Cityscapes Semantic Segmentation

Noud van de Gevel, (0890583), David Matos, (1352822)

***Abstract*—Autonomous vehicles have the potential to be the future of mobility. Convolutional neural networks can take the images taken on a vehicle and analyze them to perform semantic segmentation on camera images. Utilizing these networks, live predictions are made of objects surrounding the camera, and send signals accordingly to maneuver the vehicle. A great deal of training data is vital for the accuracy of these networks. Images from different cities or countries than the training data can already decrease the performance significantly. The architecture of a network influences its final performance. This research uses a ResNet50 network to perform the semantic segmentation task. The evaluation is improved by tuning hyper-parameters and introducing regularization. The use of full-sized images as inputs to the network, as opposed to down-sized images, along with regularization on the loss function yields a significant improvement in Intersection over Union (IoU) accuracy, having an average of 15% increase over the baseline experiments.**

I. Introduction

SEMANTIC segmentation is the task of predicting a class label for every pixel in an image. Autonomous vehicles can use these predictions to compute control signals that operate the vehicle. These predictions need to be reliable in order to create a safe autonomous vehicle [1]. The finite amount of training data makes it challenging to train networks that meet the desired performance [2].
Convolutional neural networks tackle this challenge by learning the features of the training set, giving it relatively good generalization comparing to, e.g., the k-nearest neighbour technique [3]. The architecture of these network greatly influence the performance. The most popular network backbones used in the Cityscapes challenge are Resnet-101 [4], ResNet-50[4] and DenseNet-161[5] [6]. For instance, the third highest entry on the pixel level semantic labeling task on CityScapes uses a Faster R-CCN [7] network [8]. The Faster R-CNN uses ResNet-101-FPN as backbone. It is extended with a High-Resolution Net that:"maintains high-resolutional representation by connecting high-to-low resolution convolutions in parallel and repeatedly conducting multi-scale fusions across parallel convolutions."

This research investigates the influence of image flipping, image scaling and regularization losses on the ResNet-50 performance in the Cityscapes pixel-level semantic labeling task. The goal is to increase the final score of the network compared to the vanilla ResNet-50. The model, including all relevant explanation of choices, is presented in Section II. Section III describes what experiments were done to evaluate the performance of each model. The results of these experiments are shown and evaluated in Section IV. The conclusion of the research can be found in Section V. Finally, since this research is carried out for the 5LSM0 course of the Eindhoven University of Technology, the research will be discussed in Section VI.

II. Model

The model is divided in five different clear processes in order to achieve a fully working network able to evaluate images. These processes are described in the following subsections.

A. Pre-processing

The cityscapes data-set, gtFine_trainvaltest and leftImg8bit_trainvaltest, are used to train the network. This data-set consist of 2975 training images and labels, 500 validation images and labels and 1525 test images. 200 randomly selected validation images are used to decrease the computational load.

A small amount of the data-set is uploaded to Google Drive so the files can be accessed with Google Colab. This free service is used to create the complete model and do all the debugging. The full data-set is uploaded to a bucket in the Google Cloud storage. The data is downloaded from this bucket to a Virtual Machine (VM) instance that is running on the Google Cloud servers. The VM is accelerated with a NVIDIA Tesla V100 GPU for faster training. This service has a fixed rate per hour, so it is only used to train the network.

B. Input pipeline

The input pipeline is responsible for the image and corresponding label input to the network. The TensorFlow (tf.) dataset function is used to create mini-batches of input data. This function uses a text file containing all file names to decide what images and labels to decode and feed into the network. When training data is gathered, the file names are shuffled to create a diverse mini-batch, since the raw data is achieved in a sequence. After decoding the image, is resized bilinearly to the desired size. The label is always kept at the original size of 1024 by 2048 pixels. After resizing, the images can be augmented.

The complete data-set is configured as an one_shot_iterator. Every time the network calls for new data, the iterator tells the data-set function to get the next batch of images and labels ready for the network.

