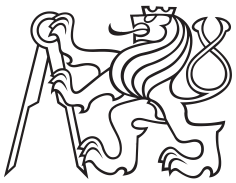
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Deep neural network for city mapping using Google Street View data

Varun Burde

Supervisor: Ing.Michal Reinštein,Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Robotics
January 2020

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Burde Varun** Personal ID number: **478596**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Deep neural network for city mapping using Google Street View data

Master's thesis title in Czech:

Hluboká neuronová sít pro mapování města s využitím dat z Google Street View

Guidelines:

The aim is to design, implement and experimentally evaluate a deep neural network based solution for city mapping using Google Street View images. The proposed software solution should allow the user to request Google Street View imagery for any location, perform analysis and feature extraction using deep neural network(s) and output vectorized description projected and visualized over the underlying map.
Instructions are as follows:
1. Study the state-of-the-art literature relevant to the thesis [1-7].
2. Explore the TensorFlow framework [7] and use it with Python to design, implement and evaluate a deep neural network model [3].
3. For experimental evaluation use publicly available datasets; existing pre-trained models should be explored first.
4. Design and implement user interface for the application execution, processing of the input images and visualization of results; Google Colab utilising TPUs is recommended.
5. Compare the results with related state-of-the-art work [4, 5, 6].

Bibliography / sources:

[1] Goodfellow, Ian, et al. „Deep Learning", MIT Press, 2016
[2] Szegedy, Christian, et al. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning." AAAI. 2017. APA
[3] He, Kaiming, et al. "Mask R-CNN" arXiv preprint arXiv:1703.06870 (2017).
[4] Liu, Ming-Yu, et al. "Layered interpretation of street view images." arXiv preprint arXiv:1506.04723 (2015).
[5] Kang, Jian, et al. "Building instance classification using street view images." ISPRS Journal of Photogrammetry and Remote Sensing (2018).
[6] Law, Stephen, Brooks Paige, and Chris Russell. "Take a look around: using street view and satellite images to estimate house prices." arXiv preprint arXiv:1807.07155 (2018).
[7] Abadi, Martın, et al. "TensorFlow: Large-scale machine learning on heterogeneous systems, 2015." Software available from tensorflow.org.

Name and workplace of master's thesis supervisor:

Ing. Michal Reinštein, Ph.D., Vision for Robotics and Autonomous Systems, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **24.01.2019** Deadline for master's thesis submission: **07.01.2020**

Assignment valid until:
by the end of summer semester 2020/2021

Ing. Michal Reinštein, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I want to acknowledge the help of all of those who made this project possible. I want to start by thanking my parents for their unconditional love and support during my thesis. I want to express my sincere gratitude to my supervisor Ing. Michal Reinštein, Ph.D., for his time, patience, guidance, and also for allowing the idea to persuade originally, and made this project successful. Furthermore, I would like to thank all those people who work on all open source projects mentioned in the reference. And all generous people who post the discussion and blogs for all useful learning resources. I am also thankful for Google LLC, who is offering free resources like Google Colab laboratory and API for the emerging developer.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, January , 2020

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, . ledna 2020

Abstract

With the advancement of computation power and large datasets, a massive improvement of the deep neural network leads to many widespread applications. One of the applications of the deep neural network is solving computer vision problems like classification and segmentation. Competition like ImageNet[1] Large Scale Visual Recognition Challenge[2], took the capability to the next level; in some cases, classification is better than human.

This thesis is an example of an application that utilizes the ability of neural networks. The document describes the implementation, methodology, experiments done for developing software solutions by using the deep neural network on image resources form Google Street View images [3].

The user provides a geojson file consists of an area of interest in the form of square or polygon as the input. Google StreetView API [3] downloads the available images. The images are first processed with the state of the art CNN (Mask R-CNN[4]) to detect the objects, classify them with the confidence score, generate a bounding box, and a pixel-wise mask around the detected object. The text file stores information like coordinates of the bounding box, name of the class, and the mask values.

An ordinary RGB (panoramic) image from GSV does not consist of any depth data. The images are processed with another state of art CNN (monodepth2[5]), to estimate the pixel-wise depth of the objects in the images.

