Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Efficient Convolutional Neural Networks Recognizing Driveable Trails

Master's Thesis

2020
Bc. Adrián Matejov

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Efficient Convolutional Neural Networks Recognizing Driveable Trails

Master's Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	Mgr. Pavel Petrovič, PhD.
Consultant:	Mgr. Marek Šuppa

Bratislava, 2020
Bc. Adrián Matejov

72835524

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Adrián Matejov
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Efficient Convolutional Neural Networks Recognizing Driveable Trails

Annotation: Robotour is an annual outdoor delivery contest for autonomous robots. Robots navigate the trails located in a natural park to reach loading, unloading and servicing areas. They are typically equipped with laser range sensors, GPS, compass, map of the trail network in the park, and some other sensors. Information obtained from these sensors and the map is seldom accurate enough. The system typically depends on the vision system and its ability to recognize driveable surfaces. The aim of this thesis is to investigate various models of convolutional neural networks that have been successfully applied to image segmentation task. Based on such analysis, the student will design, train, and evaluate an original model that could efficiently perform in real-time on the embedded GPU hardware installed in the robot (Jetson TX2). The student is expected to bring into play active learning, if it proofs to be beneficial in this context.

Literature: Evan Shelhamer, Jonathan Long, Trevor Darrell: Fully Convolutional Networks for Semantic Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.39, no.4, p.640-651, April 2017

Keywords: robotour, CNN, path recognition, autonomous driving

Supervisor:	Mgr. Pavel Petrovič, PhD.
Consultant:	Mgr. Marek Šuppa
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	07.12.2018
Approved:	03.03.2020

prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

....
Student

....
Supervisor

72835524

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Adrián Matejov
Študijný program:	informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Efficient Convolutional Neural Networks Recognizing Driveable Trails
Efektívne konvolučné neurónové siete rozpoznávajúce zjazdné chodníčky

Anotácia: Robotour je pravidelná súťaž autonómnych robotov v doručovanú nákladu vo vonkajšom prostredí. Roboty sa pohybujú na chodníčkoch v prírodnom parku s cieľom dosiahnut nakladaciu, vykladaciu a servisnú oblasť. Bežne bývajú vybavené laserovým senzorom na meranie vzdialenosti, GPS, kompasom, mapou siete chodníčkov v parku, a rôznymi inými senzormi. Informácia získavaná z týchto senzorov a z mapy je málokedy dostatočne presná. Systém preto väčšinou závisí na spracovaní obrazu z kamery a svojej schopnosti rozpoznávať zjazdný povrch chodníčkov. Cieľom tejto práce je preskúmať rôzne modely konvolučných neurónových sietí, ktoré boli na úlohu segmentácie obrazu doteraz úspešne použité. Na základe tejto analýzy študent navrhne, natrénuje a vyhodnotí svoj vlastný model, ktorý by mohol efektívne pracovať v reálnom čase na GPU hardvéri, ktorý je v robotovi nainštalovaný (Jetson TX2). Zadanie predpokladá že študent využije metódu active learning, ak sa ukáže v tomto kontexte ako užitočná.

Literatúra: Evan Shelhamer, Jonathan Long, Trevor Darrell: Fully Convolutional Networks for Semantic Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.39, no.4, p.640-651, April 2017.

Kľúčové slová: robotour, konvolučné neurónové siete, rozpoznávanie cesty, autonómna jazda

Vedúci:	Mgr. Pavel Petrovič, PhD.
Konzultant:	Mgr. Marek Šuppa
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	prof. Ing. Igor Farkaš, Dr.
Dátum zadania:	07.12.2018

Dátum schválenia: 03.03.2020

prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

....

študent vedúci práce

Acknowledgement: I would like to thank my supervisor Mgr. Pavel Petrovič, PhD. for leading this thesis, capturing images and labeling both datasets, reviewing this thesis and for the opportunity to participate at the RoboTour contests. A tremendous thanks goes to my both friend and consultant Mgr. Marek Šuppa for helpful advises, consultations, new ideas, thorough thesis review and last but not least, for providing me with his own GPU machine in order to conduct the experiments. Special thanks goes to my closest ones for their continuous support throughout my studies.

Abstract

Our faculty's students annually participate at the competition called RoboTour Outdoor Delivery Contest. The robot, which the faculty has at its disposal, had been designed, built and improved in previous works. One of the rules of this competition strictly forbids the robot from leaving driveable trail. Since the previous approaches have not been accurate enough, we focus on solving the problem of recognizing driveable trails using deep learning, whose application to various areas has achieved remarkable success in recent years. In this work we firstly deal with this problem by utilizing existing convolutional neural network models for semantic segmentation and test them right at the competition in the outdoor environment. These models reach very good results and the robot is able to distinguish between driveable and non-driveable segments more accurately, which contributes to a better navigation. In order to reach higher prediction speed, we minimize the size of these models and reach more than double speedup with prediction of the road. Subsequently, we compare our models with the ones specifically designed for devices with lower computational power. Finally, we examine the possibilities of reducing the number of images needed for training, leading to less effort dedicated to labeling. Our simulations show that by making use of image clustering combined with entropy of prediction it is possible to halve the number of training data at the cost of a very little decrease in accuracy.

Keywords: robotour, CNN, path recognition, autonomous driving

Abstrakt

Študenti našej fakulty sa každoročne zúčastňujú súťaže RoboTour Outdoor Delivery Contest. Robot, ktorého má fakulta k dispozícii, bol navrhnutý, skonštruovaný a vylepšovaný v predošlých prácach. Jedno z pravidiel tejto súťaže prísne zakazuje robotovi opustiť chodníček. Keďže predošlé prístupy neboli dostatočne presné, v tejto práci riešime problém rozpoznávania zjazdných chodníčkov z obrázka pomocou hlbokého učenia, ktorého aplikácia v rôznych oblastiach dosahuje pozoruhodné výsledky. Najskôr využívame už existujúce modely konvolučných neurónových sietí na sémantickú segmentáciu, ktoré testujeme priamo na súťaži vo vonkajšom prostredí. Tieto modely dosahujú veľmi dobré výsledky a robot vie oveľa presnejšie rozlišovať medzi zjazdnými a nezjazdnými segmentami, čo prispieva k lepšej navigácii. Aby sme dosiahli vyššiu rýchlosť predpovedania, zmenšujeme veľkosti týchto modelov, čím dosahujeme viac ako dvojnásobné zrýchlenie pri predpovedaní cesty. Následne naše modely porovnávame s takými, ktoré boli navrhované priamo na zariadenia s nižším výpočtovým výkonom. Nakoniec skúmame možnosti zníženia počtu obrázkov potrebných pri trénovaní, čo vedie k zníženiu času stráveného označovaním obrázkov. Naše simulácie ukazujú, že využitím zoskupovania podobných obrázkov v kombinácii s entropiou predpovedí vieme zredukovať počet trénovacích dát na polovicu za cenu veľmi malého zníženia v presnosti.

Kľúčové slová: robotour, konvolučné neurónové siete, rozpoznávanie cesty, autonómna jazda

Contents

List of Figures

List of Tables

Introduction

Robots are becoming a part of our daily lives nowadays. This includes helping people in their household chores, in the industry area, in the medical facilities or even in delivering packages and thus making our lives easier. The world of robotics has ever since raised enormous number of interesting problems to be solved. Many of them are subject to the current research. There are also many competitions being organized in order to attract more people into the field robotics. One of these competitions is RoboTour Outdoor Delivery Contest. Participants are challenged to build fully autonomous robot with the ability to drive in the outdoor environment through city park while delivering a load. The robots have to deal with various issues such as obstacles, terrain and correct navigation. These problems can be solved by using reliable hardware parts, proper algorithms and math behind everything.

In this work we focus on recognizing driveable path in the images taken from the camera mounted on the robot. This is quite important for the robot since it is not allowed to leave the driveable trail and breaking this rule leads to disqualification. Previous approaches have not been accurate enough due to very high sensitivity to weather conditions and the colors within the image, and because they do not take entire image as a whole into account but rather predict small regions. In recent years, deep learning has proven that many of these issues can be dealt with more precisely than reported previously. Therefore, in order to improve the vision module, we incorporate convolutional neural networks, which are suitable for computer vision applications. In our case, it is a dense semantic segmentation task that needs to be solved. Put differently, it means that, given the input image, we would like to predict which pixels correspond to driveable segments of the environment. Training the network is conducted in a supervised manner. There are Lednice and Deggendorf datasets available and their images had been captured right before the contests in 2018 and 2019. We show that the models outperform technique based on HSV image statistics used at previous contests. Our approach was tested in Deggendorf, Germany where we found out that the prediction time is not sufficient when it comes to running the robot at higher speed. To reduce the prediction time, we minimize the size of the models while keeping the accuracy almost untouched, and compare them with the so called mobile models [1, 2] designed for devices with low computational power. Moreover, we employ active learning technique [3] to reduce the number of images needed to be labeled since labeling entire dataset is a time consuming process and redundant samples which bring little value during training are best ignored.

This work is structured into five chapters. In Chapter 1 we describe semantic segmentation along with several basic techniques and convolutional neural networks. Chapter 2 discusses related work in terms of architecture of successful networks and describes previous work on the robot. In Chapter 3 we present both datasets and compare results of the models. Subsequently, in Chapter 4, we present results of minimized models and comparison to the mobile models. Finally, in Chapter 5 we present results of our active learning experiments.

Chapter 1

Preliminaries

In this chapter we introduce the RoboTour competition, discuss several basic techniques of image segmentation, describe the functionality and inner workings of convolutional neural network (CNN) as well as its impact on image segmentation. The chapter also includes description of layers relevant for CNNs applied to semantic segmentation.

1.1 RoboTour competition

RoboTour Outdoor Delivery Contest is an annual competition for outdoor robots being held in different locations, which are mostly parks. The participants are challenged to build an autonomous robot which completes the task without breaking rules. The robot starts at the position A (also called the base, where it is possible to repair or calibrate the robot) and is given GPS coordinates of position B. Then, the robot must navigate itself through the park to the location B where organizers of the competition load a five liter barrel of beer onto it. The robot is once again given coordinates, but this time of position C where it is supposed to deliver the barrel. After unloading, the final goal is to return back to the base.

The robot is forbidden both to leave the driveable path or to touch any obstacle during the round. If such event occurs, the robot is immediately disqualified from the round. Therefore, it is important for the robot to prevent breaking the rules. Avoiding obstacles is done quite accurately using laser and ultrasonic sensors mounted on the robot. Our robot (see Figure 1.1) is also equipped with camera for local navigation so it tries not to leave the road. In this work we focus on the problem of differentiating between two categories within given image (driveable or non-driveable path) using computer vision technique called convolutional neural networks solving semantic segmentation.

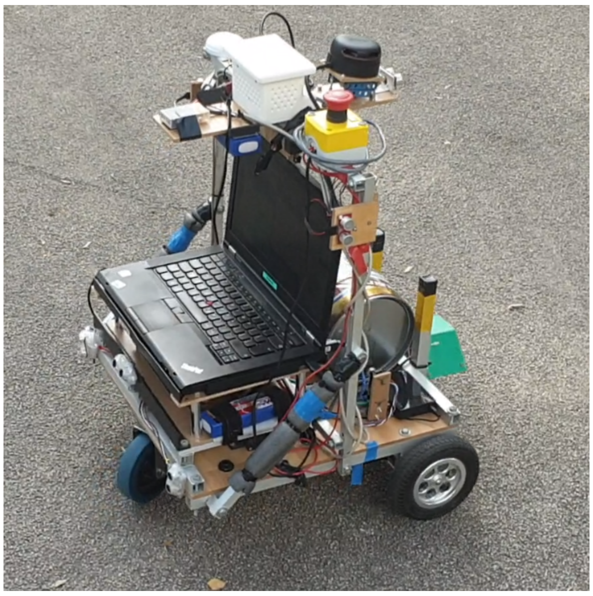


Figure 1.1: The photo of our robot Smelý Zajko, heading to delivery point in Deggendorf.

1.2 Image segmentation

Extracting useful information from images is an inevitable step for image analysis. Image segmentation is a computer vision task of automatic image analysis. Its main task is to partition an image into multiple classes and extract objects or regions of interest from background. Each pixel within this image is assigned a class or a category which it belongs to. Pixels with the same label share certain characteristics. [4]

The history of image segmentation can be traced back to 50 years ago. Since then, many techniques have been developed and evaluated. We are going to describe some of them. One of them is a machine learning technique that utilizes convolutional neural networks (CNNs), which we will describe in the Section 1.4.

1.2.1 Region-based segmentation

Threshold segmentation

Threshold segmentation is one of the most commonly used and simplest techniques. The input image is transformed into grayscale representation. Global and local threshold methods can be used afterwards. The former uses just one global threshold value and divides the image into just two regions - target and background. On the other hand, local threshold method uses multiple threshold values to divide the image into multiple target and background regions.

Simple calculations make this method very fast. If the contrast between background and the target is high enough, the segmentation can be computed quite accurately.

The disadvantage of this method is that it is difficult to obtain accurate results where there is no significant difference in grayscale image, since it does not take the spatial information into account. [5]

Regional growth segmentation

The basic idea of regional growth segmentation method is to merge neighboring pixels (or regions) that share similar properties into one. The first step is to select so called seed pixels. We then grow regions from them, merging pixels if their absolute difference is less or equal to some threshold T. Visiting input pixels is done by breadth-first search.

The main disadvantages of the aforementioned method are that it is computationally expensive local method with no global view and it is quite sensitive to noise .

In the Figure 1.2, there is an example of the algorithm starting with input (leftmost) matrix. The middle matrix is computed using threshold $T = 3$ and the second one using $T = 6$. We can clearly see that the choice of the threshold value is very important step for this algorithm to work properly. [5]

$$\begin{bmatrix} \mathbf{1} & 0 & 4 & 7 & \mathbf{5} \\ 1 & 0 & 5 & 7 & 7 \\ 0 & 1 & 5 & 5 & 5 \\ 2 & 0 & 5 & 6 & 5 \\ 2 & 2 & 5 & 6 & 4 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 5 & 5 & 5 \\ 1 & 1 & 5 & 5 & 5 \\ 1 & 1 & 5 & 5 & 5 \\ 1 & 1 & 5 & 5 & 5 \\ 1 & 1 & 5 & 5 & 5 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 1.2: Example of regional growth segmentation.

1.2.2 Edge detection segmentation

Edges represent the most significant changes in the image. That includes changes in brightness, colors, etc. This method involves detection of abrupt pixel discontinuities and arranges them into edges (for example transition from one color to another). [5]

1.2.3 Clustering-based segmentation

Clustering is widely known as a method of iterating over samples in a feature space and computing the nearest cluster they belong to. After convergence of pixels, we can map them back to the original image to get the segmentation results.

The most frequently used clustering algorithm is K-means. It consists of four steps:

1. Initialize clusters - select random samples to be centroids of the cluster

2. Assign each sample to cluster - calculate the distance from each cluster and assign the sample to the nearest one

3. Recompute the centroid of each cluster by taking the mean of its samples

4. Repeat steps 2 and 3 until none of the samples changes the cluster

The K-means algorithm is fast and easy to implement and scalable to large datasets. Unfortunately, the parameter K (the number of clusters) is difficult to estimate and has to be chosen experimentally. As the authors of [5] mention, it is distance-based partitioning method and therefore it is only applicable to convex datasets.

1.2.4 Histogram-based segmentation

Histogram-based segmentation is a very efficient method in terms of computation since it only requires one pass through the image. During that pass it computes histogram from the pixels. The histogram peaks are then used to locate objects within the image. This method can be improved by recursively applying it to clusters in order to divide them into smaller ones. One disadvantage of histogram-based segmentation is that it might be difficult to identify significant peaks.

1.3 Neural networks

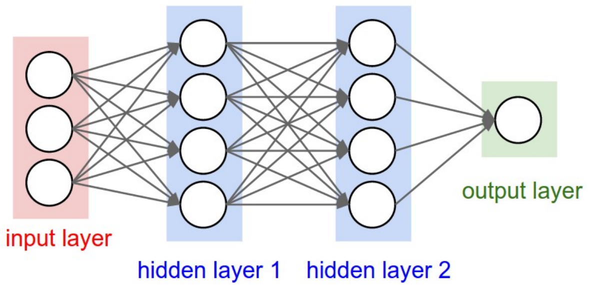


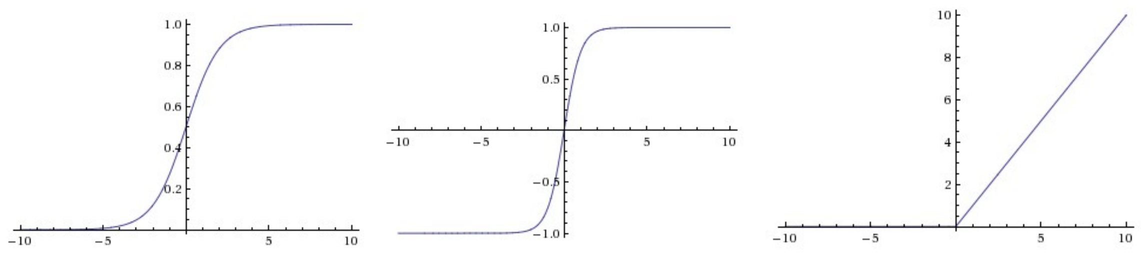
Figure 1.3: Multi-layer perceptron. (Image taken from [6])

The development of neural networks has been primarily inspired by the goal of modeling biological neural systems, but has diverged and become the matter of engineering. These networks have been adopted to a wide range of tasks while beating previous state-of-the-art techniques. The basic unit of the brain is neuron and has been mathematically defined. In the late 1950s, Frak Rosenblatt developed the so called perceptron for classifying objects. It takes a vector $\vec{x}$ as the input, multiplies it with weight vector $\vec{w}$ and sums the resulting product. If this product is above certain threshold, neuron fires and outputs the number 1, otherwise 0. In the most common literature, there is a *bias* term introduced and responsible for shifting the decision boundary. The equation can be written as $\sum_{i=1}^{n} x_i w_i + b$.