C. Network

ResNet-50 from the tf.slim library is implemented [9][4].An output-stride of 8 was used in the network. Additionally, avoiding downsizing too much helps the network in retaining more information about the image. The network is finalized by adding a 1x1 convolution with 34 out channels which corresponds to the amount of classes.

D. Training

The training is performed by minimizing the loss between the output of the network and the ground truth labels of images in the mini-batch. Minimizing the loss is a complex task in a high dimensional network like ResNet-50. An optimizer is used to smooth this process. The gradient descent, Adagrad, RMSprop and Adam will be implemented to test for the most suitable optimizer. In addition, one big advantage of Adam over the other optimizers mentioned is the fact that the learning rate is easily tuned because of the first and second moments.

The loss is computed by taking the mean of the sparse softmax cross-entropy. The mean makes sure that the loss is normalized over all the pixels. The sparse softmax cross-entropy is used because it gives an output for every pixel of the image, which is needed for semantic segmentation. Since just classifying an image as 'road' is not the task, every pixel should have 34 probabilities of belonging to a specific class.

The training is run in a tf.MonitoredTrainingSession. This function will create a graph in which all the variables can be reused. In this session the number of epochs and batches are looped to train for the desired time. During training, the training loss and the validation loss is stored for every batch. This is done to analyze if the network is overfitting, e.g. the validation loss converging and training loss going down. If this would happen the training can be stopped since the network is not likely to learn more. Every desired amount of epochs, predictions on the validation set will be stored. The predictions made on the last epoch are kept for evaluation.

E. Post-processing

The saved prediction for the validation set is evaluated using the Cityscapes evaluation method [10]. This method will output the mean Intersection over Union (IoU), for the classes and the categories. The IoU is favoured over the pixel accuracy because it not only computes the correct or incorrect predictions, but also punishes for the incorrect predictions. Pixel Accuracy can be very high in an image with a big portion of sky, but the network has to be able to predict all classes, not only sky.

The test-set predictions are uploaded to Cityscapes to be evaluated on the same criteria as the validation predictions.

III. Method

This section describes the experiments that were carried out and why. The learning rate during training was set to $1e^{-4}$. This value was obtained while testing in Google Colab and proved to result in quick loss convergence. Experiment 1 to 5 all use 1/4 of the original image as input. A summary of all experiment parameters can be seen in Table I.

TABLE I: Experiments

Exp	Lr	Epochs	Batch	Scale	Reg	Flip
1	$1e^{-4}$	5	5	1/4	No	No
2	$1e^{-4}$	8	5	1/4	No	No
3	$1e^{-4}$	8	5	1/4	No	No
4	$1e^{-4}$	8	5	1/4	No	Yes
5	$1e^{-4}$	8	5	1/4	Yes	No
6	$1e^{-4}$	4	1	1	No	No
Final	$1e^{-4}$	7	1	1	Yes	Yes

A. Experiment 1: Optimizer

The four mentioned optimizers in Subsection II-D are used to train for 30 epochs on 12 images of the train data-set. Even though a low loss does not straightforwardly mean a good performance, for this subset of images, quick and low loss convergence is desired.

B. Experiment 2: Vanilla network

The vanilla ResNet-50 is trained for 5 epochs on the full data-set. The loss and mIoU are evaluated to decide on further experiment parameters.

C. Experiment 3: Vanilla network

The vanilla ResNet-50 is trained for 8 epochs with the largest batch size possible for faster training, which is 5. The epochs were chosen For two reasons: from the results of experiments it was concluded that the loss could still decrease if more epochs were run. Yet, only 8 epochs were chosen due to constant disconnects while training, which led to a waste of credits. Therefore, the decision was made to run 8 epochs instead of 10 to make it possible to run at least 4 experiments and a final test. During training, 10 random validation images were used every batch to compute the validation loss.

