In The Name of God

University Tehran

Engineering Facility

Electrical and Computer Engineering

Deep Learning with Applications

Final Project

Arash Hatefi 810695327

Alireza Forouzandeh Nezhad 810198221

Amin Fadaeinedjad 810195442

Spring 99

Abstract

In this project, we are going to implement a Hierarchical multi-scale attention network model for semantic segmentation. The reason that we are using this structure for segmentation is that the result of this technic shows that predicting the label of a specific pixel of an image can be more accurate if it's done in different scales, so in that case, the paper suggested trying multiple scales to have a better performance in the segmentation process, other than its performance the results show that out attention mechanism is about four times more efficient in memory usage compared to the previous methods in segmentation, also, it enables faster training result, this will let us train with larger images with equal training time which will lend us to better accuracy.

In this paper, they used Cityscapes and Mapillary Vistas datasets, which have a large number of weakly labeled images, and they also leverage auto labeling to improve generalization.

In the paper their model was trained using Pytorch on Nvidia DGX servers containing 8 GPUs per node with mixed precision, distributed data parallel training and synchronous batch normalization, and they used Stochastic Gradient Descent (SGD) for their optimizer, with batch size 1 per GPU, momentum 0.9 and weight decay $5e^{-4}$ in training.

Dataset Overview

The dataset that we were using in the following project was the Cityscapes dataset, which was labeled for semantic understanding.

Datasets Features:

1) Polygonal annotation
    a. Dense semantic
    b. Instance segmentation for vehicle and people
2) Complexity
    a. 30 classes (but we used 20 of the most used classes in our project)
    b. Class Definitions
3) Diversity
    a. 50 cities
    b. Several months (spring, summer, fall)
    c. Daytime
    d. Good/medium weather conditions
    e. Manually selected frames
        i. A large number of dynamic objects
        ii. Varying scene layout
        iii. Varying background
4) Volume
    a. 5000 annotated images with fine annotations
    b. 20000 annotated images with coarse annotations
5) Metadata
    a. Preceding and trailing video frames. Each annotated images is the 20th image from 30 video snippets
    b. Corresponding right stereo views
    c. GPS coordinates
    d. Ego-motion data from vehicle odometer
    e. Outside temperature from the vehicle sensor
6) Extensions by other researchers
    a. Bounding box annotations of people
    b. Images augmented with fog and rain
7) Benchmark suite and evaluation server
    a. Pixel-level semantic labeling
    b. Instance-level semantic labeling
    c. Panoptic semantic labeling

Labeling Policy

Labeled foreground objects must never have holes, i.e., if there is some background visible ‘through’ some foreground object, it is considered part of the foreground. This also applies to regions that are highly mixed with two or more classes: they are labeled with the foreground class. Examples: tree leaves in front of the house or sky (everything tree), transparent car windows (everything car).

Class Definitions

Group	Classes
Flat	$road - sidewalk - parking^{+} - rail\ track^{+}$
Human	$person^{*} - rider^{*}$
Vehicle	$car^{*} - truck^{*} - bus^{*} - on\ rails^{*} - motorcycle^{*} - bicyle^{*} - caravan^{*+} - trailer^{*+}$
Construction	$building - wall - fence - guard\ rail^{+} - bridge^{+} - caravan^{*+} - trailer^{*+}$
Object	$pole - pole\ group^{+} - traffic\ sign - traffic\ light$
Nature	Vegetation – terrain
Sky	Sky
Void	$ground^{+} - dynamic^{+} - static^{+}$

- * Single instance annotations are available. However, if the boundary between such instances cannot be clearly seen, the whole crowd/group is labeled together and annotated as a group, e.g., car group.

- + This label is not included in any evaluation and treated as void (or in the case of the license plate as the vehicle mounted on).

Fine annotations

Below are examples of our high-quality, dense pixel annotations for a volume of 5000 images. Overlayed colors encode semantic classes. Note that single instances of traffic participants are annotated individually.

Example:

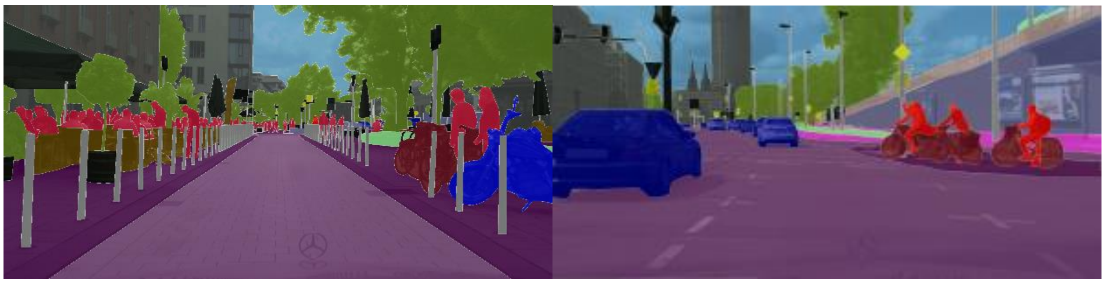


Figure 1 Fine annotations

Coarse annotations

In addition to the fine annotations, we provide coarser polygonal annotations for a set of 20 000 images in collaboration with Pallas Ludens. Again, Overlayed colors encode the semantic classes (see class definitions). Note that we do not aim to annotated single instances. However, we marked polygons covering individual objects as such.

Example:

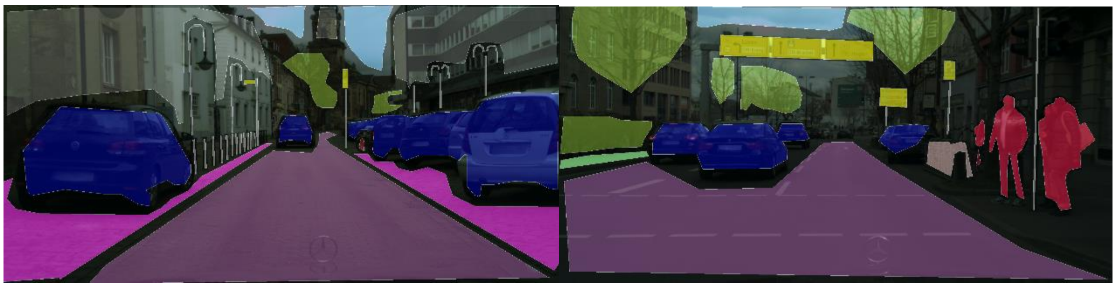


Figure 2 coarse annotations

Figure 3 shows the number of pixels which belong to every class in the mentioned dataset.

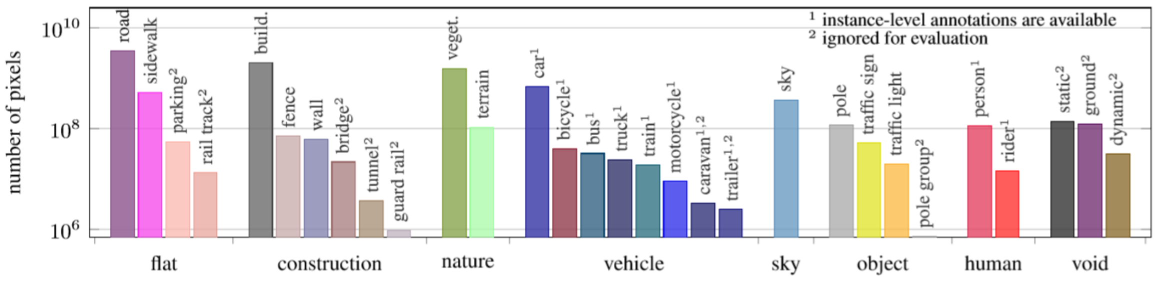


Figure 3 Number of finely annotated pixels (y-axis) per class and their associated categories (x-axis).

Because of the short memory that we have in google Colab server, we can’t use all the labels for the segmentation classification, so we used 19 of the most common labels in the current dataset, and label $unknown$ (20) for the unknown

The Labels that we are using:

desired labels = [road, sidewalk, parking, building, fence, wall, vegetation, terrain, car, bicycle, bus, truck, sky, person, pole, traffic sign, static, ground, dynamic]

Computational limitation

As it was mentioned the network was implemented by Nvidia had eight high-tech GPUs for every single node in this network, but all we have has a number of free accounts of google Colab and that was impossible for us to implement all the network together, we used several methods to make it happen because Colab only gives us 25GB Ram for the cloud that we are using and further more we can’t use the GPU for long because we have a month limit usage so that made us in hard situation, so we needed to come up with some ideas for how to implement this network without receiving errors for out using the Ram and The GPU.

We used different ways for this part, such as that we used smaller scales of the image. For example, the images that were in the dataset were quite large, and we use them with that size would have taken a large amount of memory.

The other thing that we have done and it’s probably the most important thing that we have done is that we could train all the part of the network together because only by training a particular part of the network the usage of the GPU will increase to above 90% of the maximum amount of GPU we could have used in google Colab, the way we managed this part that we started to train every part of the network individually for example we trained the segmentation with different hyper parameters and also we have done this same with the trunk as well and at final we considered joining the network parts together in order to have the best result, this job is like what some researchers when they want to fix a particular part of the network and try to train other parts of it they consider on freezing the part they want it to stay unchanged, we done a similar job as well.

As we all know that in the paper they were using a very large network for the trunk such as $ResNet$ which we know is a pretty big network with millions of parameters, and it’s clear that trying to train this network this deep is a thing beyond the instruments that we have so we tried on using smaller networks such as $MobileNet$ which we are going to discuss it in the following report.

It is needed to be mentioned that we weren’t able to train end-to-end any of the following networks that were mentioned.

Architect

In the paper, it wasn't mentioned clearly the structure of the model, and it usually mentioned structures from other papers, so there wasn't a clear structure for us to implement.

Backbone: For the ablation studies in this section, we used ResNet-50 as the trunk for our network, for the state of the art results, we use a larger, more powerful trunk, HRNet-OCR.

Semantic Head: Semantic predictions are performed by dedicated fully convolution head consisting of $(3x3\ conv) \rightarrow (BN) \rightarrow (ReLU) \rightarrow (3x3\ conv) \rightarrow (BN) \rightarrow (ReLU) \rightarrow (1x1conv)$. The final convolution output $num\_classes$ channels.

Attention Head: Attention predictions are made using a separate head that is structurally identical to the semantic head, except for the final convolutional output, which outputs a single channel. When using ResNet-50 as the trunk, the semantic and attention heads are fed with features from the final stage of ResNet-50. When using HRNet-OCR, the semantic and attention heads are fed with features out of the OCR block. With HRNet-OCR.

Auxiliary semantic head: With HRNet-OCR, there also exists an auxiliary semantic head, which takes its features directly from the HRNet trunk, before OCR. This head consists of $(1x1\ conv) \rightarrow (BN) \rightarrow (ReLU) \rightarrow (1x1\ conv)$. After attention is applied to the semantic logits, the predictions are upsampled to the target image size with bilinear upsampling.

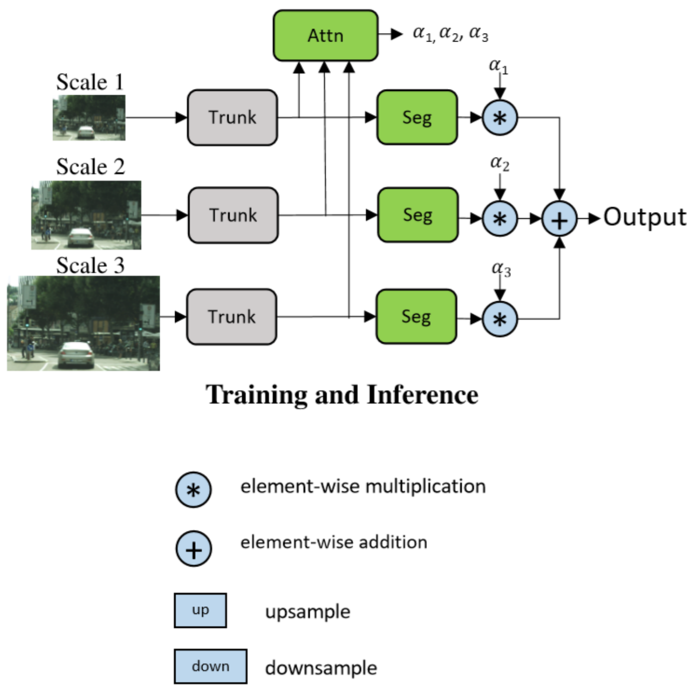


Figure 4 Explicit

Figure 4 shows the structure and architect of the explicit model which in the paper the new hierarchical method was mainly compared with this model, and the way that the paper was approaching is to show the improvement that the new network has compared the explicit one.