The averaged value of the depth within the mask is used as the distance of the object. The coordinates of the bounding box are used for positioning of the object in other axes.

The resulting outputs are markers of detected objects underlying in the map. A bar graph to visualize the number of detection per class. A text file containing the number of detection per each class. The output from each processing step above, like detections, depth images, mask values to compare and evaluate.

Keywords: Google Street View, Mask R-CNN, Monodepth2, Object detection, Deep neural network, City mapping

Supervisor: Ing.Michal Reinštein,Ph.D.
E225b,
Karlovo nam. 13,
121 35 Prague 2,
Czech Republic

Abstrakt

S rozvojem výpočetní síly a rozsáhlými datovými soubory vede masivní zlepšení hluboké neuronové sítě k mnoha rozšířeným aplikacím. Jednou z aplikací hluboké neuronové sítě je řešení problémů počítačového vidění, jako je klasifikace a segmentace. Soutěž jako ImageNet [1] Výzva pro vizuální rozpoznávání ve velkém měřítku [2] posunula schopnost na další úroveň; v některých případech je klasifikace lepší než lidská.

Tato práce je příkladem aplikace využívající schopnost neuronových sítí. Dokument popisuje implementaci, metodiku, experimenty prováděné pro vývoj softwarových řešení pomocí hluboké neuronové sítě na obrázkových prostředcích z obrázků Google Street View [3].

Uživatel poskytuje soubor geojson sestávající z oblasti zájmu ve tvaru čtverce nebo mnohoúhelníku jako vstup. Google StreetView API[3] stáhne dostupné obrázky. Snímky jsou nejprve zpracovány pomocí nejmodernějších CNN (Mask R-CNN [4]), aby detekovaly objekty, klasifikovaly je pomocí skóre spolehlivosti, vytvořily ohraničující rámeček a kolem detekovaného objektu malovaly pixely. . Textový soubor ukládá informace, jako jsou souřadnice ohraničovacího rámečku, název třídy a hodnoty masky.

Obyčejný RGB (panoramatický) snímek z GSV neobsahuje žádné hloubkové údaje. Obrázky jsou zpracovávány s jiným nejmodernějším CNN (monodepth2[5]), aby se odhadla hloubka objektů v obrazech po pixelech.

Průměrná hodnota hloubky v masce se používá jako vzdálenost objektu. Souřadnice ohraničovacího rámečku se používají pro umístění objektu v jiných osách.

Výsledné výstupy jsou markery detekovaných objektů, které jsou základem mapy. Sloupcový graf pro vizualizaci počtu detekcí ve třídě. Textový soubor obsahující počet detekcí pro každou třídu. Výstup z každého kroku zpracování výše, jako jsou detekce, hloubkové obrázky, hodnoty masky pro porovnání a vyhodnocení.

Klíčová slova: Google Street View, Mask R-CNN, Monodepth2, Detekce objektů, Hluboká neuronová síť, Mapování města

Contents

Figures

Tables

List of Abbreviations

Abbrevation	Full form
DNN	Deep Neural Network
GSV	Google Street View
CNN	Convolutional Neural Network
COCO	Comman Objects in Context
API	Application Program Interface
FCNN	Fully Convolutional Neural Network
RGB	Red Green Blue

Chapter 1
Introduction

1.1 Motivation

In recent years, many applications have been developed using neural networks. Especially the use of a CNN for image processing has opened the doors of solving computer vision problems[19] with computers. Maps provide important information to the user in terms of navigation and the landmarks. Productivity can be further improved by adding more features to the map. It can be either a bus station, a post box, or any object of interest. Google has been providing street images in form GSV images for a long time. These images carry a big set of information that can be transformed to create a useful application. With the unlimited possibility of applications using GSV images, the thesis describes a novel solution of mapping the whole city using the DNN. An approach of using neural networks to find features from images and placing over the underlying map in the form of a marker.

