Academic implementation of Faster R-CNN

Cecilia Martínez Martín
cemm@kth.se
950217-1524

Tobias Niewalda
niewalda@kth.se
930925-6833

David Villagrá Guilarte
dvg@kth.se
950904-9012

Abstract

The following project consists in the implementation and analysis of the Faster R-CNN structure [6], which was designed for object detection. We focused on understanding the underlying structures of the network and how they work together in order to correctly classify the input images. The final aim of the project is to create a network that can be implemented in an outdoor road-like environment and applied to autonomous driving. In order to do that, an academic version of the Faster R-CNN architecture was implemented and tested, using different parameter settings and with two different convNets: ZFNet and GoogLeNet. These networks were trained given as input the CIFAR-100 dataset. The complete object detection structure adds an Anchor algorithm and a RoI pooling layer before handing in an input image to the ConvNet already trained.

1 Introduction

During these last years, autonomous cars have experienced a boost in their performance thanks to (between other critical developments) the latest improvements in Deep Learning (DL) based object detection architectures. Most vision-based DL projects aim to perform classification as accurately as possible based on a standard dataset of images, but object detection requires much more, due to the fact that, for one input image, the output is completely variable (whether there are several, one or zero objects in the image). Also, the output by itself has to include both a classification problem (for detecting which object it is), and a regression problem typically, in which the boundary box of the object classified is estimated.

That's why the main aim of the project focuses on object detection, along with its position estimation. It is an interesting and very complicated challenge that is state-of-the-art at the moment. More specifically, the project aims to detect objects in an outdoor environment, mainly 'road environment', with the goal of producing results that improve autonomous driving. For our project, we decided to implement the already existing Faster-RCNN architecture.

The faster R-CNN network has three major components: a CNN architecture that converts the input image to a feature map, a RPN (region proposal network) that extracts *important* parts of the feature map (based on a parallel architecture with a Anchor extraction and a trainable CNN network), so that the last Fully Connected layers, that apply both classification and regression, only work with *significant* input. Due to the limitation of this project, an academic version of the Faster-RCNN architecture was implemented. For this, the decision of which are the *important* parts of the image has been simplified so that the original image is randomly cropped with specific scales, and the extracted pieces (anchors) are then sent to the ConvNet in order to classify the objects.

For testing the final outcome of object detection, the architecture has been tested using the Kitti dataset, which mainly represents outdoor road scenes, so that the result can be compared to what the real application would be.

With the analyzed parameters, we have come up with the conclusion that the CNN implemented is a more influencing factor in the performance than the parameters of the network. The computing speed is also affected by the complexity of the network.

2 Related work

The latest networks designed for object detection are the YOLO network, the Detectron network and the Faster-RCNN architecture, all highly complex DL architectures aiming to provide utmost efficiency. For our project, we decided to implement the Faster-RCNN, as it was proven to be more accurate than YOLO [2].

The most similar implementations, compared to this project's work, was R-CNN, but after some years it has been improving, first with Fast R-CNN and later with Faster R-CNN [6].

With regard to the CNN used for each architecture, most of the found implementations use VGG16. Nevertheless, this is a parameter that may be changed in order to improve the performance of the network. An implementation of the network can be found in `https://github.com/rbgirshick/py-faster-rcnn` using Caffee. For this project two networks have been studied: GoogLeNet and ZFNet.

GoogLeNet was the winner of the ILSVRC 2014 [8] achieving a top-5 error rate of 6.67%, which was very close to human level performance. Some implementations can be found in `https://github.com/rainer85ah/Papers2Code/tree/master/GoogLeNet`, which has a model of the Inception (V1) network implemented in Keras, as well as many other models of different netwoks.

ZFNet was the winner of the ILSVRC 2013 [8], achieving a top-5 error rate of 14.8% by just tweeking the parameters of the AlexLeNet network. There are also several implementations of this network, as the one found in `https://github.com/rainer85ah/Papers2Code/tree/master/ZFNet`

3 Data

There are several existing datasets available for training. Among the most popular, the selected ones for the project were CIFAR-100 for training and Kitti for testing.

3.1 CIFAR-100 dataset

The CIFAR-100 dataset is an image dataset designed for image classification. There are two different CIFAR datasets, depending on the number of classes that it contains. The large version (CIFAR-100, which is the one used in the project) consists on 600 images of each class (there are 100 classes grouped into 20 superclasses). Each image has one label corresponding to the class and other label corresponding to the superclass [3].

