UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Segmentare de imagini folosind retele convolutionale
Segmentarea autovehiculelor

Valeriu Prodan

Coordonator științific:

Prof.dr.ing. Adina Florea
Conf.dr.ing Irina Mocanu

BUCUREȘTI

2019

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

Image segmentation using convolutional neural networks
Car segmentation

Valeriu Prodan

Thesis advisor:

Prof.dr.ing. Adina Florea
Conf.dr.ing Irina Mocanu

BUCHAREST

2019

CONTENTS

SINOPSIS

Dupa celebrul scandal in jurul retelei de socializare Facebook in care date personale ale utilizatorilor au fost folosite fara voia lor, atat in Europa cat si in alte parti ale lumii, societatea a inceput sa puna accent pe protejarea informatiilor personale ale indivizilor. Desi pozele din modul "vedere a strazii" din hartile Google elimina elemente cum ar fi fetele persoanelor sau numerele masinilor, contextul actual impune sporirea sistemelor de protejare a datelor personale, astfel ar fi benefica eliminarea automata a masinilor in intregime. Acest proiect vine cu o solutie de identificare a masinilor in poze, generandu-se o masca de marimi identice cu cele ale imaginii analizate. Pe baza acestei masti se pot aplica filtre de eliminare, incetosare, reconstructie, etc.

ABSTRACT

After the famous scandal involving Facebook which was accused of not protecting it is users personal data resulting in misusage of those information, both in Europe and in the rest of the world the society started to focus on protecting this kind of information. Although Google's street view pictures blur elements like people's faces or cars numbers, today's contexts demands upgrading the systems that are in charge of protecting personal data, so automatically removing the cars entirely would ensure a higher level o security regarding this problem. This project brings a solution that identifies cars in pictures, by generating a mask with identical dimensions as the analyzed picture. Using this mask, applications can apply filters to crop out the cars, blur the cars, inpaint the removed cars, etc.

1 INTRODUCTION

This chapter will get the reader familiar with the purpose of this paper.

1.1 Context

In a world that runs on big amounts of data, it became necessary to create algorithms that can automatically process this data, replacing manual human work. As hardware evolved with time, new algorithms could be developed, algorithms that were thought to be inaccessible in the past. This is the case with machine learning algorithms which were introduced before 1950. Researchers started to solve problems using machine learning and for the next 20 years new methods were introduced, for example, Bayesian methods were developed for probabilistic interface. But in the 1970's there was no considerable improvements in this field, period which is called "AI winter", because people did not feel that these methods were effective.

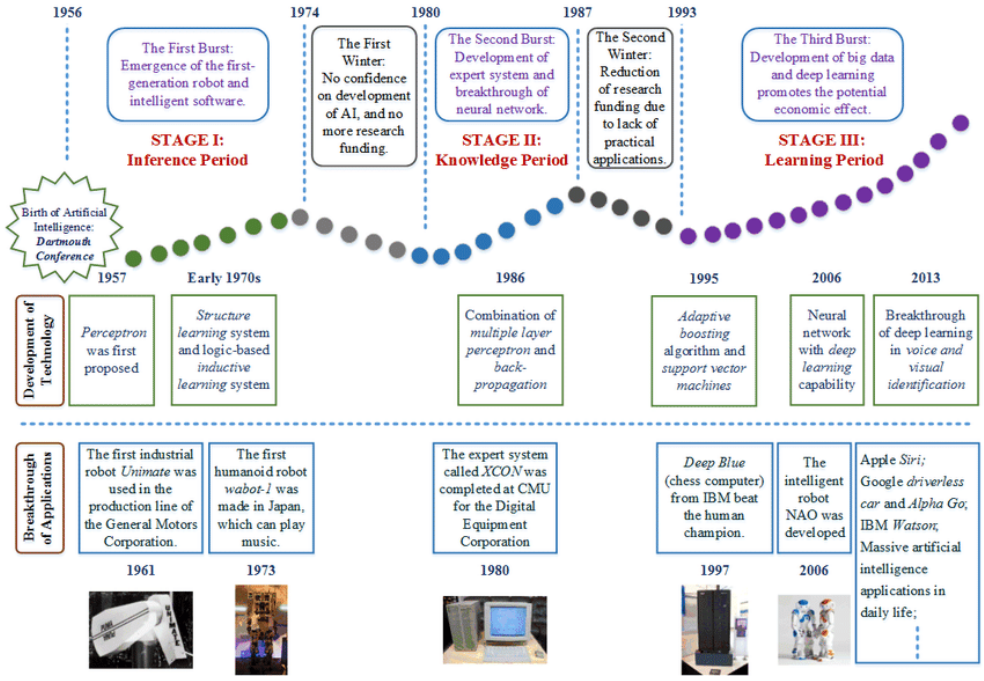

Figure 1: AI history

In the 1980's backpropagation revived the machine learning research, starting the epoch of

"learning" algorithms. This new type of algorithms would use large amounts of data to extract general features about different inputs. After the year 2000, solutions like Support Vector Machine using kernels could resolve problems better than older algorithms used before, but the big breakthrough came after 2010 when deep learning started to be more and more accessible due to new hardware capable of computing faster than ever and libraries capable of using GPUs for matrix operations.

1.2 Problem

Companies are using a lot of pictures that are given to the public for different reasons. For decades these companies struggled to keep private things private, so they started blurring elements in images, using human resources that would go over these images by hand and edit them accordingly. The problem is that the human resource is both expensive and slow. One example of the companies trying to edit images in order to keep private data away from the public is Google. Since 2007 when it added a new feature in it is maps application, called "Google Street View", the company worked intensely to protect private information from being released in public, blurring faces of persons and car number plates. Nowadays, new concerns arose about cars appearing in their photos. So a new algorithm is needed that can detect cars in pictures.

Figure 2: Google street view on "Splaiul Independentei" street[1]

We can see in Figure 2 an example of an image in "Google street view" which has cars number plates blurred.

[1]© https://www.google.com/maps/

1.3 Objective

The project's objective is to develop a way to recognize and mark cars in pictures. In order to achieve that we need to develop an algorithm that can generate a mask based on a given image. it is primary role will be to encode every pixel with a class number, "car" class would have it is own encoding, in order for an application to know which of the pixels in the image is part of a car.

After the mask is generated it can be used to automatically edit the picture by removing or blurring the cars in the picture. A more complex feature could be trying to "inpant" the removed parts to give the impression that there were no cars in the picture from the beginning.

Figure 3: Original image and car mask

Figure 3 shows an example of an image and a mask encoding only the cars in that picture. The application will then use that mask to remove cars from the picture.

1.4 Proposed solution

In order to solve this problem, I opted for a well-known approach regarding pictures: Convolutional neural network. Convolutional neural networks became very popular after libraries like CuDNN from Nvidia were released and computations on big matrices were resolved by GPUs, significantly reducing both training time and predict time.

The application is based on a convolutional neural network which is capable to generate a

mask based on a picture uploaded in the application through a user-friendly interface, which will differentiate cars from other elements.

1.5 Results

The last version of the algorithm reached a top accuracy of 97.34% on the validation dataset. These results were achieved after remodeling the dataset, living only 3 classes (background, cars and people). The application is using only the cars code to remove them out of the picture, as can be seen in Figure 4.

Figure 4: Original image and edited image with cars removed[2]

The network was able to recognize and wrap the cars in the foreground with very high accuracy, but it can be seen that the cars in the background were not even detected.

[2]© https://www.google.com/maps/

1.6 The structure of the paper

This paper contains eight chapters that describe the steps made to develop this project, the resources and the technologies used.