1.2 Aim and objective of the thesis

The aim is to design, implement, and experimentally evaluate a deep neural network-based solution for city mapping using Google Street View images [3]. The proposed software solution should allow the user to request Google Street View imagery for any given location specified as geojson[20], perform analysis and feature extraction using the deep neural network(s) and output vectorized description projected and visualized over an underlying map. The user interface for the application execution, processing of the input images, and visualization of the results should be realized using Google Colab [21] to utilize Google TPUs. Existing pre-trained models should be explored. First, a thorough experimental evaluation of publicly available datasets should follow. Comparison with related state-of-the-art is an integral part of the work and should be presented in the final thesis. Recommendation: implementation should be done in Python [22], using Keras [23] and TensorFlow [24] frameworks.

1.3 Overview of Thesis

The thesis work is a software solution to achieve the task described above, using the state-of-the-art DNN(s) [25] and data set from GSV [3].

The core implementation of the thesis depends on the GSV images [26]. GSV API for Python [22] is used to download the available GSV images within the area specified by the coordinates.

Downloaded images are then processed with DNN for classification and segmentation. For segmentation and classification, Mask R-CNN[4] with architecture developed in Keras[23] framework with TensorFlow [24] as computational backend is used. The pre-trained model with COCO[27] dataset, which consists of 81 classes, are used. The resulted images consist of classified objects with their confidence score by their class, localized within a bounding box, and segmentation with the colored mask.

The depth of detected objects is predicted with another state-of-the-art DNN [5]. Using the estimated depth of the object respect to the GSV image, objects are inserted on the map. The visualization of objects is expressed in the form of an overlay of markers on the map. The output geojson file consists of objects with their properties and classes separated by a different color. A bar graph contains information of the number of detection per class to reflect the understanding of the scenario.

1.4 Structure of thesis

Chapter 2, **Related work**, covers the latest state-of-the-art approaches related to the topic of this thesis.

Chapter 3, **Theory**, describes the conceptual knowledge about the algorithm and tools used within the thesis.

Chapter 4,**Implementation**, describes the created pipeline and its description of each block in terms of data and its processing.

Chapter 5, **Methodology**, this chapter describes the methods and algorithms used to solve the tasks mentioned in the chapter implementation. The comprehensive description of transformation and manipulation of data, what approaches and parameters chosen for the tools are described in this section.

Chapter 6, Experimental **Evaluation**, involves examining the outputs from the state-of-the-art networks. The performance of different tools is shown during there development.

Chapter 7, **Results** from the outputs are described — the visual representation of output in terms of their features and properties with various tools. The shortcomings of the solution are mentioned in this chapter.

Chapter 8, **Conclusion** this chapter describes the usability and applications of the software solution with different approaches for various problems, future scope, and aim fulfillment.

Chapter 2

Related Work

The smart way of utilizing Google Street View images[3] as dataset using deep learning algorithms[25], to develop a software solution has been seen in the last few years. One such example "Google Street View image of a house predicts car accident risk of its resident", where images of the house from GSV is used to annotate house feature like age, type and condition manually and this data is used to predicts car accident risks of its resident using probabilistic model[28]. Another such application “Take a Look Around: Using StreetView and Satellite Images to Estimate House Prices”, where GSV and satellite images are used to extract features like age, size and accessibility as a visual feature and using DNN to estimate the house prices [29] .

With the popularity of autonomous driving [30], layered interpretation of GSV images, [31] was developed in Mitsubishi Electric Research Labs with the use of DNN on GSV images. In the paper, the author planned a stratified street model to encode depth and semantic data on the street pictures for autonomous driving. The author proposes a 4-layer street model, layers categories as the ground, pedestrians, vehicles, buildings, and sky. The input used for the experiment was the pair of stereo images. The deep neural network was used to extract the appearance features for semantic classes.

Another example of an application developed with GSV and deep neural network is building instance classification using Street View images[32]. The author projected a general framework for classifying the practicality of individual buildings. The projected technique relies on Convolutional Neural Networks (CNNs), that classify facade structures from Street View pictures, additionally to remote sensing pictures, that sometimes solely show roof structures. Geographic info was used to mask out individual buildings and to associate the corresponding street view images. Additionally, the tactic was applied to get building classification maps on each region and town scales of many cities in the USA.

