Pixel Level Image Segmentation of Cityscapes

Csenge Kilián, Beáta Csilla Kovács

***Abstract* — Image understanding is highly relevant in the context of many of today's most exciting problems in the field of information technology, such as mobile robotics and autonomous vehicles. To solve these problems, accurate information of the objects in the scene may be applied for decision making or safe navigation, among others. Semantic segmentation is probably one of the best approaches towards complete scene understanding. This project's aim is to create a deep neural network trained with the UC Berkeley's dataset, which is capable of semantic segmentation of cityscapes. This project is experimental, our aim was to explore the most relevant deep learning solutions.**

***Index Terms* — deep learning, image segmentation, ResNet, U-Net**

I. Introduction

Inferring knowledge from images plays an important role in various fields of today's modern technological landscape, such as medical analysis, object detection in satellite images, iris recognition, or autonomous driving systems. Though self-driving vehicles got a bad reputation recently because of some accidents, they will certainly become an evidential part of our lives in the future, like many other autonomous systems that are able to assist humans in everyday tasks. In our work we investigate the problems of applying deep learning for autonomous driving related tasks: the training data problem is investigated and a possible deep neural network architecture is evaluated for the given task.

II. Related Work

A. Semantic Segmentation

Image segmentation means assigning a semantic annotation label to each pixel in the image, so that each pixel is labeled with the class of the pixel's enclosing object. This is a supervised learning problem which requires training a set of classifiers from data labelled at the pixel level.
Unlike classification, where the end result of the network is a summary for the whole image, semantic segmentation attempts to partition the image into semantically meaningful parts, to classify each part into one of the pre-determined classes, and it also requires the ability to project the discriminative features learnt at different stages of the network to the pixel space. Therefore, image segmentation is also categorized as a dense prediction task. In itself it does not distinguish between object instances, only object categories.

Semantic segmentation related research [7, 8, 10, 11] has grown significantly over the last decade. The accuracy of these state-of-the-art techniques advanced the most by training different types of Convolutional Neural Networks (CNNs). From the countless CNN architectures available, the convolutional encoder-decoder network pattern is particularly well adapted for the problem of pixel labeling. The encoder part is usually responsible for partitioning the image into semantically meaningful parts with the help of a pre-trained classification network, while the decoder's responsibility is to project the lower resolution features onto the higher resolution pixel space.

After some research in the field and considering other architectures as well, we first chose SegNet [2] to experiment with, but later we moved on to a simple U-net [1], because we run into memory limitations.

B. SegNet

In this study, first we choose the SegNet architecture, because it was designed exactly for pixel-level semantic segmentation motivated by the need for understanding cityscapes. Also the reduced model, often referred to as SegNet Basic seemed to be providing a great balance between accuracy and computational cost, just perfect for the equipment currently available for us.
The SegNet architecture also follows the encoder-decoder pattern, as most semantic segmentation architectures. Direct adoption of classification networks for pixel wise segmentation yields poor results mainly because max-pooling and subsampling reduce feature map resolution and hence output resolution is reduced. In contrast with this, the encoder part of the network creates a rich feature map representing the image content and the decoder transforms the feature map into a map of class probabilities for every pixel of the input image.

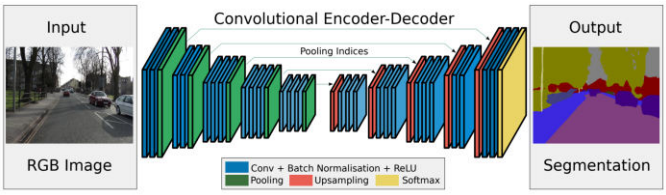


Figure 1 SegNet's architecture

The encoder:
The encoder of the network is topologically identical to the first 13 layers of the fully convolutional VGG-16. The last three fully connected layers are removed. VGG-16 uses a stack of convolutional layers with small receptive fields in the first layers instead of fewer layers with bigger receptive fields. By reducing its layers down to only 13 layers, the number of parameters is minimized from 134 million features to only 14.7
million parameters.
Each encoder applies convolution, batch-normalization, a non-linearity, and then max-pooling on the result, while storing the index of the value extracted from each window.