- The first chapter, **Introduction**, provides a quick overview of the paper.
- The chapter **Motivation** provides a reason for using these solution into the real world. It represents a problem that needs to be solved in a better way than the ones that already exist.
- The third chapter, **State of the art**, overviews current methods used for similar problems that work as an inspiration and guidance in developing the new solution. Four architectures are described in this chapter, one representing the foundation of this project.
- **Proposed solution** details the Resnet50 encoder used in combination with SegNet decoder to form the network used in predicting masks describing classes for every pixel in images given as input. A demo application is also presented for better and quicker visualization of the solution.
- The fifth chapter, **Architecture**, describes the components used to build the encoders and the decoders, as well as the components used for building the demo application.
- **Technologies used** offers support for researchers interested in applying this solution into their work by listing the resources used.
- In the **Evaluation** chapter steps taken into developing these solutions are presented, each one with the results achieved. The datasets are also defined here, due to their big impact on the behavior and results of the networks tried.
- The **Conclusions and further work** chapter offers different paths that can be taken in order to improve the results already achieved and also new features that can be implemented.

2 MOTIVATION

As stated before, more and more concerns about the private data of individuals are rising in society. As this concerns evolve, they can become laws, as was the case with General Data Protection Regulation (GDPR)[1]in Europe. So it is in the best interest of the companies affected by these laws to take measures in order to remain in legal terms and also to ensure their clients that their business does not break any moral laws.

Developing networks that can locate elements in pictures becomes crucial in the battle of processing private data, because the networks that can recognize elements like cars can be modified to recognize other elements, depending on the business.

More specifically, the solution that I presented fits perfectly in Google's "Street View" feature, removing cars from pictures that can be accessed by anybody that has a connection to the internet.

[1]© https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr/index_en.htm

3 STATE OF THE ART

This chapter will present some state of the art solutions used nowadays in the world and a comparison between them and the solution that this paper proposes.

The review made by Arthur Ouaknine [4] and the one made by Sik-Ho Tsang [8] were great inspirations for this chapter.

3.1 Fully Convolutional Network (FCN)

Fully Convolutional Networks [9] were introduced in 2015 by Jonathan Long, Evan Shelhamer and Trevor Darrell with the purpose of training deep learning networks to perform image segmentation. Their architecture contains only convolutional layers.

The network was inspired by other known networks at that time that were constructed with fully connected layers, for example, AlexNet [6], VGG16 [3] and GoogLeNet [7], and the authors replaced those layers with convolutional layers. Another modification is the possibility to give non-fixed size input to the network.

The output given by this convolutions represented small feature maps that have to be put together in order to generate a mask with the same size as the input. This was is by a convolutional layer with a stride inferior to 1, which is called "deconvolution".

The results were 62.2% mIoU score on the 2012 PASCAL VOC with pre-trained models on the 2012 ImageNet dataset.

3.2 U-Net

U-net [5] was also introduced in 2015, extending FCN by Olaf Ronneberger, Philipp Fischer, and Thomas Brox. The name comes from the form of the architecture that resembles a 'U',

[1] © https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

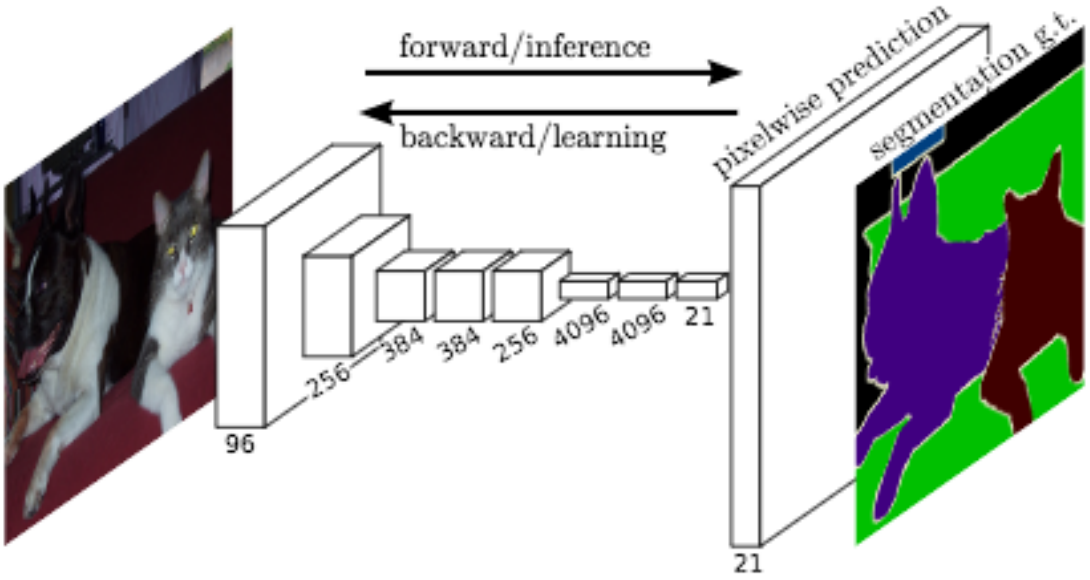

Figure 5: Architecture of the FCN. Note that the skip connections are not drawn here.[1]

as can be seen in Figure 6.

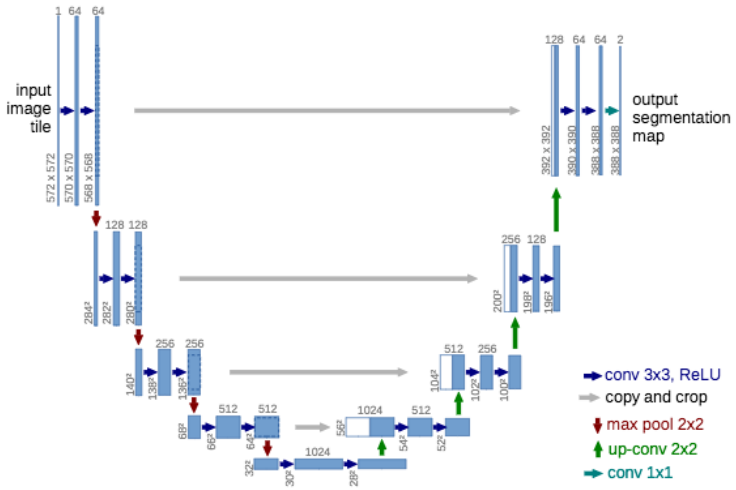

Figure 6: Architecture of the U-net.The blue boxes represent feature maps blocks and the white boxes represent copied and cropped feature maps. [2]

The purpose of this network was to segment biological microscopy images. The network is made with two parts:

- The downsampling part extracts features with an architecture very similar to FCN, using 3x3 convolutions, slowly increasing the number of channels but reducing their size.

[2]© https://arxiv.org/abs/1505.04597

- The upsampling part uses up-convolutions (also called "deconvolution") slowly reducing the number of feature maps while increasing their size. Also, cropped feature maps form the first part of the network are copied as input to upsampling layers to avoid losing pattern information. The final step is a 1x1 convolution which assigns each pixel a class and outputs the mask generated.

Given the fact that no fully connected layers are used, there are not many parameters to be trained, consequently, small number of examples are needed in the training dataset. The authors of the architecture used only 30 images for training in their experiments.

3.3 SegNet

SegNet [1] was introduced in 2016 by Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. This architecture is also composed of two parts: encoder and decoder.