Multi-layer perceptron (MLP) is the basic and probably the most widely known feedforward artificial neural network. We can think of MLP as just a mathematical function that maps inputs to output categories. It is formed by layers, which contain several perceptrons accompanied by some nonlinearity (also reffered to as *activation function*), where each of this neurons reads input from all of the neurons in previous layer. The standard MLP consists of *input* and *output* layers, and several layers in between called *hidden* layers, see Figure 1.3. [7] Once we obtain the output from the network, we compute the loss and the weights are then updated using backpropagation, see detailed description in [7]. The loss is a function denoting how much the predicted values differ from the ground truth. Furthermore, it needs to be differentiable so we are able to compute direction in multidimensional space which leads to update of the model's weights that improve our results.

1.3.1 Activation functions

As we mentioned in Section 1.3, neurons are accompanied by an activation function. This is always some nonlinear function, since the linear function would not bring any advantage and we could collapse all the layers into one. Let's have a look at some of the most commonly used nonlinearities presented in Figure 1.4.



(a) Sigmoid (b) Hyperbolic tangent (c) ReLU

Figure 1.4: Nonlinear functions.

Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ takes real value number and squashes it into a range between 0 and 1. Recently, sigmoid has fallen out of favor because it suffers from saturation. If the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero, which essentially stops the learning process. This phenomenon is called *vanishing gradient* in the literature [7].

Tanh $\tanh(x) = 2\sigma(2x) - 1$ (hyperbolic tangent) activation is very similar to the sigmoid. It also saturates, but on the other hand, its output is zero centered. Note, that tanh is just a scaled sigmoid function.

ReLU $\text{relu}(x) = \max(0, x)$ (rectified linear unit), is another nonlinear activation function heavily used nowadays, mostly in computer vision applications. It greatly accelerates the convergence of stochastic gradient descent as found in [8]. If we compare it to sigmoid or tanh activations, ReLU is much easier to compute and can be implemented by thresholding a matrix activations at zero. However, neurons with ReLU activation might die during training because a large gradient could cause the weights to update in such a way that these neurons will never activate and the gradient flowing through them will forever be zero. With the proper setting of the learning rate, however, this is less frequently an issue. One may encounter other variants of ReLU trying to fix this problem, such as *Leaky ReLU* [9] or *Parametric ReLU* [10]. [6]

1.3.2 Learning concepts

There are three major learning concepts covering most of the contemporary machine learning:

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

Supervised learning represents subset of machine learning techniques where the model is provided with data along with ground truth labels. During training, the model makes predictions, compares its results with the correct ones and adjusts parameters to decrease the error.

Unsupervised learning stands for searching for patterns and similarities within dataset, grouping similar ones together and detects anomalous samples. That means there are no ground truth labels at all. Some popular examples of such models utilizing unsupervised learning are *K-means*, *Self-organizing map* etc.

In reinforcement learning, there is an agent exploring some sort of environment being rewarded for actions. Either positively or even negatively. The goal of the agent is to observe the environment and learn underlying model to maximize the reward. [11]

In the convolutional neural network setting, applications usually make use of supervised learning. In this work, our labels are ground truth masks - binary images of the same resolution as input images, but each pixel being marked either as driveable (value 1) or non-driveable (value 0).

1.3.3 Parameter updates

Once we are training a neural network, we are evaluating our loss function. Computing the gradient of this function directs us towards updating the parameters in such a direction that decreases the error computed by the loss function. There are several approaches we can use for performing the update of parameters. The most basic form of update is *Gradient Descent.* After evaluating the loss function, we can compute the negative gradient and update the parameters by the following formula

$$\theta = \theta - \alpha \nabla_\theta C(\theta)$$

where θ denotes the parameters, α denotes learning rate (fixed constant), $C(\theta)$ is the loss function and $\nabla_\theta C(\theta)$ is gradient of the loss function with respect to θ. Updating the parameters once after evaluating the entire dataset becomes very inefficient in machine learning setting, especially when the dataset is very large. The modification of this algorithm performs the update after each training sample is evaluated and is called *Stochastic Gradient Descent* (SGD). However, it is a common practice nowadays to use *minibatch SGD* which essentially takes the average of gradients of small portion of training examples and performs the parameter update accordingly.

In order to converge even faster, *Momentum update* is another approach for deep networks. It helps accelerate SGD in relevant direction and dampens oscillations. The loss can be interpreted as hilly terrain and we push a ball down a hill. The ball accumulates its velocity and is able to pass through slight hills in its way in order not to end up in local minimum. The momentum technique modifies gradient descent by introducing a new variable v representing the velocity and a smoothing constant β, which helps in controlling the value of v. We can use the following formula to compute velocity in step t

$$v_t = \beta v_{t-1} - \alpha \nabla_\theta C(\theta)$$

and the parameters are updated by the velocity $\theta = \theta + v_t$.

Nesterov momentum, based on Nesterov accelerated gradient [12], has gained popularity for its stronger theoretical convergence guaranty for convex functions. Moreover, it is in practice slightly better than standard momentum. Its core idea is to look ahead, so our ball is smarter and knows to slow down before the hill slopes up again. Computing $\theta + \beta v_{t-1}$ gives us an approximation of the next position of parameters. The revised velocity update is

$$v_t = \beta v_{t-1} - \alpha \nabla_\theta C(\theta + \beta v_{t-1})$$

and the parameters update stays the same.

Previously discussed approaches manipulate the learning rate globally and equally for all parameters. A lot of work has been put into methods that can adaptively tune the learning rates, and even do so per parameter. One of the most used method especially for convolutional neural networks is *Adam* (Adaptive Moment Estimation) [13]. The update of the parameters is done by following formula:

$$m_t = \beta_1 m_{t-1}(1 - \beta_1)\nabla_\theta C(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla_\theta C(\theta)^2$$
$$\theta = \theta - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}$$

where β_1 and β_2 denote decay rates. Authors suggest to set them to $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The ϵ prevents from dividing by zero and is set to $\epsilon = 10^{-8}$. The full Adam update also includes a bias correction mechanism, which tries to fix that both m and v are in the beginning biased at zero. With the bias correction mechanism the update looks as follows

$$\hat{m}_t = \frac{1}{1-{\beta_1}^t}$$
$$\hat{v}_t = \frac{1}{1-{\beta_2}^t}$$
$$\theta = \theta - \alpha \frac{\hat{m}_t}{\sqrt{v_t} + \epsilon}$$

There are also many other optimizers available. Sometimes, it is worth trying SGD combined with Nesterov momentum as it might work slightly better in some cases. [6]

1.4 Convolutional neural networks

Convolutional neural networks (CNNs or ConvNets) are a class of deep neural networks. They are quite similar to ordinary neural networks, because they are made up of neurons that have learnable weights and biases. By *ordinary* we mean well known multi-layer perceptron (MLP). CNNs are often applied to computer vision problems like image classification, instance segmentation, semantic segmentation, object detection etc.

The CNN is a sequence of layers. It takes an image as the input, transforms it layer by layer and outputs the final prediction. The CNN is given so called training data, which are used during the training process. Each sample image is fed into CNN and the output is compared to the ground truth and based on the result both loss and accuracy functions are computed.

There should be also validation data available which are often small portion of training data and network should never see them. Their purpose is to see approximation of model's performance in real-world scenario and stop training process in order to prevent overfitting.

We can clearly see from the results published that ConvNets are so powerful that they have overcome classic computer vision techniques known before and are now state-of-the-art in various areas.

Networks for semantic segmentation utilize an encoder network followed by a decoder network. Encoder is usually a pre-trained model. Architectures mostly differ in their choice of the decoder. The job of the encoder is to learn discriminative features at different stages and encode them into high-dimensional feature vector, whilst decoder takes this vector and produces semantic segmentation mask. [14]

Every ConvNet architecture consists of several types of layers. Let's have a deeper look on these building blocks in following subsections.

1.4.1 Fully-connected layer

Fully-connected or FC layer has been adopted from the ordinary multi-layer perceptron. Each neuron is connected to all the neurons from previous layer and fires activations to all the neurons in following layer. Such layers are mostly located at the tail of ConvNet. Features extracted by convolutional layers are fed into fully-connected ones which firstly transform input into a single vector and then decide the class presence.

These layers usually contain most of the network's parameters and thus are a very expensive part in terms of computation and model's size. Most of the modern architectures doing semantic segmentation avoid using fully-connected layers, resulting in more efficient models.

1.4.2 Convolutional layer

This type of layer is a core building block of a CNN. Each one consists of a set of trainable filters with weights. These filters, also called kernels, are usually small spatially (along width and height) but extend through the full depth of input volume. Each slice of depth channel can be referred to as a *depth slice*. The number of filters corresponds to the depth of output volume. For example, filters in the first layer might have size $5 \times 5 \times 3$ (5 pixels wide, 5 pixels high and depth 3, because our input image has 3 color channels - red, green and blue). During the forward pass, we convolve (or slide) filter by filter through the image and compute dot product at each region. Denoting filter's weights $W_{w,h,d}$ and particular image region $R_{w,h,d}$ where w is width, h is height, d is depth and b is a bias, we can compute their dot product by the following formula

$$y_{i,j} = b + \sum_{i=1}^{w} \sum_{j=1}^{h} \sum_{k=1}^{d} w_{i,j,k} \cdot r_{i,j,k}$$

Each filter produces a 2D activation map $Y_{i,j}$. These maps are then stacked along the depth dimension and passed as the input volume for activation. Intuitively, ConvNets learn filters that activate when they see specific feature such as an edge, color and so on. Typically, in standard multi-layer perceptron neural networks, each neuron receives input from all of the neurons from previous layer. In ConvNets, each neuron receives input only from restricted region from input volume. This region is defined by aforementioned filter size, also referred to as *receptive field*.

Authors of ConvNet architectures sometimes make use of 1×1 convolutions. This might sound a bit confusing, but recall that filters extend through the full depth of input volume. Therefore, we are able to control the depth of output volume by creating a layer with several 1×1 filters.

The basic convolutionalization downsamples the image spatially since dot product is not defined for border pixels. This is undesirable behaviour in most cases. To keep width and height dimensions of input and output volumes equal, we have to apply *padding* operation beforehand. The idea is to add enough zeros around the image borders.

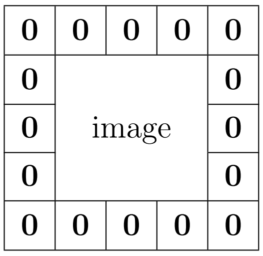


Figure 1.5: Padding image with zeros. Padding with one 0 on each side is usually done before applying 3×3 filter. For 5×5 we pad the image with two 0s and for 7×7 with three 0s.

On the other hand, we might sometimes need to downsample the image via ConvNet layer. The output's volume size can also be controlled by *stride* hyperparameter. Stride is the number of steps we make to shift the filter to next region. In practice, if we want to reduce the output size by convolutional layer, stride is mostly set to 2 (uncommonly 3). When the stride is set to 1, we move the sliding window one pixel at a time.

To summarize the convolutional layer: [6]

- accepts input of size $W_1 \times H_1 \times D_1$
- sets hyperparameters $\boldsymbol{K}$ (number of filters), $\boldsymbol{F}$ (filter's size), $\boldsymbol{S}$ (stride) and $\boldsymbol{P}$ (amount of zero padding)
- produces output of size $W_2 \times H_2 \times D_2$ where:
  - ☐ $W_2 = (W_1 - F + 2P)/S + 1$
  - ☐ $H_2 = (H_1 - F + 2P)/S + 1$
  - ☐ $D_2 = K$

1.4.3 Pooling layer

The common practice in ConvNets architectures is to use *pooling layer* in between convolutional layers. The purpose of the pooling layers is to achieve spatial invariance by reducing the resolution of the feature maps which in the end leads to dramatic reduction of computing power, memory consumption and also control of overfitting.

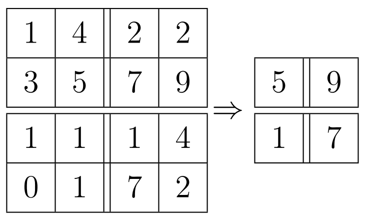


Figure 1.6: Max pooling example.

Each depth slice is processed individually and therefore depth of the volume remains unchanged. Similarly to the convolutional layer, it contains the sliding window, but this time we are not computing dot product. The window takes values from the input volume's region and computes a simple function like *max* (called *max pooling*, see Figure 1.6) or *average* (called *average pooling*). In practice, it has been shown that max pooling performs better than average pooling [15]. Most widely used pooling layer is 2×2 with a stride of 2, which reduces both width and height by a factor of 2. We may sometimes encounter 3×3 pooling layers with a stride of 2 (also called *overlapping*) being used in ConvNet architectures, but in general there is no significant improvement over using non-overlapping ones. [6, 15]

1.4.4 Upsampling layers

The upsampling technique has been mostly used in semantic segmentation architectures. When we input an image to the ConvNet, an encoder takes care of downsampling the image to the very low resolution and classifying all the pixels. The decoder's role is to upsample this output to the original size while preserving as much information as possible. There are more ways to upsample the image. In the following subsections we describe two of them, namely *Unpooling* and *Transposed convolution.*

Unpooling layer

Unpooling is a simple upsampling layer with no trainable parameters. In practice, the pooling operation is non-invertible, but we can approximate this inversion using unpooling layer. For example, many architectures remember indices of maxima within each region when doing max-pooling and reuse them to fill appropriate region. The rest of the grid is usually filled with zeros, see Figure 1.7. The most common unpooling layer with kernel size of 2×2 upsamples input of size $n \times n$ to $2n \times 2n$. [16]

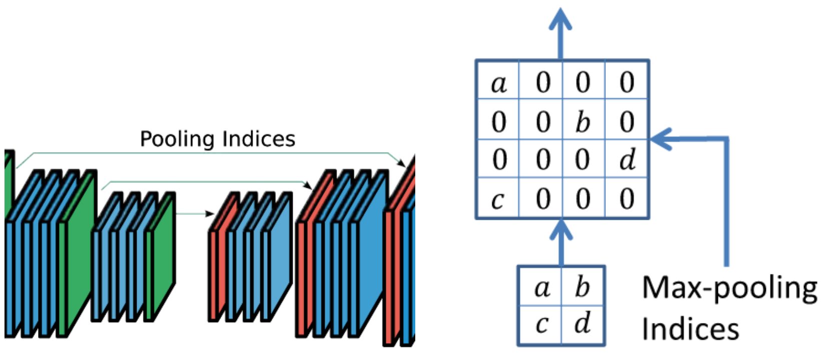


Figure 1.7: Demonstration of how pooling indices are reused in the decoder part. (Image taken from [17])

Transposed convolutional layer

Transposed convolution (also called fractionally-strided convolution) is a layer used for upsampling the given input. Since we do not have to use predefined interpolation method, we let the network train the layer's parameters to upsample optimally.

The core element behind transposed convolution is kernel (or matrix) comprised of trained parameters. It works similarly to ordinary convolutional layer, but in backward direction. Even though it is called transposed convolution, it does not mean that we take some existing convolution matrix and use the transposed version. The association between input and output is handled in backward fashion (one-to-many instead of many-to-one). Values in kernel window are multiplied by scalar from input and the kernel is slid to next region. This process is repeated until there are no more scalars to be used. Overlapping output values are then summed together.

1.4.5 Dilated convolutional layer

Dilated convolutions have been specifically designed for dense prediction where the output has a similar size and structure to the input image. In such applications, one wants both to get locally precise information and to integrate wider context. This is often dealt with by using pooling and subsampling layers. Authors in [18] propose a model based on dilated convolutions which supports exponential expansion of the receptive field without the loss of resolution or coverage while the number of parameters grows linearly with layers. The main idea is to set *dilation rates* to some value d_h and d_v, which expands the kernel filter by adding $d_h - 1$ zeros between active points in rows and $d_v - 1$ zeros between active points in columns.
The resulting filter's size is $(k + (k-1) \cdot (d_v - 1), k + (k-1) \cdot (d_h - 1))$ where k denotes the size of standard filter (k, k). See example presented in Figure 1.8.

a	0	b	0	c
0	0	0	0	0
d	0	e	0	f
0	0	0	0	0
g	0	h	0	i

Figure 1.8: Example of a dilated convolutional layer. The kernel of size 3×3 and dilation rate of 2.

1.4.6 Depthwise-separable convolutional layer

Depthwise-separable convolutions are known as a key building block in efficient network architectures. It is composed of two separate layers called *depthwise convolution* and *pointwise convolution*. The former filters each channel from the input volume by separate filter and stacks their outputs on top of each other, which means that the number of input and output channels is equal. The latter takes these stacked outputs and builds new features through computing linear combination with 1×1 filter. The number of output channels depends on the number of pointwise filters.