The decoder:
The decoder has repeated upsampling layers followed by convolutional layers. The decoder network is almost like the inverse of the encoder: it has the same type of convolutions as the encoder network, regarding filter sizes, and the channels of corresponding layers, except they don't have a non-linearity. That is how after the encoder's downsampling process, the image is upsampled again to the original image size.
In SegNet, upsampling does not take part in the learning process. It is a sort of backward max-pooling operation.

Residual connections:
During the forward pass in the encoder, while downsampling, the max-pooling indices are stored – meaning the location of the highest value pixel in the max-pooling window at each sliding position of the layer. The residual connection in principle is a kind of skip connection that bypasses the non-linear transformations. They are responsible for transmitting the max-pooling indices from the encoder to the decoder, so they can be then used when the corresponding upsampling layer is working in the decoder. The remaining pixels in the upsampled output are set to zero.

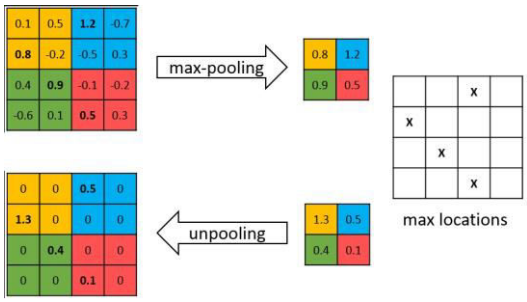


Figure 2 Visualizing downsampling and upsampling

Outer Layers:
The encoder and decoder are followed by two outer layers. The first outer layer is responsible for feature extraction. The output is then transmitted to the next layer adjacent to the very last layer of the network. This layer is responsible for pixel-wise classification, in other words determining which pixel belongs to which class. In the last layer, softmax is used for the pixel-wise classification and giving the final prediction. The prediction is going to be an n-channel image, where each channel corresponds to one of the semantic categories.

SegNet Basic

SegNet Basic is a smaller, reduced variation of SegNet. It has 4 encoders and 4 decoders. All encoders perform max-pooling, and the corresponding decoders upsample using the max-pooling indices received through the residual connections.
We found it a great tool to experiment with, but because of memory limitations, we had to work with very small batch-sizes (2 or 4), and training the network was way too time consuming (training with only 70 training samples took us around 40 minutes per epoch). Another reason for switching to U-Net was the poor results.

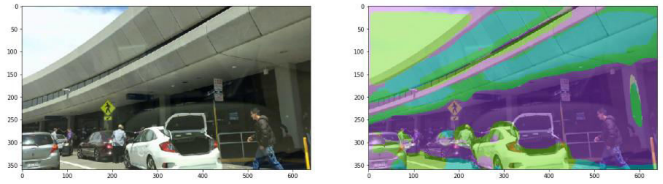


Figure 3 An example of prediction with SegNet Basic

C. U-Net

U-Net is a very popular encoder-decoder network originally invented for biomedical image segmentation trained on a small dataset. The whole architecture consists of 23 layers.

The encoder
The network's encoder, the contracting path, follows the typical architecture of a convolutional network. At each step, 3x3 unpadded convolutions are followed by a ReLU layer and a 2x2 max pooling layer, so that the number of feature channels are doubled.

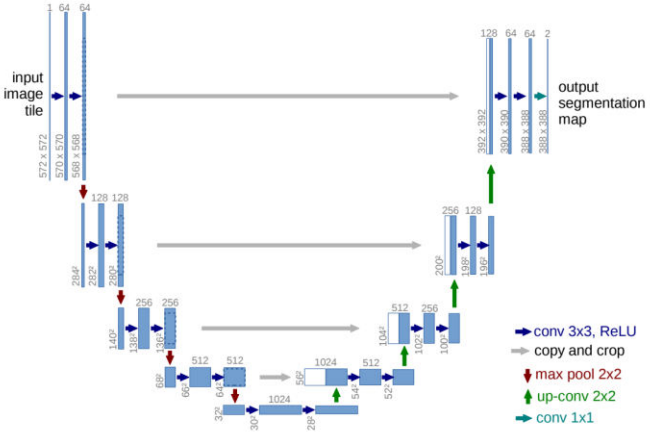


Figure 4 U-Net's architecture

The decoder
The decoder, the expansive path, consists of upsampling layers with 2x2 backward convolutions that halves the number of feature channels. There is an additional operation in every step of the decoder: the corresponding encoder feature maps are concatenated with the upsampled output in the given decoder. This is done in order to counterweight the effect of lost information during the pooling operation. This way of better localization and learning of the representation is what makes U-Net special among encoder-decoder patterned networks, the way residual connections make SegNet special.