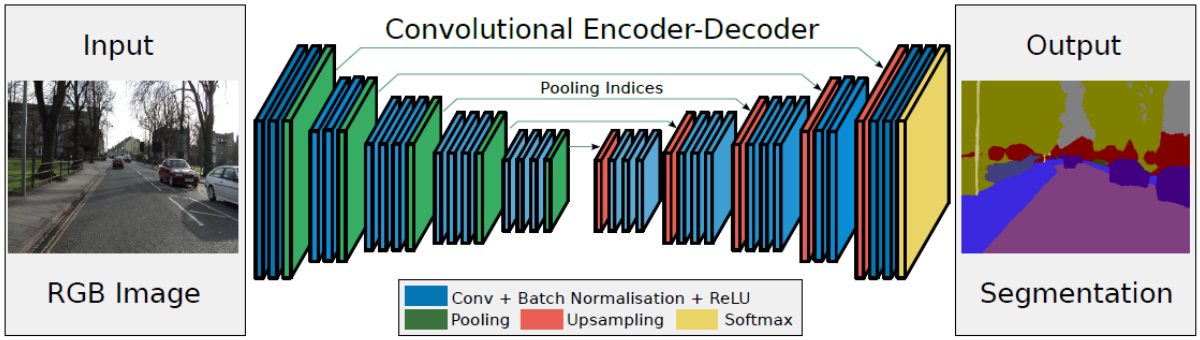

Figure 7: SegNet architecture[3]

The encoders are usually pre-trained networks like VGG-16 [3] and ResNet50 [**?**] have the fully connected layers replaced by decoders. This also means that the network has fewer parameters than the other networks which usefully connected layers to upscale the features map's size.

3.4 Mask R-CNN

Mask R-CNN [2] was introduced in 2017 by Kaiming He, Georgia Gkioxari, Piotr Dollar and Ross Girshick. This network is a bit different, the biggest difference being that it outputs 3

[3]© https://arxiv.org/pdf/1511.00561.pdf

branches: one with bounding boxes, the second one with the class and the third one a binary mask to segment the object.

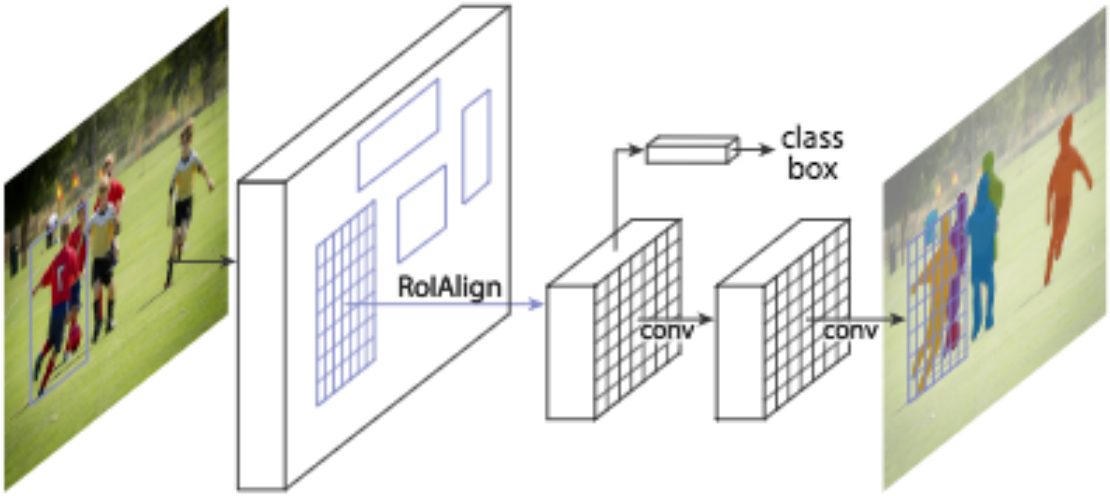

Figure 8: Mask R-CNN architecture[4]

The way it works is the input is firstly processed by a Region Proposal Network (RPN) to propose bounding box candidates that will be processed next. These regions of interest (RoI) are then feed to a RoIAlign layer that computes features from this region that become inputs for the three branches. The branch which outputs the mask is an FCN layer. Mask R-CNN was based on Faster R-CNN, adding output layers.

The results where a 37.1% AP (Average Precision) score on the 2016 COCO segmentation challenge and a 41.8% AP score on the 2017 COCO segmentation challenge.

[4]© http://openaccess.thecvf.com/content_ICCV_2017/papers/He_Mask_R-CNN_ICCV_2017_paper.pdf

4 PROPOSED SOLUTION

I implemented a SegNet architecture which is composed of two main parts: encoder and decoder. This can be seen in Figure 9.

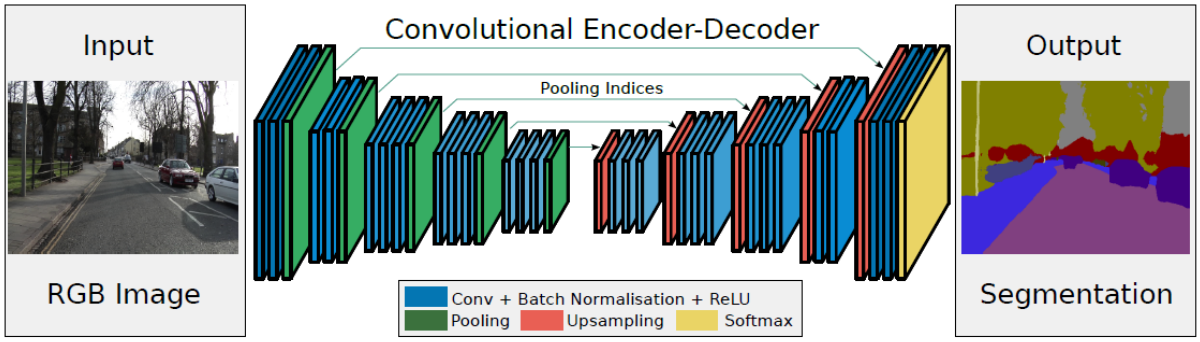

Figure 9: SegNet architecture[1]

The encoder is based on convolution layers and pooling layers, whose role is to extract features from the input, represented by an array of pixels from the image which has to be processed. The first layers extract more general features, for example, horizontal, vertical and diagonal vertices from the input, and as we go deeper in the network layers start recognizing car parts. When the decoder outputs the result, the output is represented by channels of smaller dimensions than the original image.

The decoder gets the output from the encoder and does the upscaling by using "UpSampling" layers, which are the opposite of pooling layers, as many times as it is necessary to output a mask with the desired dimensions. Given the fact that I used Max-Pooling in the encoder, the decoder uses the Max-Pooling indices to restore the information back in the mask. Figure 10 gives an example of upscaling a matrix, using indices of the max values extracted by the Max-Pooling layers.

In order to get better results, I also implemented the encoder as a ResNet architecture. The ResNet architectures, which is based on Residual Networks, reuses inputs from the previous

[1]© https://arxiv.org/pdf/1511.00561.pdf
[2]© https://towardsdatascience.com/review-segnet-semantic-segmentation-e66f2e30fb96

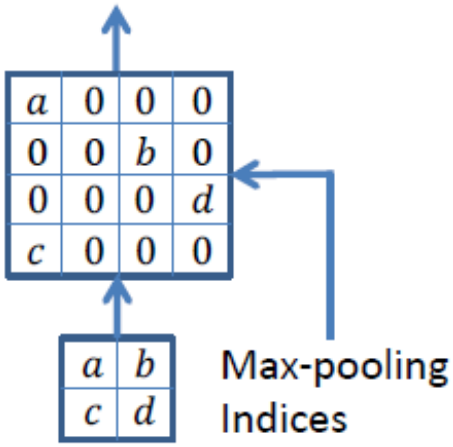

Figure 10: Up-Pooling using Max-Pooling indices[2]

layers in layers which are deeper in the network, in order to minimize the chance that some information is lost by a block in the network, which would create a "bottleneck" for the network.