This approach leads to drastic reduction of computation and model size. Let's calculate the difference. Ordinary convolutional layer takes input of size $h \times w \times c_i$ and applies c_0 kernels of size $k \times k \times c_i$ which produces output of size $h \times w \times c_o$ assuming the stride is set to 1. Its computation cost is $w \cdot h \cdot c_i \cdot k \cdot k \cdot c_o$. Depthwise-separable convolution is the sum of depthwise and pointwise convolutions. Formally, taking the same input mentioned above, the resulting output will be the same and computation cost is $h \cdot w \cdot c_i \cdot (k^2 + c_o)$. If we take for example filters of size 3×3, the computational cost is 8 to 9 times smaller compared to standard convolution. [1, 19]

1.4.7 Batch normalization

Proper initialization of the network might bring a lot of headaches. The activations of layers are not controlled and a small change to the network parameters amplify as the network goes deeper. The learning process is slowed down because one needs to set lower learning rate in order to make the network converge. The work of S. Ioffe and Ch. Szegedy [20] proposes a layer called *Batch normalization*. It is supposed to normalize activations of previous layer which prevents values from exploding too high or vanishing by being too low. Moreover, it reduces overfitting and thus makes the model better at generalization. For a layer with d-dimensional input $x = (x^{(1)} \dots x^{(d)})$ (d is number of channels in our case) and mini batch B (small portion of training images, since it is almost impossible to pass entire training set through the network at once) we normalize each dimension $\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]_B}{\sqrt{\mathrm{Var}[x^{(k)s}]_B}}$. Such a simple normalization of layer's inputs may change what the layer can represent and would constrain it to the linear regime of nonlinearity. To address this, authors introduce two more trainable parameters γ (scaling) and *beta* (shifting). The result of batch normalization layer is $\hat{y}^{(k)} = \hat{x}^{(k)} \cdot \gamma^{(k)} + \beta^{(k)}$.

Chapter 2

Related work

In this chapter we go through the related work in the field of deep learning, image classification and image segmentation while describing various model architectures as well as recent active learning approaches.

2.1 Convolutional neural networks

2.1.1 First CNN

The first convolutional neural network was proposed by LeCun et. al. in late 1980s. [21]. Their approach has been successfully applied to recognition of handwritten digits. Their dataset [22] consisted of roughly 9,298 images (later extended to 60,000 images), which vary in sizes and styles. Image transformation is applied to fit them to 16×16 pixel area, which prepares the images for feeding to the network. The output consists of ten neurons, each one computing probability of the given input being digit 0-9. The further development of this project was delayed because of limited computational capabilities at that time. Later, Matan et al. [23] extended this project to recognize strings of digits because previous work was limited to only one-dimensional input strings. Their approach includes recognizing 5-digit ZIP codes taken from the U.S Mail using Viterbi module. Since then, CNNs have been ignored by the computer vision community until the mid 2000s because of lack of computational power.

Figure 2.1: Example of digits from MNIST dataset [22].

2.1.2 AlexNet

The great boom of deep neural networks has started after proposing AlexNet [8]. Krizhevsky et al. competed in the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012. ImageNet [24] is a dataset of over 15 million labeled high-resolution images with around 22,000 categories. ILSVRC uses just a subset of them which is 1,000 categories.

AlexNet consists of 60 million parameters and 650,000 neurons. These neurons are spread over five convolutional layers and 3 fully connected layers. The last layer is 1000-way softmax, which produces the distribution over the 1,000 class labels. Convolutional layer filters are of sizes 11×11, 5×5 and 3×3. Standard activation functions of neuron's input are *tanh* or *sigmoid*. However, in terms of training time with gradient descent, ReLU (rectified linear unit) activation function is considered to be much faster. ConvNets with ReLU train several times faster. To prevent model from overfitting, data augmentation and dropout methods are applied.

The full architecture is shown in Figure 2.2.

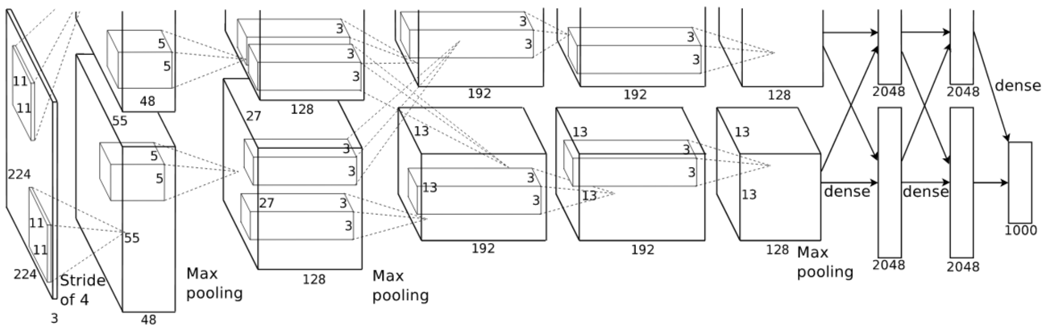


Figure 2.2: AlexNet architecture. One GPU runs the top part while the other runs the bottom part. They communicate only at certain layers. (source [8])

AlexNet outperforms previous feature-based methods lowering the error rate down by 40% compared to hand-engineering approaches. On the test data, they achieve top-1 and top-5 error rates of 37.5% and 17.0%. Their success is based on using ReLU activation function, local response normalization, dropout and stochastic gradient descent and in addition, usage of GPU during training and testing process. The network became state-of-the-art in computer vision classification.

2.1.3 VGGNet

VGGNet is an architecture proposed by Simonyan et al. [25]. The idea is to use deeper networks with much smaller filters. The work contains models which ranges from 16 to 19 layers and each one utilizes the smallest possible filters, which is 3×3 in order to reduce number of parameters. Some of the convolutional layers are followed by 2×2 max pooling layers with stride 2. The stack of three 3×3 layers has the same receptive field as one 7×7 layer and also fewer parameters assuming that both input and output have C channels, because $3 \cdot (3^2C^2) < 7^2C^2$. In addition, they incorporate three nonlinear functions instead of one, which make the decision function more discriminative. A downside of the VGGNet is that it uses a lot of memory due to expensive fully connected layers. Most of the parameters are in the first fully connected layer, but it was found that these FC layers can be completely removed with no significant performance downgrade [17]. Although the network's top-5 accuracy 92.7% did not beat GoogLeNet (see subsection 2.1.4), it won the 1st prize in localization task in ILSVRC'14.

2.1.4 GoogLeNet

GoogLeNet [17] won ILSVRC competition in 2014. It became the new state-of-the-art CNN while achieving top-5 error rate of 6.67%, which is very close to human level performance.

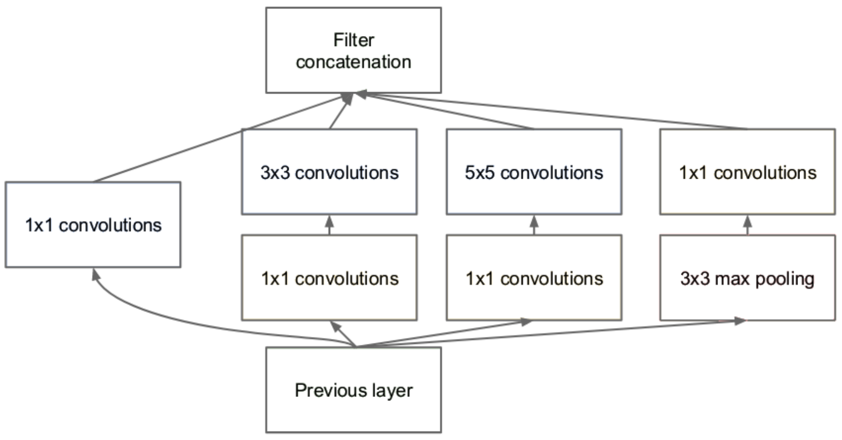


Figure 2.3: Inception module. (Image taken from [17])

Impressively, it consists of 22 layers, but the number of parameters is reduced to just 4 million (compared to 7 layer AlexNet with 60 million parameters). The main contribution this work has brought are *Inception modules* which are stacked on top of each other resulting in dramatic parameter reduction. Each one contains independent convolution and pooling layers and their results are subsequently concatenated into one output volume. Also, the authors got rid of expensive fully connected layers.

The original naive inception module would be very computationally inefficient. Each module would just increase the volume depth. Thus, the authors introduce an updated version of the inception module (see Figure 2.3) with 1×1 convolution put right before both 3×3 and 5×5 convolutions and additionally after max pooling layer to reduce dimension.

2.1.5 ResNet

Looking at the popular architectures people started asking a simple question: what happens if we continue stacking deep layers on a plain CNN? Does it increase accuracy? The answer is no, as He et al. describe and experimentally prove in [26]. Even though deep models perform worse, it is not caused by overfitting. They set a hypothesis that this problem is an *optimization problem*, and deeper models are harder to optimize than more shallow networks. The reasoning behind this idea is that deeper networks should perform at least as well as shallower ones (the solution is to copy a shallow network and add identity layers on top). The network won the 1st prize in 2015 on ILSVRC classification task with 3.75% top-5 error. The network itself brings a "revolution in depth" since it is supposed to contain a huge number of layers (it was 152 on ImageNet competition). The network features special *skip connections* and heavy use of *batch normalization*. The key idea is that instead of fitting $H(x)$ directly, they rather use network layers to fit residuals $F(x) = H(x) - x$, which is easier for the network to learn, see Figure 2.4. For deeper ResNet networks (50+ layers) they use so called bottleneck layers (similarly to GoogleNet) in order to improve efficiency. These layers contain additional 1×1 filters to decrease the volume depth.

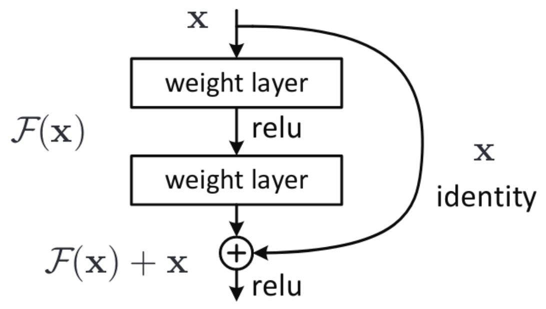


Figure 2.4: Residual block. (Image taken from [26])

2.2 Semantic segmentation using CNNs

2.2.1 Fully Convolutional Networks for Semantic Segmentation

Fully convolutional neural networks proposed by Long et al. [27] in 2014 popularized the use of end-to-end ConvNets for semantic segmentation of natural images.

They adapt contemporary classification networks like AlexNet, VGGnet, and GoogLeNet and fine-tune them to segmentation task. The fully connected layers of these classification networks are converted to fully convolutional layers (see Figure 2.5).

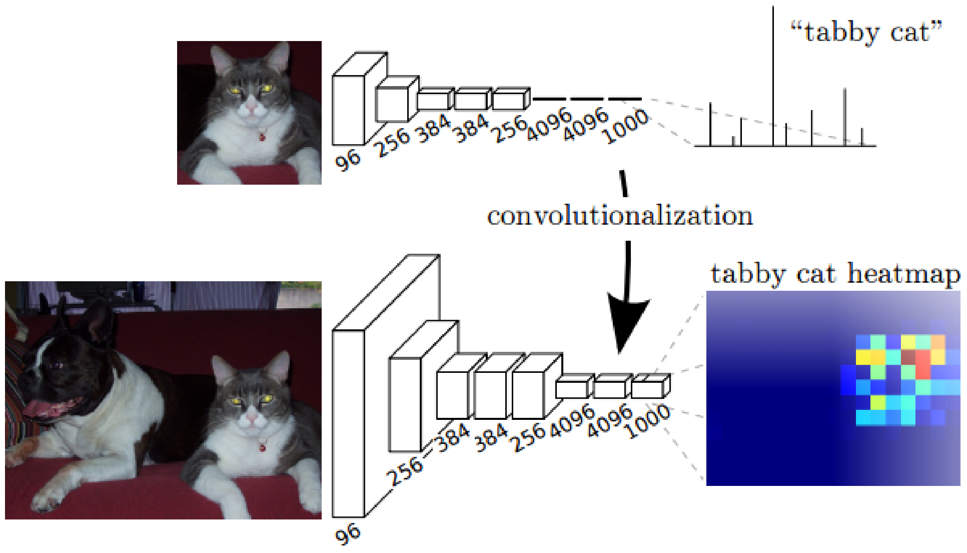


Figure 2.5: Transformation of fully connected layers into convolutions. (source [27])

After convolutionalization, these layers produce class presence heatmaps in low resolution. Some spatial information is lost because of pooling or strided convolutions, so the mask must be upsampled at each stage using billinearly initialized deconvolutions.

They achieve state-of-the-art segmentation on PASCAL VOC dataset [28] with 20% relative improvement to 62.2% mean IU in 2012.

2.2.2 U-Net

U-Net network [29] got popular in biomedical image analysis. Since the researchers in this area were forced to train the network with very little data, image augmentation played a key role in this problem. The dataset the network is trained on contains only 30 annotated images and outperforms the best methods on the ISBI challenge for segmentation of neuronal structures in electron microscopic stacks. As discussed in [29], it takes less than a second to segment image on a recent GPU.

The architecture resembles the character "U", that is why it is called U-Net. On the left side, there is encoder (or contracting path) which utilizes typical ConvNet architecture. It contains 3×3 convolutions, followed by ReLU and max-pooling operations for downsampling. Each layer in expanding path consists of upsampling of the feature map, followed by 2×2 convolution, a concatenation of the feature map from the contracting path and 3×3 convolution followed by ReLU. In terms of data augmentation on microscopical images, they essentially just need shift and rotation invariance as well as robustness to deformations and gray value variation.

This network achieves 92% mean IU on PhC-U373, beating the second best algorithm with 83%.

2.2.3 SegNet

SegNet [30] is another architecture of deep convolutional neural networks for pixelwise segmentation. The architecture of encoder consists of 13 convolutional layers topologically identical to first 13 layers of VGGnet [25]. The contribution of this work is mainly in the decoder, which uses pooling indices from the encoder to perform upsampling, thus leaving high frequency details intact in the segmentation. On the other side, neighboring information is missed when unpooling. Therefore, the decoder does not have to learn upsampling and whole architecture makes it more memory efficient.

2.2.4 R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN

Mask R-CNN published by He et al. [31] in 2017 builds on previous object detection works R-CNN [32], Fast R-CNN [33] and Faster R-CNN [34].

The original R-CNN is a four step process:

1. Input an image
2. Extract regions potentially containing objects (region proposals)
3. Compute features from each region proposal using pre-trained CNN
4. Classify each proposal using linear SVM

The problem of R-CNN is that it is enormously slow. Also, the model does not learn to localize objects using deep CNN. The contribution of Fast R-CNN is region of interest (ROI) pooling module, which essentially extracts fixed-size window from feature map. The network became end-to-end trainable but the performance is dependent on selective search and therefore suffers at prediction time.

To make it even faster, Ren et al. proposed Faster R-CNN. It utilizes a so called Region Proposal Network (RPN), which alleviates the need for selective search. Regions of interests are generated and top N are kept according to their objectness score. In the original paper N is set to 2,000. Faster R-CNN architecture is capable of running at 7-10 FPS, which is a huge step towards real-time object detection with deep networks.

Mask R-CNN has been published with two major contributions:

1. Replaced ROI pooling module with more accurate ROI align module

2. Inserted an additional branch out of the ROI align module to produce mask of the object

The network is therefore able to produce three outputs - bounding box of the object, its mask and label.

2.2.5 MultiNet

In 2016, Teichmann et al. [35] published an interesting work on road segmentation. In their approach they efficiently perform classification, detection and semantic segmentation simultaneously.

Model architecture is traditional encoder-decoder. The difference is that there is just one encoder producing rich features shared among all the tasks. These features are then utilized by task specific decoders. Encoder's weights are pre-trained on ImageNet classification data. They perform experiments on VGG16 and ResNet architectures. The first one is called VGG-pool5 as it discards all fully-connected layers from VGG and pool5 is the last layer. The second implementation is called VGG-fc7 because it only discards the final softmax layer. Layers fc6 and fc7 are replaced by 1×1 convolutions. This allows the network to process images of arbitrary size. For ResNet they implement 50 and 101 layer version of the network and discard fully-connected softmax.

Decoder architecture follows the FCN architecture [27]. Segmentation is applied to input downsampled by encoder to size 39×12 using 1×1 convolution layer. Then, it is upsampled using three transposed convolution layers.

MultiNet's training strategy is fine-tuning. The encoder is trained to perform classification on ILSVRC2012 dataset. But in practice, this step is omitted and encoder is initialized using pre-trained weights of the respective networks they are using. The second step is dedicated to replacing fully connected layers with convolutional ones.

Authors evaluate the proposed model on KITTI dataset [36], which contains images from various street situations captured from a moving platform in the city. The segmentation module reaches MaxF1 score of 94.88% with average precision of 93.71% beating all other submissions. Moreover, the speed of prediction is 42.48ms on GPU for all tasks, which is very efficient for real-time predictions.

According to their comparison of VGG and ResNet they observed that both ResNets outperform VGGs, but noticed that there is a trade-off because VGGs are faster.

2.3 Mobile models

Most of the work on semantic segmentation has focused on increasing accuracy of proposed models with little attention to computational efficiency. While these networks have brought state-of-the-art results, they often lack real-time prediction ability on devices with limited computational power.