One example which is similar to work done in the thesis is Automatic Discovery and Geotagging of Objects from GSV imagery [27]. This paper describes the solution to localize the object from multiple views using

geometry[33]. To geolocate the object in an image, they developed a Markov Field model to perform object triangulation. They use two state-of-the-art FCNN for semantic segmentation and monocular depth estimation of the object of interests. The geolocalization is done with Google street view images with Triangular based MRF(Markov random field) model described in the paper. The result from the DNN is a map with an overlay of tags on the map. The algorithm requires images from two or more different location to complete MRF based localization.

Photo localization with deep neural network [34] (Deepgeo), another such example of the application of the deep neural network. The author uses a deep neural network and trains it over the panoramic image of Google Street View. The outcome is the prediction of the location of the image. It's trained with the 50 states10K[35] datasets, which they created with the GSV API. The author presented Resnet[12] architecture with three types of integration, and their results. The early integration, in all four views, are concatenated to form twelve channels. It allows information to be shared between images in all layers. In medium integration, the feature is extracted from each image before concatenation. In late integration, along with feature extraction, a layer of the perceptron is connected to integrate the prediction, which simply takes the maximum over each output class with the max pool layer. The conclusion shows the results of the game GeoGuessr with a proposed solution against humans. With the best variant of the network out of 5 games, the neural network outperforms the human.

Geolocation by embedding maps and images, show similar work done [36]. Here the author presents an approach to geolocate images on a 2D map based on learning a low dimensional embedded space. The neural network is trained with GSV panoramic images cropped with different angles along with the geolocated map tiles. The map tile is taken from an OpenStreetMap [37], which consists of the visible junction, building, and green areas on a map illustrating semantic features that leverage for geolocation. The network has two independent sub-networks, one for location images and other for map tiles. First sub-network which process location image extracts the feature and based on Resnet50[12] architecture. In the sub-network top layer is removed and coupled a trainable NetVLAD layer[38]. The other sub-network extracts the features from the map and has similar architecture. Instead of using Resnet50, Resnet18 is used and coupled with the NetVLAD layer. In both sub-networks, projection modules go through the same layers, which reduce the dimensional of descriptor down to the embedding size and help to project semantically similar input near to each other. The results show the methodology to correlate 360-degree location images and 2D cartographic map tile into a common low dimensional space using a deep learning approach.

Performance of state-of-the-art CNN(s) for segmentation has been evaluated in following project. Identification of cell nuclei based on deep neural

	Input size	Output size	Mean average precision(mAP)
U-Net[39]	512x512x3	128x128x1	0.325
Mask R-CNN[39]	512x512x3	56x56x1	0.476
DenseUNet[41]	512x512x3	128x128x1	0.442

Table 2.1: Segmentation results on data science bowl 2018 challenge[17]

network[17] shows evaluation of three neural networks for segmenting cell nuclei in images; Mask R-CNN [4], U-Net[39], Denset[40]. The task was to segment each cell nuclei and background with the use of 640 microscopic images. The Mask R-CNN shows the best results with 0.476 mean average precision for the segmentation. The detailed results can be inferred from table 2.1

Chapter 3

Theory

3.1 Image Classification

Image classification is the process of classifying an image based on its visual features present in the raster. It is a task of identifying whether the given visual feature is present in the image or not. It can be done by finding the relationship between the nearby pixels. The relationship of the nearby pixel can be calculated using classifiers; one way is to compare images using the nearest neighbor classifier in which pixel-wise absolute value differences can be used to show the relationship between two images [19]. The figure 3.1 is an example in which the object is classified as the predefined class car with a confidence score of 0.98.

3.2 Semantic Segmentation

Image segmentation is the process of partitioning of digital image into multiple segments for further analysis. The pixels of the image are organized into higher-level units that are either more meaningful or more efficient for further analysis (or both). Figure 3.2 shows the example of semantic segmentation.