As the Kitti dataset has 'background' labels, there was a need of including a new set of images labeled as 'background' in the original CIFAR dataset. In order to do it, some weighted random images of size 32x32x3 (same shape as the original dataset) have been created. The number of added images is 500 for the test set and 2500 for the training dataset, corresponding to a 5% of the total images of each dataset. These images have been manually added to the binary files downloaded from the CIFAR page [3]. Since we had to start the simulations earlier the extended dataset has not been used for training.

The labels used for training are the coarse labels from the dataset, i.e. the ones corresponding to the superclasses.

3.2 Kitti dataset

The Kitti dataset consists on images on a road-like environment, with more than one object. Each object has a label and a ground truth bounding box. It contains a total of 7481 training images and 7518 test images. As each image contains more than one object, the total number of labeled objects is bigger than the number of images (unlike in the CIFAR set), being the total number of labeled objects 80.256. The images have been downloaded from [7].

The bounding boxes of each labeled object are represented by their top left and bottom right corner. They are rectangle boxes, unlike the blobs that other datasets, such as COCO's annotated data.

The images from this set haven't been touched before the cropping, but the labels have. As the labels should correspond to the ones in the CIFAR dataset in order to be able to test the results obtained from the network, they have been changed and all the objects from the images have been transformed into labels from the CIFAR dataset. This way, 'car', 'truck', 'bike' and similar labels have been relabeled to 'vehicle' and 'person sitting' or 'pedestrian' to 'people'.

The dataset has been analyzed in order to find the best cropping sizes, as well as the distribution of the amount of labels of each class. After the analysis, we found that most of the labels corresponded to cars and pedestrians, and that the distribution of heights and widths of the bounding boxes followed a Poisson distribution, with the peak around 60 pixels for the width (maximum value at around 750 pixels) and 35 pixels for the height (maximum value at around 350 pixels). This shows that most of the main two labels have the same anchor box size.

4 Methods

Architectures as R-CNN, Fast R-CNN and Faster R-CNN have most characteristics in common: they all feed the pre-processed input image to a network that creates a feature map, which is fed to a region proposal network. This network first generates a big number of anchors, of different sizes, strides and with different aspect ratios. Then the number of anchors is pruned using non maximal suppression, and the regions of interest (RoI) are created. After this, a pooling layer "normalizes" the RoIs and feeds them to a final network that finally classifies the object or predicts the object frame. A simplified scheme of the network is shown in Figure 1 [1], where the difference of the Faster R-CNN with respect to the other two networks is that, while the other two use selective search to find out the region proposals, Faster R-CNN uses a network to learn them [2].

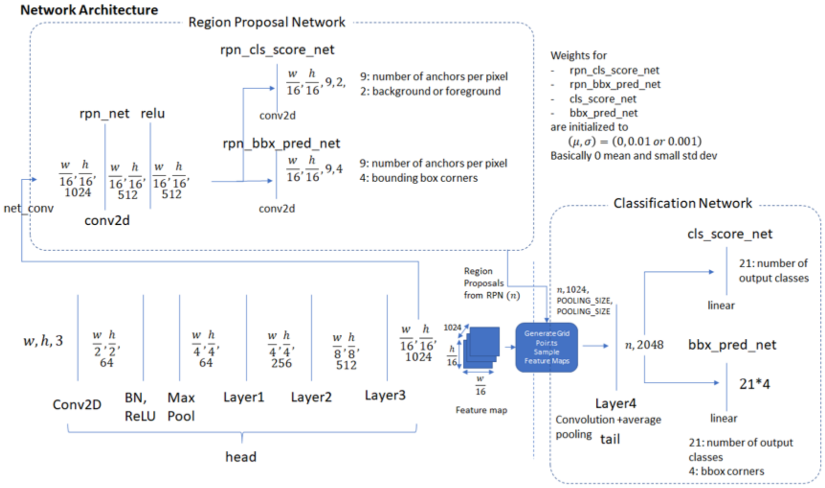
Network Architecture
Region Proposal Network
rpn_cls_score_net
$\frac{w}{16}, \frac{h}{16}, 9, 2,$
9: number of anchors per pixel
2: background or foreground
conv2d
rpn_net
relu
rpn_bbx_pred_net
$\frac{w}{16}, \frac{h}{16}, 9, 4$
9: number of anchors per pixel
4: bounding box corners
$\frac{w}{16}, \frac{h}{16}, 1024$
$\frac{w}{16}, \frac{h}{16}, 512$
$\frac{w}{16}, \frac{h}{16}, 512$
net_conv
conv2d
conv2d
Weights for
- rpn_cls_score_net
- rpn_bbx_pred_net
- cls_score_net
- bbx_pred_net
are initialized to
$(\mu, \sigma) = (0, 0.01\ or\ 0.001)$
Basically 0 mean and small std dev
Classification Network
cls_score_net
21: number of output classes
linear
Region Proposals from RPN (n)
n, 1024, POOLING_SIZE, POOLING_SIZE
$w, h, 3$
$\frac{w}{2}, \frac{h}{2}, 64$
$\frac{w}{4}, \frac{h}{4}, 64$
$\frac{w}{4}, \frac{h}{4}, 256$
$\frac{w}{8}, \frac{h}{8}, 512$
$\frac{w}{16}, \frac{h}{16}, 1024$
1024
$\frac{h}{16}$
$\frac{w}{16}$
Feature map
Generate Grid Points Sample Feature Maps
n, 2048
bbx_pred_net
21*4
linear
Conv2D
BN, ReLU
Max Pool
Layer1
Layer2
Layer3
head
Layer4
Convolution +average pooling
tail
21: number of output classes
4: bbox corners