D. Experiment 4: Data augmentation

In this experiment the input images and labels are flipped horizontally with a probability of 0.5. Essentially doubling the size of the data-set. Having more data means—for a large validation set—a better general understanding of the features. The network is less prone to memorizing the training data when it less likely to have the same input all the time.

E. Experiment 5: Regularization

Tf.get_collection gathers the regularization losses for the network. These losses are added to the previous implemented losses. The training loss will therefore increase, but the validation loss is expected to decrease more than in the other experiments.

F. Experiment 6: Full scale input

The full scale (1024x2048) is used in this experiment. The maximum batch size that fits on the GPU is 1, making it more time consuming. For that reason the epoch range is set to 4. The hypothesis for this experiment is that the network can predict the small object, like poles, better than the other networks. These object can be only a few pixels wide, downsizing the images before inputting in the network might already decrease the chance of detecting them.

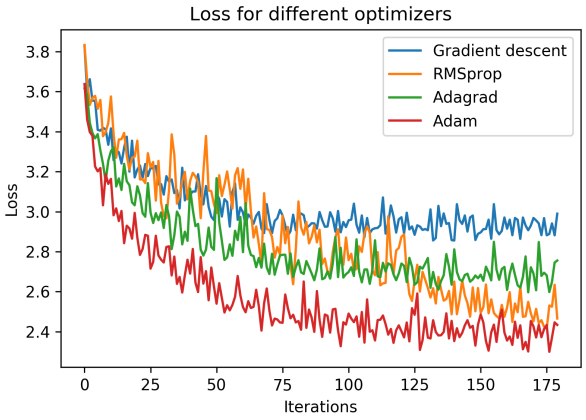

Fig. 1: Different optimizers tested on 12 images from the train data-set.

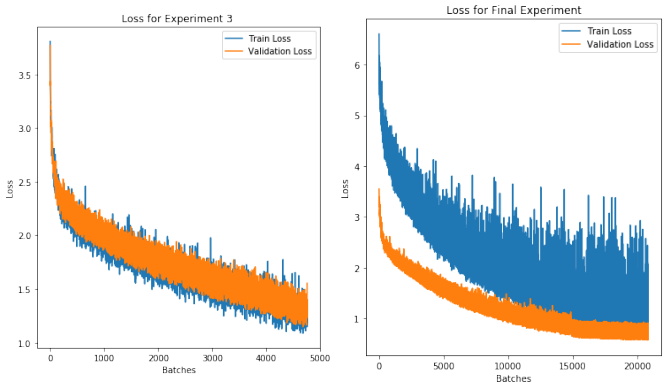

Fig. 2: Loss convergence of experiments 3 and the final test.

G. Final test

Using the results of previous experiments, the final testing setup is created. Flipping, regularization and full-scale images are used to train the network for 7 epochs.
The test prediction will be made using this network to get the final results by submitting it to the Cityscapes website with the name: *TUE-5LSM0-g12-N00bN3t*.

IV. Results

Figure 1 shows the loss convergence for the four tested optimizers. Adam clearly shows better results, converging quicker and to a lower value than the others.

From the loss of the experiment 2 is could be seen that the loss has potential to converge to a lower value. Therefore, the next experiments were executed with more epochs. Fig. 2 shows the loss for experiment 3 and the final test. It can be seen that the validation loss is tracking the training loss, and, that both losses are not converging. This means the network is not overfitting and it can decrease the loss for a longer training time. During the final test, the initial loss value is higher, this is a results of adding the regularization loss. The training loss is fluctuating very much, which is caused by the batch size. Every image is different, resulting in a different loss. When using a bigger batch, the images and the loss are more generalized and shows less fluctuation. It can be seen that the loss of the final test is converging to its minimum value.