Outer Layer:
The final convolutional layer is a 1x1 convolutional layer. It is used to map the features vectors to the desired number of classes.

III. Dataset

We have overviewed many datasets (like CamVid, CityScapes [9], KITTI, and ApolloScape) and we finally opted for the dataset collected and annotated by Berkeley university.
UC Berkeley has open sourced this dataset, and they claim it to be the largest and most diverse self-driving dataset currently available for the general public. Two of the main advantages of the dataset are the rich environmental and weather conditions, and the balance between the day-time and night-time scenarios. The demand for such dataset has been consistently high for a long time, so the generous release of this huge dataset means that there is more diversity of data available for researchers and scientists to use.
The researchers behind collecting and annotating this dataset have also suggested that they will further advance the dataset by expanding it from only having monocular videos to including panorama and stereo videos as well as other types of sensors like LiDAR and radar.

For our project, there are over 10,000 samples and corresponding pixel-level annotations for both class-level and instance-level segmentation. After a quick registration, they provide an easy way for downloading the data through Google Drive.

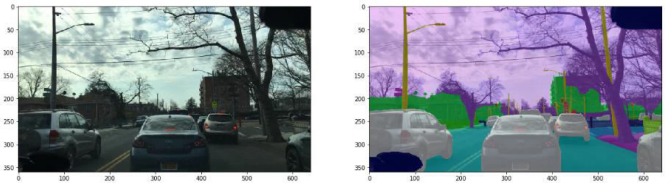


Figure 5 An example of raw data and its annotation

IV. Methods

A. Data Preprocessing

Building an effective network not only requires careful consideration of the network architecture, but also of the input data and its format.

Image size

Since the images of our dataset all have the same dimensions of 1280x780 pixels, there was no need for us to crop or manipulate them to have uniform size.

Standardization

As the general requirement of most machine learning methods, we also standardized our data with the help of Keras's function, originally implemented for ImageNet. It scales pixels between -1 and 1, sample-wise.

Reducing dimensionality

We reduced the dimensionality of our annotated images. Our dataset originally contained pixel-level annotations in the form of .png files. We trained our model with the corresponding annotation-matrices, built so that each pixel has the id of the class annotated to them.
The annotations are serialized on the same filename, as the corresponding raw data images for easier use later.

Shuffling the data

Shuffling the data was definitely necessary, because some of the samples that follow each other were taken from the same video, and therefore look quite alike, both in terms of the environment and the position of the vehicles. This is exactly the opposite of what would be ideal: samples following each other being as different from each other as possible, because that is when the error will be the highest, and therefore that is when the model will learn the most.
On a side note: Initially, for experimentation, we only used the first 100 samples of the dataset to train our network. The samples in the training set had little to no variance in terms of the environment, and as a consequence our model overfitted on their features, resulting in very poor predictions on the test set. Later we paid more attention to diversity in the dataset even when having a "quick train".

Splitting the data

We chose to split our data into training, validation and test sets with the ratios being 70:15:15.
Training data includes the samples used to fit the model. This is the part of the dataset that the model actually sees and learns from. Validation dataset is used to provide unbiased evaluation of the model after each epoch. Test dataset is for evaluating the final model, so it is only used after the model is completely trained.

Augmentation

Data augmentation is taking an image and generating new ones through manipulation of the original. It is extremely useful when working with smaller datasets or datasets that are not diverse enough. Although UC Berkeley's dataset is excellent at both of these points, we still wanted to explore some possible augmentation techniques. Turns out, through some experimentation, one can even create snowy roads on a sunny day.

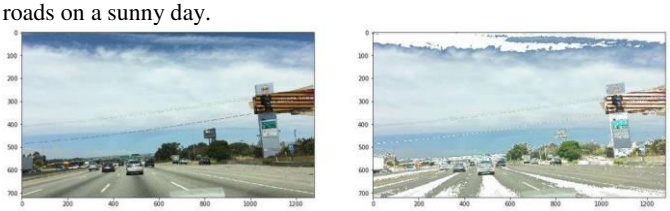


Figure 6 An example of augmentation

B. Data Generator

Using a Data Generator is great when working with huge datasets, because loading your entire training dataset at once may cause you some headache, as it may not be able to fit in the available memory at the same time. The data generator, on the other hand, generates your data on the fly. The training process becomes less memory intensive this way.
Besides, the data generator is also responsible for shuffling the data.