Figure 1: Scheme of the Faster R-CNN network [1]

Firstly our aim was to replicate the Faster-RCNN architecture (using Tensorflow backend), testing its parameters and optimizing it based on our Kitti dataset. But the structure proved to be too complex for the limited scope of time, and given that our implementation was aimed to be done from scratch, a simplified version of the network, based on Pytorch backend, was then decided to be implemented. The scheme is shown in Figure 2.

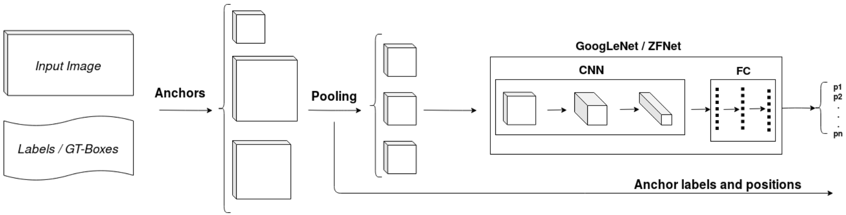

Figure 2: Scheme of the designed network

The network takes an input image, then crop it in several squared sections and pool the cropped images in order to get a group of 32x32x3 images. Then each image is fed into a CNN architecture in order to obtain the probability of that section being an object (and which object), or background.

4.1 The input image

The input image will be taken from the Kitti dataset when testing. The aim will be to find pedestrians and/or vehicles, as those are the two most annotated classes in the dataset . Therefore, we use a selfmade dictionary to relable the images corresponding to the CIFAR dataset that we used for the training. The cars, trams or transporters have, for example, been labeled to vehicles, while pedestrians are relabeled as people.

4.2 The anchors

The anchors have been designed in order to avoid shape distortion after the pooling, That means, we only use quadratic anchors of different scales. Therefore, the pooling will just down-sample larger scales to the required size but not squeeze them. There will be different anchor scales tests, according to the distribution of heights and widths from the Kitti dataset as described in 3.2 . To capture the most commonly annotated objects correctly we recommend anchors of the following size:

- 50 x 50 x 3
- 100 x 100 x 3
- 150 x 150 x 3

and to use a stride of 8 to 20. The anchor labels, as we don't have a region proposal CNN implemented, are retrieved by measuring the distance from anchor position to the ground-truth labels and boxes. Using an overlap based measure turned out to be nice for reduction of non-background labeled anchors, but also crucial to tune. With more time we would have expanded our implementation to detect the most meaningful anchor boxes CNN-based and, therefore, have also reduced the number of output images. Operations as non-maximum-suppression can be used to further reduce the anchors to classify.

4.3 The pooling

The pooling layer has been designed so that the final shape of the input to the CNN layer is 32x32x3. It creates images of that fixed size before serving them as an input to the CNN. Therefore the trained weights on CIFAR-100 are valid for other images (Kitti for testing and generic images for classifying what is in them).