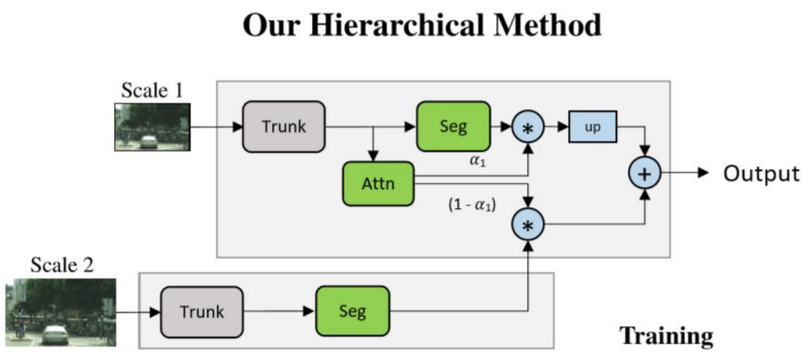


Figure 5 Our hierarchical method training

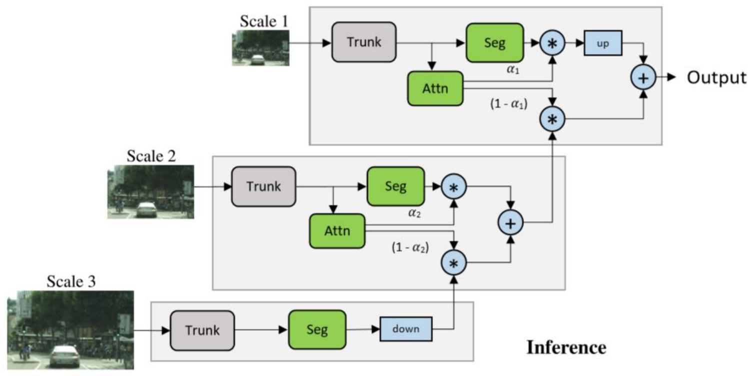


Figure 6 Our hierarchical method Inference

Figures 5 and 6 show the structure and architect of the new hierarchical network, it can be noticed that in the new there has been added some up and downscaling and other new methods, with our hierarchical method, instead of learning all attention masks for each of a fixed set of scales, we learn a relative attention mask between adjacent scales. When training the network, we only train with adjacent scale pairs. As shown in Figure 4 5 6, given a set of image features from a single (lower) scale, we predict a dense pixel-wise relative attention between the two image scales. In practice, to obtain the pair of scaled images, we take a single input image and scale it down by a factor of 2, such that we are left with a 1x scale input and an 0.5x scaled input, although any scale-down ratio could be selected. It is important to note that the network input itself is a re-scaled version of the original training images because we use image scale augmentation when we train. This allows the network to learn to predict relative attention for a range of image scales. When running inference, we can hierarchically apply the learned attention to combine N scales of predictions in a chain of computations, as shown in Figure and described by the equation below. We give precedence to Lower scales and work our way up to higher scales, with the idea that they have a more global context and can choose where predictions need to be refined by higher scale predictions.

More formally, during training, a given input image is scaled by factor r where $r = 0.5$ denotes downsampling by a factor of 2, $r = 2$ denotes upsampling by factor of 2, $r = 1$ denotes no operation, For our training, we choose $r = 0.5$ and $r = 1$. The two images with $r = 1$ and $r = 0.5$ are then sent through the shared network trunk, which produces semantic logits L and also an attention mask α for each scale, which are used to combine logits L between scales. Thus for two scales training and inference, with U being the bilinear upsampling operation, * and + are pixel-wise multiplication and addition respectively, the equation can be formalized as:

$$L_{(r=1)} = U\left(L_{(r=0.5)} * \alpha_{(r=0.5)}\right) + \left(\left(1 - U\left(\alpha_{(r=0.5)}\right)\right) * L_{(r=1)}\right)$$

There are two advantages to using our proposed strategy:

- At inference time, we can now flexibly select scales, thus adding new scales such 0.25x or 2.0x to a model trained with 0.5x, and 1.0x is possible with our proposed attention mechanism chains together in a hierarchical way. This differs from previously proposed methods that limited to using the same scaled that were used during model training.
- This hierarchical structure allows us to improve on the training efficiency as compared to the explicit method. With the explicit method, if using scales 0.5, 1.0, 2.0 the training cost is $0.5^2 + 1.0^2 + 2.0^2 = 5.25$, relative to single-scale training. With our hierarchical method, the training cost is only $0.5^2 + 1^2 = 1.25$.

Trunk:

One of the key parts in the architect of the model was the structure of the trunk. There were several network models for this part of the network, we used $ResNet\ 50, ResNet\ 101$, and $MobileNet$ for this part on the network.

ResNet: Short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks. This model was the winner of ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully. Prior to ResNet training very deep neural networks was difficult due to the problem of vanishing gradients.

ResNet first introduced the concept of skip connection. The diagram below illustrates skip connection. The figure on the left is stacking convolution layers together one after the other. On the right, we still stack convolution layers as before, but we now also add the original input to the output of the convolution block. This is called skip connection:

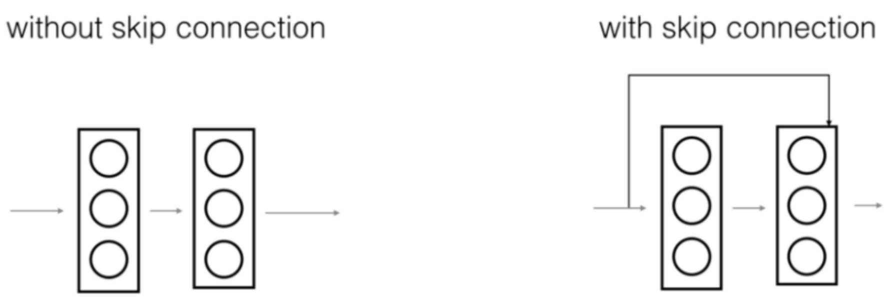


Figure 7

This is an interesting question. I think there are two reasons why Skip connections work here:

- They mitigate the problem of vanishing gradient by allowing this alternate shortcut path for the gradient to flow through
- They allow the model to learn an identity function which ensures that the higher layer will perform at least as good as the lower layer, and not worse

MobileNet: In the previous version, $MobileNet\ V1$ is introduced which dramatically reduces the complexity cost and model size of the network, which is suitable to Mobile devices or any devices with low computational power. In MobileNetV2, a better module is introduced with an inverted residual structure. Non-linearity's in narrow layers are removed this time. With MobileNetV2 as a backbone for feature extraction, state-of-the-art performances are also achieved for object detection and semantic segmentation.

Some of the Blocks in $MobileNet\ V2$

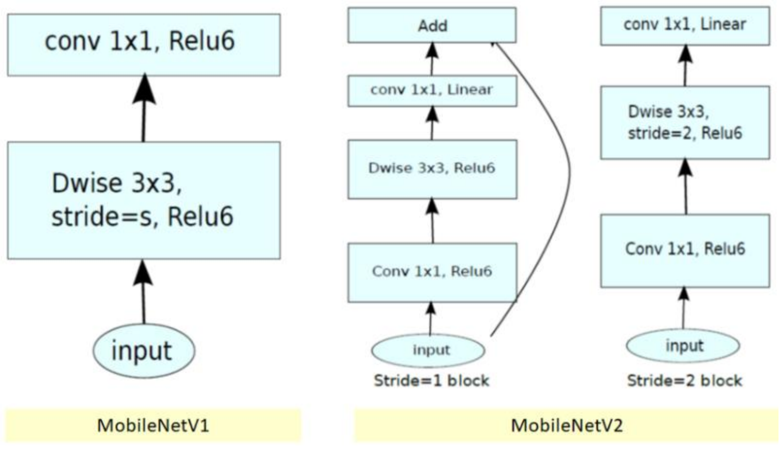


Figure 8

- In MobileNetV2, there are two types of blocks. One is a residual block with stride of 1. Another one is a block with stride of 2 for downsizing.
- There are 3 layers for both types of blocks.
- This time, the first layer is 1×1 convolution with ReLU6.
- The second layer is the depthwise convolution.
- The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if $ReLU$ is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.
- And there is an expansion factor t. And t=6 for all main experiments.
- If the input got 64 channels, the internal output would get 64×t=64×6=384 channels.

Segmentation:

One of the other key parts of the network was the segmentation part of the model wherein the following paper hasn't been clearly mentioned what kind of segmentation model they have used, so we tried $DeepLab\ V3$ and $DeepLab\ V3^{+}$ for the segmentation part, in the following we are going to give some information about this two networks.

DeepLab V3: This network was invented by Google, which is quite similar to the previous networks of $DeepLab\ V1, V2$. This model has a few outline which we are going to mention.