3.3 Instance segmentation

Instance segmentation is the task of semantic segmentation with the identification of boundary at the detailed pixel level for each classified object.

Figure 3.3 is an example of instance segmentation of different color segmentation masks. For example, the bus is segmented with a green mask.

3.4 Feature extraction

Feature extraction is the process of transforming the pixel data of an image to a set of feature points or something more meaningful, which can be used in other techniques, such as point matching or machine learning, and using

Figure 3.1: Example of image classification where the object is classified as the car in the image

point matching.

3.5 Neural Network

A neural network is made up of a set of connecting units or nodes called neurons. A neuron is the basic unit of the neural network, which is simple models like linear, logistic regression.

Consider a neural network model built from a linear regression model.

3.5.1 Regression task

In supervised learning, linear regression is the task of creating a linear model by finding the relationship between inputs (independent variable) and the output (dependent variable).

Consider the input variable

$$x = (x_1 \dots x_D) \tag{3.1}$$

and output variable y

Linear regression is a function which is made to learn the relationship between input and output is given by

$$\widehat{y} = h(x) = w_0 + w_1 x_1 + \ldots + w_D x_D = w_0 + \langle w, x \rangle = w_0 + x w^T [7] \tag{3.2}$$

Figure 3.2: Semantic segmentation, where the girl and horse are segmented from the whole image[6]

where,
$\widehat{y}$ is model, $h(x)$ is hypothesis, $w_0 \dots w_D$ and other are weights, $\langle w, x \rangle$ is the dot product of vector w and x

Often the data are being represented in homogeneous coordinates and matrix notation by

$$\boldsymbol{X} = \begin{pmatrix} 1 & x^{(1)} \\ \vdots & \vdots \\ 1 & x^{(|T|)} \end{pmatrix} \tag{3.3}$$

$$y = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(|T|)} \end{pmatrix} \tag{3.4}$$

An accurate model can be created by estimating the value of weights. Training set T = (X,y) consists of a set of known inputs with their output and used to train the model.

Learning is the process of finding such a model parameter w^*, which minimizes the certain loss function.

$$w^* = \underset{w}{\operatorname{argmin}} J(w, T)[7] \tag{3.5}$$

3.5.2 Loss function

The function we want to minimize in order to get a low error rate for the training data is called the loss function. The loss function reduces all the aspects of a possibly complex system (dense or deep network) down to a

Figure 3.3: Instance segmentation of class bus with the green mask

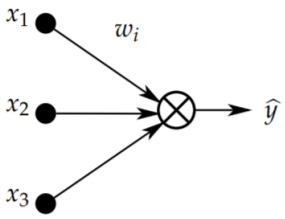


Figure 3.4: Multiple regression model as linear neuron [7]

single scalar value, which allows solutions to be ranked and compared[42].

The minimum of loss function can be found using numerical optimization techniques like mean square error which is given by,

$$J_{MSE}(\boldsymbol{w}) = \frac{1}{|T|}\sum_{i=1}^{|T|}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 [7] \tag{3.6}$$

where |T| is the set of training examples given by

$$T = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{|T|} \tag{3.7}$$

The simple linear regression task can be further combined together to make a multiple linear regression model. Consider the neuron in figure 3.4 with three inputs and one node.

The loss function $J(w)$ can be minimized using Gradient descent algorithm is given by

$$\begin{aligned} w &\leftarrow w - \eta \nabla J(w), \text{ i.e.} \\ w_d &\leftarrow w_d - \eta \frac{\partial}{\partial w_d} J(w) \end{aligned} \tag{3.8}$$

where η is learning rate

Loss function for training, with T number of examples ,

$$J(\boldsymbol{w}) = \sum_{i=1}^{|T|} E\left(\boldsymbol{w}, \boldsymbol{x}^{(i)}, y^{(i)}\right) \tag{3.9}$$

To understand the concept for calculating error function for T number of examples, let's find the loss function for a single training example and assuming the squared error loss.

$$E(\boldsymbol{w}, \boldsymbol{x}, y) = \frac{1}{2}(y - \widehat{y})^2 = \frac{1}{2}\left(y - x\boldsymbol{w}^T\right)^2 \text{ [8]} \tag{3.10}$$