4.4 The CNN + FC

The CNN architecture to use has been a design variable in the project. In this case, two networks have been implemented and tested: ZFNet and GoogLeNet. Due to the complexity of the GoogLeNet architecture, we strongly based our implementation in an existing implementation [9], but the ZFNet architecture was designed from scratch using Pytorch backend, based on the original architecture [10].

For the training of both of the CNNs, the CIFAR-100 dataset has been used using the coarse labels (the ones for the superclasses). For the case of classifying a part of the image as background, and given that the CIFAR-100 dataset does not have annotated data for "background", there were two options:

- Assume that anything with highest probability lower than a threshold for any of the classes is background.
- Create annotated data for "background" and train on those images as well.

After discussing what would be the best option for the task, the selected option was to create new annotated data for the new class. This class has been created by selecting a weighted combination of

RGB values (based on either a β or an unit distribution) and creating images with noised pixels based on those values.

The output of the complete network consists on the probability values of each part of the original image being from one of the trained classes. It is also possible to find the location keeping track of the original position of the cropped image in the input image.

5 Experiments

In order to find the best performing parameters and network for the application stated in the Introduction, several experiments, both in training phase and testing phase have been done.

5.1 During training phase

There are many parameters that can be changed during the training phase, as this is the CNN is the biggest element of the whole architecture.

5.1.1 GoogLeNet vs ZFNet

The first experiment that has been done is a comparison between the two proposed networks: ZFNet and GoogLeNet.

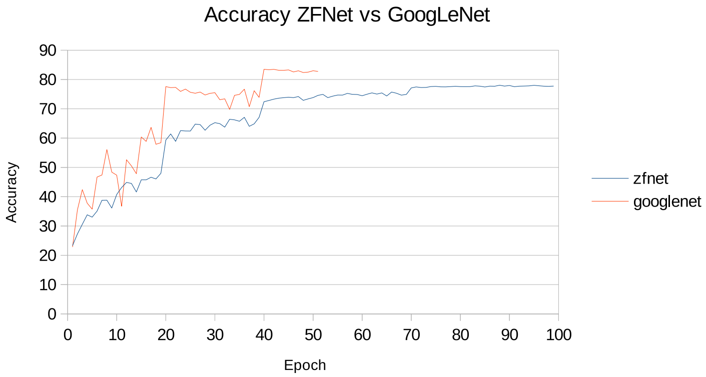

Figure 3: Comparison between accuracies of the two networks

As shown in Figure 3, GoogLeNet CNN has better performance since the beginning, reaching higher accuracies in less epochs. Nevertheless, the performance varies more than in the ZFNet, which has a much smoother graph. The training ran up to 100 epochs for the ZFNet, and for 50 epochs for the GoogLeNet. One observed feature is that, having the same parameter settings, the ZFNet trains much faster than the GoogLeNet, and with higher learning rates the accuracy changes in a smoother way for ZFNet. The GoogLeNet network is much complex (and with more parameters) than the ZFNet. This makes it slower to train and with more accuracy changes between epochs.

For the simulation, a learning rate scheduler was applied, reducing the learning rate by 80%. This way, during the first 20 epochs the learning rate was 0.1, from 20 to 40 it was 0.02, from 40 to 70, 0.004 and up to 100 it was set to 0.0008.

Once the main differences between the two networks were analyzed, the different parameters of each one were tweaked in order to find the best parameter settings. Every experiment has run for 20 epochs and with a batch size of 100.[1]

5.1.2 Default parameters performance

To be able to correclty evaulate the influence of the parameters, the first run has been done using "neutral" parameters so that the comparison can be correctly done.

[1] The batch size influenced the GPU's memory usage. As there was a limitation of space, the influence of the batch size couldn't be analyzed. For a batch size of 100 the CPU memory usage is 5,4 GB, while for a batch size of 50 it is 2,9 GB.