TABLE II: IoU class scores of experiments [%]

Class	Exp2	Exp3	Exp4	Exp5	Exp6	Final
Road	83	90	85	93	90	94
Sidewalk	51	53	50	61	53	65
Building	73	71	71	78	72	79
Wall	18	**8**	14	13	17	16
Fence	4	7	7	9	7	12
Pole	20	24	24	33	25	38
Traffic light	**0**	4	5	11	5	27
Traffic sign	**5**	15	15	23	19	39
Vegetation	77	75	77	84	77	85
Terrain	29	24	28	38	26	39
Sky	81	75	77	87	77	86
Person	32	28	30	38	30	42
Rider	23	**10**	7	2	7	9
Car	65	62	65	73	65	76
Truck	5	5	4	0	6	1
Bus	0	4	**11**	1	3	5
Motorcycle	0	2	5	5	3	5
Bicycle	27	21	26	31	29	40
Average	30	31	32	**36**	32	40

The loss of experiment 2 and 4 were comparable to experiment 3. Experiment displayed the same shape, but with higher initial train loss due to the addition of regularization loss. Experiment 6 showed the same shape and values as experiment 3 but with higher fluctuation.

The quantitative class scores of each experiment on the validation set can be found in Table II. Notable values are given in bold. Adding regularization losses (Exp5) increased the scores significantly for almost every class, even more than training on full scale images. Training the vanilla network for 8 epochs instead of 5 decreased the score for ”Wall” and ”Rider” drastically. The other way around, training for 8 epochs resulted in increase for traffic signs and lights. Training for longer results in mor detailed segmentation, shown by seeing classes traffic signs and traffic lights.

Flipping the images increased the score for ”Bus”. There are not many pixels of class ”buss” in the validation set. Flipping the train images can result in a network that not only learns the shape of a bus, but also two different orientations and scales, of each train image, of the bus. Giving it more chance to recognize a bus in the validation set. This theory does apply for the class ”truck”.

The score of the final test is higher for all classes, except ”sky”. The hypothesis that full-scale images increase on the small objects like poles and signs is correct, but regularization causes an even greater increase. The average scores in the table do not increase as much as some particular classes. This is caused by some (not shown, e.g. train) classes are close to 0 and drastically limit the average score.

Fig. 3 gives the qualitative results of experiment 3 and the final test next to the input image. The vanilla network seems to be guessing, because it learned that a ”car” is mostly found on the road. The final test clearly sees more detail in the image, not predicting ”car” but ”person” walking on class ”sidewalk”. It even predicts the bicycles carried by the people.

Table III shows a reduced confusion matrix of the final test containing 8 classes. Notably, is that every class, although

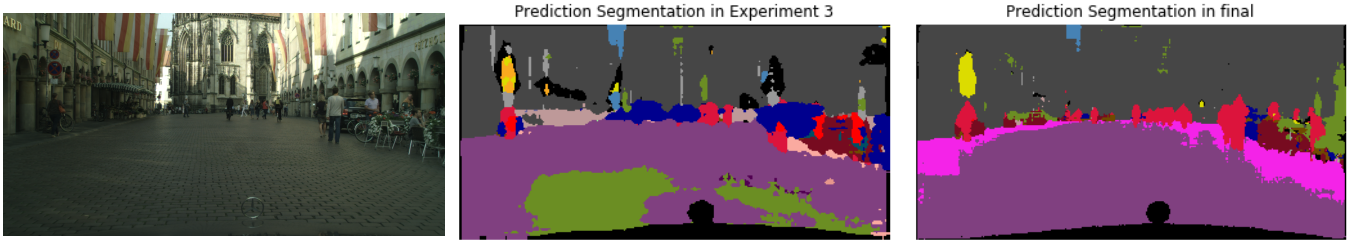

Fig. 3: Munster_000145_00019 image with the predictions from experiment 3 and the final test.