Finding the derivative of loss function using the chain rule

$$\begin{aligned} \frac{\partial E}{\partial w_d} &= \frac{\partial E}{\partial \widehat{y}} \frac{\partial \widehat{y}}{\partial w_d}, \text{ where} \\ \frac{\partial E}{\partial \widehat{y}} &= \frac{\partial}{\partial \widehat{y}} \frac{1}{2}(y - \widehat{y})^2 = -(y - \widehat{y}), \text{ and [8]} \\ \frac{\partial \widehat{y}}{\partial w_d} &= \frac{\partial}{\partial w_d} x w^T = x_d \end{aligned} \tag{3.11}$$

which gives,

$$\frac{\partial E}{\partial w_d} = \frac{\partial E}{\partial \widehat{y}} \frac{\partial \widehat{y}}{\partial w_d} = -(y - \widehat{y}) x_d [8] \tag{3.12}$$

The process can be iterated over batch of training example with a Gradient descent algorithm given by equation 3.8 to find the optimum value for the weights.

These neurons are grouped together to form a layer which is connected to each other. These layers change their weight during the process of learning. Training changes the weight of each layer to create a filter that allows the specific form of features to pass and thus can be used as a feature detector. Making these network deep enough and combining with several forms of different layer makes it possible to create a filter which can detect very complex features.

A neural network should have at least three layers: input layer, hidden(can be one or many) layers and output layer.

Figure 3.5 shows the structure of the three layers neural network.

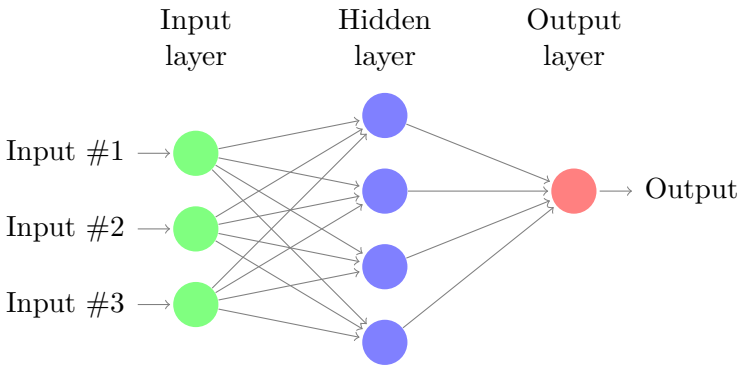


Figure 3.5: Structure of three layers of neural network
Source: Created with LaTex package TikZ [43]

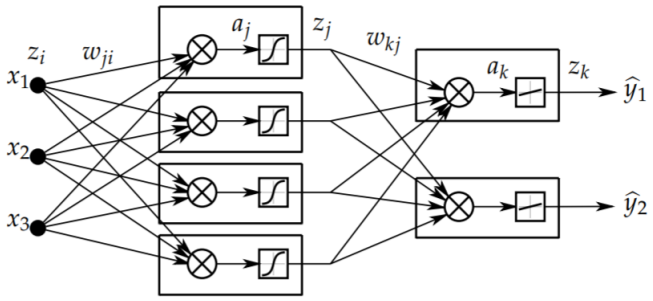


Figure 3.6: Three layer neural network with parameter[8]

The layer between the starting point (input) and the endpoint (output) layer is called a hidden layer. There are layers which can be trained called trainable layer and layer which can not be trained called non-trainable layer like pooling. The number of parameters is layer dependent.

Example of the three layers neural with a parameter associated with the network can be seen form 3.6

3.5.3 Forward propagation

The weights of the layer change when the input is passed through the network. The input is feed into the layer, changing the weights (from the result of the loss function), then passed to the next layer, and the process continues until the output layer. Each layer can have a different set of functions. This process takes place from left to right, i.e., from the input of the network to its output and called forward propagation.

