training time.

References

[1] James Bergstra and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization”. en. In: (), p. 25.