- Atrous Separable Convolution
  - Atrous Convolution

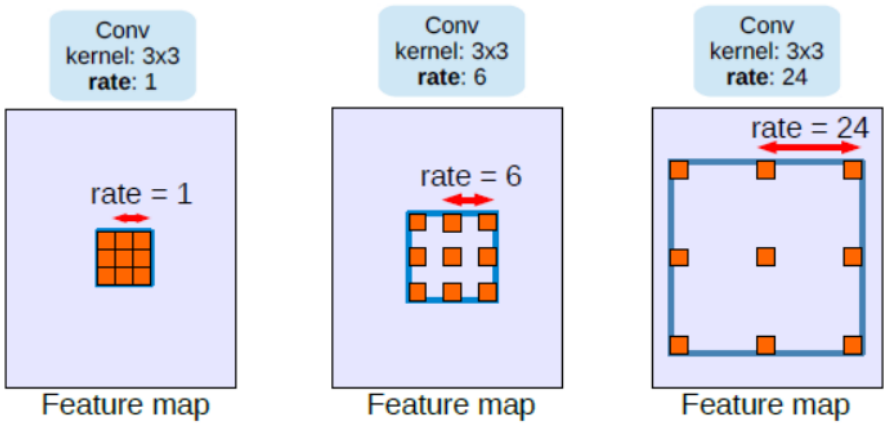


Figure 9 Atrous Convolution with Different Rates r

$$y[i] = \sum_{k} x[i + r.k]\omega[k] \quad Atrous\ Convolution$$

  - Atrous Separable Convolution

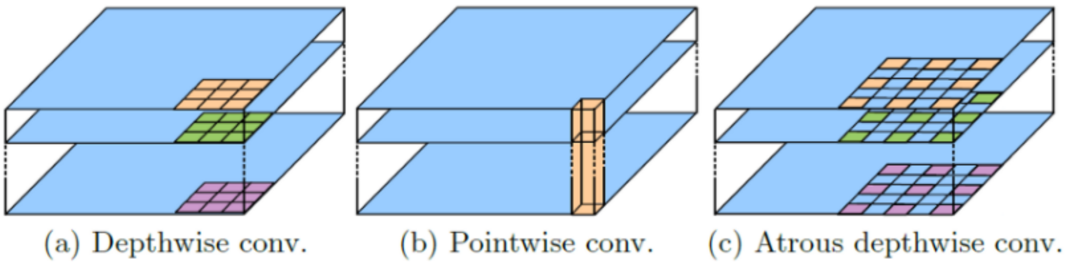


Figure 10 Depthwise Separable Convolution Using Atrous Convolution

- Encoder-Decoder Architecture

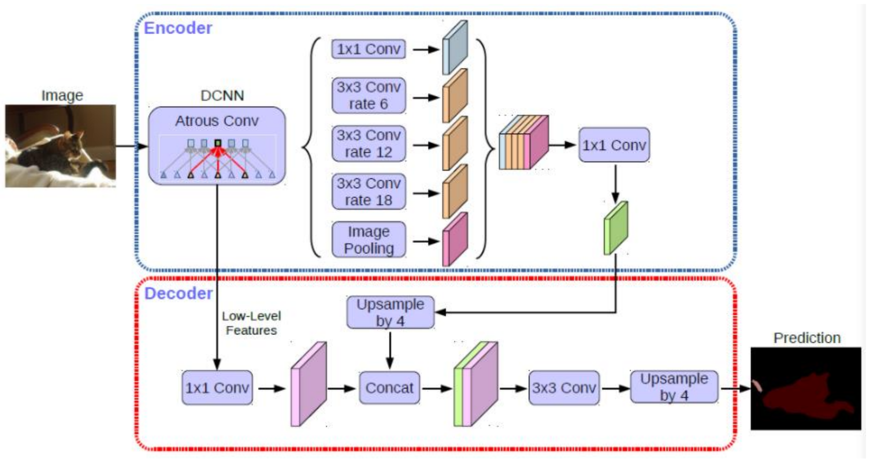


Figure 11

- Modified Aligned Xception
  - To be brief, some of the max pooling operations are replaced by separable conv in the entry flow. The number of repeating is increased from 8 to 16 in the middle flow. One more conv is added in the exit flow.
  - All max pooling operations are replaced by depth wise separable convolution with striding, in which atrous separable convolution is applied to extract feature maps at an arbitrary resolution.
  - Extra batch normalization and $ReLU$ activation are added after each 3×3 depth wise convolution.
- Ablation Study
  - It is found that 48 channels of 1×1 convolution used to reduce the channels of low-level feature map has the best performance.
  - And it is most effective to use the Conv2 (before striding) feature map and two extra [3×3 conv; 256 channels] operations.

This were some on the features of the networks that we were using in our implementation.

The Attention Architecture

There was no specific architecture mentioned for the attention block in the paper so we decided to design an attention model with the following architecture:

Layer 1:

- **ConvTranspose2d**
    - Kernel Size: 6*6
    - Input Channels: Trunk Output Features
    - Output Channel: 512
    - Stride: 2
    - padding: 2
- **Batch Normalization Layer**
- **LeakyRelu Activation Function**

Layer 2:

- **ConvTranspose2d**
    - Kernel Size: 6*6
    - Input Channels: 512
    - Output Channel: 265
    - Stride: 2
    - padding: 2
- **Batch Normalization Layer**
- **LeakyRelu Activation Function**

Layer 3:

- **ConvTranspose2d**
    - Kernel Size: 6*6
    - Input Channels: 256
    - Output Channel: 64
    - Stride: 2
    - padding: 2
- **Batch Normalization Layer**
- **LeakyRelu Activation Function**

Layer 4:

- **ConvTranspose2d**
    - Kernel Size: 5*5
    - Input Channels: 64
    - Output Channel: 1
    - Stride: 1
    - padding: 2
- **Batch Normalization Layer**
- **LeakyRelu Activation Function**

Data Augmentation

As we all know augmenting data is one of the best ways to prevent overfitting in the network and a good for generalizing the network, so the network will become robust to different images.

We used different methods to augment our data such as:

1. **Random crop**: By giving the image it will automatically crop the image for us and will give us the cropped part of the image, we can see some example coming figures.
2. **Changing contrast**: As we all know the images may come from different cameras with different features and contrast is one of those features that can be different from one camera to another so we need to make out network robust to this kind of problems, the point is that all the pixels of the image are influenced by this method.
3. **Gaussian Noise**: Unlike the contrast method this one doesn't influence all the pixels with the same way because every pixel is independent to other pixels therefor this will help the network to be more robust to damaged images, for example there might be an image where a particular part of it is damaged it will do less effect in the output result of the network.
4. **Horizontal flip**: Rotating and flipping always has been of the augmentation methods, we need to consider that we are not going to see an image upside down so rotating it with 90 degrees wouldn't help must so we just tried to flipping the image horizontally.

The following images are some sample of data augmentation that we have done without dataset

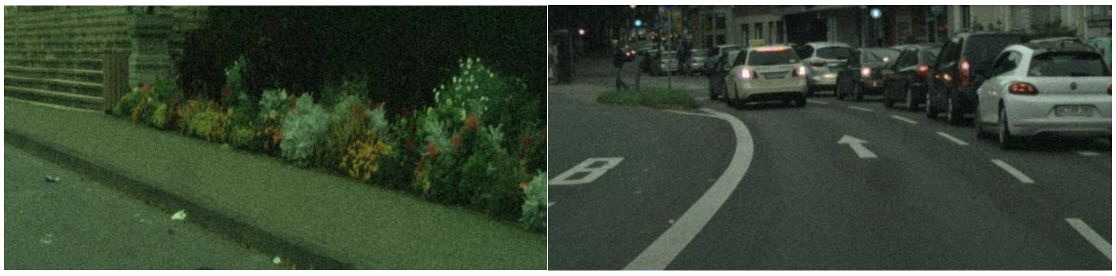


Figure 12

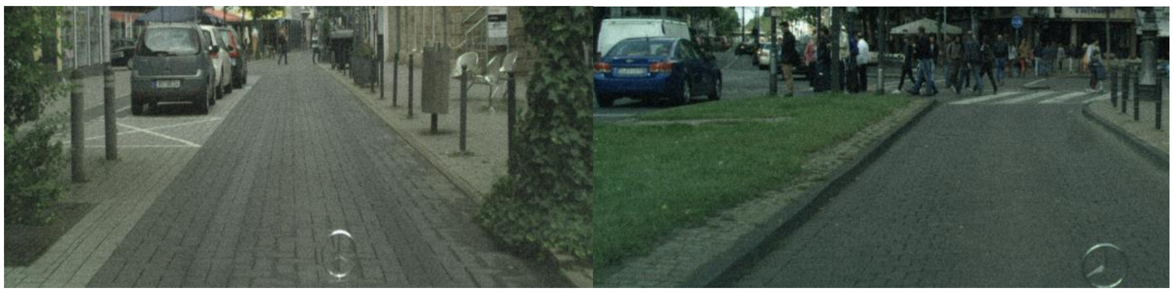


Figure 13

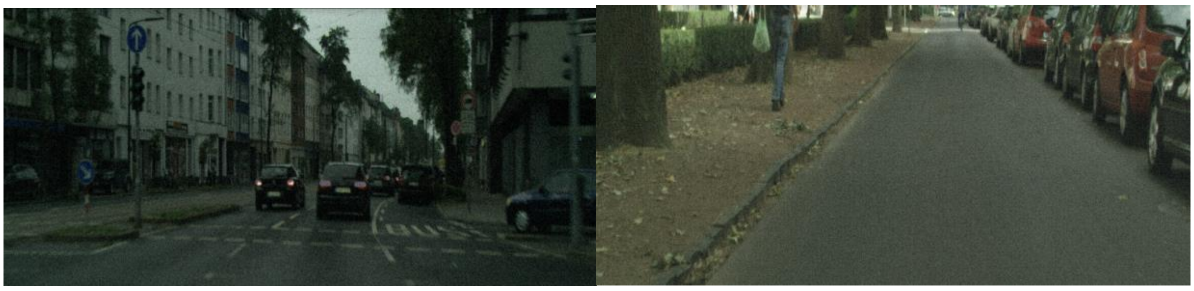


Figure 14

We can see from the images from figures 12 13 14 that the images we had used for example figure 12 left has been added Gaussian noise to it and in figure 14 the images are randomly cropped (we can't see the Benz logo in the image as a proof)

Semi-supervised learning

Learning can be divided to supervised learning and unsupervised learning, where is supervised learning we work with labels data's and in unsupervised learning we work with unlabeled data's.

Supervised: Most of the time, we need labeled data to do supervised machine learning. I particular, we use it to predict the label of each data point with the model. Since the data tells us what the label should be, we can calculate the difference between the prediction and the label, and then minimize that difference.

Unsupervised: another category of algorithms called unsupervised algorithms don't need labels but can learn from unlabeled data. Unsupervised learning often works well to discover new patterns in a dataset and to cluster the data into several categories based on several features. Popular examples are K-Means and Latent Dirichlet Allocation (LDA) algorithms.

As we all know labeling data is quite time consuming and is expensive, so we can't afford using small number of data or spending large amount of money just for labelling the datasets (it is obvious that the labels for our network are quite expensive because it takes a very long time just to label those images, so we decided to use semi supervised learning for this case which means at first we trained the network with labeled data and after that the network was trained fairly we used the unlabeled data as the networks input and used the result as the label of the particular image so we have a quite bigger dataset now (needed be mentioned that we have more unlabeled data in our dataset compared to the labeled data) so this will help us to get a better result, because as you know the key in deep learning is lots of data. So we did the same job to get a better performance from the network and labeled the unlabeled images and used them again for training the network.

Loss

Semantic segmentation is a fundamental problem in computer vision, and its goal is to assign semantic labels to every pixel in an image. Recently, much progress has been made with powerful convolutional neural networks such as (VGGNet, ResNet, Xception), and fancy segmentation models (FCN, PSPNet, SDN, DeepLab, ExFuse, EncNet). These segment approaches treat semantic segmentation as a pixel-wise classification problem and solve it by minimizing the average pixel-wise classification loss over the image. The most commonly used pixel-wise loss for the semantic segmentation is the softmax cross-entropy loss:

$$L_{ce}(y,p) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} y_{n,c}\, log\, log\, (p_n, c)$$

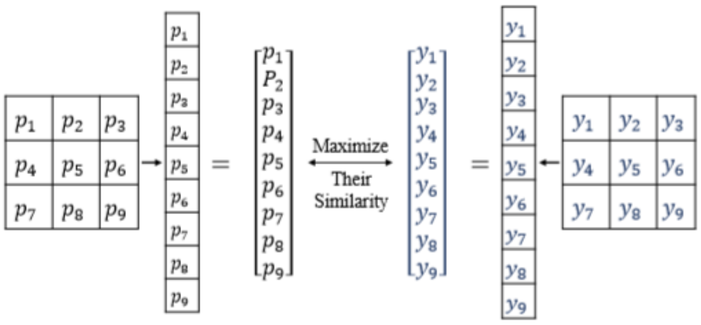


Figure 15

The figure above image region and its corresponding multi-dimensional point. Following the same strategy, an image can be cast into a multi-dimensional distribution of many high dimensional points, which encode the relationship between pixels.

Entropy and mutual information

Let X be a discrete random variable with alphabet X probability mass function (PMF) is $p(x)$, $x \in X$. For convenience, we denote $p(x)$, and $p(y)$ refer to two different random variables, and they are actually two different PMFs, $p_X(x)$ and $P_Y(y)$, respectively. The entropy of X is defined as:

$$H(X) = -\sum_{x \in X} p(x)\, log\, log\, p(x)$$

The entropy is a measure of the uncertainty of a random variable. Then the joint entropy $H(X,Y)$ and conditional entropy $H(Y)$ of a pair of discrete random variable (X,Y) with a joint distribution $p(x,y)$ and conditional distribution $p(y,x)$ are defined as:

$$H(X,Y) = -\sum_{x \in X}\sum_{y \in Y} p(x,y)\, log\, log\, p(x,y)$$

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \; log \; log \; p(y|x)$$

Indeed, the entropy of a pair random variables is the entropy of one plus the conditional entropy of other:

$$H(X,Y) = H(x) + H(X)$$

We now introduce mutual information, which is a measure of the amount of information that X and Y contain about each other. It is defined:

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \; log \; log \; \frac{p(x,y)}{p(x)p(y)}$$

The equation suggests that $I(X;Y)$ is very natural measure for dependence. It can also be considered as the reduction in the uncertainty of X due to knowledge of Y and vice versa:

$$I(X;Y) = H(X) - H(Y) = H(Y) - H(Y|X)$$

Where:

$$I(Y;P) = \int_Y \int_P f(y,p) \; log \; log \; \frac{f(y,p)}{f(y)f(p)} dydp$$

$$I_l(Y;P) \approx \frac{1}{2d} \; Tr(log \; log \; (M) \;)$$

Overall objective function: The overall objective function used for training the models is:

$$L_{all}(y,p) = \lambda \, L_{ce}(y,p) + (1-\lambda) \frac{1}{B} \sum_{b=1}^{B} \sum_{c=1}^{C} \left(-I_l^{b,c}(Y;P)\right)$$

where $\lambda \in [0,1]$ is a weight $L_{ce}(y,p)$ is the normal cross-entropy loss between y and p, B denotes the number of images in a mini-batch, and the maximization of RMI is cast as a minimization problem. The role of the normal cross-entropy loss is a measure of the similarity between the pixel intensity of two images, and RMI can be considered as a measure of the structural similarity between two images. Following the structural similarity (SSIM), the importance of pixel similarity and structural similarity is considered equally, so we simply set $\lambda = 0.5$.