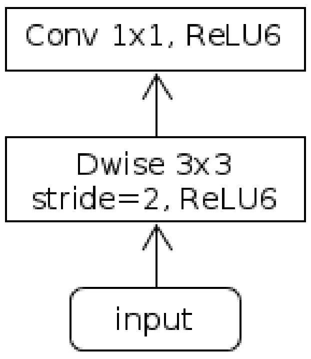
2.3.1 MobileNets

Probably one of the most popular networks designed specifically for small devices in terms of speed and computational power are **MobileNets** [1]. These models come with two hyper-parameters, namely α and ρ. The former, also called *width multiplier*, is responsible for adjusting number of channels and thins the network uniformly at each layer. For a given layer and multiplier α, the number of input channels becomes αM and the number of output channels αN, where M and N are original numbers. ρ, called *resolution multiplier*, is applied to input image reducing size of input as well as internal representation of subsequent layers. The architecture is composed of 3×3 depthwise-separable convolutions except the first layer which is full convolution. Authors test several variations of hyper-parameters and compare results on ImageNet dataset and get 70.6% top-1 accuracy. Interestingly, MobileNet with $\alpha = 1$ is almost as accurate as VGG16, while being 32 times smaller and 25 times less compute intensive.

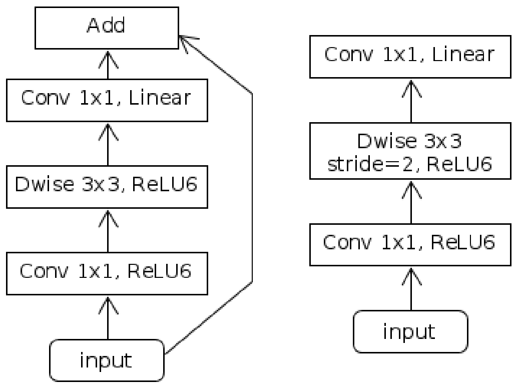
Sandler et al. [19] continue in the work of searching for efficient mobile models. Their architecture expands on aforementioned MobileNet and they call it **MobileNetV2**. Each bottleneck stage consists of several bottleneck layers, where the first one downsamples the representation with stride = 2 (with the exception in some stages). The rest of the layers in stage utilize residual connection from input which is summed with output of the pointwise convolution. Similarly to original MobileNet, it makes use of depthwise-separable convolution which is preceded by 1×1 convolution as shown in Figure 2.6. The model with $\alpha = 1$ reaches 72.0% top-1 accuracy beating original MobileNet and reducing the number of parameters, as well as the inference time. With $\alpha = 1.4$ they get 74.7% accuracy at the cost of doubling the inference time.

Authors also compare semantic segmetation abilities of MobileNet and MobileNetV2 as feature extractors on PASCAL VOC dataset. Encoders are followed by *DeepLabV3* decoder [37] with various modifications. Benchmarks show that their performance is almost equal but MobileNetV2 achieves it with fewer parameters.

The new generation of MobileNet is presented in the work of Howard et al. [38]. **MobileNetV3** is based on MobilenetV2 and MnasNet [39] building blocks. Automated platform-aware search NAS is used to look for global network structures by optimizing each network layer and applies NetAdapt algorithm afterwards to find out the number of filters per layer. Authors observe that some of the earlier and final layers are more expensive than others, so they redesign and further optimize them. Second half of bottleneck layers use newly proposed *h-swish* nonlinearity, since original swish



(a) MobileNet bottleneck



(b) MobileNetV2 bottlenecks

Figure 2.6: Comparison of MobileNet and MobileNetV2 bottlenecks.

(swish$(x) = x \cdot \sigma(x)$) includes computation of sigmoid which is much more expensive in mobile environments. Therefore, sigmoid has been replaced by its piece-wise linear hard analog and the h-swish function is defined as hswish$(x) = x\frac{\text{ReLU6(x+3)}}{6}$. The final model comes in two versions, namely *MobileNetV3-Large* and *MobileNetV3-Small*, which are targeted to high and low resource use cases respectively. The former reaches 75.7% top-1 accuracy and the latter 67.4%, while being much more efficient. The authors employ the models in semantic segmentation task and propose *Lite R-ASPP* segmentation head which is based on R-ASPP head that was proposed by Sandler et al. [19]. Experiments are conducted on Cityscapes dataset [40] and results reported in mean IoU metric. MobileNetV3-Large attains similar performance as MobileNetV2 while being faster and MobileNetV3-Small performs similarly to MobileNetV2-0.5.

2.3.2 ShuffleNets

Another family of light-weight convolutional network models is called **ShuffleNet**. Zhang et al. [2] propose the first model of this family. The architecture makes use of *pointwise group convolutions* and new idea called *channel shuffling*. The former has been introduced in [8] for distributing the model over two GPUs. If we stacked two group convolutions on top of each other, the output channels would gain information only from small fraction of previous channels. To address this, authors incorporate channel shuffling, which divides previous groups into subgroups and then feeds each group in next layer with different subgroup. Network's architecture starts with full 3×3 convolution followed by max pooling and three stages with different numbers of shufflenet units where the first one is used for spatial downsampling. Their experiments show that channel shuffle increases accuracy and ShuffleNet with $\alpha = 1$ (width multiplier) and group = 8 reached 67.6% top-1 accuracy. With $\alpha = 2$ they beat MobileNet by 3.1% while keeping the inference time.

Building upon ShuffleNet, Gamal et al. [41] experiment in the field of semantic segmentation. Their network, called **ShuffleSeg**, employs ShuffleNet as encoder. Experiments are conducted on Cityscapes dataset and compare decoders U-Net, SkipNet, Dilation Frontend 8s and Dilation 4s. Results indicate that SkipNet pretrained on coarse annotations with more labeled images beats other decoders and provides a good trade-off between accuracy and real-time prediction. It is reported to be capable of running at 15.7 FPS on Jetson TX2. [41, 42]

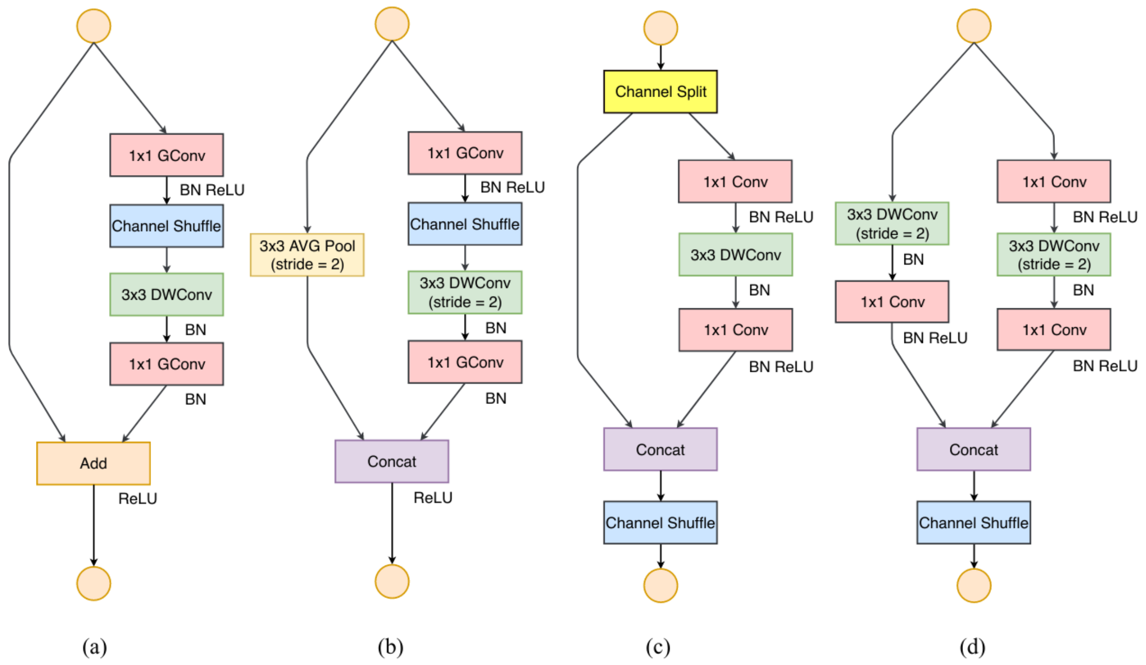


Figure 2.7: ShuffleNet (a), (b) and ShuffleNetV2 (c), (d) bottleneck units. (b) and (d) are units used for spatial downsampling with stride set to 2. (source [43])

In the work of Ma et al. [43], several practical guidelines for efficient ConvNet architecture design have been raised. These are targeted mostly to reduce MAC (memory access cost or the number of memory access operations). Based on these guidelines, authors propose new architecture called **ShuffleNetV2**. They replace group convolutions with pointwise 1×1 convolutions and use channel split in downsampling unit. Otherwise, the number of stages and units within them stays the same. Figure 2.7 shows the difference between ShuffleNet and ShuffleNetV2 units. Authors report their model is faster than MobileNetV2, ShuffleNet and Xception. Moreover, with $\alpha = 1$ it reaches 69.4% top-1 accuracy.

ShuffleNetV2 has been studied further in order to employ it in semantic segmentation. Türkmen et al. [44] take ShuffleNetV2 as an encoder and modify the last stage to omit downsampling and utilize dilated convolution within depthwise-separable convolution. Decoding process is carried by *DeepLabV3+* [45] which contains *Dense Prediction Cell* [46] at the beginning and also makes use of dilated convolutions. Authors report reaching 70.33% mean IoU on Cityscapes dataset and 15.41 FPS on a mobile phone with an input image size of 224×224.

2.4 Active learning

Labeling a dataset for supervised learning is quite expensive and time consuming process. *Active learning* [3] is a technique where the model being trained participates in the selection of its own training data. Selected samples are sent to the oracle (which is a human expert in many cases) to annotate them and training continues on the enlarged dataset. Training is stopped either when there are no more samples to annotate or the model performs good enough and early stop condition is met. [47]

There are several ways the model queries for new data. *Pool-based* sampling is often used in supervised learning. In theory, one sample is picked in each iteration and model is retrained. However, during training ConvNet, querying one sample would be quite inefficient and training would take too much time. Therefore, querying batches is a common practice in the deep learning world. [48, 49, 50, 51, 52, 53, 54, 55]

Joshi et al. [48] propose active learning framework for training an SVM classifier. They experiment with and compare random sampling, entropy based sampling and margin sampling (called *Best-versus-Second-Best* in their work) on three datasets. Margin sampling showed up to be the best choice as they were able to reach higher accuracy with less data.

The work of Vezhnevets et al. [49] discusses active learning in the field of semantic segmentation. The authors try to bridge the gap between weakly supervised and fully supervised methods by using active learning. The problem is modeled with a pairwise conditional random field (CRF) defined over superpixels. Their active learning starts from the output of weakly supervised learninng, meaning the labeling is partially incorrect. The model is able to ask oracle about ground truth of specific superpixel. They introduce a novel *Expected Change* (EC) score. Instead of querying for most uncertain images, they query for the ones that induce largest expected change in the labeling of the whole training set in terms of upper-bound accuracy. The disadvantage of this approach is that computing EC score is slow. On the other hand, their algorithm beats standard entropy sampling and reaches 97% of total pixel accuracy of the corresponding fully supervised model while querying less than 17% of the superpixel labels.

Wang et al. [50] study active learning and propose so called *Cost-Effective* active learning framework. Training the model starts with small amount of unlabeled data.

The models queries images to be annotated according to one of least confidence, margin sampling or entropy. Moreover, unlabeled images the model is very confident on are given *pseudo-labels* and are added to training set with no human labor cost. At the end of iteration, these pseudo-labeled images are returned to the pool of unlabeled ones. Authors state that their algorithm performs better than random selection. However, the analysis in [52] shows it actually performs worse than random sampling.

Several acquisition function are studied by Gal et al. [51] in their Bayesian ConvNet setting recognizing hand-written digits from MNIST dataset. These function are max entropy, BALD, variation ratios, mean STD and random baseline. Both mean STD and random sampling under-perform compared to the other functions.

Another work is presented by Sener et al. [52]. Their sampling strategy is based on *k-center* algorithm which strives for finding k points which minimize distance from all points to their closest core point. The authors demonstrate that random sampling is in many cases much more effective than uncertainty methods. This is because images the model is most uncertain on are often similar and thus do not cover the sample space appropriately. They evaluate models on several datasets achieving state-of-the-art results.

Active learning for biomedical image segmentation has also been successfully used by Yang et al. [53]. Image sampling is based on the uncertainty information as well as similarity information. Last layer in the encoding part produces high level feature embedding which is then used for computing cosine distance between images. Their approach achieves state-of-the-art performance by using only 50% of the training data.

It seems that awareness of image diversity is one of the most important criterion. Images may be embedded into vector space and clustered by K-means algorithm as stated in [55].

2.5 Previous approaches with Smelý Zajko

The robot *Smelý Zajko* (*Brave Rabbit* in English), shown in Figure 1.1, began its journey to discover and drive through city parks in 2011. The work of M. Nadhajský [56] aims to propose a design of the outdoor robot for this kind of competition. This design includes construction from aluminium and wooden parts as well as different hardware parts like servo motors, sensors and wheels. The author also implements algorithms for controlling the robot and verifies its functionality in the outdoor environment. The proposed solution for dealing with recognition of driveable path makes use of multi-layer perceptron (MLP).

M. Moravčík continued on robot's improvements in his thesis [57]. The author focuses on improving robot's localization and planning based on Open Street Maps.

Moreover, the author introduces vision module for predicting driveable path which utilizes MLP and various preprocessing methods. Camera's image is first being converted into CIELAB color space and then cut into smaller patches of size 5×5. The author also tests various MLP architectures as well as other handcrafted features such as histograms, regions with high driveable probability etc.

The work of J. Dúc [58] describes integration of a new laser sensor and based on the analysis of the previous state of the robot, he decides to change the system architecture and rewrite the software into *Robot Operating System* (ROS) [59]. ROS aims to be modular with the ability to divide subsystems into modules which are able to communicate between each other and are portable when it comes to integrating them to other system. It also supports recording and logging data from sensors and replaying them afterwards. The author also designs new reactive algorithm for detecting and avoiding obstacles and tests it in both indoor and outdoor environment.

Jariabka et al. [60] extended previous MLP approaches in aforementioned works. They evaluate baseline MLP models on their dataset with images of size 224×224 pixels and propose the first ConvNet model used on *Smelý Zajko*. Its architecture is rather simple having several convolutional layers with 10 filters followed by ReLU nonlinearity and max-pooling 2×2 layers. Authors also experiment with deeper architecture containing fully-connected layer and two convolutional blocks.

The solution used at RoboTour 2018 is based on HSV data extracted from training images. Every image is converted into HSV color space and processed pixel by pixel. Only **H** and **S** components are taken into account. The model creates 2D matrix with values expressing how likely is the pixel with specific H and S driveable.

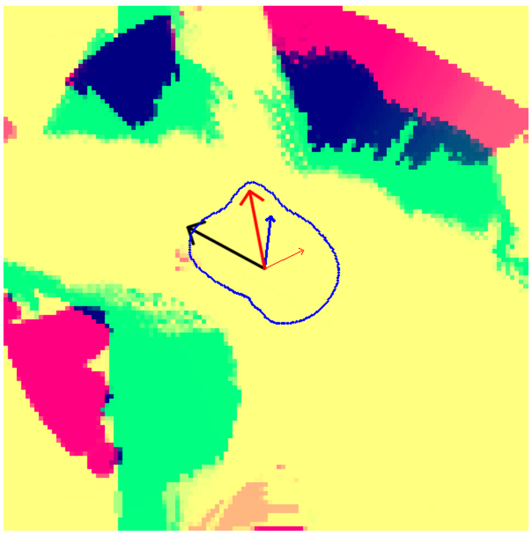


Figure 2.8: Example of Local Map at the crossroad. Robot is deciding which exit to use. The yellow color denotes driveable trail and blue or red are obstacles calculated by laser sensor.

In 2019, M. Fikar [61] continued on improving robot's algorithm for planning and localization. The robot reads data from sensors, tries to combine them and creates so called *Local Map*. Based on this data, the robot calculates the best direction towards destination point and sends the information about next move to motor control unit, which is in our case Arduino. The map also takes predicted driveable path into account. An example of this map is presented in Figure 2.8.

All of the aforementioned models for predicting driveable path are looking at specific regions discarding spatial information which makes them less precise. This causes the models to make mistakes in classifying shadows, segments with color similar to the road and images with varying lighting conditions. To deal with such a problem we will focus on incorporating a ConvNet taking whole image as input and outputting dense pixel-wise segmentation mask.

Chapter 3

Testing and comparing models for RoboTour

The RoboTour 2019 competition took place in Deggendorf, Germany. It was our first opportunity to train and test several segmentation models. In this chapter, we introduce our datasets, the way we preprocess and augment data before feeding it to ConvNet and present the HSV baseline model. We also compare several models we tested before the competition and report results.

3.1 Datasets

The first dataset that has already been published in [60] contains **333** labeled pictures in 4:3 ratio taken in the city park of *Lednice*. See Figure 3.1 for examples from this dataset.