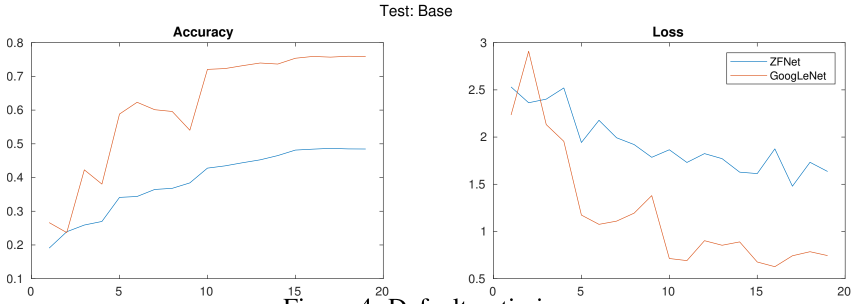

Figure 4: Default optimizer

As expected for the default parameters, and as it had been shown in the previous experiments, GoogLeNet achieves a better performance for the same parameters but with a more edgy evolution.

5.1.3 Learning rates

The aim of this experiment will be comparing a cyclic learning rate (Figure 5) with a scheduled one (Figures 6 to 8), varying the number of epochs when it changes and the decay with every change.

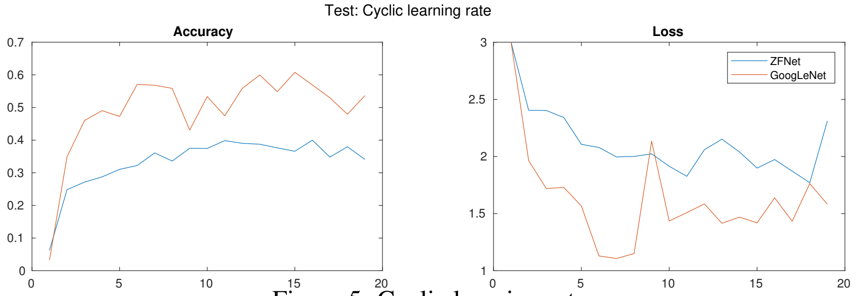

Figure 5: Cyclic learning rate

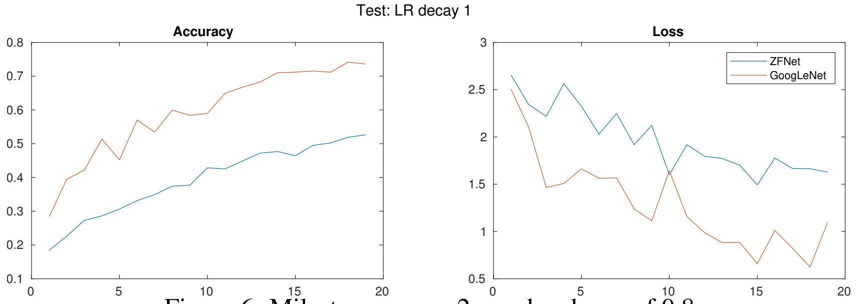

Figure 6: Milestones every 2 epochs, decay of 0.8

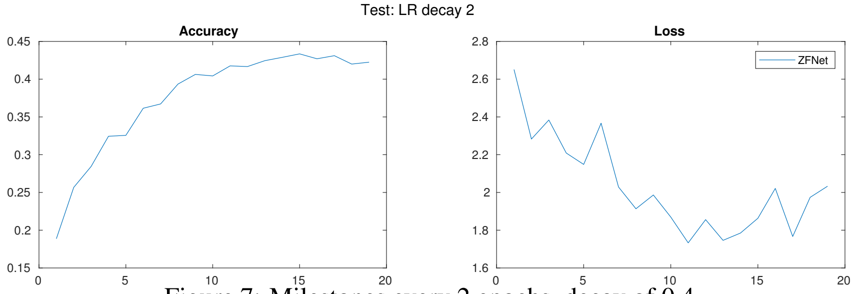

Figure 7: Milestones every 2 epochs, decay of 0.4

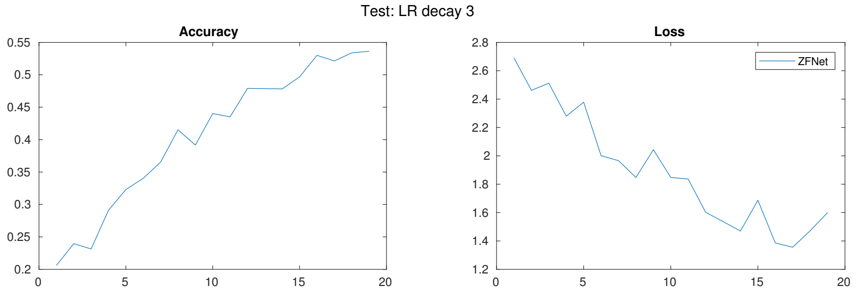

Figure 8: Milestones every 4 epochs, decay of 0.4

For the cyclic learning rate, the training was unfortunately done for half a cycle, starting from 0.01 and increasing every epoch until reaching the 0.1 learning rate. Therefore, the performance is not a good as expected and cannot confirm that cyclical learning rate proves to be worse. When leaving the training during more epochs and completing cycles, the performance would improve as seen in the course assignments.