The network was designed to work with images in 960x448px format as input and to generate masks with the same size. So, when given different size images, they are preprocessed and resized to the desired dimensions of 960x448px.

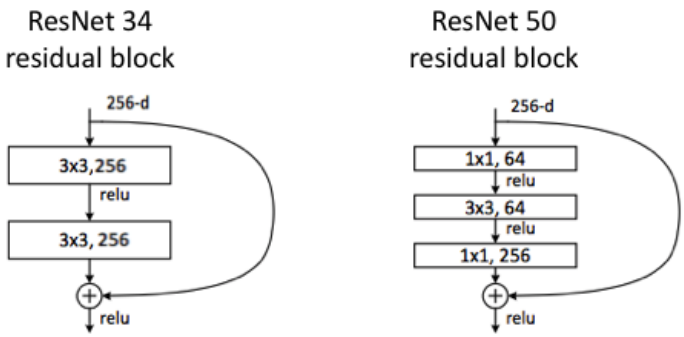

Figure 11: ResNet architecture[3]

[3]© https://www.jeremyjordan.me/convnet-architectures/

5 ARCHITECTURE

The flow of the entire application can be seen in Figure 12. The two main components, the **network** and the **demo application** (GUI), will be described next.

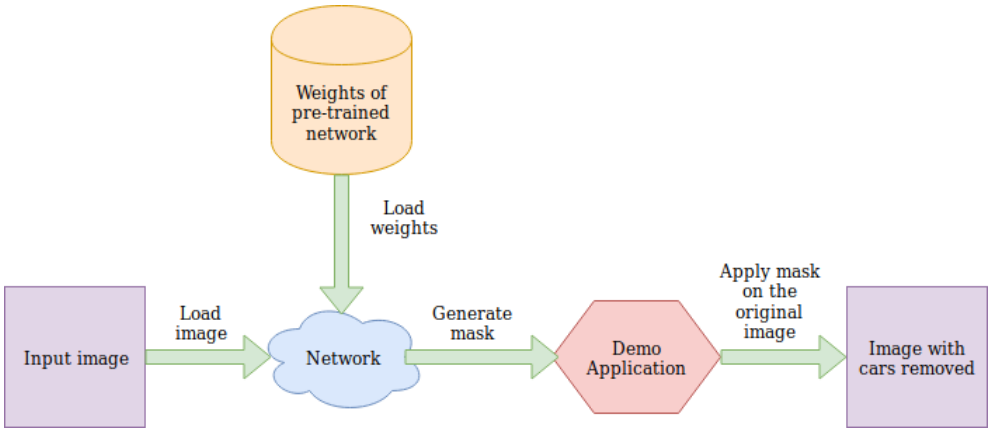

Figure 12: Application's diagram

5.1 Network's architecture

The final version of the demo application uses a ResNet50 encoder with a SegNet decoder.

- ResNet50 encoder

The encoder uses two kinds of blocks - Conv_blocks and Identity_blocks.

The Conv_block is made up of two branches, one being the "shortcut" whose role is to provide to the next layer the input as unprocessed as it can, only applying one convolution to match the output size. The other branch is designed to extract features, starting with the convolution to reduce the size of the output, the same as the first branch, and continuing with two other convolutions with 1x1 filters. BatchNormalization and Activation (relu) layers are used between the conv layers. An illustration of the Conv_block can be seen in Figure 13.

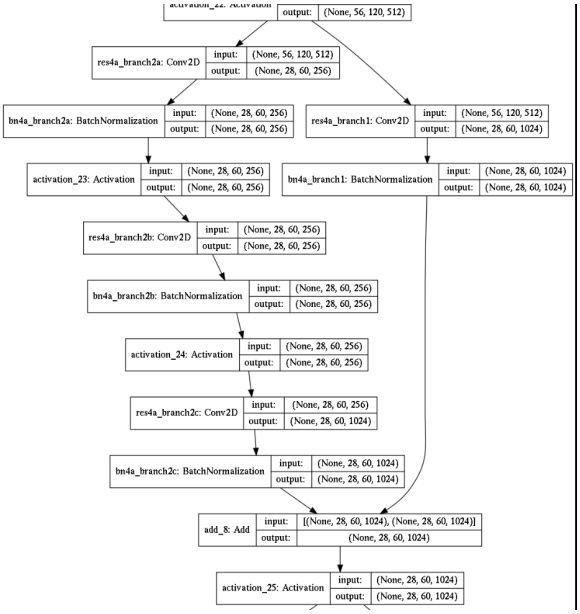

Figure 13: ResNet50 Conv_block

The Identity_block is also made up of two branches, but the "shortcut" uses the input without applying any layer on it. The first branch processes the input without modifying the size, but modifying the channels number decreasing it at first and then increasing it back to the original value. An illustration of the Conv_block can be seen in Figure 14.

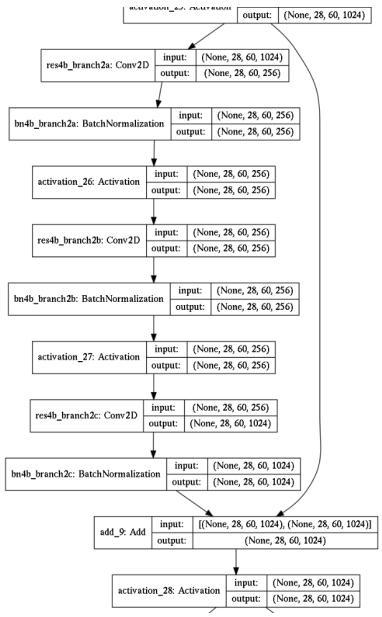

Figure 14: ResNet50 Identity_block

Using the blocks together we can see in Table 2 the architecture of the encoder. It can be observed that the size slowly decreases from 488x960 to 28x60 as the number of channels

slowly increases from 64 to 1024.

- SegNet decoder

The SegNet decoder uses the features extracted by the encoder and generates a bitmap by slowly increasing the size for 28x60 to 488x960 and slowly decreasing the number of channels from 1024 to 64

5.2 Demo application's architecture

I downloaded code from [link] and kept the button "Open File" and it is functionality and removed the other buttons in order to add my own 'Blur cars' button.

The way the application works is this.

- 1. The user opens the application. An interface with two buttons, Figure 15, gives the user the option to open an image.