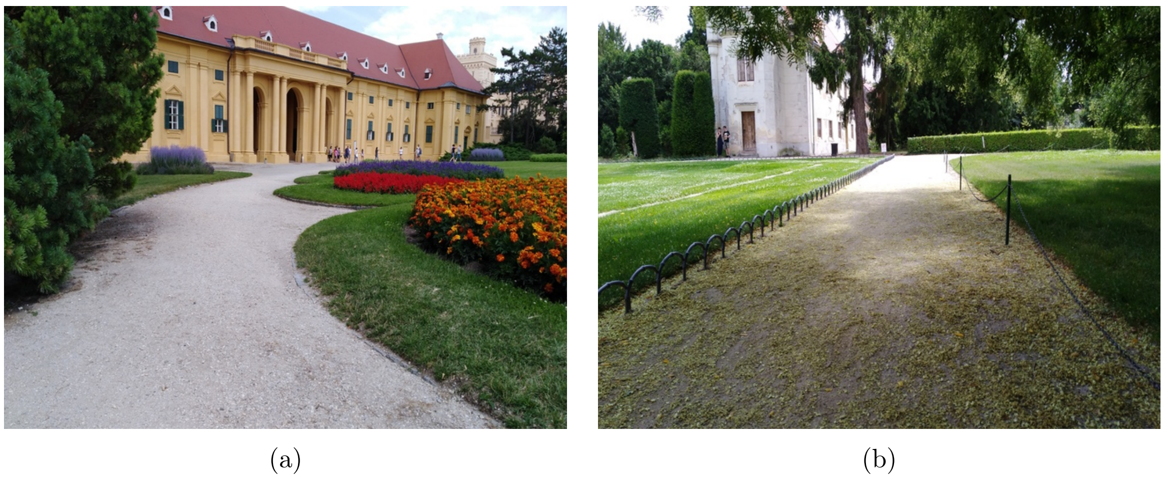
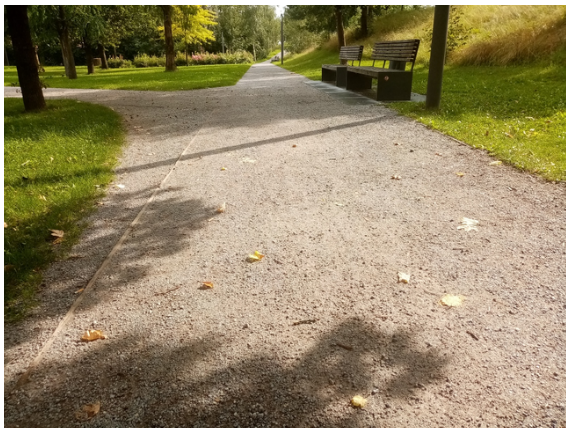


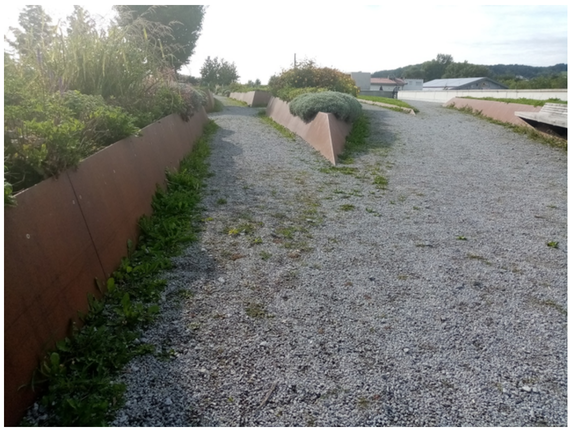
Figure 3.1: Examples from the city park of Lednice.

Since the latest contest took place in Deggendorf, we wanted to create a new dataset capturing the park which surrounds *Deggendorf Institute of Technology*. Images were captured by Android Huawei phone in 4:3 ratio catching various weather conditions, which prevented models overfit to specific conditions. One can find there are various types of pavements, wet and dry roads, sun shining right to the camera lens, people walking or riding a bicycle, bridges, benches, shadows and leaves on the roads, river, grass, flowers, gravel and railway. Examples are shown in Figure 3.2.

Testing one day before the competition revealed that the dataset was quite imbalanced because all of the images contained driveable path. Models had always searched for some sort of road, which resulted in very bad predictions of images containing mostly grass or non-driveable segments. Therefore, we let the robot capture additional pictures with grass covering huge part of images. See predictions before and after the dataset has been extended in Figure 3.4. We picked 61 and manually labeled them. The Deggendorf dataset now contains **344** images in total.

(a)


(b)


(c)

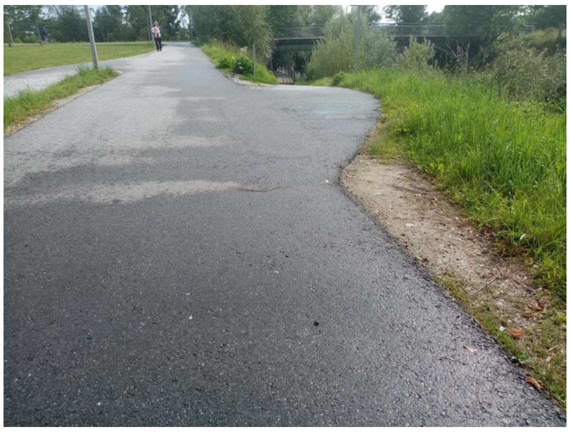

(d)

Figure 3.2: Examples from the city park of Deggendorf.

3.2 Data preprocessing and augmentation

In order to feed data to network one has to preprocess the image in some way. Pictures produced by phone were in quite high resolution which would make the inference time very high. We decided to scale them down to widely used resolution of 640×480 pixels. Labels in Lednice dataset contain 53.90% of driveable segment pixels and labels in Deggendorf dataset contain 61.14% of driveable segment pixels.

Our images are in the **RGB** color space, which means that each one contains three channels with values in range from 0 to 255. In order to make the model learn it is a common practice to normalize these values so they are in range either from 0 to 1 or from -1 to 1. The former can be reached by dividing all the channels by 255 and the latter by subtracting mean and dividing by standard deviation per channel. Ground truth labels (masks) are just 2D matrices with binary values where **1** denotes driveable pixel and **0** non-driveable one. Since these masks contain values 0 or 255 once loaded from disk, we just divide them by 255 to get zeros and ones.

Image augmentation is essential when it comes to extending the dataset because of the need for better generalization and preventing the model from overfitting. From every image we generate several augmented ones using *imgaug* library [62].

- **Rotation** - random rotation by 5 degrees
- **Horizontal Flip** - flip the image horizontally
- **Motion Blur** - the robot captures images while moving and they are often blurred. Motion Blur blurs images by random kernel $k = randint(3, 7)$ to resemble real world scenario
- **Add Brightness** - add brightness by severity $s = 2$
- **Fog** - add fog to image

Rotation and *Horizontal Flip* transformations require the mask to be augmented as well and thus we apply such transformation on mask. Both augmented and original images are then being fed to ConvNet.

3.3 Models and learning process

3.3.1 Losses

The *loss* function (also called *cost* or *error* function) denotes a function computing how bad is the model at predictions. The optimizer updates model's weights to decrease the loss. Since we are not able to compute these weights perfectly, training neural networks is an optimization problem where the loss function navigates us through the space of possible configurations.

There are many loss functions to choose from. The most common ones for semantic segmentation used in literature are *Binary crossentropy* and *Dice coefficient loss.*

- **Binary crossentropy**

$$\text{BCE}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(\hat{y_i}) + (1 - y_i) \cdot log(1 - \hat{y_i})$$

- **Dice coefficient loss**

$$\text{DICE}(y, \hat{y}) = 1 - \frac{2 \cdot y \cdot \hat{y} + \epsilon}{y + \hat{y} + \epsilon} = 1 - \frac{2 \cdot \sum\limits_{i=1}^{N}(y_i \cdot \hat{y_i}) + \epsilon}{\sum\limits_{i=1}^{N} y_i + \sum\limits_{i=1}^{N} \hat{y_i} + \epsilon}$$

The N denotes the number of pixels, y_i denotes ground truth value for pixel at i-th position and $\hat{y_i}$ is the output of the network at i-th position, which is interpreted as the probability of pixel being driveable.

3.3.2 Metrics

Metrics are used to measure the performance of the model. Choosing the right metric is a crucial part in order to compare models appropriately. We might sometimes also use loss function as a metric, but in general a metric function does not have to be differentiable. In our work, we are working with two most widely used metrics for segmentation task:

- **Binary Accuracy**

$$\text{ACC}(I) = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Intersection over Union (IoU)**

$$\text{IoU}(I) = \frac{1}{|I|}\sum_{i=1}^{|I|} \frac{|Y_i \cap \hat{Y_i}|}{|Y_i \cup \hat{Y_i}|}$$

Since output of the network is in range between 0 and 1 for each pixel, we need to round $\hat{Y}$ values. Threshold is set to 0.5 meaning pixels with probability above 0.5 get classified driveable and conversely, pixels with probability below 0.5 are classified non-driveable. The argument I is a set of images the metric is measured on.

The nominator in *binary accuracy* metric represents the number of pixels that are classified correctly, whilst denominator represents the total number of pixels present across entire image set.

Intersection over Union is measured as a mean of IoUs over the set I. Simply put, it is the area of intersection between predicted segmentation mask and ground truth divided by their union. IoU is the most popular metric in semantic segmentation community sometimes referred to as *Jaccard index.*

Binary accuracy can sometimes provide misleading results and that's why IoU is better in terms of correct overlaps. Now let's try to understand why this metric is better than binary accuracy. Imagine the model is asked to predict an image with a small area of driveable segment. If the model classifies all the pixels as non-driveable (returns mask filled with zeros) we get very high accuracy due to correct classification of these zeros. However, the model's IoU is zero in this case which is desired result.

3.4 Results

3.4.1 HSV model

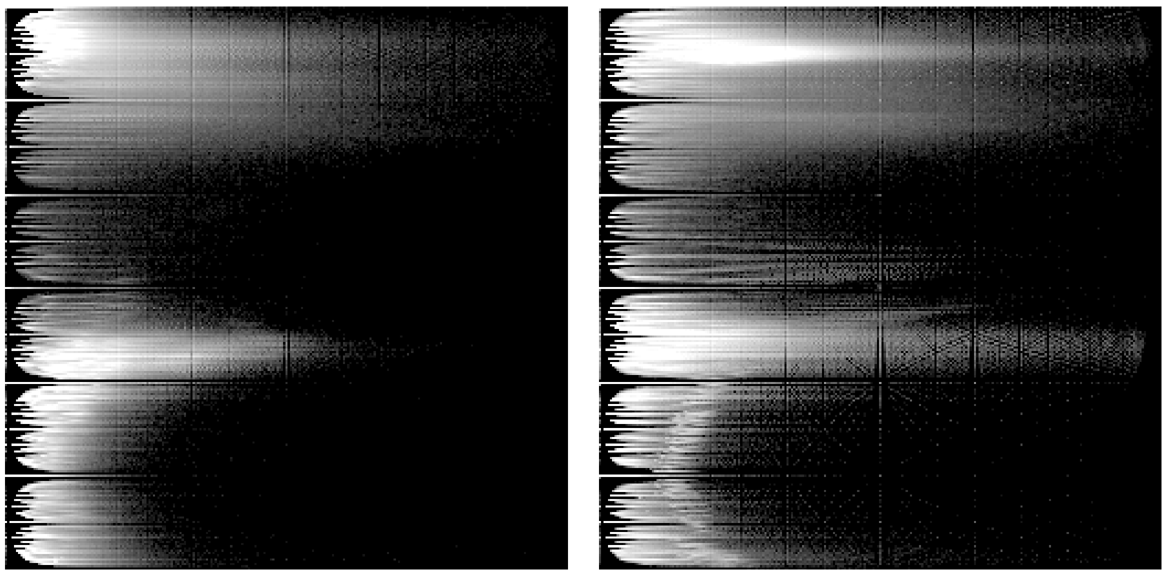

(a) Trained on Lednice dataset (b) Trained on Deggendorf dataset

Figure 3.3: HSV model trained on Lednice and Deggendorf datasets. Y axis represents H values, X axis represents S values and pixel intensity indicates how likely is the pixel driveable.

We took HSV model used on RoboTour 2018 and set it as our baseline.

dataset	test accuracy	test IoU
Lednice	0.9037	0.8504
Deggendorf	0.8006	0.7403

Table 3.1: Results of HSV model measured on both datasets.

Both datasets were split into *train* and *test* subsets. Training the model produced matrices shown in Figure 3.3.

According to pixel intensities we may notice higher uncertainty in Deggendorf dataset which proves our assumption that this dataset is much more difficult to learn and predict. There is a large number of pixels with same or almost the same color but classified differently (i.e. grey pavement and grey wall). Model's performance is captured in Table 3.1. Although the Lednice results might be acceptable, in case of Deggendorf the accuracy is not sufficient.

3.4.2 KittiSeg

One of the decoders in MultiNet architecture [35] is dedicated to semantic segmentation task. We wanted to test its performance on our Lednice dataset in 2018 right before the competition in the Lednice park.

Encoders in this architecture were pretrained on ImageNet dataset [24] and we only wanted to fine-tune the model to our Lednice dataset. Since this architecture is designed to perform binary semantic segmentation of driveable roads, it took only few epochs to fine-tune it to city park. Results are captured in Table 3.2. The reason for not using it on robot is we were not able to load it to Jetson's memory because of the model's huge size. NVIDIA Jetson TX2 GPU mounted on our robot contains 8 GB of RAM, 256 NVIDIA CUDA cores and 6 CPU cores.

type	train acc	val acc
no augmentation	0.9571	0.9563
augmentation used	0.9580	**0.9566**

Table 3.2: Results of KittiSeg model training on Lednice dataset.

3.4.3 ConvNet models

We decided to test four different models before RoboTour competition. Among many available we picked these:

- Unet

- SegNet
- ResNet with 16 identity blocks
- FCN VGG16 with 32x upsampling

The FCN model was not able to start learning, thus we discarded it from our experiments. As long as we have had access to external server with graphics processing unit (GPU) NVIDIA GeForce GTX 1080, training procedure was far more efficient in terms of speed than on CPU. Our experiments involve training the models on both datasets, testing both loss functions and reporting pixel accuracy as well as intersection over union.

The environment we chose to work in is *Python 3.6* which has support for many useful tools necessary for computer vision and especially deep learning. Furthermore, in order to work with neural network we decided to choose widely used *TensorFlow* [63] framework. On top of *TensorFlow* backend we utilize *Keras* [64]. It is an open source library capable of running on top of several backends. It focuses on being user-friendly by providing interfaces for higher level abstractions and contains implementations for commonly used building blocks in deep learning such as layers, optimizers, metrics, activation functions as well as many utilities for manipulating data and evaluating models.

model	disk size	# params	prediction time on Jetson
ResNet	33 MB	2,753,729	0.24169 sec
SegNet	60 MB	7,818,117	0.36903 sec
Unet	356 MB	31,032,837	1.08655 sec

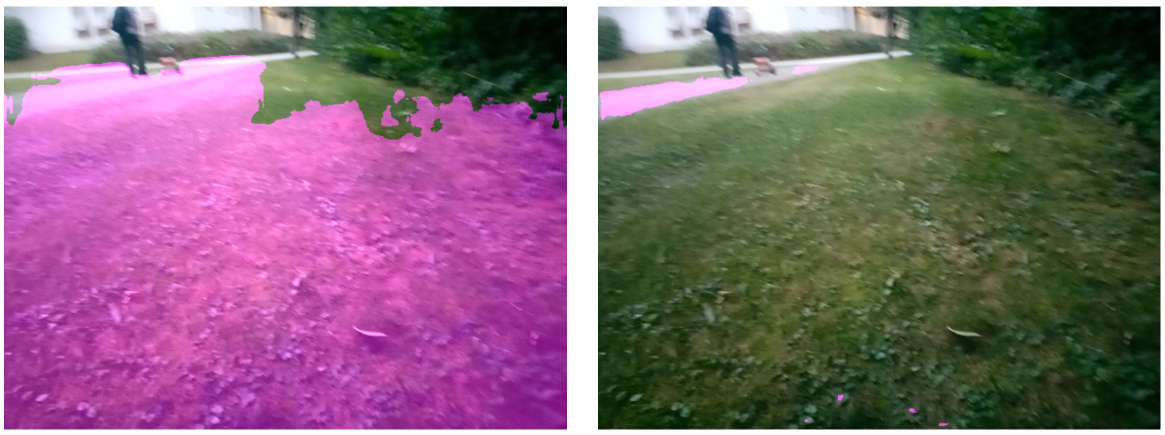
Table 3.3: Basic information about models. The size of the input image is 640×480. The prediction time is computed as the average of 20 predictions.

Table 3.4 captures results of aforementioned models. These results are split into two subtables in order to differentiate between Lednice and Deggendorf dataset.

All the models were trained from scratch meaning all weights were randomly initialized. We test both binary crossentropy and dice coefficient loss as well as both *RGB* and *HSV* color spaces. We split both datasets into *train*, *validation* and *test* subsets. Deep learning models are usually trained with some kind of *early stopping condition* which prevents the model from overfitting to training data and decreases the time needed for learning. In our case, we set early stopping condition to monitor validation loss and stop the training process when this loss does not decrease in 5% of total epochs.

In Table 3.3 we report basic information about proposed models and compare their average inference times on Jetson TX2 GPU.

When it comes to optimizing test intersection over union, binary crossentropy outperformed dice coefficient loss (but in general it does not have to be the case). Furthermore, results indicate that models show better performance in RGB color space. ResNet is almost always better than SegNet while having the lowest number of parameters and the best inference time. Unet's disadvantage is its number of parameters and undesirable inference time. Evaluation of models shows that Lednice dataset is easier to predict than Deggendorf dataset, like we observed with HSV baseline model. An interesting fact is that we are often able to train these models in less than one hour which makes it usable especially in competition setup where we sometimes need to retrain model to specific conditions.