For the milestone configuration, the higher the decay value was, the more constant the increase of the performance was (basing the results on the outcome of the ZFNet network). The last two milestones scenarios are not available currently for GoogLeNet. For the learning rate decay of 0.4, changing its value less frequently gave better results.

To conclude with the experiments for this section, a summary of the effect of each parameter has been done for each network, and the results are shown in Figures 9 and 10.

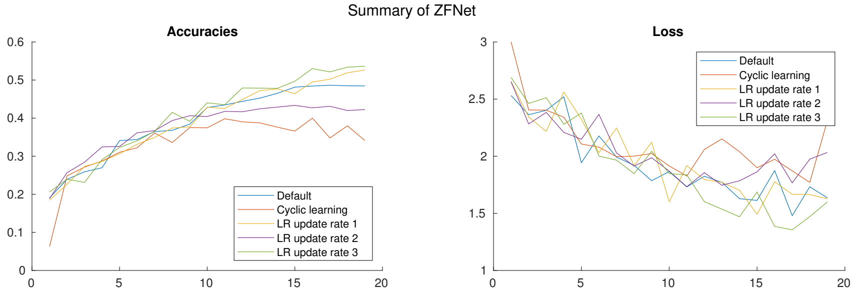

Figure 9: ZFNet summary

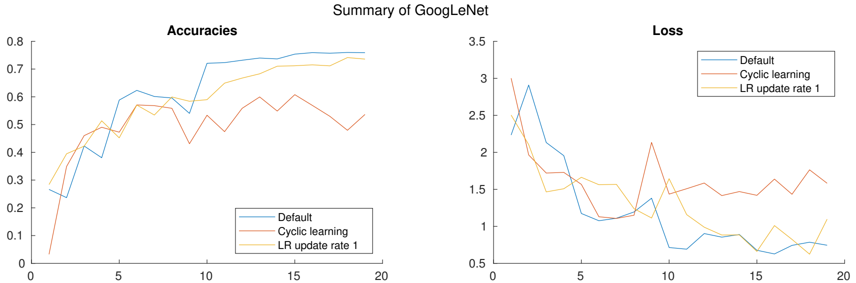

Figure 10: GoogLeNet summary

5.2 During test phase

The experiments during the test phase have been done once the network was already trained. Picking the weights of the best performing network, three different sizes of the anchors for the Kitti image have been tested. The sizes of the anchors used are the ones proposed in the section 4.2. The generated anchors were then filtered based on the ground-truth values of the bounding boxes, and applying non-maximum suppression (NMS) to evade duplicated anchors.

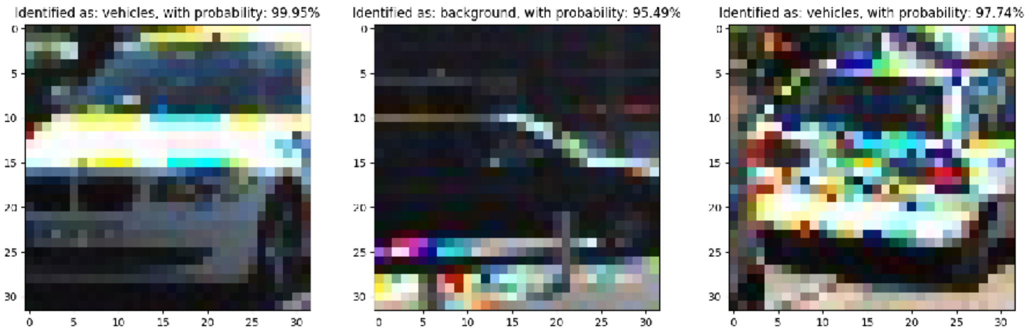
Identified as: vehicles, with probability: 99.95%
Identified as: background, with probability: 95.49%
Identified as: vehicles, with probability: 97.74%
0
5
10
15
20
25
30

Figure 11: GoogLeNet classification output for extracted anchors from KITTI

As it can be appreciated from the output given by the trained network, if the vehicle provided by the RPN layer to the CNN is clear enough, the CNNetwork can recognize with high probability that it is indeed a vehicle (with probabilities of 99,95%, and 97,74%). Other images of general background were provided to the CNN, which were identified as miscellaneous classes for our road-like application.