C. Network Characteristics

Mini Batch

Mini-batch is a combination of batch and stochastic training. Instead of using all training data items to compute gradients (as in batch training) or using a single training item to compute gradients (as in stochastic training), mini-batch training uses a user-specified number of training items. The size of each mini-batch is very important; if it is very large, then it will behave as batch gradient descent, which is very slow, and if it is very small it becomes stochastic gradient descent with very large noise. One of the biggest advantages of using mini-batch training is not having all training data in memory and the batched updates provide a computationally more efficient process than stochastic gradient descent.

Early Stopping

When training a neural network, it is important to wisely decide the number of epochs we want to have. If we use too many, the model might overfit the problem, fit the noise in the training data. On the other hand, if we use too few, we might underfit the problem, meaning that the model did not exploit all the learning possibilities from the training data.
The idea behind early stopping is to remove the need to manually set this value. At the end of each epoch, if the network outperforms the previous best model, we save its weights. If the validation accuracy is unable to improve in a given time, the training stops, and the weights of the best model are loaded back.

Batch Normalization

The original paper of batch normalization [4] was created to tackle the fact, that the distribution of each layer's inputs changes during training, thus slowing down the training process because of the phenomenon called internal covariate shift. Batch normalization means the normalization of the data's internal representation as part of the model architecture. The input feature and the layer's output before activation is both normalized in each hidden layer. It reduces the impact of earlier layers on later layers and speeds up training. Batch normalization also acts as a regularizer.

Optimizer: Adam

Adam is an adaptive learning rate method, meaning that it computes individual learning rates for different parameters. It is straightforward, computationally efficient, and has little memory requirements at the same time. Therefore, this optimizer is great for problems that deal with a large dataset or huge parameter space.
Adam is a combination of RMSProp [6] and momentum. It uses the squared gradients to scale the learning rate like RMSProp and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

Loss function: categorical cross-entropy

Categorical cross-entropy is a loss function often used in multiclass classification tasks.

D. *Our U-Net architecture: U- Lite*

The model's input is a 3-channel image with the default size being 360x640.
This architecture has 3 encoder and 3 decoder steps. The decoders have max-pooling subsampling layers, and the encoders have their corresponding upsampling layers. Each step has batch normalization layers and ReLU as their activation function.
The final layer is responsible for classification, with softmax as its activation function.

V. Evaluation

Evaluating the model is an essential part of any project. But choosing the right evaluation technique is not as trivial as one might think. Your model may give you satisfying results when evaluated using one metric, while it may give poor results with an other.

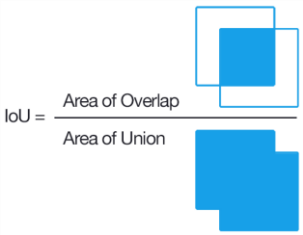
A. *Techniques*

Pixel Accuracy

When evaluating a binary classifier, we usually classify our predictions into the four categories of true positives, false positives, true negatives, and false negatives. For pixel-level semantic segmentation and other dense prediction tasks, it is not so trivial to tell what counts as true positive. Because the prediction is pixel level, we have to interpret these concepts at pixel level, too.
Pixel accuracy means dividing the number of correctly classified pixels (the sum of true positive and true negative pixels) with the total number of pixels. This is usually calculated per class.
Our first approach was to use pixel accuracy as our performance metric, but soon we realized, that it is far from ideal. It only works well if the classes are equally represented in the database, and the object to image size ratio is around 50%. Otherwise it provides misleading results, as the measure will be biased in mainly reporting how well the model identifies negative case.

Intersection over Union

Intersection over Union [5, 7] takes the number of true positive pixels and divides them by sum of the number of true positive, false positive and false negative. If we think of the pixels predicted to be of given class as one set and the pixels annotated to be of this class, then this metric gives the fraction of these two sets' intersection and their union. It essentially gives you the percent of overlap between the ground truth mask and your prediction.
Opposed to pixel accuracy, the IoU, also known as the Jaccard index, is scale invariant, meaning that it gives appropriate relevance to small objects, and it is unbiased against negative cases.