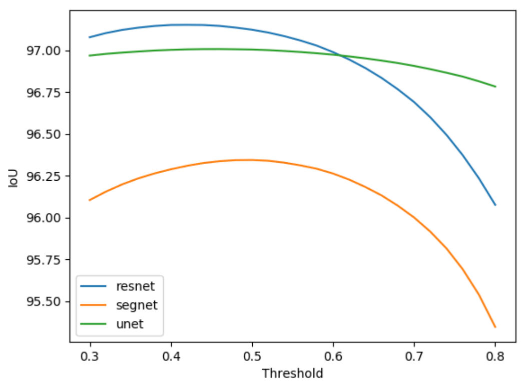


(a) Before (b) After

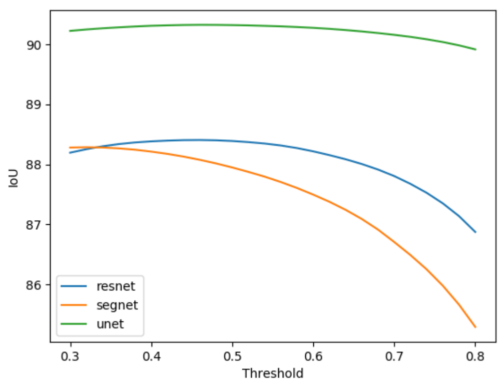
Figure 3.4: Predictions of images containing mostly grass. Image (a) shows the prediction before this kind of images have been added to the dataset while image (b) is the prediction after the dataset has been extended. Both images are predicted by the same ResNet. The only thing that has changed is dataset.

AUC-ROC comparison

Since the output from ConvNet is probability of each pixel being driveable, we need to define threshold for rounding these probabilities to one and zero respectively. All the results in Table 3.4 were acquired using standard threshold of 0.5, but we also examined thresholds in range from 0.3 to 0.8. According to Figure 3.5, ResNet seems to work a little bit better with thresholds close to 0.45 on Deggendorf dataset whilst SegNet's best threshold values are close to 0.3. Interestingly, SegNet works better with threshold around 0.5 on Lednice dataset and Unet's accuracy stays almost unchanged on both datasets. Once we noticed our models make mistakes on images covered mostly by non-driveable pixels we extended Deggendorf dataset right before the competition which improved predictions in production by huge margin, see Figure 3.4.



(a) Lednice dataset

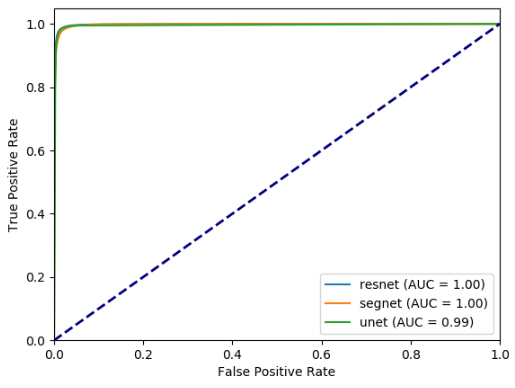


(b) Deggendorf dataset

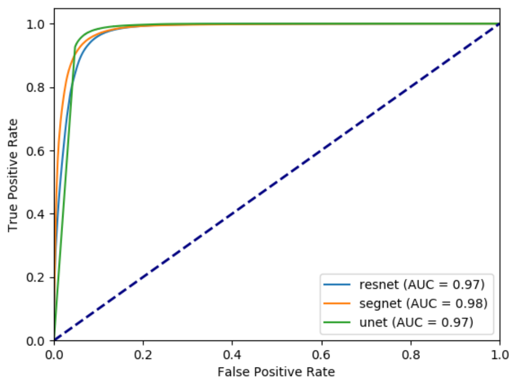
Figure 3.5: Comparison of various rounding thresholds for ResNet, SegNet and Unet (binary crossentropy loss with early stopping condition). The IoU is measured on test set.

In order to select the best model which should be used in production, a selection metric is chosen. One of the most commonly used selection metrics is *AUC-ROC* (Area Under Curve - Receiver Operating Characteristics) curve. It is used to show performance of models at various threshold settings. The plot consists of two parameters:

- True Positive Rate denoted as $TPR = \frac{TP}{TP+FN}$
- False Positive Rate denoted as $FPT = \frac{FP}{FP+TN}$



(a) Lednice dataset

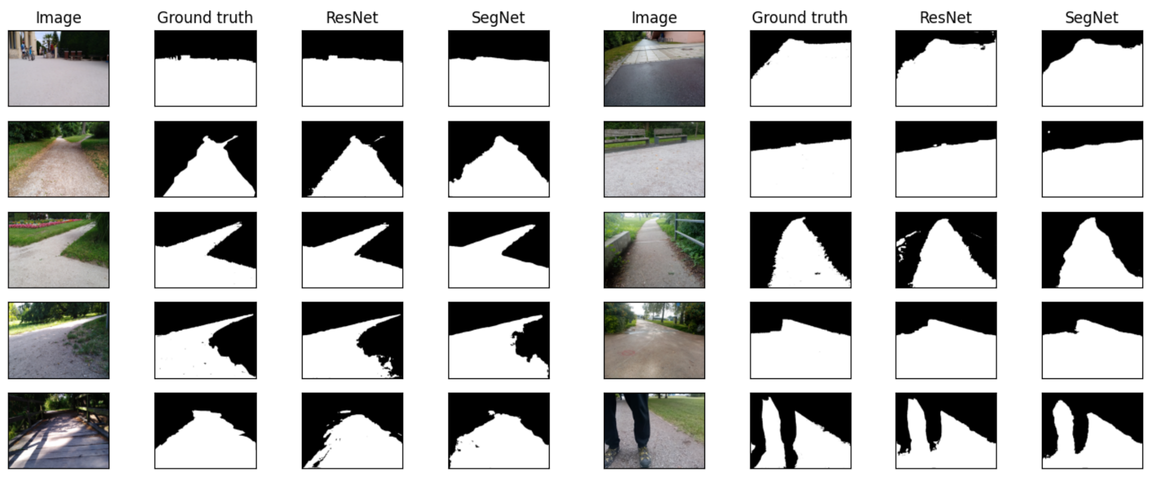


(b) Deggendorf dataset

Figure 3.6: AUC - ROC comparison of ResNet, SegNet and Unet (binary crossentropy loss with early stopping condition).

In terms of the predicted probability, ROC tells us how good the model is at distinguishing pixel's class. Since AUC stands for Area Under Curve, it measures the entire two-dimensional area under the ROC curve and provides an aggregate measure across all possible thresholds. The bigger the area, the better the model is in distinguishing between driveable and non-driveable pixels. If the AUC is equal to 0.5, model's prediction are almost random with no ability to distinguish between classes.

We compared our models using AUC-ROC curve. Figure 3.5 captures these comparisons on both datasets. Their performance is very similar on Lednice dataset, but SegNet works a bit better on Deggendorf dataset. Since ResNet shows the best results overall (the best trade-off between test IoU and inference time), we decided to use it at the competition in Germany. During official runs predictions were quite good in terms of accuracy, but the model's inference time is very high giving us only **four** frames per second (FPS). This makes it very difficult for the robot to drive faster because it would suffer from high latency between predictions. The visualization of test set predictions by ResNet and SegNet is presented in Figure 3.7.



(a) Lednice

(b) Deggendorf

Figure 3.7: Comparison of test set predictions by ResNet and SegNet against ground truths.

model	clr	loss	time	eps	train acc	train IoU	test acc	test IoU
ResNet	rgb	dcl	81m	77	0.9847	0.9673	0.9840	0.9700
SegNet	rgb	dcl	191m	167	0.9808	0.9578	0.9709	0.9484
Unet	rgb	dcl	523m	157	0.9925	0.9842	0.9846	**0.9715**
ResNet	rgb	bce	76m	70	0.9867	0.9698	0.9845	0.9711
SegNet	rgb	bce	114m	100	0.9845	0.9649	0.9809	0.9643
Unet	rgb	bce	434m	130	0.9948	0.9888	0.9841	0.9705
ResNet	hsv	dcl	45m	42	0.9758	0.9496	0.9815	0.9660
SegNet	hsv	dcl	144m	125	0.9769	0.9493	0.9756	0.9562
ResNet	hsv	bce	54m	50	0.9818	0.9594	0.9853	**0.9725**
SegNet	hsv	bce	52m	45	0.9768	0.9487	0.9751	0.9554
ResNet	rgb	dcl	155m	150	0.9848	0.9680	0.9828	0.9683
SegNet	rgb	dcl	171m	150	0.9796	0.9552	0.9767	0.9580
ResNet	rgb	bce	159m	150	0.9823	0.9604	0.9839	**0.9694**
SegNet	rgb	bce	172m	150	0.9854	0.9668	0.9790	0.9619

model	clr	loss	time	eps	train acc	train IoU	test acc	test IoU
ResNet	rgb	dcl	44m	40	0.9605	0.9259	0.9378	0.8878
SegNet	rgb	dcl	122m	104	0.9660	0.9302	0.9222	0.8551
Unet	rgb	dcl	499m	145	0.9834	0.9606	0.9506	0.9017
ResNet	rgb	bce	42m	38	0.9625	0.9269	0.9366	0.8838
SegNet	rgb	bce	57m	48	0.9671	0.9357	0.9403	0.8795
Unet	rgb	bce	451m	131	0.9908	0.9715	0.9521	**0.9031**
ResNet	hsv	dcl	61m	55	0.9546	0.9165	0.9376	**0.8868**
SegNet	hsv	dcl	192m	167	0.9656	0.9303	0.9329	0.8788
ResNet	hsv	bce	86m	75	0.9750	0.9410	0.9409	0.8861
SegNet	hsv	bce	97m	81	0.9738	0.9415	0.9308	0.8748
ResNet	rgb	dcl	160m	150	0.9784	0.9504	0.9445	0.8953
SegNet	rgb	dcl	176m	150	0.9687	0.9357	0.9189	0.8494
ResNet	rgb	bce	164m	150	0.9766	0.9459	0.9491	**0.8958**
SegNet	rgb	bce	177m	150	0.9784	0.9486	0.9386	0.8793

Table 3.4: Results - the first table describes results on **Lednice** dataset and the second table on **Deggendorf** dataset. Models in last four rows in both tables were trained for 150 epochs, whilst the other ones used early stopping condition monitoring validation loss. Dice coefficient loss is marked as *dcl* and binary crossentropy as *bce*. *Clr* denotes color space of images, *time* is training time and *eps* is the number of epochs.

3.4.4 Testing models on unseen dataset

Imagine the robot is participating at the competition in environment we do not have dataset from. Since obtaining a new dataset along with labeling all of its images is very time consuming we cannot afford to prepare the model in such a short period of time. Thus, we want to train our models on other dataset (obtained at previous contests) and use it in a new environment. This approach is often reffered to as transfer learning.

Table 3.5 shows performance of models described in Section 3.4.3 on unseen datasets.

Surprisingly, if we compare these results with Table 3.4, we can clearly see that there is not a big difference in terms of accuracy and models perform relatively well on unseen datasets. The IoU is worse by 3% on average, but the IoU is high enough for predicting driveable path and therefore we can conclude that models trained on specific dataset may be used in a similar environment.

model	clr	loss	acc	iou
ResNet	rgb	dcl	0.9144	0.8475
SegNet	rgb	dcl	0.9157	0.8502
Unet	rgb	dcl	0.9360	0.8810
ResNet	rgb	bce	0.9036	0.8278
SegNet	rgb	bce	0.9146	0.8439
Unet	rgb	bce	0.9357	**0.8812**
ResNet	hsv	dcl	0.8939	0.8189
SegNet	hsv	dcl	0.8893	0.8171
ResNet	hsv	bce	0.8887	0.8110
SegNet	hsv	bce	0.8835	0.8003
ResNet	rgb	dcl	0.9230	**0.8616**
SegNet	rgb	dcl	0.9103	0.8383
ResNet	rgb	bce	0.9192	0.8509
SegNet	rgb	bce	0.9093	0.8351

(a) Lednice models on Deggendorf dataset

model	clr	loss	acc	iou
ResNet	rgb	dcl	0.9624	0.9289
SegNet	rgb	dcl	0.9563	0.9149
Unet	rgb	dcl	0.9654	0.9324
ResNet	rgb	bce	0.9626	0.9271
SegNet	rgb	bce	0.9597	0.9206
Unet	rgb	bce	0.9689	**0.9382**
ResNet	hsv	dcl	0.9637	**0.9300**
SegNet	hsv	dcl	0.9532	0.9108
ResNet	hsv	bce	0.9617	0.9242
SegNet	hsv	bce	0.9526	0.9074
ResNet	rgb	dcl	0.9644	0.9312
SegNet	rgb	dcl	0.9535	0.9103
ResNet	rgb	bce	0.9656	**0.9323**
SegNet	rgb	bce	0.9583	0.9199

(b) Deggendorf models on Lednice dataset

Table 3.5: Results of models tested on full unseen dataset

Chapter 4

Faster models

Models presented in chapter 3 are quite slow, giving us only $\sim$ 3-4 frames per second on NVIDIA Jetson TX2. In this chapter we experiment with light-weight variations of ResNet and SegNet. We also test various mobile models proposed in recent years capable of running on devices with low computational power. From now on, we use only Deggendorf dataset, binary crossentropy loss and images in RGB color space.

4.1 Reducing the number of parameters

As stated above, our models (namely ResNet and SegNet) are slow in terms of inference time. The first idea that comes to our mind is to reduce the number of parameters in order to get more FPS.

Our **ResNet** contains so called *identity blocks* which we put in pairs next to each other. The model also takes parameter f which denotes the filter multiplier (default 16) being used in all layers. **SegNet** contains too many layers so we decided to reduce them and experiment with the number of filters in each convolutional layer.

List of modified models:

- ResNet-1 - original model, $f = 8$
- ResNet-2 - original model, $f = 4$
- ResNet-3 - every other identity block skipped, $f = 16$
- ResNet-4 - every other identity block skipped, $f = 8$
- SegNet-1 - filters by layers: 64, 128, 256, 256, 256, 256, 128, 64, 32
- SegNet-2 - filters by layers: 32, 64, 128, 128, 256, 128, 128, 64, 32
- SegNet-3 - filters by layers: 32, 32, 64, 64, 128, 128, 64, 32, 32

- SegNet-4 - filters by layers: 32, 32, 64, 64, 64, 64, 64, 32, 32

model	# params	on Jetson	train acc	train IoU	test acc	test IoU
ResNet-1	698,017	0.13288 sec	0.9795	0.9509	0.9408	0.8836
ResNet-2	179,297	0.10143 sec	0.9678	0.9332	0.9360	0.8796
ResNet-3	1,895,649	0.16817 sec	0.9783	0.9488	0.9446	**0.8919**
ResNet-4	480,817	**0.09807 sec**	0.9662	0.9284	0.9399	0.8850
SegNet-1	2,534,401	0.31107 sec	0.9444	0.9024	0.9316	0.8784
SegNet-2	1,075,009	0.22216 sec	0.9497	0.9073	0.9365	0.8816
SegNet-3	391,105	0.14389 sec	0.9631	0.9276	0.9417	**0.8889**
SegNet-4	206,145	**0.13818** sec	0.9625	0.9268	0.9380	0.8843

Table 4.1: Results of modified ResNet and SegNet training.

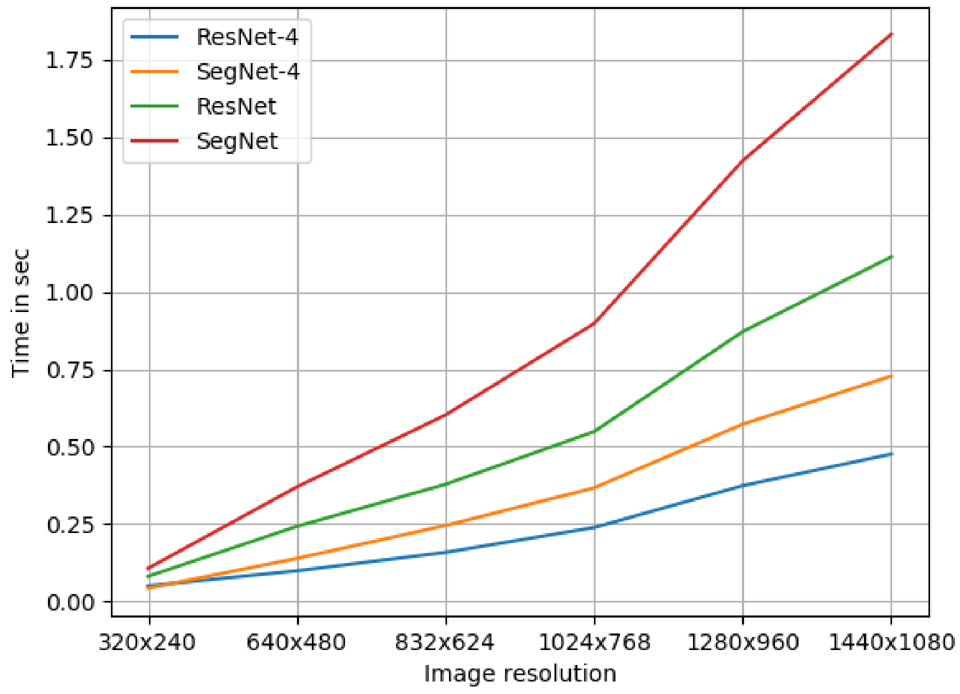


Figure 4.1: Comparison of inference times against various image resolutions on Jetson TX2 GPU.

By reducing ResNet's number of parameters we are able to increase the FPS to 10 while preserving the IoU. The same applies to SegNet where we reach 7 FPS with the same IoU, see Table 4.1.

We have also tried incorporating dilation rate to encoder convolutional layers [18]. In case of SegNet, we set the dilation factors in ascending order (1, 2, 4, 8). ResNet contains identity and bottleneck blocks, both of them having one 3×3 convolutional layer. We tried setting dilation rate just in identity block. Other experiments with ResNet involved setting dilation rate in both identity and bottleneck blocks. The experiments show that dilation convolutions do not bring improvement in terms of accuracy to our models.