We adopt the sigmoid operation rather than softmax operation to get predicted probabilities. This is because RMI is calculated channel-wise; we do not want to introduce interference between channels. Experimental results demonstrate the performance of models trained with softmax and sigmoid cross-entropy losses is roughly the same.

Cross Entropy Loss:

The Cross-Entropy Loss is actually the only loss we are discussing here. The other losses names written in the title are other names or variations of it. The CE Loss is defined as:

$$\mathrm{CE} = -\sum_{i}^{C} t_i \log s_i$$

Where t_i and s_i are ground truth and the CNN score for each class i in C. As usually an activation function (sigmoid/softmax) is applied to the scores before the CE loss computation, we write $f(S_i)$ to refer to activations.

In a binary classification problem where $C' = 2$ the cross entropy loss can be defined also as:

$$CE = -\sum_{i}^{C} t_i \log s_i = -t_1 \log(s_1) - (1 - t_1)\log(1 - s_1)$$

and C_2. $t_1[0,1]$ and s_1 are the ground truth and Where it's assumed that there are two classes: C_1 the score for $C_1$and $t_2 = 1 - t_1$ and $s_2 = 1 - s_1$ are the ground truth and the score for C_2 That is the case when we split a Multi-Label classification problem in C binary classification problems. See next Binary Cross-Entropy Loss section for more details.

Metrics

We need different kinds of metrics in order to show if the training has enhanced the network of not so in this case we are going to use mean intersection over union and pixel accuracy for the metrics.

Mean Intersection Over Union(MIOU): In an image there are several classes, we need to calculate the IOU for every class and get the average of these metrics and give it as the result

Intersection Over Union(IOU): The Intersection-Over-Union (IOU), also known as the Jaccard Index, is one of the most commonly used metrics in semantic segmentation… and for good reason. The IOU is a very straightforward metric that's extremely effective. Take a look at figure 16 simply put, the IOU is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth, as shown on the image to the left. This metric ranges from 0–1 (0–100%) with 0 signifying no overlap (garbage) and 1 signifying perfectly overlapping segmentation (fat dub).

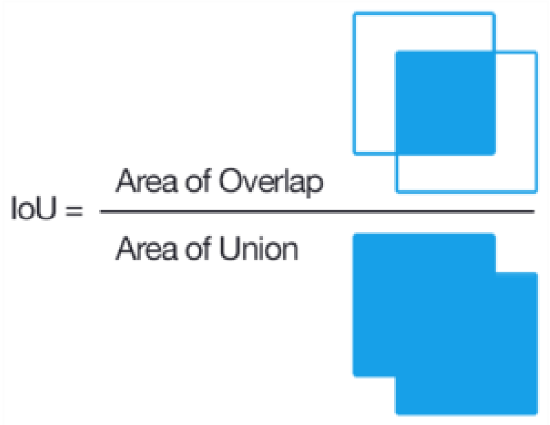


Figure 16

There is also another example for the pixel accuracy in segmentation, you see in the figures below:

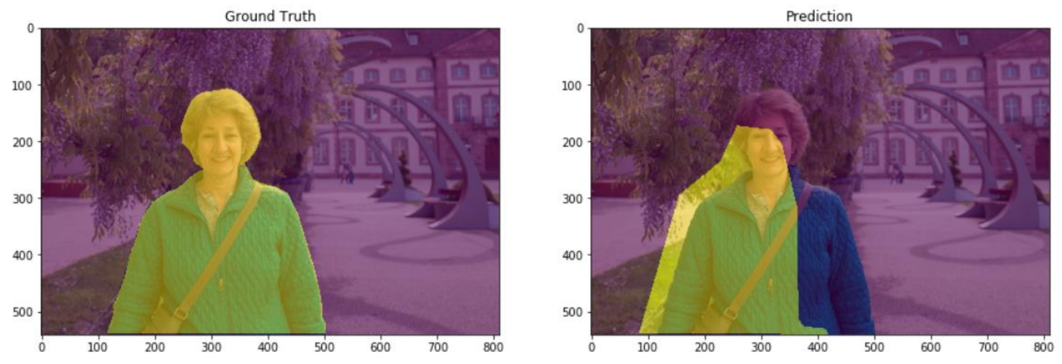


Figure 17

So now we want to identify the intersection and the union between the Ground Truth and the Prediction like below:

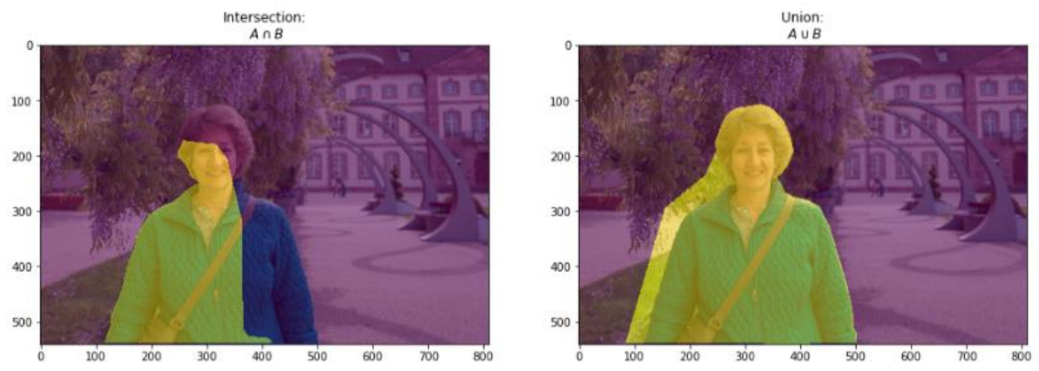


Figure 18

The result for the IOU can be seen in the figures above.

Pixel Accuracy: Pixel accuracy is perhaps the easiest to understand conceptually. It is the percent of pixels in your image that are classified correctly. While it is easy to understand, it is in no way the best metric. As it was mentioned from the previous part the IOU metric is much better that pixel accuracy. This issue is called class imbalance. When our classes are extremely imbalanced, it means that a class or some classes dominate the image, while some other classes make up only a small portion of the image. Unfortunately, class imbalance is prevalent in many real world data sets, so it can't be ignored.

- True Positive (TP): pixel classified correctly as X
- False Positive (FP): pixel classified incorrectly as X
- True Negative (TN): pixel classified correctly as not X
- False Negative (FN): pixel classified incorrectly as not X

The formula for calculating pixel accuracy is:

$$Pixel\ Accuracy = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN}$$

For example

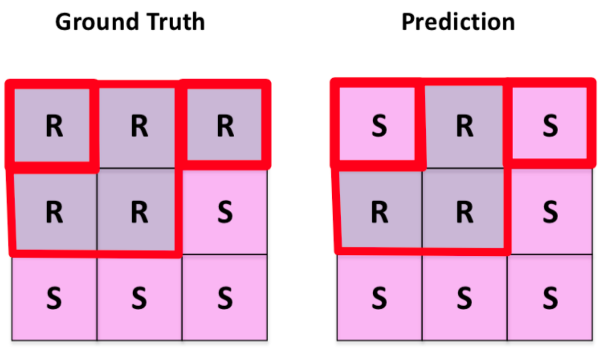


Figure 19

$$Pixel\ Accuracy = \frac{3+4}{3+4+0+2} = \frac{7}{9} = 77.8\%$$

Why IOU is better than Pixel accuracy:

Studies have shown that the IOU metric is better that pixel accuracy because in many cases where there are two classes for segmentation the network might fall to a local minimum for example the look at the images:

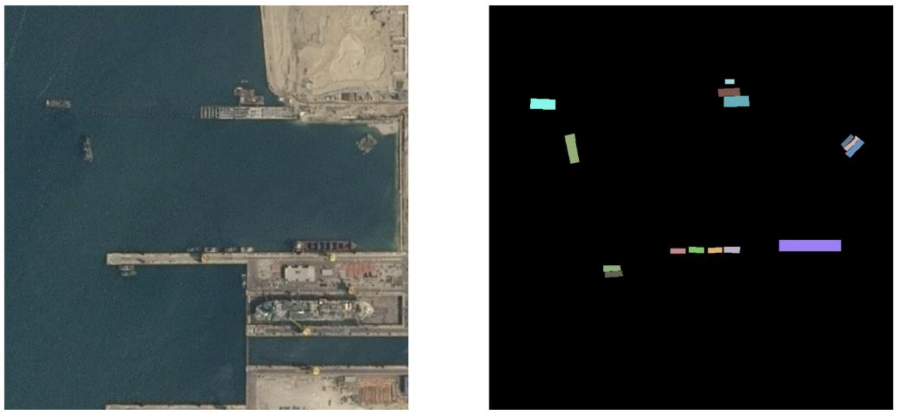


Figure 20

Where the output of the network is stuck in a local minimum and gets same result every time like figure 21.

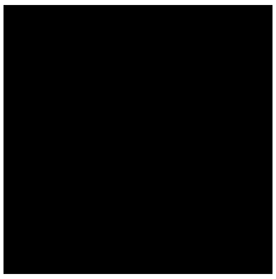


Figure 21

Where the pixel accuracy is 95% but as we can notice the result isn't good but for IOU metric this accuracy number is about 47.5%, which in this case the 47.5% is much reasonable.

So this reason shows why IOU is better than pixel accuracy in segmentation tasks.

Training Process and results

Trained Models

At the beginning, our ultimate goal was to build up two-stage and three-stage hierarchical multi-scale attention by trying both Resnet-50 and HRNet-OCD as trunk and the DeepLab v3+ as the segmentation block. This, however, did not become possible due to our computational limitations (specially the small GPU memory). Inevitably, we replaced the HRNet-OCD architecture with MobileNet in our experiences due to its smaller size and fewer parameters. Even after this, we were still not able to train our models end to end and instead, we relied transfer learning methods for obtaining acceptable results.

Our main approach in training the hierarchical models, was to train different 3 segmentation models for segmenting images with three different scales (2, 1, and 0.5) and eventually use the trained trunk and segmentation blocks for the original models. We use a same architecture to the hierarchical models' initial stage for training truck and segmentation blocks on images with various sizes.

Training the initial stages

Here, we trained six segmentation blocks using two trunk architectures and on three image sets with different scales.

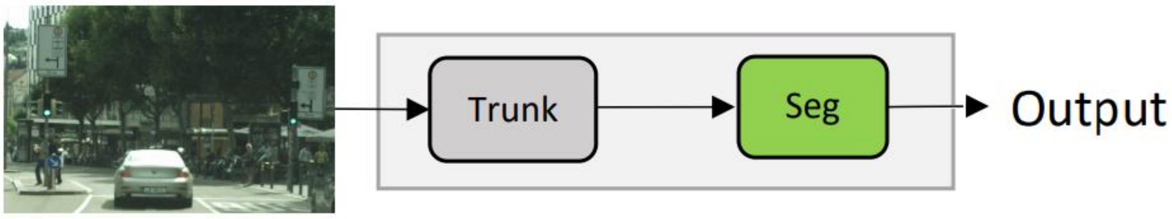


Figure 22

index	Trunk	Segmentation Block	Image Scale
1	MobileNet	DeepLab V3+	0.5
2	MobileNet	DeepLab V3+	1
3	MobileNet	DeepLab V3+	2
4	ResNet	DeepLab V3+	0.5
5	ResNet	DeepLab V3+	1
6	ResNet	DeepLab V3+	2

In order to speed up the training process we used pre-trained MobileNet and ResNet parameters in our models and fine tune them doing the training process. Other training parameters are explained bellow:

- we initialized the learning rate to be 1e-3 and decay it after each epoch by a factor of 0.8.
- To delay over fitting, we used a weight decay rate of 0.01

- For each model, we set the mini-batch size to the biggest possible number considering the accessible memory. This number was 4 for the first model, 3 for the second and the third, and 2 for others.