TABLE III: Reduced confusion matrix of the final test

	building	pole	traffic light	person	car	truck	bus	motorcycle
building	**91**	1	0	0	1	0	0	0
pole	30	**48**	0	2	1	0	0	0
traffic light	41	4	**33**	1	0	0	0	0
person	15	2	0	**60**	6	0	0	0
car	4	0	0	2	**89**	0	0	0
truck	37	0	0	3	45	**1**	2	0
bus	28	1	0	1	39	1	**6**	0
motorcycle	5	0	0	27	26	0	0	**7**

some more than others, are predicted as a building. Vehicles like: ”truck”, ”bus” and ”motorcycle” are commonly predicted as a car, all scoring very low own their own class. This can be explained by the composition of the data-set. The class ”car” is 15 times more represented in the images as the other mentioned vehicle classes [11]. This makes the data-set unbalanced, giving the highest chance of being right when prediction for a car if the correct prediction is not known. The ”motorcycle” is predicted as person for 27%. Which makes sense since in most cases, the person is sitting on the motorcycle.

The results of the final test on the complete test set is submitted to Cityscapes, resulting in position 163 of 166 entries for the Pixel-Level Semantic Labeling Task [12]. The IoU on the classes is 41.67 % and 74.37% on the categories.

V. Conclusion

The Adam optimizer shows the quickest loss convergence to the lowest of the four tested optimizers. Training on full-scale images, adding regularization and flipping images is beneficial for the performance of the network. ResNet-50, is not able to predict all vehicles in the correct class and many classes are predicted as a building. Even though only minor additions were added to the baseline network, an increase of 29% is achieved. Comparing the best experiment to the final test, the increase is 11%.

VI. Discussion

In future work it is advised to train the networks for more epochs. In this case, the results would make more sense, because the experiments might converge to different values. However, the results in this research were limted by the hardware. Results can be improved by further implementation of the experiments. Data augmentation techniques like skew or color correction can improve the average scores. Regularization techniques like dropout can be added to make the network rely less on certain features and train on less used ones.

Different backbone networks could be tested to investigate the performance difference. Network with more parameters might be able to store more features and predict correctly on, for instance, the vehicle category.

References

[1] H. Lamba, “Understanding Semantic Segmentation with UNET Towards Data Science,” 2017. [Online]. Available: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

[2] M. Larsson, E. Stenborg, L. Hammarstrand, T. Sattler, M. Pollefeys, and F. Kahl, “A Cross-Season Correspondence Dataset for Robust Semantic Segmentation,” 3 2019. [Online]. Available: http://arxiv.org/abs/1903.06916

[3] E. Fix and J. Hodges, “Discriminatory Analysis, Nonparametric Discrimination: Consistency Properties,” *Technical Report, USAF School of Aviation Medicine, Randolph Field*, vol. 4, 1951.

[4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” 12 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[5] “Dataset Overview Cityscapes Dataset.” [Online]. Available: https://www.cityscapes-dataset.com/dataset-overview/

[6] “GitHub - lxtGH/GALD-Net: Source code and models of GALD net.” [Online]. Available: https://github.com/lxtGH/GALD-Net

[7] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature Pyramid Networks for Object Detection,” Tech. Rep.

[8] K. Sun, Y. Zhao, B. Jiang, T. Cheng, B. Xiao, D. Liu, Y. Mu, X. Wang, W. Liu, and J. Wang, “High-Resolution Representations for Labeling Pixels and Regions,” 4 2019. [Online]. Available: http://arxiv.org/abs/1904.04514

[9] “GitHub - resnet_v1.py.” [Online]. Available: https://github.com/tensorflow/models/blob/master/research/slim/nets/resnet_v1.py

[10] “cityscapesScripts/evalPixelLevelSemanticLabeling.py at master · mcordts/cityscapesScripts · GitHub.” [Online]. Available: https://github.com/mcordts/cityscapesScripts/blob/master/cityscapesscripts/evaluation/evalPixelLevelSemanticLabeling.py

[11] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, D. A. R&d, and T. U. Darmstadt, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” Tech. Rep. [Online]. Available: www.cityscapes-dataset.net

[12] “My Submissions Cityscapes Dataset.” [Online]. Available: https://www.cityscapes-dataset.com/submit/?action=anonymousLink&submissionID=4594