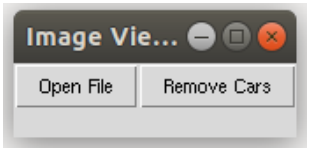

Figure 15: App's home page

- 2. The user clicks the 'Open File' button and a dialog appears on the screen, Figure 16.
- 3. The user chooses a picture and the picture is loaded in the application, Figure 17. The picture's size is saved for further use.
- 4. The user clicks on the 'Remove cars' button and the application calls the predict method of the model selected. The weights are loaded from the given location and the model outputs a 960x448px mask. The mask is resized using the saved size of the image opened by the user, so a pixel by pixel comparison between the original image and the mask can be performed. For each value that represents a car in the mask, the associated pixel in the original image is removed (it's value is set to 0). In the end, the edited image is saved locally as 'removed_cars.png' and opened automatically by the application to replace the original image, Figure 18

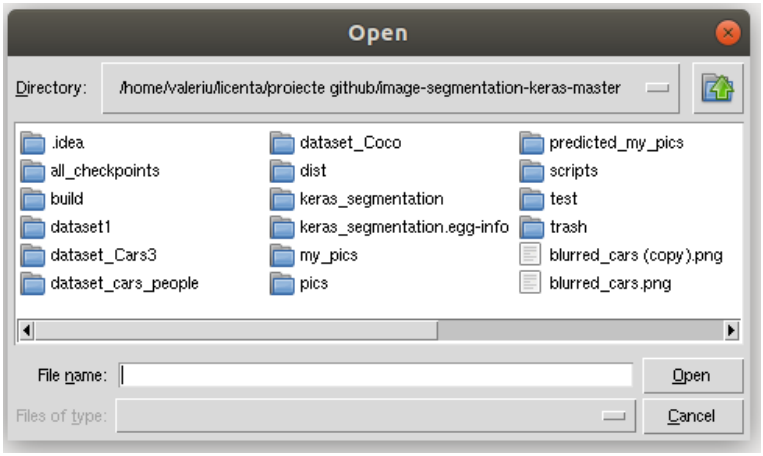

Figure 16: App's open file dialog

Figure 17: App's opened image

Figure 18: App's removed cars

6 TECHNOLOGIES USED

The experiments were run on two environments - local and online.

- 1. Local.

The local environment represents my personal computer. it is CPU is an Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz and the GPU is an Nvidia GeForce GTX 950M with 4GB GDDR5. More information can be seen in Figures 19 and 20.

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 94
Model name:            Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
Stepping:              3
CPU MHz:               800.006
CPU max MHz:           3500,0000
CPU min MHz:           800,0000
BogoMIPS:              5184.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              6144K
NUMA node0 CPU(s):     0-7
```

Figure 19: CPU local

This environment was used to create the analysis in Figure 30 which represents a comparison between the behavior of different architectures after 5 epochs on dataset1 (which will be described later).

it is limitations include the small number of dedicated RAM in the graphics card, which does not allow allocations of big chunks of memory for the networks, consequently, the input image size and the mask size were too small, losing information and generating too general masks.

For the final results, the online environment was used to train the network.

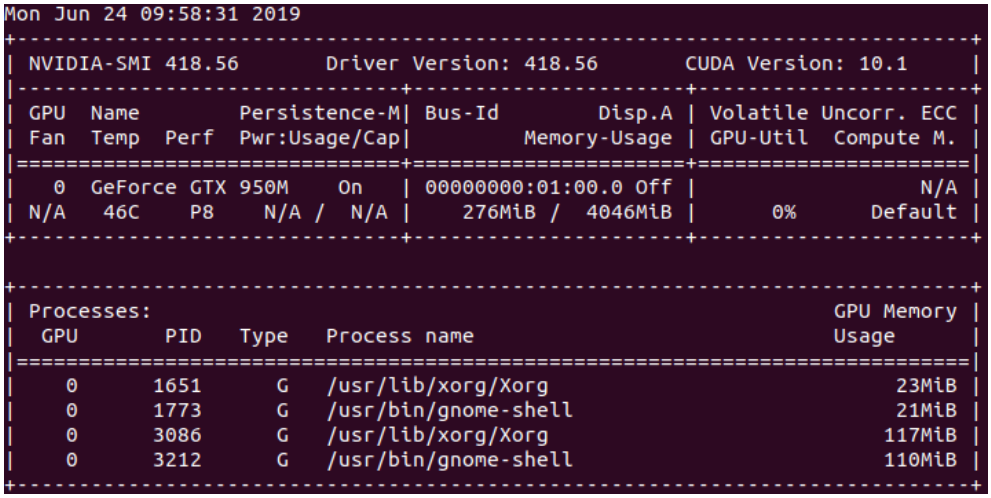

Figure 20: GPU local

- 2. Online.

The online environment represents virtual machines on Google's colab platform. These virtual machines are equipped with Intel(R) Xeon(R) CPU @ 2.30GHz as CPUs and Nvidia Tesla K80 with 12GB GDDR5 as GPUs. More information can be seen in Figures 21 and 22.