The average time for training the model on a single batch was approximately 3 seconds which was dramatically high and led to long training process and so we could only train each model for few epochs.

The final results of training each of the six models are shown in the table below:

index	**Trunk**	**Segmentation Block**	**Image Scale**	**#Epochs**	**Mean IOU On Validation Set**	**Average Accuracy On Validation Set**	**Training Time per Epoch**
1	MobileNet	DeepLab V3+	0.5	7	51.4%	71.3%	45 min
2	MobileNet	DeepLab V3+	1	7	50.6%	68.7%	60 min
3	MobileNet	DeepLab V3+	2	5	50.3%	68.2%	60 min
4	ResNet	DeepLab V3+	0.5	6	50.1%	68.4%	1.5 h
5	ResNet	DeepLab V3+	1	6	48.0%	76.3%	1.5 h
6	ResNet	DeepLab V3+	2	5	49.0%	68.1%	1.5 h

Initial Stage outputs:

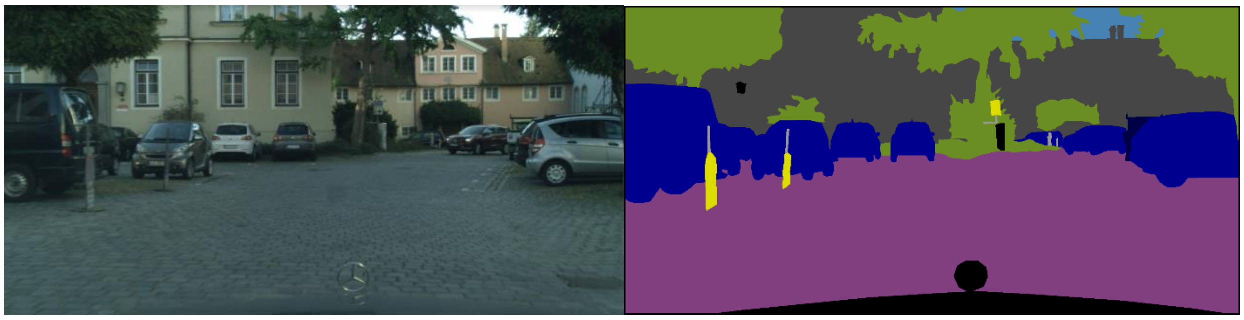


Figure 23 Real image and ground truth

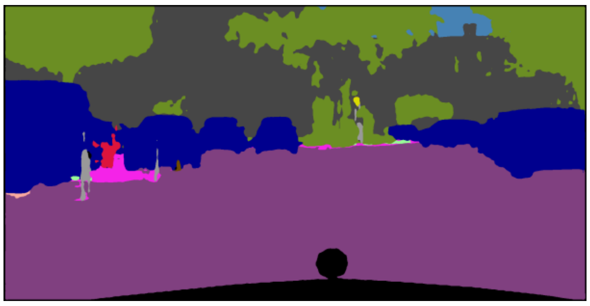


Figure 23 Scale 2

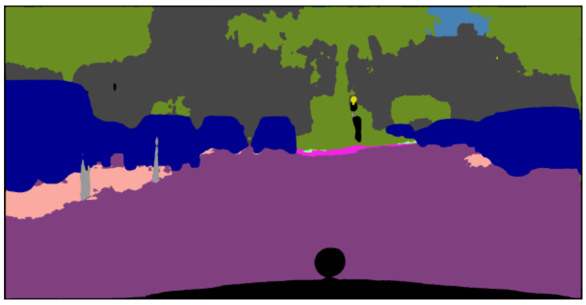


Figure 24 Scale 1

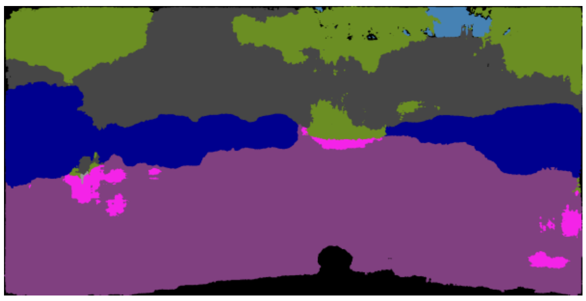


Figure 25 Scale 0.5

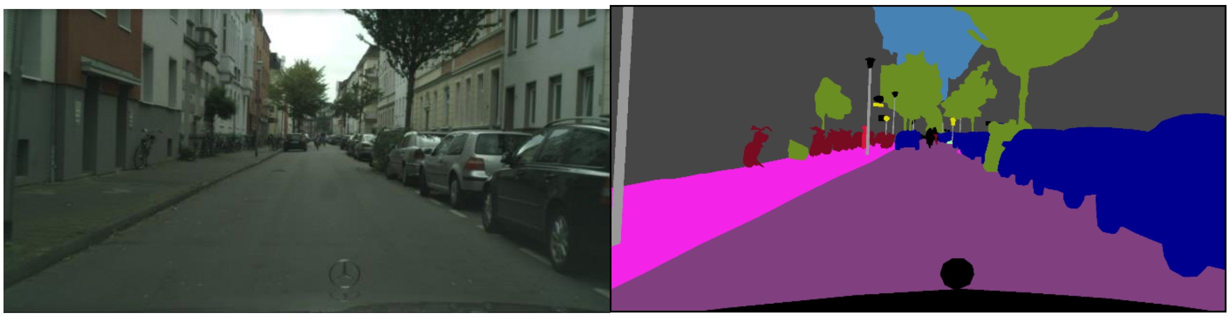


Figure 26 Read image and ground truth

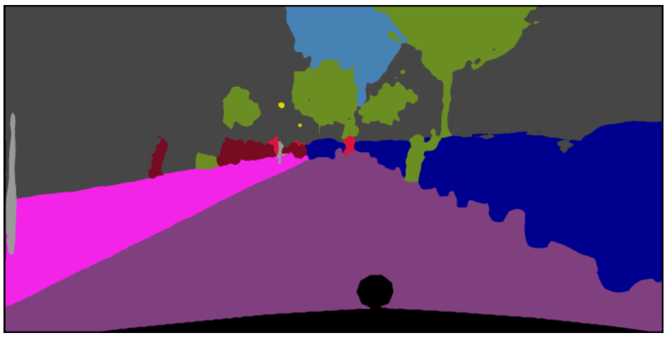


Figure 27 Scale 2

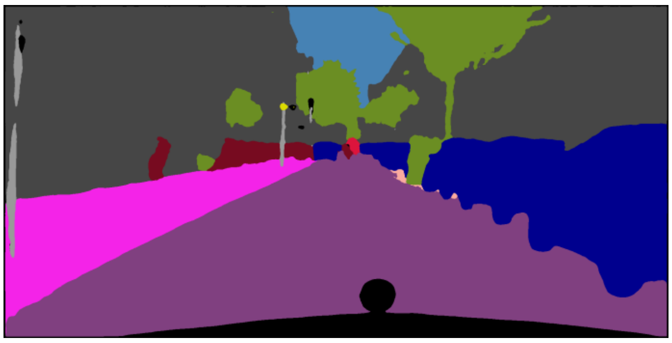


Figure 28 Scale 1

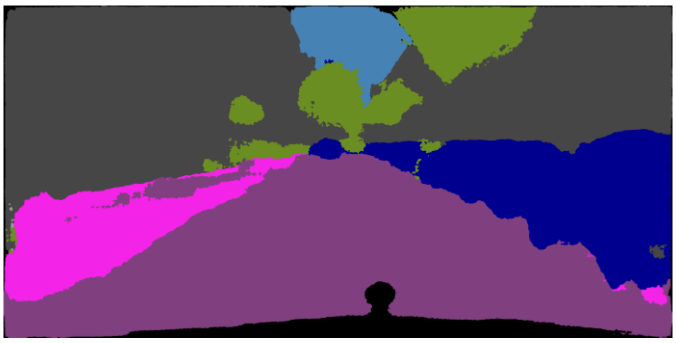


Figure 29 Scale 0.5

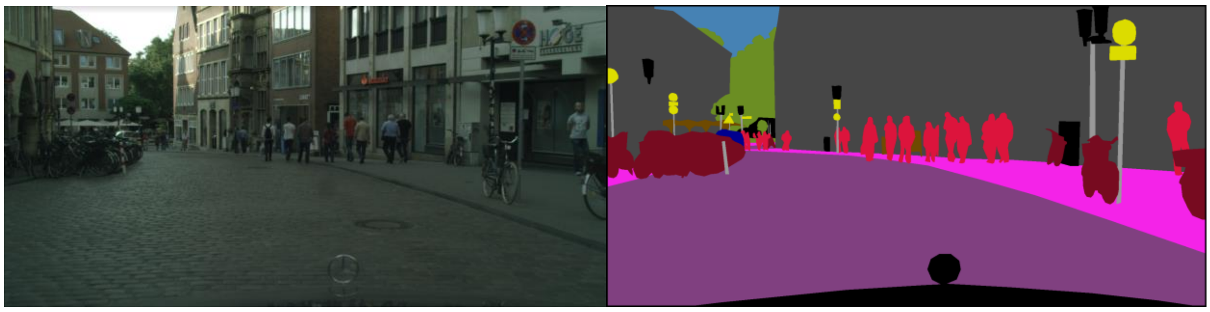


Figure 30 Real image and ground truth

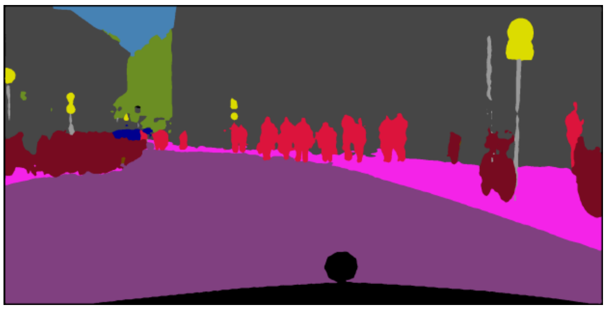


Figure 31 Scale 0.5

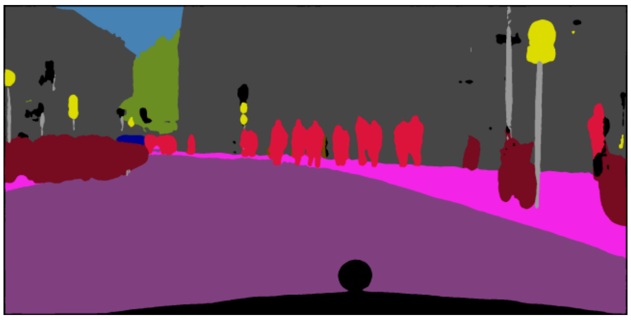


Figure 32 Scale 1

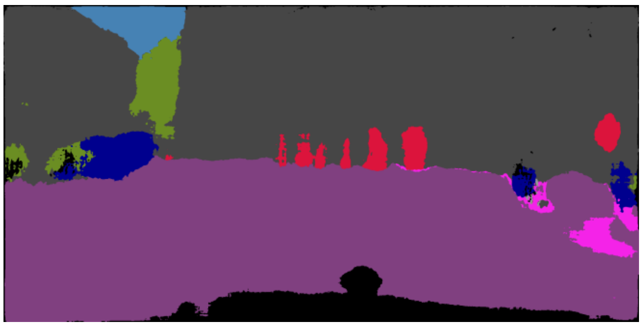


Figure 33 Scale 2

Training the Main Models

As mentioned above, we used the trained trunk and segmentation blocks for the previous section to train the final models. As the obtained results were much better for the models with a MobileNet trunk, we only used the MoblieNet architecture in the models that will be discussed in this section.