6 Conclusions

The learning outcome for all of the members in the group has been great during the evolution of this project. Not only we learnt how to implement a complex Deep Learning architecture for object detection, but we all got acquainted with state-of-the-art solutions to the problem. We firstly realized that the structure was much more complex than it actually looked like in the beginning. Faster-RCNN is a very powerful tool for object detection in short time, and the parameters chosen for the training of the networks don't have as much influence as the network itself, but we have been able to implement an academic version of it, and analyze the effect of several editable hyperparameters.

As future improvements, a non maximum suppression should be implemented to reduce the number of anchors generated, as well as a better implementation of the RPN, adding a trainable network that discriminates anchors. The weights should also be adjusted based on the Kitti images/labels.

References

[1] *Object Detection and Classification using R-CNNs*, http://www.telesens.co/2018/03/11/object-detection-and-classification-using-r-cnns/#_Anchor_Target_Layer (2018)

[2] Rohith Gandhi (2018). *R-CNN, Fast R-CNN, Faster R-CNN, YOLO. Object Detection Algorithms*, https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[3] Alex Krizhevsky (2009). *Learning Multiple Layers of Features from Tiny Images*, https://www.cs.toronto.edu/ kriz/cifar.html

[4] Deval Shah (2017). *Activation functions*,https://towardsdatascience.com/activation-functions-in-neural-networks-58115cda9c96

[5] Ravindra Parmar (2018). *Common Loss functions in machine learning*, https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

[6] S.Ren, K.He, R.Girshick, J.Sun (2016). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, https://arxiv.org/abs/1506.01497

[7] http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark

[8] Siddharth Das (2017). *CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more . . .* ,https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

[9] *95.16% on CIFAR10 with PyTorch*, https://github.com/kuangliu/pytorch-cifar

[10] Sik-Ho Tsang (2018). *Review: ZFNet–Winner of ILSVRC 2013 (Image Classification)*, https://medium.com/coinmonks/paper-review-of-zfnet-the-winner-of-ilsvlc-2013-image-classification-d1a5a0c45103

Learning outcomes

For the project, and for each person, the learning outcomes are the following:

Cecilia Martínez Martín

- Get to know the main architecture of the Faster R-CNN network.

  This skill was acquired when searching for documentation on how to build out own network. Wrote the report and some functions for the architecture.
- Understand how each parameter affects to the network

  Analyzing the results obtained and coming up with the conclusions on how the parameters affect.
- Get to know the CIFAR dataset is structured and how to modify it adding new classes or relabeling the existent ones.

Tobias Niewalda

- Not only did I understand the basic structure of Faster RCNN, but I also focused in particular on the Region Proposal Network (PRN). Taking the step backwards in the project and programming this (at least one elementary function) myself made me understand why the previously used repo is so complex.
- The experience with the first used repo (tf-faster-rcnn) also shows that this was not the ideal way to learn Tensorflow. Instead using Pytorch gave me a good bridge between the well known mathematics (like programming in Matlab) and building more complex neural networks with Python.
- Working with IDEs, the cloud platform and Linux is much easier for me after this project. There are many mistakes you can make when using and installing packages and codes. When debugging our algorithm I learned a lot about the correct use of Pytorch. The project also made clear to me how important it is to have a detailed look at the topic before starting it. We underestimated the project and we could have gained a lot of time for exciting investigations.

David Villagrá Guilarte

- I got to learn the underlying concepts behind the currently mostly used DL backends: Tensorflow and Pytorch, and I am now able to implement a Deep Learning project using those backends with confidence.
- I got to know in detail a state-of-the-art Object Detection architecture such as Faster R-CNN, and how it could be implemented to other specific input images, changing and optimizing the parameters.
- I got a great overall understanding of how the Object Detection Deep Learning based architectures work and the differences between the most common ones nowadays.

The project code can be found in:

`https://github.com/david-villagra/plan_b_DL`

The trained models that were used for testing can be as well found in:

`https://github.com/TNeverWood/DL_LargeFiles`