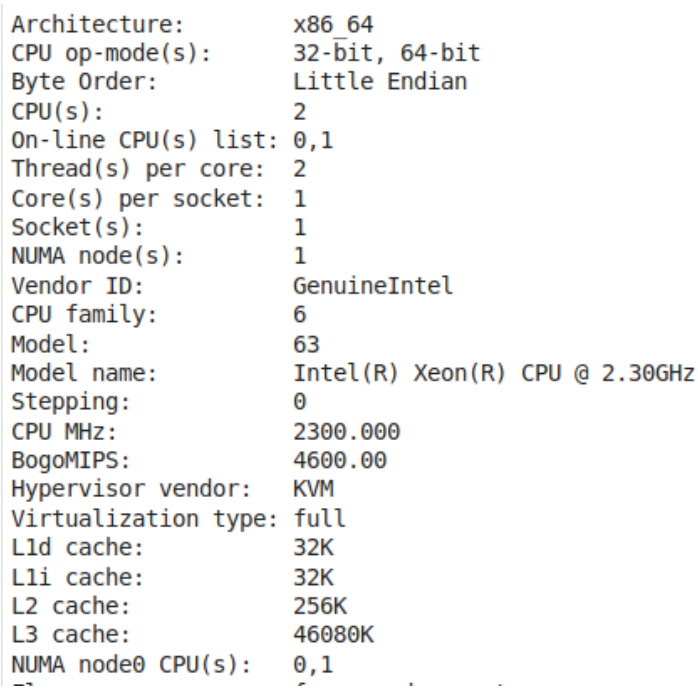

Figure 21: CPU colab

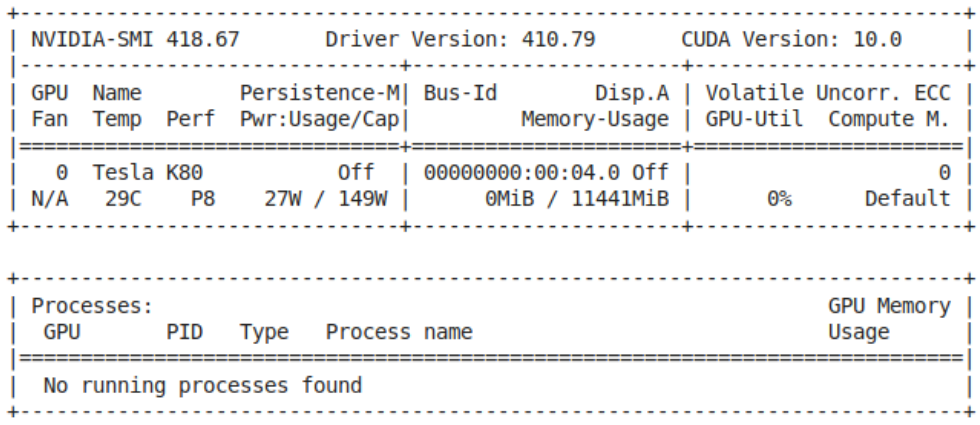

Figure 22: GPU colab

This environment was used to train the networks which had bigger input size and output size, including the final network used in the application.

In order to train networks and to implement an interface, the following libraries were used:

- numpy.
- h5py.
- tqdm.
- tensorflow.
- keras.
- matplotlib.
- six.
- os.
- json.
- argparse.
- cv2.
- PIL.
- tkinter.

7 EVALUATION

The evaluation depends both on the datasets and the metrics used.

7.1 Datasets

Thorughout my experiments I used three datasets, each of them divided into a training set and a validation set. The accuracies invoked in this project involve evaluations over the validation sets for models trained on the training sets.

- Dataset 1

Dataset 1 contains dash view camera images taken in traffic in a city. The format of the images is 480x360px in RGB, as seen in Figure 23.

Figure 23: Images from dataset1

The training set totals 367 images and the validation set 101 images.

The annotations sets contain bitmaps with 10 classes, cars being coded as '8' and persons as '9', the format being 480x360px. Figure 24 shows an example of an image and a mask.

Figure 24: Dataset1, image and mask (each bit in the mask multiplied by 20)

- Dataset 2

Dataset 2 contains dash view camera images taken in traffic both in a city and outside the city. The format of the images is 1242x375px in RGB, as seen in Figure 25.

Figure 25: Images from dataset2

The training set totals 151 images and the validation set 50 images.

The annotations datasets contain bitmaps with 30 classes, cars being coded as '26' and persons as '24', the format being 1242x375px. Figure 28 shows an example of an image and a mask.

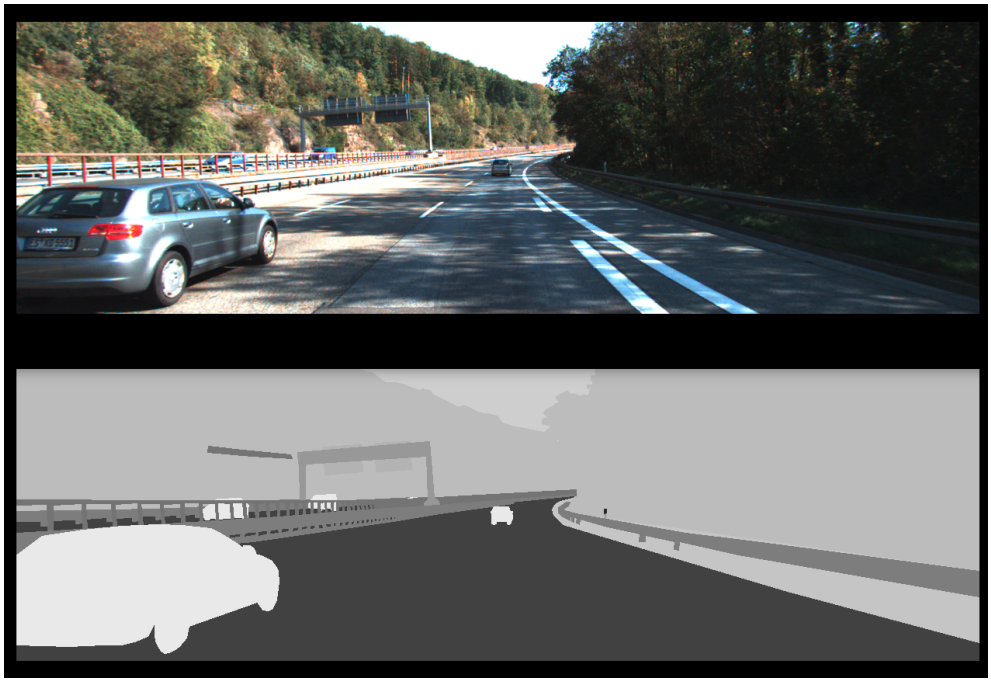

Figure 26: Dataset2, image and mask (each bit in the mask multiplied by 9)

- Dataset 3

Dataset 3 contains dash view camera images from both dataset 1 and dataset 2 by creating new bitmaps for every image. The code that I used for such a modification can be seen in Figure 27 . The two formats of the images are 1242x375px and 480x360px in RGB.

The training set totals 557 images and the validation set 110 images.

The annotations sets contain bitmaps with 3 classes, cars being coded as '1' and persons as '2', the format being both 1242x375px and 480x360px (accordingly to the original image). Figure 28 shows an example of an image and a mask.