Aforementioned models are able to process images of various resolutions. Figure 4.1 shows the comparison of inference times of these models against several resolutions (with aspect ratio 4:3). Predictions of images with resolution 320×240 take similar amount of time. However, with growing resolution we can see modified ResNet beats other models and is a good choice when dealing with higher resolutions.

4.2 Mobile models

name	encoder	decoder	α	# params
SSeg-1	ShuffleNet ($g = 1$)	SkipNet 8s	0.25	970,825
SSeg-2	ShuffleNet ($g = 1$)	SkipNet 8s	0.5	1,920,793
SSeg-3	ShuffleNet ($g = 2$)	SkipNet 8s	0.25	959,241
SSeg-4	ShuffleNet ($g = 2$)	SkipNet 8s	0.5	1,889,841
SSeg-5	ShuffleNet ($g = 4$)	SkipNet 8s	0.25	921,833
SNetV2-1	ShuffleNetV2	DeepLabV3+	1.0	1,871,428
SNetV2-2	ShuffleNetV2	DeepLabV3+	0.5	694,886
MNetV2-1	MobileNetV2	DeepLabV3+	1.0	11,978,698
MNetV2-2	MobileNetV2	DeepLabV3+	0.5	4,643,466
MNetV2-3	MobileNetV2	DeepLabV3+	0.25	2,742,418
MNetV2-4	MobileNetV2*	DeepLabV3+	0.25	1,252,498
MNetV3-L-1	MobileNetV3-Large	DeepLabV3+	1.0	4,641,674
MNetV3-L-2	MobileNetV3-Large	DeepLabV3+	0.5	2,484,146
MNetV3-L-3	MobileNetV3-Large	DeepLabV3+	0.25	1,902,602
MNetV3-S-1	MobileNetV3-Small	DeepLabV3+	1.0	2,142,074
MNetV3-S-2	MobileNetV3-Small	DeepLabV3+	0.75	1,775,474

Table 4.2: Description of mobile models we experiment with. α denotes width multiplier. MNetV2-4 contains encoder MobileNetV2 where we removed final convolution layer to reduce the number of parameters.

Mobile models are quite popular when it comes to real-time performance on devices with low computation resources. Since our robot lacks real-time prediction ability, we decided to search for a modern mobile model, that would allow the robot to deal with this problem. Based on previous works in this field, we test *ShuffleSeg* [41], *ShuffleNetV2* [44], *MobileNetV2* [19] and both large and small versions of *MobileNetV3* [38]. Although authors of MobileNetV2 and MobileNetV3 use other decoders in their original articles, we decided to use DeepLabV3+ [45] in our experiments. In fact, we were not able to reproduce learning ability of *Lite R-ASPP* as a decoder proposed for MobileNetV3 and DeepLabV3+ seems like a good fit for our purposes. In Table 4.2 we present architecture and hyper-parameters of mobile models. All model variations are trained using *Adam* optimizer and the training procedure took from 25 to 40 minutes in all cases.

name	train acc	train iou	test acc	test iou	on Jetson
SSeg-1	0.8776	0.8849	0.8862	0.8548	0.05559 sec
SSeg-2	0.8112	0.9141	0.8096	0.8565	0.07858 sec
SSeg-3	0.8442	0.8919	0.7391	0.8618	0.07048 sec
SSeg-4	0.7327	0.9166	0.7676	0.8581	0.08932 sec
SSeg-5	0.8013	0.9144	0.8060	0.8343	0.08605 sec
SNetV2-1	0.9811	0.9547	0.9506	0.8979	0.08615 sec
SNetV2-2	0.9731	0.9386	0.9375	0.8799	0.06318 sec
MNetV2-1	0.9757	0.9457	0.9433	0.8852	0.24385 sec
MNetV2-2	0.9760	0.9441	0.9424	0.8849	0.14537 sec
MNetV2-3	0.9746	0.9411	0.9341	0.8721	0.09902 sec
MNetV2-4	0.9757	0.9453	0.9396	0.8770	0.08698 sec
MNetV3-L-1	0.9711	0.9369	0.9486	0.8918	0.13953 sec
MNetV3-L-2	0.9744	0.9440	0.9549	**0.9000**	0.10666 sec
MNetV3-L-3	0.9734	0.9417	0.9485	0.8908	0.07009 sec
MNetV3-S-1	0.9733	0.9418	0.9506	0.8933	0.06125 sec
MNetV3-S-2	0.9739	0.9430	0.9536	0.8977	**0.05534** sec

Table 4.3: Results of mobile models trained on Deggendorf dataset along with their inference times measured on Jetson TX2.

Results of trained models are summarized in Table 4.3. ShuffleSeg's all modifications are quite fast, but their test IoUs are much lower compared to other models. SNetV2-1 gives us promising results with 11.5 FPS. All versions of MobileNetV2 do not reach test IoU higher than 89% and are not that fast as expected. With decreasing α the IoU decreases as well. MobileNetV3 seems to be most efficient one. Although its large version's test IoU is the best compared to all mobile models tested, the inference time is too high. A good trade-off between IoU and inference time provides MNetV3-L-3 with 14.2 FPS. But surprisingly, both small versions of MobileNetV3 beat other models with their significantly better performance. We are able to get to 18 FPS with unchanged IoU.

We picked one variation of each model which gives us the best trade-off between IoU and speed and compared them with our small models using AUC-ROC curve. The graph presented in Figure 4.2 shows that all MobileNets are the most powerful ones while ShuffleNet being equal with our modified ResNet and SegNet.

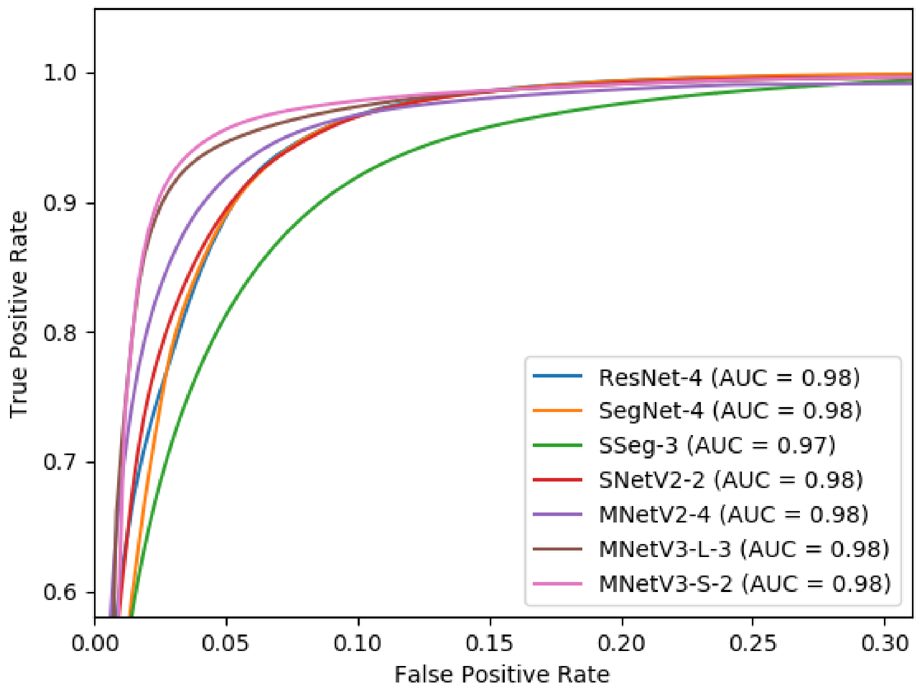


Figure 4.2: Comparison of mobile models using AUC-ROC curve. The graph is zoomed to the upper-left corner for better view.

Chapter 5

Active learning

In this chapter we experiment with the idea of decreasing the number of images needed to be labeled for training. We present various methods for sampling batches of images in order to make the training and labeling processes more efficient.

5.1 Motivation

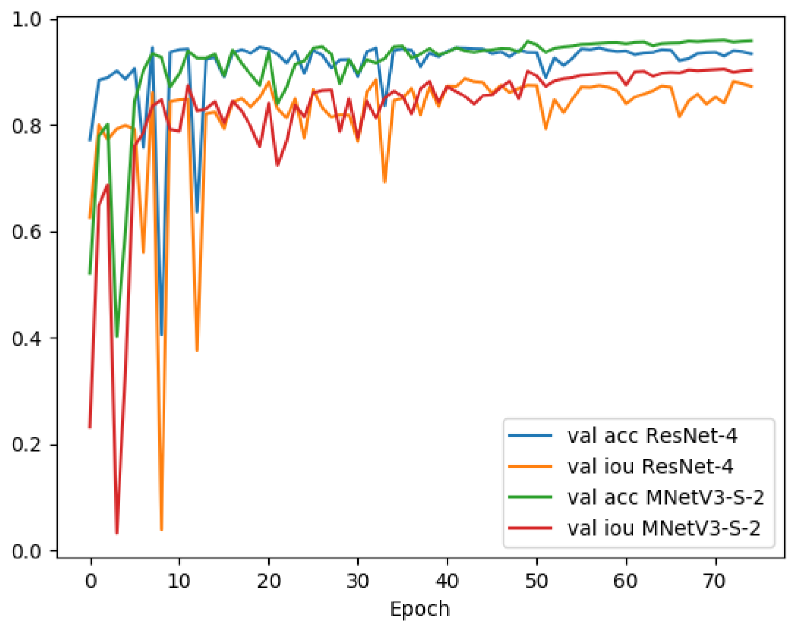


Figure 5.1: Active learning experiment with random sampling. Started with 30 random images and sampling 10 randomly in each round. Three epochs are executed within each round. Models: ResNet-4 and MNetV3-S-2. Y-axis represents IoU.

Each RoboTour contest is held annually in different locations. That means the environment always changes and datasets gathered and labeled at previous parks might not be sufficient. Creating new dataset does not only involve capturing images, but also labeling them in order to produce ground truths for our model. Since labeling entire dataset of images is often an expensive and time consuming process, we might ask if it is possible to label just a smaller portion of them and reach comparable accuracy.

Active learning may help us in such a situation. The key idea is to label a very small portion of images at the beginning and start training the model. At some point, the model asks for next portion of images to be labeled and continues learning. We may stop the training process once we think the validation loss has converged enough or some other stopping criterion has been met.

Our first experiment is presented in Figure 5.1. We started training with 30 random images and add 10 random images in each round. Validation accuracy and IoU converged after $\sim$ 45 epochs with only half of the training images used. It seems that this approach may help us in order to spend less time labeling.

We prepared an environment on the same machine with graphics processing unit where we conducted our previous experiments. In order to understand if it might be profitable to use active learning for training our models constrained to small amount of data, we decided to perform active learning simulations. We utilize *ResNet-4* and Deggendorf dataset in our experiments, but in Section 5.5 we show ablation study in which we compare this model to the other ones.

5.2 Random sampling

init	pick	reps	stop	imgs	rnds	eps	val acc	val iou	test acc	test iou
30	10	3	e-val 20	216	20	59	0.9521	0.9078	0.9316	**0.8728**
30	10	3	e-val 15	190	17	51	0.9536	0.9081	0.9308	0.8716
30	15	3	e-val 15	244	16	47	0.9509	0.9030	0.9278	0.8638
30	5	3	e-val 15	90	13	39	0.9395	0.8852	0.9238	0.8670
30	10	6	e-val 10	100	8	48	0.9472	0.9031	0.9155	0.8486
30	10	10	e-val 10	56	4	36	0.9414	0.8821	0.9103	0.8465
30	15	10	e-val 10	70	4	36	0.9306	0.8848	0.9079	0.8431
30	20	6	e-val 10	150	7	42	0.9493	0.9103	0.9247	0.8647
60	20	6	e-val 10	133	5	28	0.9400	0.8939	0.9187	0.8586

Table 5.1: Random sampling results. Average of three separate runs. *init*: number of images picked initially. *pick*: number of images picked after each round. *reps*: number of epoch in one round. *stop*: stopping condition. *imgs*: number of images involved in training. *rnds*: number of rounds in total. *eps*: number of epochs in total.

As a baseline, we sample images randomly from training data. We pick a bigger bulk of images initially and train for several round epochs (*reps*). Next samples are picked afterwards and also trained for the same number of epochs. Since our validation subset is deemed to be completely labeled, we are able to monitor validation loss. This information helps us to stop training sooner with early stopping condition. For example, *e-val 10* means we stop training if the validation loss did not change in last 10 epochs.

According to results in Table 5.1, the test intersection over union is lower by 2% on average than training the model on full dataset.

5.3 Entropy sampling

init	pick	reps	stop	imgs	rnds	eps	val acc	val iou	test acc	test iou
30	10	3	e-val 20	230	21	64	0.9529	0.9114	0.9320	0.8755
				231	22	65	0.9553	0.9089	0.9331	0.8782
30	10	3	e-val 15	203	18	55	0.9538	0.9133	0.9280	0.8688
				156	14	41	0.9490	0.8993	0.9273	0.8691
30	15	3	e-val 15	262	17	51	0.9575	0.9151	0.9375	**0.8815**
				252	16	48	0.9529	0.9047	0.9338	0.8759
30	5	3	e-val 15	111	17	52	0.9510	0.9071	0.9248	0.8636
				135	22	66	0.9512	0.9107	0.9307	0.8744
30	10	6	e-val 10	86	7	40	0.9442	0.8978	0.9210	0.8573
				70	5	30	0.9285	0.8830	0.9079	0.8456
30	10	10	e-val 10	66	5	46	0.9435	0.8931	0.9234	0.8652
				63	4	43	0.9363	0.8858	0.9143	0.8532
30	15	10	e-val 10	55	3	26	0.9358	0.8861	0.9049	0.8373
				65	3	33	0.9375	0.8847	0.9119	0.8508
30	20	6	e-val 10	123	6	34	0.9479	0.8989	0.9259	0.8678
				116	5	32	0.9441	0.8951	0.9220	0.8625
60	20	6	e-val 10	140	5	30	0.9516	0.9130	0.9305	0.8731
				133	5	28	0.9397	0.8852	0.9211	0.8586

Table 5.2: Entropy sampling results. Average of three separate runs. The first subrow in each row describes results of averaging entropy over pixels and the second subrow describes results of summing the entropy.

A straightforward way to improve random sampling might be to sample images based on the level of model's uncertainty predicting them. Before the images are picked, we need to sort them according to some score. A good candidate seems to be entropy. The output of the model is a 2D matrix with the same shape like input image, where each cell contains the number in range from 0 to 1. This number denotes the probability of pixel being driveable. Therefore, entropy of i-th pixel might be computed by the following formula:

$$H_i = -(p_i \log_2(p_i) + (1 - p_i) \log_2(1 - p_i))$$

where p_i denotes the probability of pixel being driveable. An aggregate function is applied afterwards to attach single score number to the image. In our case, we use *average* and *sum*. Table 5.2 captures results of both average and sum entropy sampling. There is no obvious winner between sum and average functions in entropy sampling because both perform quite similarly. Reducing the number of epochs for stopping condition to execute also reduces the number of rounds and images sampled as expected. *e-val 10* seems to be a good candidate for stopping since we do not end up labeling entire dataset. Batches of 20 images seem to work better in combination with *e-val 10*. If we compare these results, we can see that there is a very slight improvement over random sampling. However, we are able to get nice test IoU with only 50% images of entire dataset, which makes the labeling process more efficient.

In Figure 5.2 we demonstrate a simulation of how both entropy and validation IoU evolve with each round. Clearly, we can see that at some point the model does not learn pretty much no new information and the training procedure might be stopped.

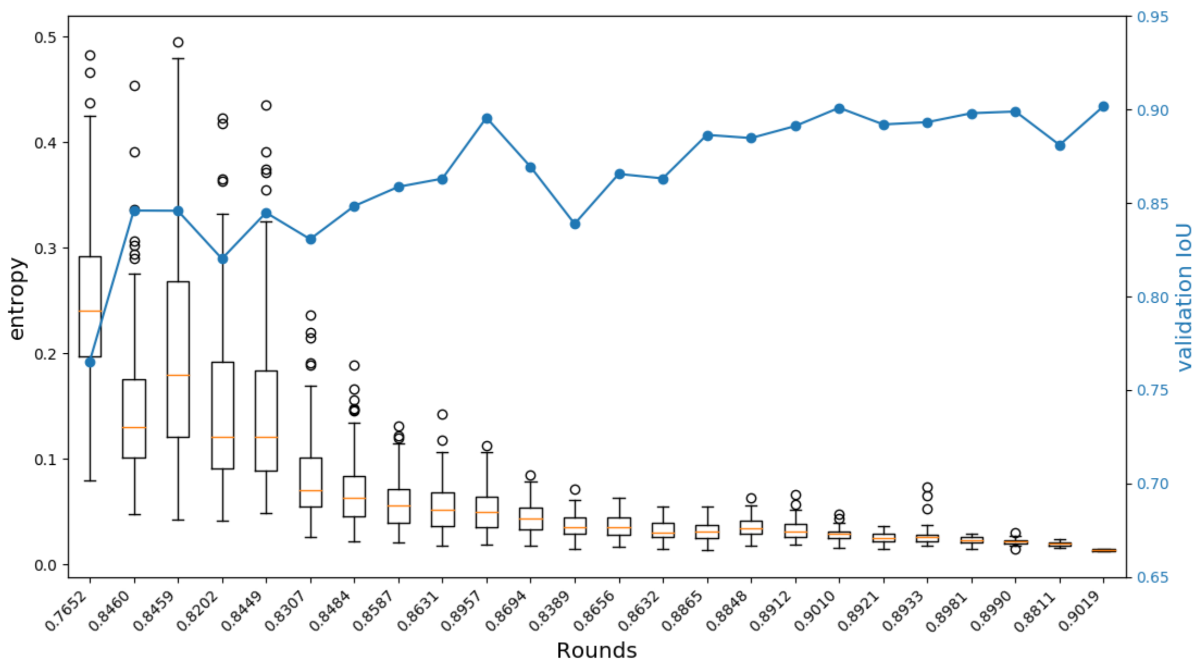


Figure 5.2: Comparison of entropies and validation IoU in each round. Started with 30 labeled images and within each round we sample 10 images. Entropies (averaged over pixels) are computed over all of the unlabeled images.