By freezing the obtained parameters form the trained segmentation and trunk blocks in the two-stage and three-stage architecture, we became capable of training the attention masks for 5 epochs in both models. Afterwards, we fine-tuned the truck and segmentation blocks of each stage (one stage at each epoch) while the attention masks were frozen. In this way, we overcame the GPU memory challenges and enhanced the mean IOU of the networks. For training the attention masks and fine tuning the remained parts, we used learning rates equal to 5e-4 and 1-e4 respectively.

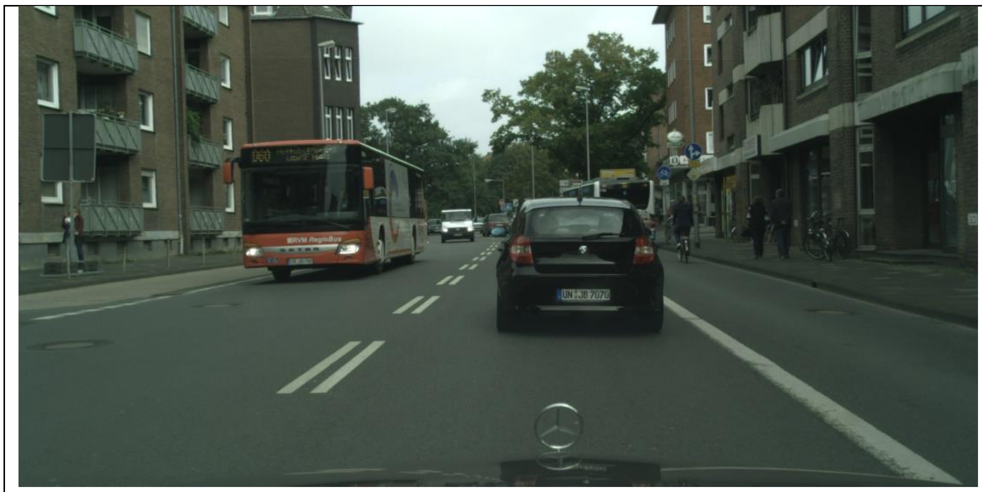
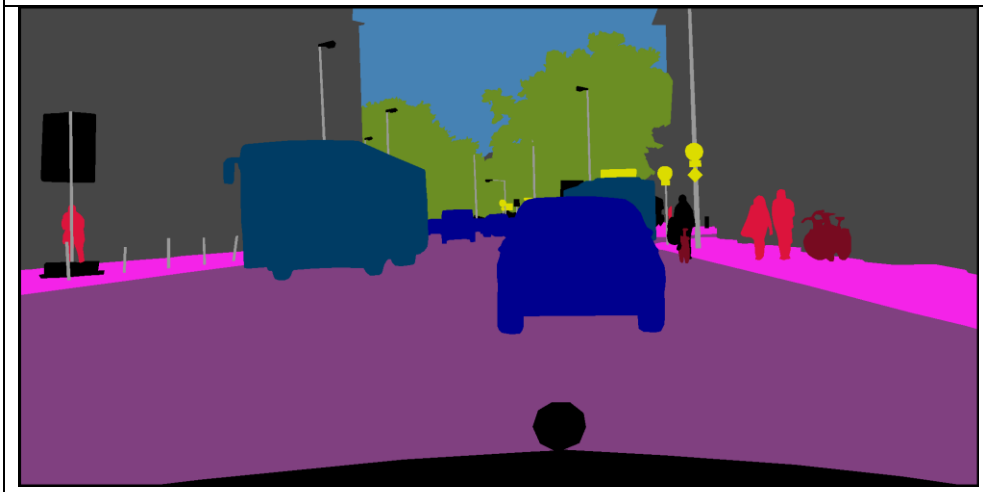
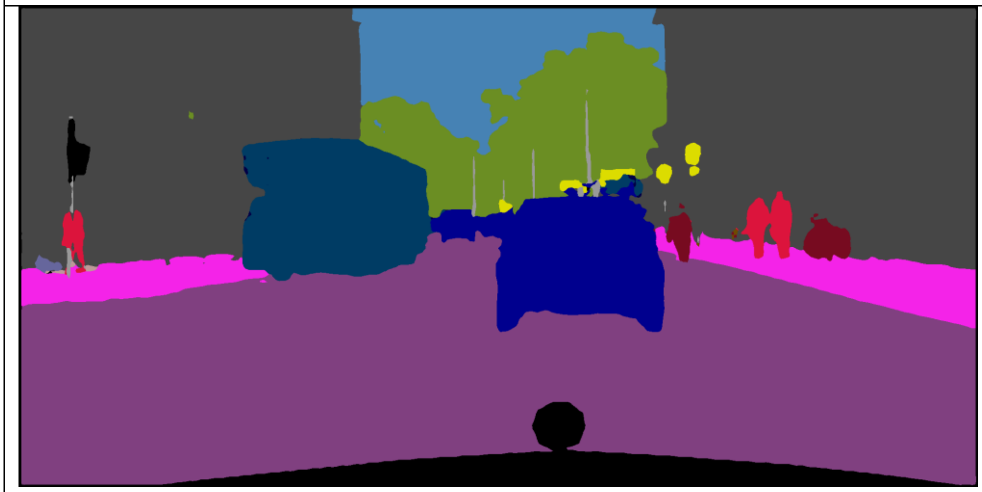
#Stages	**Image Scales**	**#Epochs**	**Mean IOU On Validation Set**	**Average Accuracy On Validation Set**
2	1/0.5	10	53.5%	75.2%
3	2/1/0.5	10	51.8%	74.6%

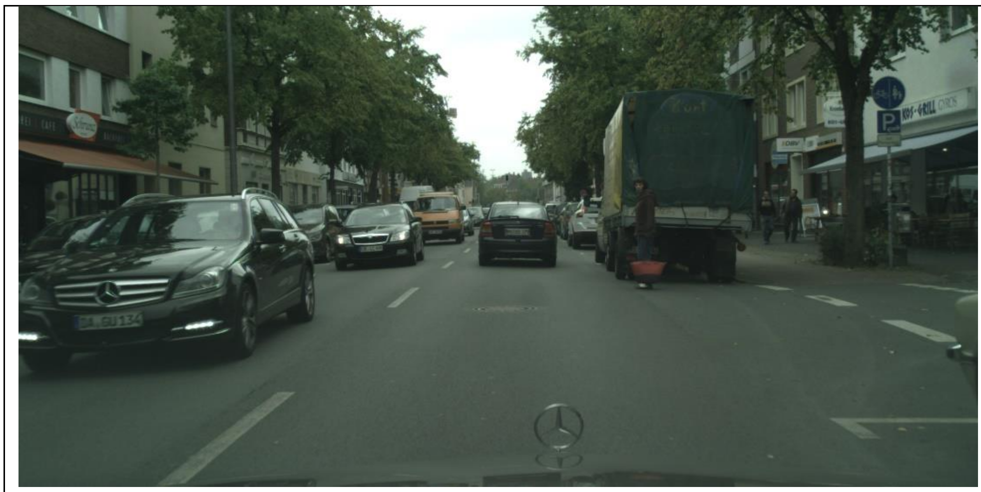
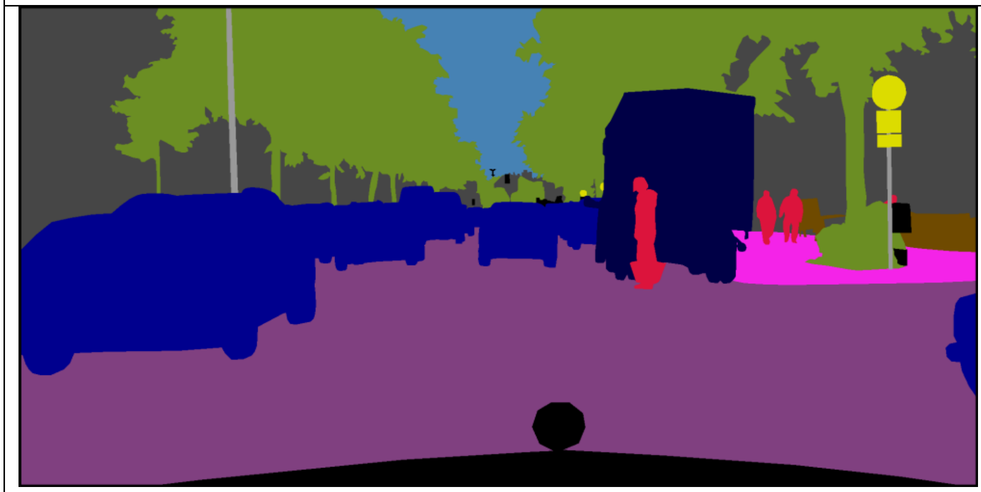
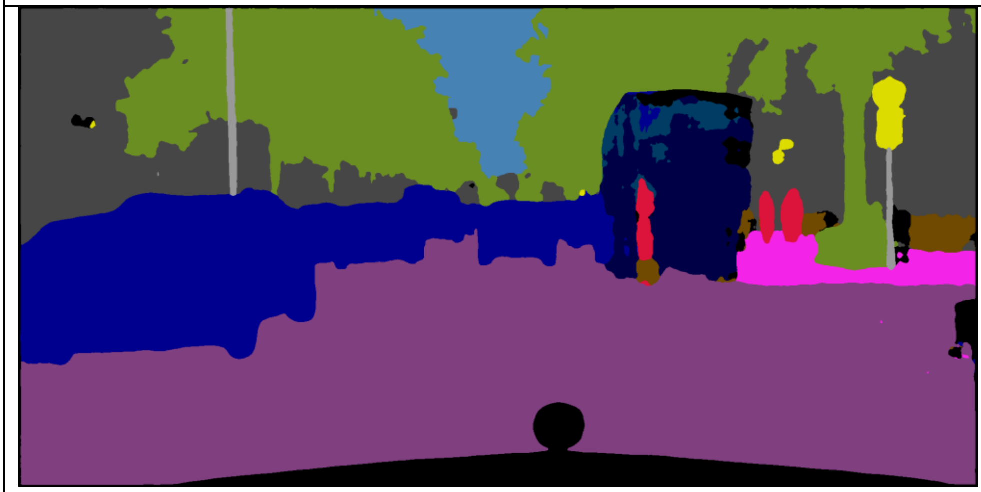
Some Points about the Training:

- Also here, we used the explained semi-supervised method to increase the efficiency of the segmentation model. Using the pest model trained in the previous section, we labeled the test photos which were not labeled before. Then, we trained the two-stage and the three-stage models using the train-set and the auto-labeled test-set.
- Computing the RMI loss is very time and memory consuming due to its complexity. Due to our limitations, in order to increase the speed of the training process, we replaced the RMI loss with the rather simpler Cross Entropy loss. The results however, were not seen to change dramatically and this was while a considerable amount of memory was saved which could be used for increasing the mini batch size.