```python
import os
import cv2
import numpy as np

CAR_CODE = 26
PEOPLE_CODE = 24

CODE1 = 1
CODE2 = 2

FOLDER_NAME = "semantic"
NEW_FOLDER_NAME = "semantic_cars_people"

for filename in os.listdir(os.getcwd() + "/" + FOLDER_NAME):
    print(filename)
    image = cv2.imread(FOLDER_NAME + "/" + filename)
    image = image.astype(np.float32)
    image[:, :, 0] = ((image[:, :, 0] == CAR_CODE) * (CODE1)).astype('uint8')
                   + ((image[:, :, 0] == PEOPLE_CODE) * (CODE2)).astype('uint8')
    image[:, :, 1] = ((image[:, :, 1] == CAR_CODE) * (CODE1)).astype('uint8')
                   + ((image[:, :, 1] == PEOPLE_CODE) * (CODE2)).astype('uint8')
    image[:, :, 2] = ((image[:, :, 2] == CAR_CODE) * (CODE1)).astype('uint8')
                   + ((image[:, :, 2] == PEOPLE_CODE) * (CODE2)).astype('uint8')
    cv2.imwrite(NEW_FOLDER_NAME + "/" + filename, image)
```

Figure 27: Code to generate dataset3

7.2 Results

I started this project with an analysis of different architectures. I included in the experiments different combinations of encoders and decoders. These are the options chosen for experiments:

- Pre-trained: Yes, on the ImageNet dataset.
- Loss function: categorical cross entropy
- Dataset: dash camera pictures with 10 classes annotated (dataset1)
- Number of epochs: 5
- input format: 640x320px
- output format: 320x160px
- Hardware: (local)

  CPU: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

  GPU: GeForce GTX 950M, 4GB GDDR5

The results can be seen in Figure 29.

After these experiments, I decided to work with the resnet_50 encoder and segnet architecture. In Figure 30 we can see the results of prediction for a random picture (not in the training set or validation set).

Figure 28: Dataset3, image and mask (each bit in the mask multiplied by 20)

Firstly, I tried to replace simple Conv2D layers with SeparableConv2D but results were not better, the validation accuracy dropped to 0.74 from 0.85, so I returned to Conv2D layers. I observed that the mask is pixelated and I checked the model. I realized that the mask was half the size of the original input on both axes. So I decided to add another Up-Sampling block (from 3 to 4) in order to generate a mask with the same size as the input image. Another optimization that I thought would increase performance would be doubling the input size, so the training images would not be down-scaled losing information and also the mask would be twice as big. But my hardware was not able to run bigger networks because it ran out of memory when trying to allocate matrices.

So I uploaded my project on Google colab. But both of my ideas could not be implemented together, because the network would become so big that even 12GB of GPU memory available on colab could not run it. Then I decided to check the performance impact on the results implementing both of the features separately and see which one would give better results. Because the dataset1 had images with 480x360px format I downloaded another dataset with 1242x375px pictures, so doubling the input size would have more information to work with.

Firstly I ran the network unchanged on the new dataset:

- Pre-trained: Yes, on the ImageNet dataset.

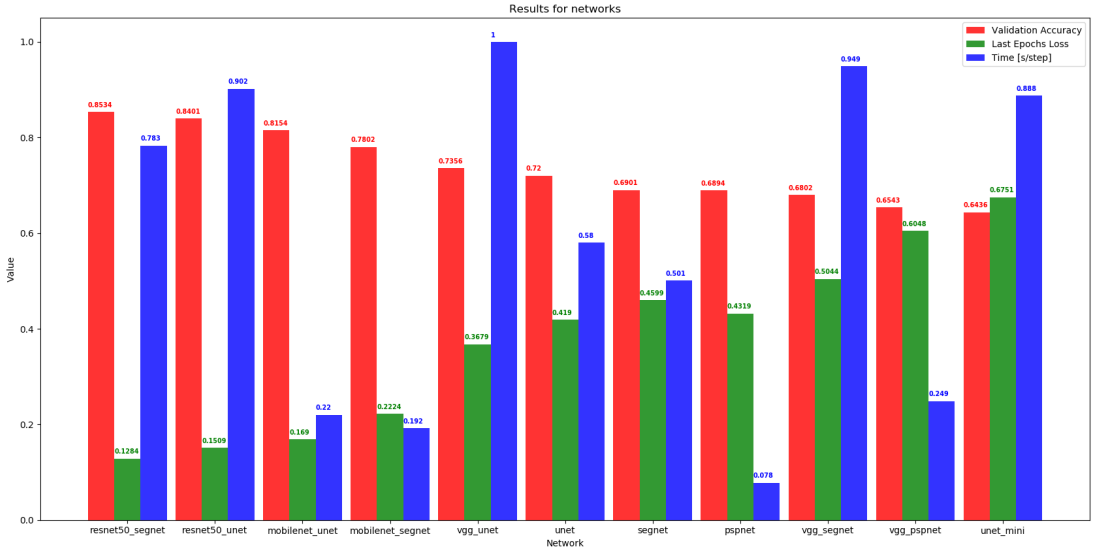

Figure 29: Analysis results

Figure 30: Removed cars, Resnet-50, Segnet, trained on dataset1

- Loss function: categorical cross entropy
- Dataset: dash camera pictures with 30 classes annotated (dataset2)
- Number of epochs: 10
- input format: 640x320px
- output format: 320x160px
- Hardware: (online)

  Intel(R) Xeon(R) CPU @ 2.30GHz

  GPU: Tesla K80, 12GB GDDR5

After 10 epochs the accuracy on the validation dataset was 0.8323.

Then I doubled the input size:

- Pre-trained: Yes, on the ImageNet dataset.
- Loss function: categorical cross entropy
- Dataset: dash camera pictures with 10 classes annotated (dataset2)
- Number of epochs: 10
- input format: 1280x640px
- input format: 640x320px
- Hardware: (online)

  Intel(R) Xeon(R) CPU @ 2.30GHz

  GPU: Tesla K80, 12GB GDDR5

After 10 epochs the validation accuracy was 0.8364. Not a big improvement, too small to be taken into account.

Then I added a new Up-Sampling layer to double the output size, add also increased the size of the input as much as the hardware let me to.

- Pre-trained: Yes, on the ImageNet dataset.
- Loss function: categorical cross entropy
- Dataset: dash camera pictures with 10 classes annotated (dataset2)
- Number of epochs: 10
- input format: 960x448px
- output format: 960x448px
- Hardware: (online)

  CPU: Intel(R) Xeon(R) CPU @ 2.30GHz

  GPU: Tesla K80, 12GB GDDR5

After 10 epochs the validation accuracy was 0.8440. Again no improvement, but at least, the mask stopped looking so pixelated, as can be seen in Figure 31. I also observed that increasing the size of the input determined the network to recognize cars that were in the background with a small number of pixels.

My next idea was to train the network on a dataset which has fewer classes. So I created a new dataset by modifying the two datasets that I already had to contain only 3 classes:

- 0 - Background

Figure 31: Removed cars, Resnet-50, Segnet, trained on dataset2, output size = input size

- 1 - Car
- 2 - Person

This is called dataset3.

I ran the network again with these specifications:

- Pre-trained: Yes, on the ImageNet dataset.
- Loss function: categorical cross entropy
- Dataset: dataset3
- Number of epochs: 20
- input format: 960x448px
- output format: 960x448px
- Hardware: (online)
  CPU: Intel(R) Xeon(R) CPU @ 2.30GHz
  GPU: Tesla K80, 12GB GDDR5

After 10 epochs the validation accuracy increased to 0.9571. So having datasets with a smaller number of classes helps the network to focus on specific elements. After 20 epochs the validation accuracy hit 0.9627.

The last step I took was implementing new loss functions. I researched the internet and tryied two custom loss functions:

Dice loss:

```
def dice_loss(y_true, y_pred):

    numerator = 2  tf.reduce_sum(y_true * y_pred)
        denominator = tfreduce_sum(y_true + tf.square(y_pred))

        return - (numerator / (denominator + tf.keras.backend.epsilon()))
```

Standard loss tensor:

```
import keras.backend as K
...
_EPSILON = K.epsilon()
nb_class = 3
def standard_loss_tensor(y_true, y_pred):
y_pred = K.clip(y_pred, _EPSILON, 1.0 - _EPSILON)

loss = 0
for cls in range(0, nb_class):
loss += y_true[:, :, cls] * K.log(y_pred[:, :, cls])