B. *Results*

When using SegNet Basic to train our network, the model produced really poor predictions, and its per class IoU evaluation also turned out to be disappointing. It is important to say, that the model's low results are not due to the architecture, but the lack of proper resources on our part. We didn't have the equipmesnt to train it properly.

After switching to a simpler U-Net architecture, and training it with 700 training samples and over 38 epochs, our results improved significantly.

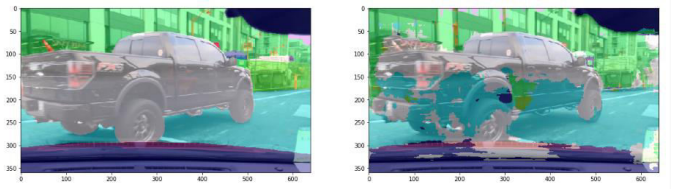


Figure 7 An example of prediction with U-Lite

Category	Segnet Basic	U Lite
Unlabeled	0.000	0.478
Dynamic	0.000	0.000
Ego vehicle	Nan	Nan
Ground	0.000	0.000
Static	Nan	Nan
Parking	0.000	0.000
Rail track	Nan	Nan
Road	0.111	0.697
Sidewalk	0.000	0.095
Bridge	0.000	0.037
Building	0.124	0.524
Fence	0.000	0.020
Garage	0.000	Nan
Guard Rail	0.030	0.165
Tunnel	0.000	0.000
Wall	0.000	0.000
Banner	0.000	0.000
Billboard	Nan	0.000
Lane divider	0.000	Nan
Parking Sign	0.000	0.021
Pole	0.000	0.153
Polegroup	Nan	Nan
Street light	0.000	0.000
Traffic cone	0.000	0.000
Traffic device	Nan	0.000
Traffic light	0.000	0.053
Traffic Sign	0.000	0.093
Traffic sign frame	0.000	0.078
Terrain	0.000	0.171
Vegetation	0.410	0.677
Sky	0.440	0.912
Person	Nan	Nan
Rider	0.000	0.000
Bicycle	Nan	0.000
Bus	0.000	0.000
Car	0.007	0.472
Caravan	Nan	0.000
Motorcycle	0.000	0.000
Trailer	0.000	0.000
Train	0.000	Nan
Truck	0.000	0.000

Table 1 IoU results per class

VI. Conclusion

After reviewing already existing solutions to semantic segmentation using deep learning, SegNet seemed to be a great model for our project. However, we were not prepared for how demanding it is both regarding computation, memory and time. For this reason we chose to change our model to a simplified U-Net, which produced relatively good results.

We are perfectly aware of the endless options to improve our solution to semantic segmentation. Still, we feel like we fulfilled our initial goal of deepening our knowledge in the field [12], and gaining some technical skills to support what we have learned about the theory of deep learning.

REFERENCES

[1] Olaf Ronneberger, Philipp Fischer, Thomas Brox "U-Net: Convolutional Networks for Biomedical Image Segmentation" 2015

[2] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation" 2016

[3] Diederik P. Kingma, Jimmy Lei Ba "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION" 2015

[4] Sergey Ioffe, Christian Szegedy "Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shif" 2015

[5] Gabriela Csurka, Diane Larlus, Florent Perronnin "What is a good evaluation measure for semantic segmentation?" 2013

[6] Tijmen Tieleman, Geoffrey Hinton. Lecture 6.5-rmsprop: "Divide the gradient by a running average of its recent magnitude" COURSERA: neural networks for machine learning, 4(2):26–31, 2012.

[7] Liang-Chieh Chen , George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs" 2018

[8] A. Garcia-Garcia, S. Orts-Escolano, S.O. Oprea, V. Villena-Martinez, J. Garcia-Rodriguez "A Review on Deep Learning Techniques Applied to Semantic Segmentation" 2017

[9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding" 2016

[10] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, Alan L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs" 2016

[11] Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation", 2017

[12] Dalia Bedewy, Adel Gabriel, "Examining perceptions of academic stress and its sources among university students: The Perception of Academic Stress Scale", 2015