Full Network Result:

3 Stage Result:

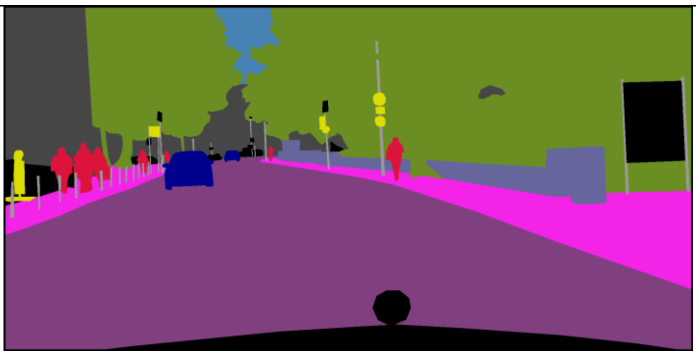
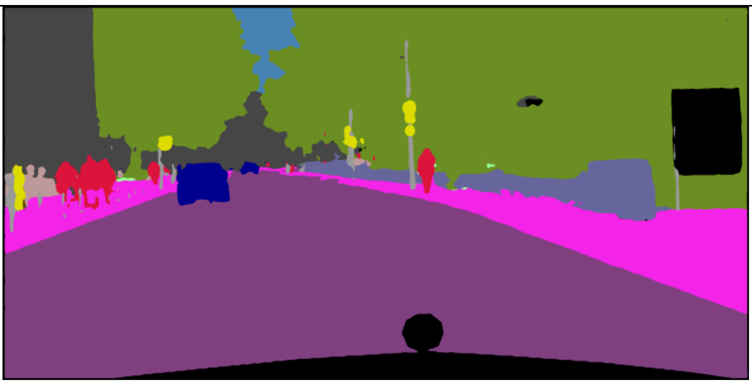
	Original Image
	Ground Truth
	Predicted Labels

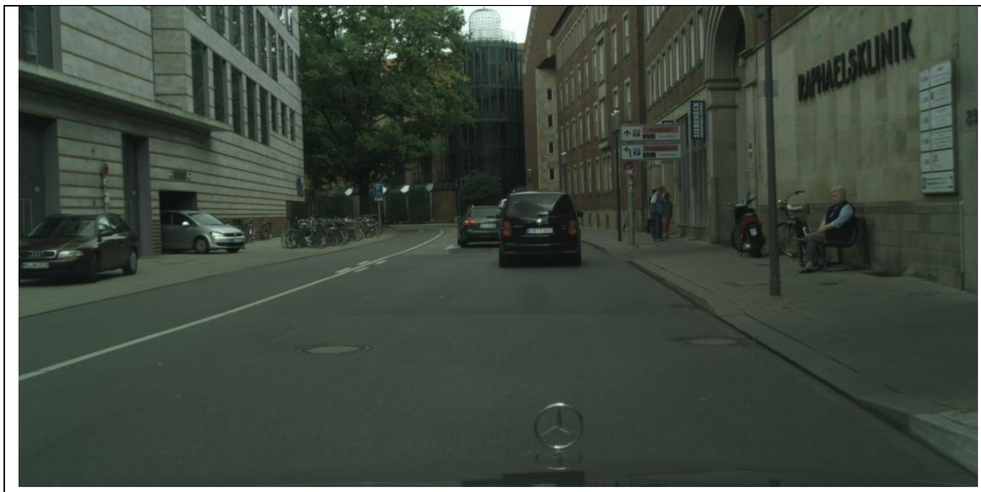
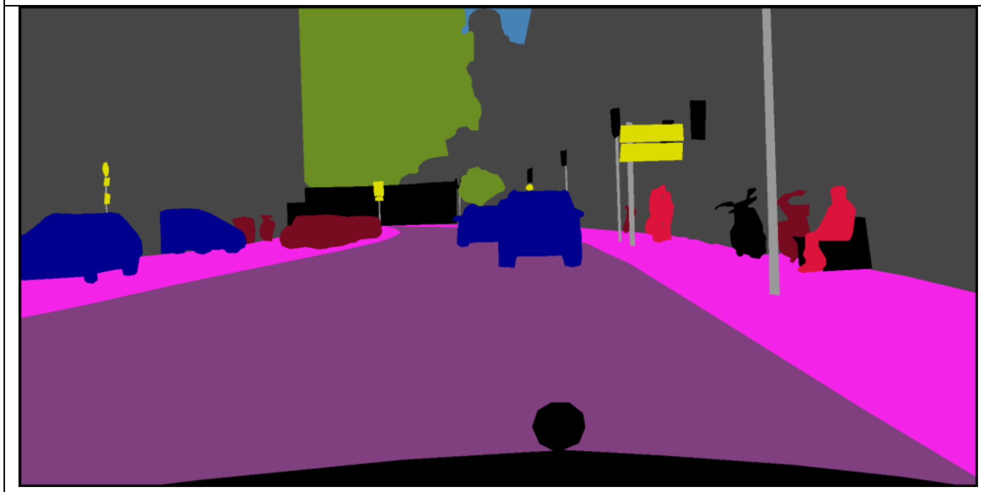
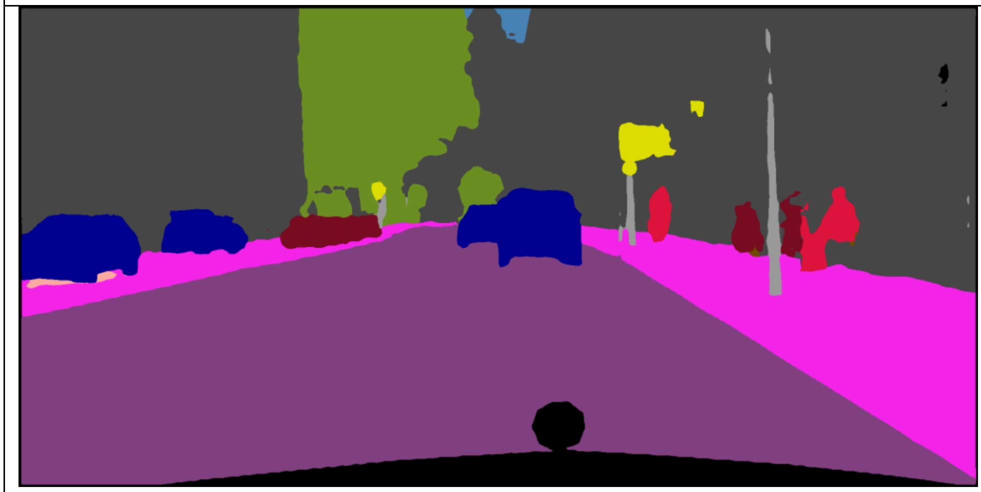
	Original Image
	Ground Truth
	Predicted Labels

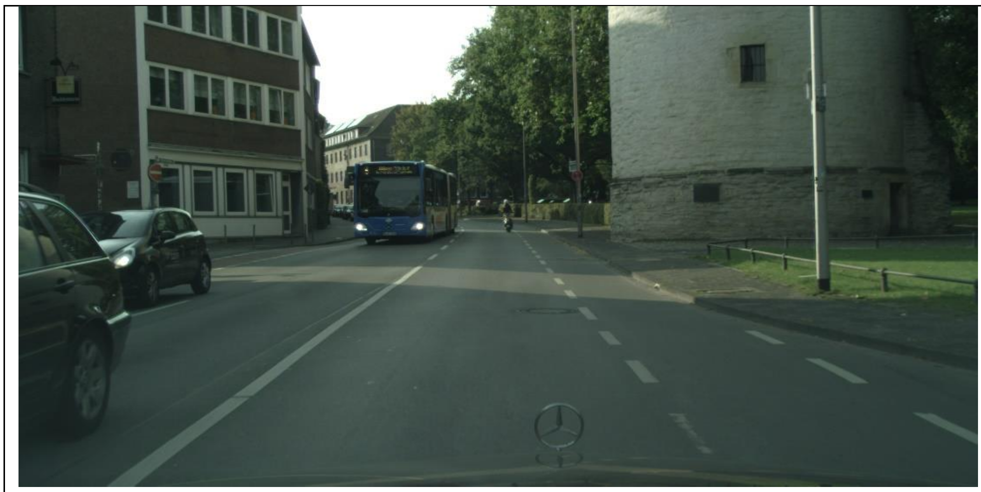
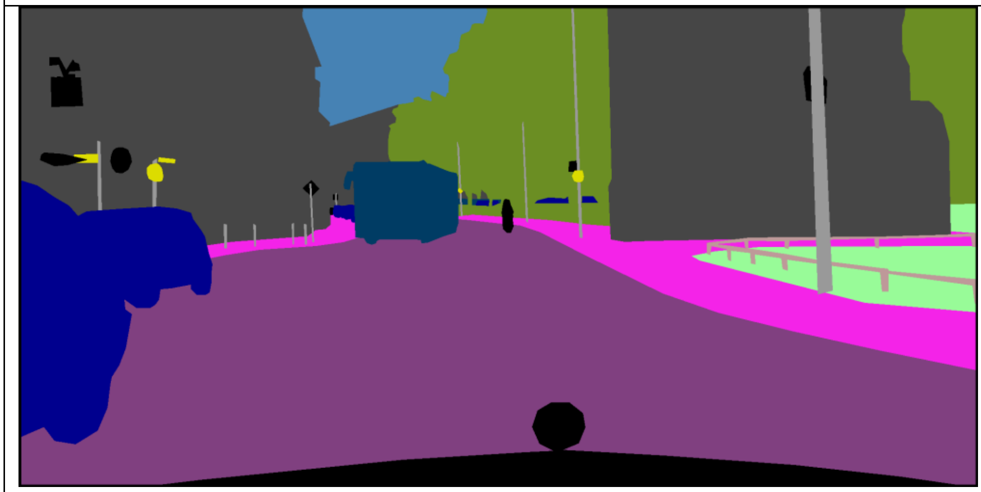
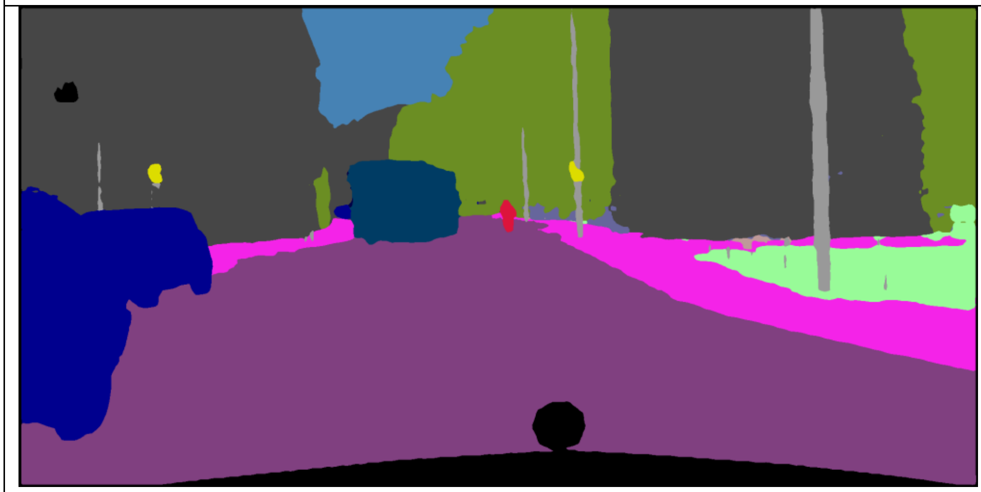
The Table of 3 stage result IOU:

Class	IOU
bicycle	0.5225412765215
building	0.83234612851222
bus	0.4223421657216
car	0.8623421512512
dynamic	0.0924312534592
fence	0.273320724334
ground	0.02184629573245
parking	0.200143135151414
person	0.57551240134615
pole	0.41231415715891
road	0.914164761415154
sidewalk	0.67112471641415
sky	0.9141457158615
static	0.14526527627525
terrain	0.419253912365234
traffic sign	0.5512461235602523
truck	0.123425621616262
untitled	0.4834512561025
vegetation	0.8721351261246
wall	0.221516346234632

2 Stage Result:

	Original Image
	Ground Truth
	Predicted Labels

	Original Image
	Ground Truth
	Predicted Labels

	Original Image
	Ground Truth
	Predicted Labels

The Table of 2 stage result IOU:

Class	IOU
bicycle	0.5473074047941955
building	0.8441790674148015
bus	0.46123703963929596
car	0.8861487497468359
dynamic	0.10016630230039981
fence	0.2903161366749986
ground	0.022144504338950342
parking	0.2011630471468155
person	0.590386473130647
pole	0.435283320151004
road	0.9278358910729865
sidewalk	0.6826796078174527
sky	0.9074193849757702
static	0.15528786671154604
terrain	0.4263387491560082
traffic sign	0.5764002194274289
truck	0.15848751565554117
untitled	0.5
vegetation	0.8820577262634581
wall	0.2539469992215929

Networks Graph:

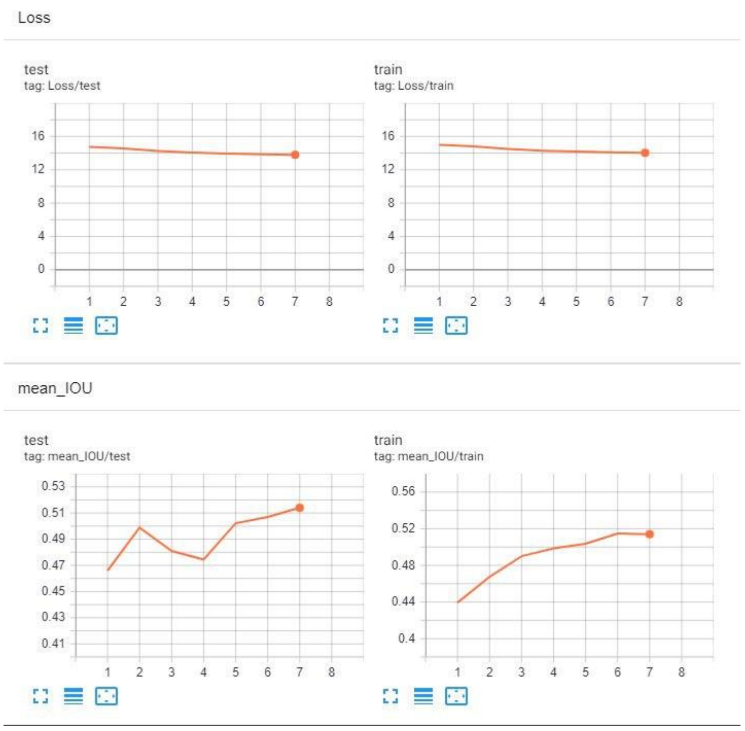


Figure 34 MobileNet initial stage scale 0.5

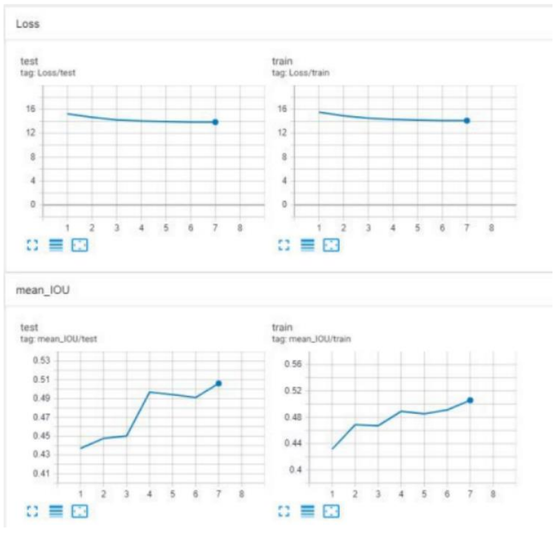


Figure 35 MobileNet initial stage scale 1

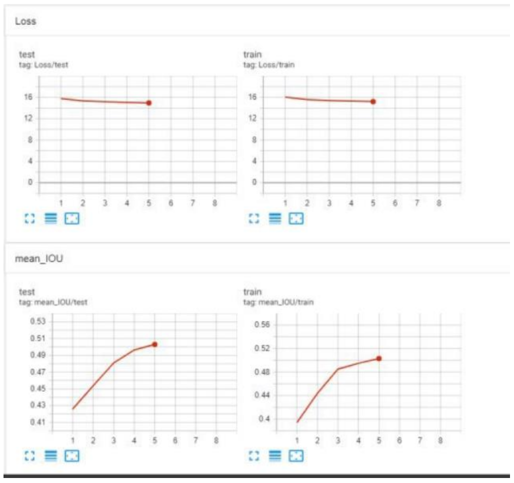


Figure 36 MobileNet initial stage scale 2

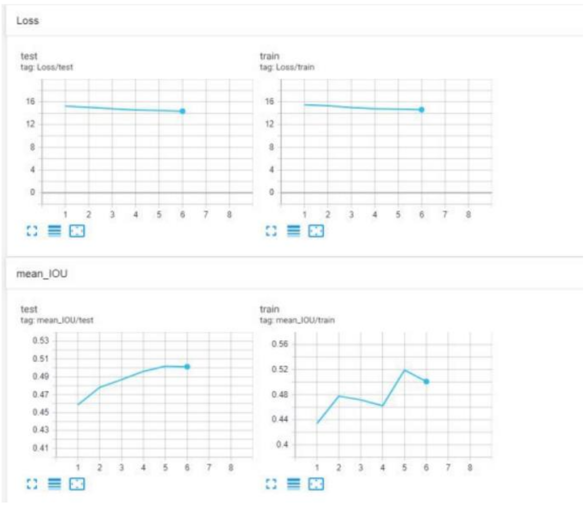


Figure 37 ResNet initial stage scale 0.5

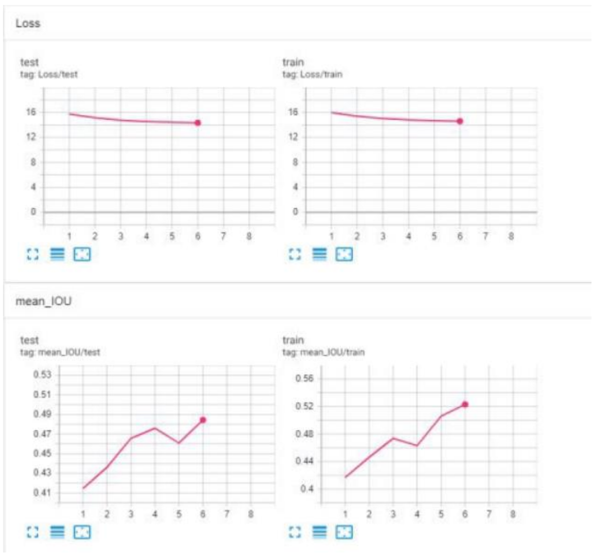


Figure 38 ResNet initial stage scale 1

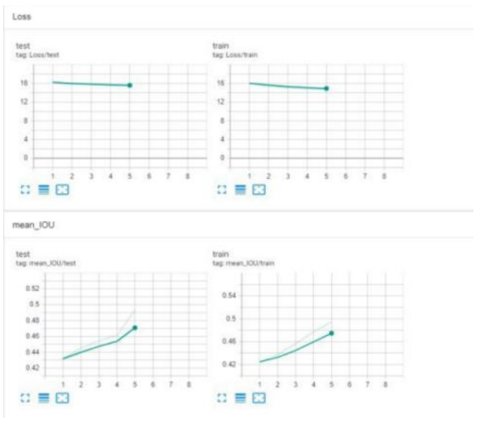


Figure 39 ResNet initial stage scale 1

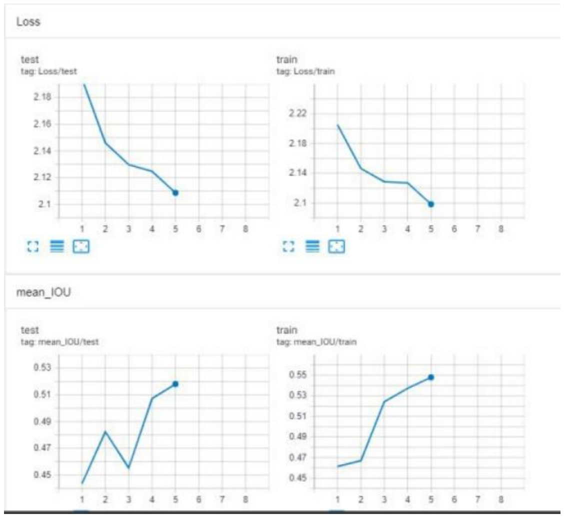


Figure 40 MobileNet initial stage scale 0.5_1

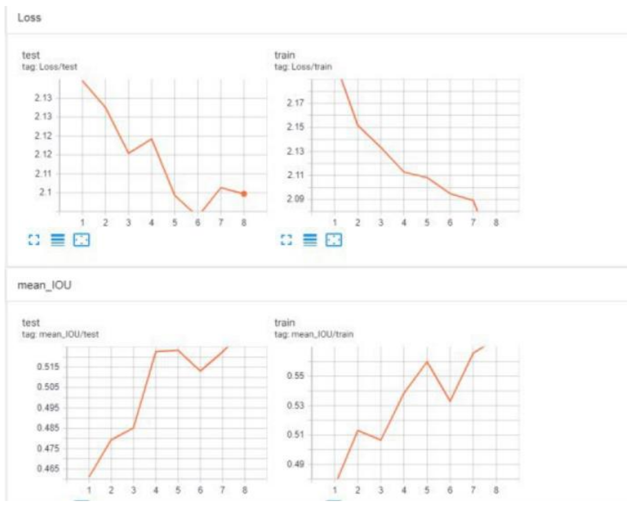


Figure 41 MobileNet initial stage scale 0.5_1_2

Conclusion

In this project, we have shown the result of the multi-scale attention has a better effect on the semantic segmentation result. We used different methods to improve our result such as augmenting the data that we were using. And the overall result shows that even though we had no High-Tec equipment's and had several problems that were clearly mentioned in the limitation part of the report, but we were still able to achieve the papers results for example we shown the if we used a small scale the network is better in finding the general features in the image as if we used larger scales it will help the network to recognize the smaller details in the image, but we weren't able to reach the accuracy of paper. We also found out some objects are better detected by a particular scale for example small objects are better detected by large scales.

As it has mentioned we trained different part of the network and in the end we combined them and used attention blocks as well as hierarchical method and the result of the combination was superior to the result from a single part.

References

[1] [Online]. Available: https://www.cityscapes-dataset.com/dataset-overview/.

[2] [Online]. Available: https://www.cityscapes-dataset.com/examples/#fine-annotations.

[3] [Online]. Available: https://www.cityscapes-dataset.com/dataset-overview/#class-definitions.

[4] [Online]. Available: https://keras.io/api/metrics/segmentation_metrics/.

[5] [Online]. Available: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c.

[6] [Online]. Available: https://towardsdatascience.com/review-deeplabv3-atrous-convolution-semantic-segmentation-6d818bfd1d74.

[7] [Online]. Available: https://medium.com/@sh.tsang/review-deeplabv3-atrous-separable-convolution-semantic-segmentation-a625f6e83b90.

[8] [Online]. Available: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33#:~:text=The%20ResNet%2D50%20model%20consists,over%2023%20million%20trainable%20parameters..

[9] [Online]. Available: https://towardsdatascience.com/supervised-learning-but-a-lot-better-semi-supervised-learning-a42dff534781.

[10] [Online]. Available: https://towardsdatascience.com/evaluating-image-segmentation-models-1e9bb89a001b.

[11] [Online]. Available: https://medium.com/inside-machine-learning/placeholder-3557ebb3d470.

[12] [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2.

[13] [Online]. Available: https://github.com/mcordts/cityscapesScripts/blob/master/cityscapesscripts/helpers/labels.py.

[14] [Online]. Available: https://github.com/CSAILVision/semantic-segmentation-pytorch/blob/4a4d88a8885b63177c9fc812a104bcad4ae9cee8/mit_semseg/utils.py%23L136.

[15] [Online]. Available: https://github.com/ZJULearning/RMI/blob/master/losses/rmi/rmi.py.

[16] [Online]. Available: https://github.com/aharley/segaware.

[17] [Online]. Available: https://towardsdatascience.com/intersection-over-union-iou-calculation-for-evaluating-an-image-segmentation-model-8b22e2e84686.

[18] Zhao, Shuai; Yang Wang; Zheng Yang; Deng Cai, "Region Mutual Information Loss for Semantic Segmentation," in *Advances in Neural Information Processing Systems*, 2019.

[19] Liang-Chieh Chen; Yi Yang; Jiang Wang; Wei Xu; Alan L. Yuille, "Attention to Scale: Scale-Aware Semantic Image Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[20] Qizhe Xie; Minh-Thang Luong; Eduard Hovy; Quoc V. Le;, "Self-Training With Noisy Student Improves ImageNet Classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* , 2020.