return -loss
```

I trained the network with the 'Standard loss tensor' function:

- Pre-trained: Yes, on the ImageNet dataset.
- Loss function: Standard loss tensor
- Dataset: dataset3
- Number of epochs: 20
- input format: 960x448px
- output format: 960x448px
- Hardware: (online)
    CPU: Intel(R) Xeon(R) CPU @ 2.30GHz

GPU: Tesla K80, 12GB GDDR5

After 10 epochs the validation accuracy reached 0.9692. After 20 epochs it reached a maximum of 0.9747.

I trained the network with the 'dice loss' function:

- Pre-trained: Yes, on the ImageNet dataset.
- Loss function: Dice loss
- Dataset: dataset3
- Number of epochs: 20
- input format: 960x448px
- output format: 960x448px
- Hardware: (online)

  CPU: Intel(R) Xeon(R) CPU @ 2.30GHz

  GPU: Tesla K80, 12GB GDDR5

After 10 epochs the validation accuracy reached 0.9691. After 20 epochs it reached a maximum of 0.9734.

I concluded that both of them would increase the accuracy with 1%.

The final network which is used in the demo application has these specifications:

- Pre-trained: Yes, on the ImageNet dataset.
- Loss function: Dice loss
- Dataset: dataset3
- Number of epochs: 20
- input format: 960x448px
- output format: 960x448px

The plotted loss of the training and validation sets can be seen in Figure 32 and in Figure 33 the accuracy of the training and validation set is the accuracy plotted.

The final result can be seen in Figure 34.

More examples of removed cars can be seen in Figures 35, 36 and 37.

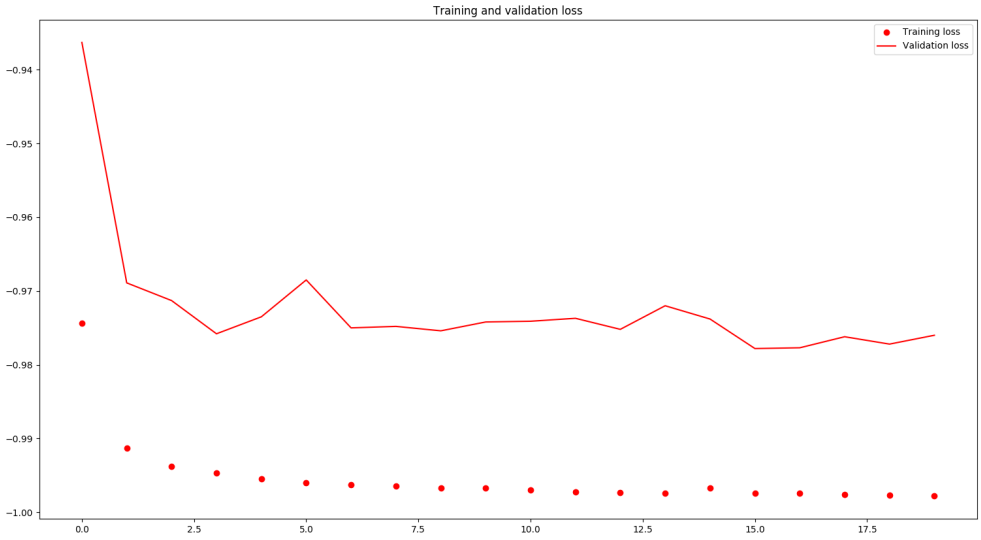

Figure 32: Loss of training and validation sets

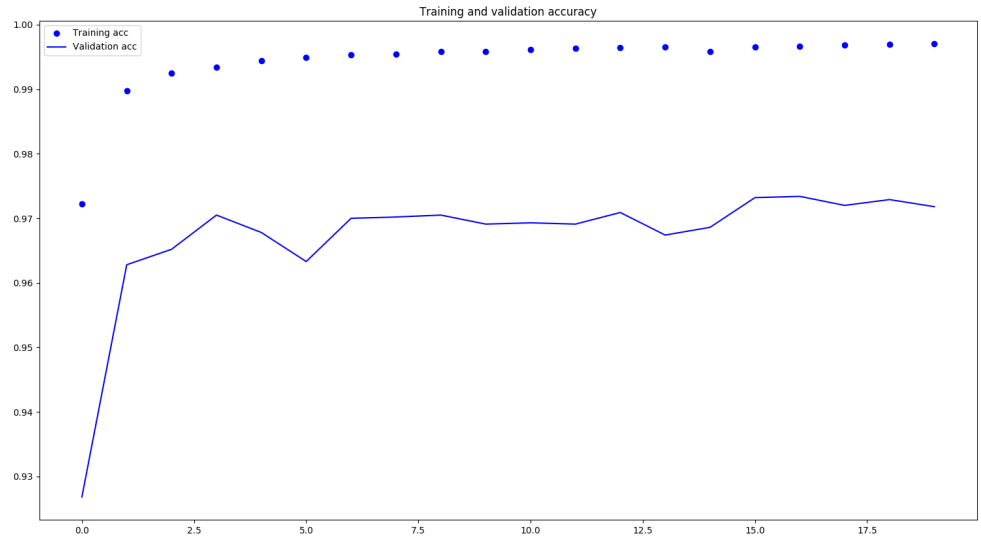

Figure 33: Accuracy of training and validation sets

Figure 34: Removed cars, Resnet-50, Segnet, trained on dataset3, output size = input size, dice loss function

Figure 35: Removed cars, Resnet-50, Segnet, trained on dataset3, output size = input size, dice loss function

Figure 36: Removed cars, Resnet-50, Segnet, trained on dataset3, output size = input size, dice loss function

Figure 37: Removed cars, Resnet-50, Segnet, trained on dataset3, output size = input size, dice loss function

8 CONCLUSIONS AND FURTHER WORK

Although the cars in the background cannot be detected by the network the purpose of keeping private information from the public view is achieved because the dimensions of the cars in question are too small to give out anything particular about a car.

Further work could include training on bigger and more diversified datasets, not only on dash view camera images. This would help the network with overfitting problems.

Another feature that would be worth working on would be an "inpainting" solution that will help to complete the removed areas in the pictures. I looked up on the internet and I found a project on github which offers a GUI which allows the user to upload a picture and "inpaint" desired areas in the pictures, as can be seen in Figures 38 and 39.

Figure 38: Inpaint using areas drawn by the user

Replacing the mask that is drawn by the user with the mask that is generated by the network would fit perfectly in the project achieving an automated solution that is able to recognize cars in pictures and remove them and then reconstruct the image, leaving no trace of the

Figure 39: Inpaint using areas drawn by the user

cars.

Figure 39 reveals that, although the second network works well with cars that do not occupy a lot of the original image and give enough context around them for the network to work with, the cars that are near the camera that takes the picture, consequently, occupying a lot of space in the image, are harder to replace leaving spaces that can be recognized as being processed.

BIBLIOGRAPHY

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[2] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[3] Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[4] Arthur Ouaknine. Review of deep learning algorithms for image semantic segmentation. `https://medium.com/@arthur_ouaknine/review-of-deep-learning-algorithms-for-image-semantic-segmentation-509a600f7b57`. Last accessed: 25 June 2019.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[6] Ilya Sutskever, Geoffrey E Hinton, and A Krizhevsky. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[8] Sik-Ho Tsang. Review of deep learning algorithms for image semantic segmentation. `https://towardsdatascience.com/`

review-segnet-semantic-segmentation-e66f2e30fb96. Last accessed: 25 June 2019.

[9] Xundong Wu. Fully convolutional networks for semantic segmentation. *Computer Science*, 2015.

A RESNET50 ENCODER

Table 1: ResNet50 Encoder

Layer or Block	**Input size (h, w, c)***	**Output size (h, w, c)***
ZeroPadding2D	(448, 960, 3)	(454, 966, 3)
Conv2D	(454, 966, 3)	(224, 480, 64)
BatchNormalization	(224, 480, 64)	(224, 480, 64)
Activation	(224, 480, 64)	(224, 480, 64)
MaxPooling2D	(224, 480, 64)	(111, 239, 64)
conv_block	(111, 239, 64)	(111, 239, 256)
identity_block	(111, 239, 256)	(111, 239, 256)
identity_block	(111, 239, 256)	(111, 239, 256)
conv_block	(111, 239, 256)	(56, 120, 512)
identity_block	(56, 120, 512)	(56, 120, 512)
identity_block	(56, 120, 512)	(56, 120, 512)
identity_block	(56, 120, 512)	(56, 120, 512)
conv_block	(56, 120, 512)	(28, 60, 1024)
identity_block	(28, 60, 1024)	(28, 60, 1024)
identity_block	(28, 60, 1024)	(28, 60, 1024)
identity_block	(28, 60, 1024)	(28, 60, 1024)
identity_block	(28, 60, 1024)	(28, 60, 1024)
identity_block	(28, 60, 1024)	(28, 60, 1024)

*(h, w, c) = (height, weight, channels)

B SEGNET DECODER

Table 2: SegNet Decoder

Layer or Block	**Input size (h, w, c)***	**Output size (h, w, c)***
ZeroPadding2D	(28, 60, 1024)	(30, 62, 1024)
Conv2D	(30, 62, 1024)	(28, 60, 512)
BatchNormalization	(28, 60, 512)	(28, 60, 512)
UpSampling2D	(28, 60, 512)	(56, 120, 512)
ZeroPadding2D	(56, 120, 512)	(58, 122, 512)
Conv2D	(58, 122, 512)	(56, 120, 256)
BatchNormalization	(56, 120, 256)	(56, 120, 256)
UpSampling2D	(56, 120, 256)	(112, 240, 256)
ZeroPadding2D	(112, 240, 256)	(114, 242, 256)
Conv2D	(114, 242, 256)	(112, 240, 128)
BatchNormalization	(112, 240, 128)	(112, 240, 128)
UpSampling2D	(112, 240, 128)	(224, 480, 128)
ZeroPadding2D	(224, 480, 128)	(226, 482, 128)
Conv2D	(226, 482, 128)	(224, 480, 128)
BatchNormalization	(224, 480, 128)	(224, 480, 128)
UpSampling2D	(224, 480, 128)	(448, 960, 128)
ZeroPadding2D	(448, 960, 128)	(450, 962, 128)
Conv2D	(450, 962, 128)	(448, 960, 64)
BatchNormalization	(448, 960, 64)	(448, 960, 64)
Conv2D	(448, 960, 64)	(448, 960, 3)

*(h, w, c) = (height, weight, channels)