5.4 Diversity sampling

The problem with entropy (or uncertainty) sampling is that images with highest score might be in many cases similar and the change will not be as high as expected [52, 53, 55]. Therefore, before we actually sample images based on some score, we want to put them into categories based on their similarity. A good candidate for this task might be well known *K-means* algorithm described in Section 1.2.3.

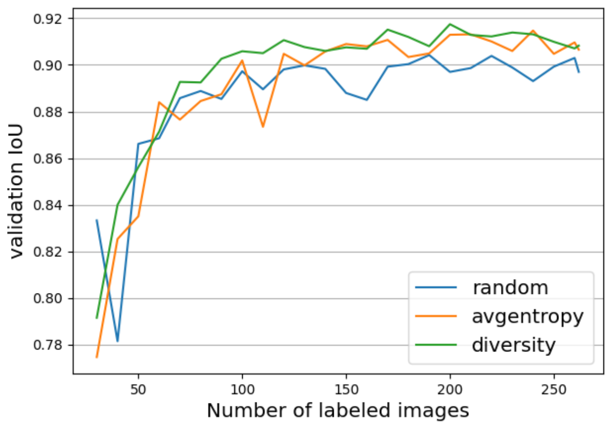
Within the model, encoder's role is to produce feature vector, which holds the information that represent the input. Hence, it is a good candidate to represent the image when running K-means. In case of the ResNet-4, the last encoder's layer produces 128 feature maps with resolution of 15×20. Each cell's value is set to be the average over all of the feature map's cells and the resulting matrix is flattened into one long vector with dimension of 300. The number of clusters, which is a hyperparameter in K-means setting, is set to be the number of images we sample within each round.

After running K-means from package called *Scikit-learn* [65], we compute entropies and sample one image with highest entropy per cluster. Table 5.3 shows the results of diversity sampling. It beats both random and entropy sampling. Moreover, results are more consistent so we do not end up with very low IoU.

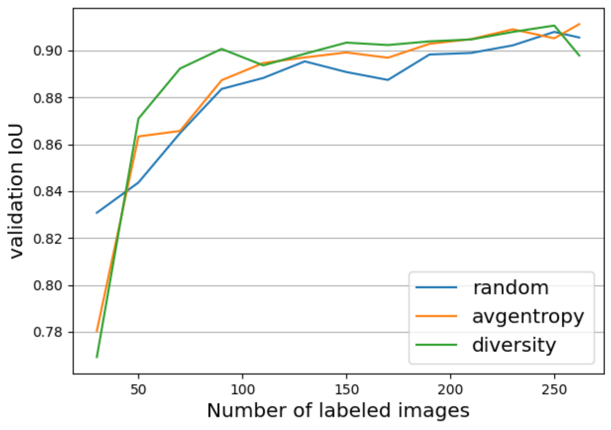
init	pick	reps	stop	imgs	rnds	eps	val acc	val iou	test acc	test iou
30	10	3	e-val 20	210	19	58	0.9522	0.9127	0.9320	0.8758
30	10	3	e-val 15	210	13	40	0.9486	0.9041	0.9206	0.8550
30	15	3	e-val 15	202	13	38	0.9501	0.9068	0.9213	0.8608
30	5	3	e-val 15	97	14	43	0.9477	0.8995	0.9228	0.8617
30	10	6	e-val 10	83	6	38	0.9441	0.8984	0.9224	0.8629
30	10	10	e-val 10	60	4	40	0.9442	0.8973	0.9168	0.8525
30	15	10	e-val 10	75	4	40	0.9443	0.8934	0.9197	0.8601
30	20	6	e-val 10	143	7	40	0.9517	0.9078	0.9261	0.8644
60	20	6	e-val 10	153	6	34	0.9503	0.9046	0.9306	**0.8770**

Table 5.3: Diversity sampling results. Average of three separate runs. The *pick* denotes the number of clusters as well as the number of images sampled in a round.

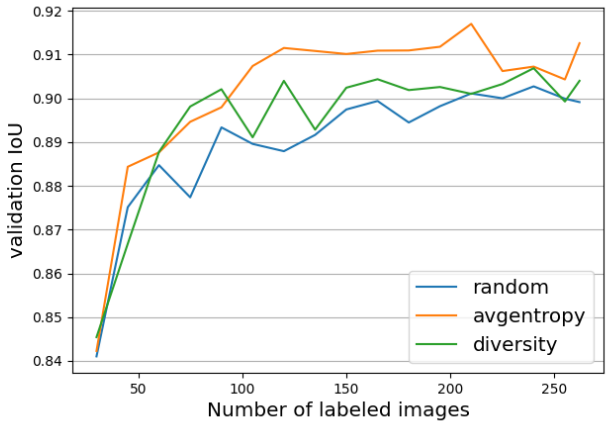
In order to compare these three aforementioned methods for sampling batches of images, we ran our experiments without early stopping condition. Three separate runs are conducted for each method within each experiment and the results are averaged. The results are presented in Figure 5.3. Clearly, both *average entropy* and *diversity* sampling perform comparably similar and better than the random sampling, and reach higher accuracy sooner. The difference is not very significant, but might help with efficient training in a setting with lower number of images in dataset. A small difference in IoU with fully labeled dataset is caused probably by not sufficient number of epochs the models were trained in last rounds.



(a) 10 samples and 6 epochs per round.



(b) 20 samples and 6 epochs per round.



(c) 15 samples and 10 epochs per round.

Figure 5.3: Comparison of sampling methods based on three separate experiments.

The initial batch of images is still being sampled randomly, so we would like to test whether choosing images in this initial batch might be done more wisely. The input image contains $640 \cdot 480 \cdot 3 = 921600$ pixels. This would be a very high dimensional vector as an input to K-means and thus dimension reduction is essential in this case. Recently, *UMAP* [66] (Uniform Manifold Approximation and Projection) algorithm for dimension reduction has been proposed. It is competitive with *t-SNE* [67] in terms of visualization quality, while preserving more of the global structure with superior run time performance.

Fortunately, there is UMAP python package [68] available for a free usage. All the images available in the train subset are loaded into memory and normalized to values in the range from 0 to 1. Channels are subsequently flattened into one long vector and dimension is reduced by UMAP algorithm. The output is then fed to the K-means.

After a couple of experiments with UMAP, we found out that in our case of such a small dataset, there is no improvement at all, since such a small portion of initial images does not affect the overall efficiency.

5.5 Ablation study

Experiments in sections above are conducted with ResNet-4 minimized model. In this Section, we test MNetV3-S-2 and SNetV2-1 in active learning setting. We tested both models within four experiments, see Table 5.4.

model	init	pick	reps	stop	imgs	rnds	eps	val acc	val iou	test acc	test iou
MNetV3-S-2	30	10	6	e-val 10	117	10	58	0.9391	0.8912	0.9245	0.8640
	30	15	10	e-val 10	125	7	73	0.9439	0.8980	0.9319	0.8704
	30	20	6	e-val 10	183	26	52	0.9449	0.9044	0.9231	0.8668
	60	20	6	e-val 10	227	9	56	0.9535	0.9131	0.9394	**0.8799**
SNetV2-1	30	10	6	e-val 10	123	10	62	0.9475	0.9046	0.9298	0.8680
	30	15	10	e-val 10	135	8	80	0.9521	0.9114	0.9351	0.8772
	30	20	6	e-val 10	197	9	56	0.9476	0.8943	0.9381	0.8799
	60	20	6	e-val 10	234	10	60	0.9506	0.9079	0.9398	**0.8805**

Table 5.4: Results of MNetV3-S-2 and SNetV2-1 in active learning setting. Average of three separate runs. For clarification of the header see Table 5.1.

The results of both models are comparably similar to the ResNet-4. The only significant change can be seen in the second row where the model uses much less data and performs good enough. The mobile models make use of more images on average. That might be caused by the number of parameters, which is almost four times higher and the early stopping condition is met later because the models still manage to update some of their weights, leading to slightly better accuracy.

Conclusion

In this work we focused on the problem of classifying pixels of the image into two categories, namely driveable segments and non-driveable ones. This problem has been raised by the need for a better vision module for the robot named Smelý Zajko. Students annually participate at the competition called RoboTour Outdoor Delivery Contest with this robot. The robot had been designed, built and continuously improved in previous works. However, the vision module did not take that much attention and we could not rely on predictions and information about driveable path. The weather and diversity of the environment are the main causes why previous approaches have not been successful enough. Based on recent advances with deep convolutional neural networks applied to computer vision, we decided to replace the robot's vision module with CNN models.

We made a survey of the best contemporary ConvNet architectures for semantic segmentation and chose some of them to be tested on our robot. Training was conducted in supervised manner, meaning we provided images along with their ground truth labels to the models. There are two datasets of images and labels released along with this thesis. The images in these datasets were taken prior to RoboTour contests in Lednice and Deggendorf.

Models trained on these images performed relatively well, so we chose ResNet to be used in Deggendorf since it showed the best trade-off between accuracy and prediction speed. The robot had some issues with localization and planning algorithm in the beginning, but in the end we were able to fix it and the robot worked fine. As the only one it drove beyond load point in that particular round. We obtained 13 points in this competition, which brought us second place overall. However, these models were not fast enough in terms of prediction time and we tried to minimize them by reducing the number of filters and removing some layers. This step allows us to run the robot at higher speed. We compared these minimized models to the so called mobile models that have been proposed recently. Interestingly, small version of MobileNetV3 reached almost 20 frames per second while our small ResNet topped at 10 frames per second, which is also nice and quite efficient at the image resolution of 640×480.

Labeling entire dataset is often an expensive and time consuming process. We used Active Learning to study whether we are able to train the models with less labeled images while preserving reasonable accuracy. The models started to train on a small subset of training images and queried another ones to be labeled after each round. This process was repeated until the stopping condition was met. We used three sampling strategies, namely random, entropy and diversity sampling. Their detailed description is presented in Chapter 5. Diversity and entropy methods performed better than random sampling and needed only half of the training dataset to reach an accuracy comparable with that of the models trained on the full dataset. Thus, we can conclude that it is worth considering to use active learning when training the model.

In the future, we could study if it is possible to get rid of the labeling process completely and train the model in a fully unsupervised way. It is currently a deeply studied topic and the first works [69, 70, 71] have been proposed recently at the NeurIPS conference. We also bought ZED mini stereo camera capable of creating depth map, tracking the position and giving us odometry information. It would be worth trying to incorporate the depth map into prediction of driveable trail, which could further improve the accuracy.

Bibliography

[1] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017.

[2] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[3] Burr Settles. Active Learning Literature Survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[4] Yu-Jin Zhang. Image segmentation in the last 40 years. In *Encyclopedia of Information Science and Technology, Second Edition*, pages 1818–1823. IGI Global, 2009.

[5] Song Yuheng and Yan Hao. Image Segmentation Algorithms Overview. *arXiv preprint arXiv:1707.02051*, 2017.

[6] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition. `http://cs231n.github.io/`. Accessed 2018-02-02.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of the 30th International Conference on Machine Learning*, volume 30, page 3, 2013.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[11] Richard S Sutton, Andrew G Barto, et al. *Introduction to Reinforcement Learning*, volume 135. MIT press Cambridge, 1998.

[12] Yurii Nesterov. A method of solving a convex programming problem with convergence $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[13] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Meet Shah. Semantic Segmentation using Fully Convolutional Networks over the years. `https://meetshah1995.github.io/semantic-segmentation/deep-learning/pytorch/visdom/2017/06/01/semantic-segmentation-over-the-years.html`. Accessed 2019-02-02.

[15] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

[16] Matthew D Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[17] 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.

[18] Fisher Yu and Vladlen Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[19] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[20] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015.

[21] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural computation*, 1(4):541–551, 1989.

[22] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: `http://yann.lecun.com/exdb/mnist`*, 2, 2010.

[23] Ofer Matan, Christopher JC Burges, Yann LeCun, and John S Denker. Multi-Digit Recognition Using A Space Displacement Neural Network. In *Advances in neural information processing systems*, pages 488–495, 1992.

[24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.

[25] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[27] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[28] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[29] 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.

[30] 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.

[31] 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.

[32] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[33] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[35] Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving. *arXiv preprint arXiv:1612.07695*, 2016.

[36] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[37] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[38] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[39] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[40] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[41] Mostafa Gamal, Mennatullah Siam, and Moemen Abdel-Razek. ShuffleSeg: Real-time Semantic Segmentation Network. *arXiv preprint arXiv:1803.03816*, 2018.

[42] Mennatullah Siam, Mostafa Gamal, Moemen Abdel-Razek, Senthil Yogamani, Martin Jagersand, and Hong Zhang. A Comparative Study of Real-Time Semantic Segmentation for Autonomous Driving. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[43] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[44] Sercan Türkmen and Janne Heikkilä. An efficient solution for semantic segmentation: ShuffleNet V2 with atrous separable convolutions. In *Scandinavian Conference on Image Analysis*, pages 41–53. Springer, 2019.

[45] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[46] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for Efficient Multi-Scale Architectures for Dense Image Prediction. In *Advances in neural information processing systems*, pages 8699–8710, 2018.

[47] Michael Sörsäter. Active Learning for Road Segmentation using Convolutional Neural Networks. Master's thesis, Linköping University in Linköping, Sweden, 2018.

[48] Ajay J Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. Multi-Class Active Learning for Image Classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2372–2379. IEEE, 2009.

[49] Alexander Vezhnevets, Joachim M Buhmann, and Vittorio Ferrari. Active Learning for Semantic Segmentation with Expected Change. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3162–3169. IEEE, 2012.

[50] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. Cost-Effective Active Learning for Deep Image Classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600, 2016.

[51] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep Bayesian Active Learning with Image Data. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1183–1192. JMLR. org, 2017.

[52] Ozan Sener and Silvio Savarese. Active Learning for Convolutional Neural Networks: A Core-Set Approach. *arXiv preprint arXiv:1708.00489*, 2017.

[53] Lin Yang, Yizhe Zhang, Jianxu Chen, Siyuan Zhang, and Danny Z Chen. Suggestive Annotation: A Deep Active Learning Framework for Biomedical Image Segmentation. In *International conference on medical image computing and computer-assisted intervention*, pages 399–407. Springer, 2017.

[54] Radek Mackowiak, Philip Lenz, Omair Ghori, Ferran Diego, Oliver Lange, and Carsten Rother. Cereals - cost-effective region-based active learning for semantic segmentation. *arXiv preprint arXiv:1810.09726*, 2018.

[55] Fedor Zhdanov. Diverse mini-batch Active Learning. *arXiv preprint arXiv:1901.05954*, 2019.

[56] Miroslav Nadhajský. Robotour. Master's thesis, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Slovakia, 2011.

[57] Michal Moravčík. Autonomous mobile robot for Robotour contest. Master's thesis, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Slovakia, 2013.

[58] Jozef Dúc. Robotour with Laser Range Sensor. Master's thesis, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Slovakia, 2017.

[59] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[60] Ondrej Jariabka, Marek Šuppa, and Ondrej Rudolf. Single Camera Path Detection for Outdoor Navigation. In *Proceedings of CESCG 2017: The 21st Central European Seminar on Computer Graphics*. CESCG, 2017.

[61] Michal Fikar. Local map for a robot for the Robotour contest. Master's thesis, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Slovakia, 2019.

[62] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, Weng Chi-Hung, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug. `https://github.com/aleju/imgaug`, 2019. Accessed 2019-01-22.

[63] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[64] François Chollet et al. Keras. `https://keras.io`, 2015.

[65] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine Learning in Python. *the Journal of Machine Learning Research*, 12:2825–2830, 2011.

[66] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv preprint arXiv:1802.03426*, February 2018.

[67] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[68] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. UMAP: Uniform Manifold Approximation and Projection. *The Journal of Open Source Software*, 3(29):861, 2018.

[69] Mickaël Chen, Thierry Artières, and Ludovic Denoyer. Unsupervised Object Segmentation by Redrawing. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 12726–12737. Curran Associates, Inc., 2019.

[70] Tam Nguyen, Maximilian Dax, Chaithanya Kumar Mummadi, Nhung Ngo, Thi Hoai Phuong Nguyen, Zhongyu Lou, and Thomas Brox. DeepUSPS: Deep Robust Unsupervised Saliency Prediction via Self-supervision. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 204–214. Curran Associates, Inc., 2019.

[71] Adam Bielski and Paolo Favaro. Emergence of Object Segmentation in Perturbed Generative Models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 7256–7266. Curran Associates, Inc., 2019.

Appendix A

Attached CD contains the source code and datasets, which are also published on GitHub at the following url: `https://github.com/Adman/road-segmentation`. The folder `data/datasets` contains both datasets along with their ground truth labels. The file `README.md` describes how to setup the environment and install dependencies as well as how to run the training process or active learning simulations.