Considering the layer given in figure 3.6, if all weight w and activation function g are available then for input vector x we can estimate the $\hat{y}$ by

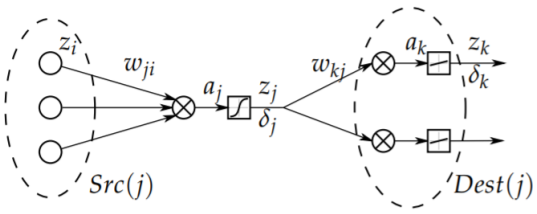


Figure 3.7: Backpropagation error[8]

iterative evaluating in individual layers. This process is forward pass.

$$\begin{aligned} a_j &= \sum_{i \in Src(j)} w_{ji} z_i \\ z_j &= g\left(a_j\right) \end{aligned} \tag{3.13}$$

In equation 3.13 z_i are inputs of hidden layers neuron x_i and z_j are outputs for hidden layer neuron.

From equation 3.9 the gradient of the loss function w.r.t to individual weight :

$$\nabla E(\boldsymbol{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_W}\right) [8] \tag{3.14}$$

Gradient decent algorithm to update weight :

$$w_d \longleftarrow w_d - \eta \frac{\partial E}{\partial w_d} \text{ for } d = 1, \ldots, W \tag{3.15}$$

where η is learning rate

Individual derivatives $\frac{\partial E}{\partial w_d}$ for each parameter can be computed using backpropagation and ultimately finding the weights.

3.5.4 Backpropagation

From figure 3.7,loss function E depends only on w_{ji} and a_j

Error δ_j is given by

$$\delta_j = \frac{\partial E}{\partial a_j} \tag{3.16}$$

and δ_j is the error of the neuron on the output of hidden layer and z_i is the input from i to j and known form forward pass.

For output layer δ_k depends only on a_k via $g(a_k)$ and can be written as:

$$\delta_k = \frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial \widehat{y}_k} \frac{\partial \widehat{y}_k}{\partial a_k} = g'\left(a_k\right) \frac{\partial E}{\partial \widehat{y}_k} [8] \tag{3.17}$$

and for hidden layer δ_j E depends on a_i via all a_k and computed as :

$$\delta_j = g'(a_j) \sum_{k \in Dest(j)} w_{kj}\delta_k [8] \tag{3.18}$$

Hence the derivative $\frac{\partial E}{\partial w_d}$ can be computed with

$$\frac{\partial E}{\partial w_{ji}} = \delta_j z_i [8] \tag{3.19}$$

The training is performed repeatedly to reduce error by minimizing the loss function. In each iteration, the weights get changed to improve its performance. Often a large data set is required to train the neural network. Initially, the network starts with some random number as the results from the forward propagation. These results from forward propagation can have error. The measure of error is found with the loss or cost. This measure of error is done with the loss function on the desired output and prediction of training examples. The learning process should be efficient to change the required network parameter to reduce the loss (error). Consequently, the negative gradient of the loss with respect to parameters is calculated by recursively applying the chain rule layer by layer towards the input. This process is repeated for each example, and the parameter learning rate of the obtained negative gradient is summed up to weight and update them. This process is called backpropagation. The learning algorithm must be optimized enough(should have proper value for parameter like learning rate and batch size so the algorithm do not stuck) such as stochastic gradient descent (SGD) [44] to get all parameters to converge to atleast local minima.

3.5.5 Activation functions

It is assumed neural networks learn the easy parameters like line detection, curve detection, etc. during the starting layer(closer to input), and the later layer can filter more complex structure like the human face or the object which is trained by the training examples. Activation function controls the output of the node, whether it should be fired or not and hence crating the filter for the feature. Some of the most common functions used as an activation function are :

Softmax

The softmax activation[9] is used to perform multi-class classification, as it ensures that all the activation in a single layer is summing up to 1.

$$y_k = \frac{\exp(\phi_k)}{\sum_j^c \exp(\phi_j)}, \tag{3.20}$$

$$y_k = \frac{\exp(\phi_k)}{\sum_j^{c-1} \exp(\phi_j) + 1}, k = 1, 2, \ldots, c-1, \tag{3.21}$$

Soft max function can be viewed from figure 3.8

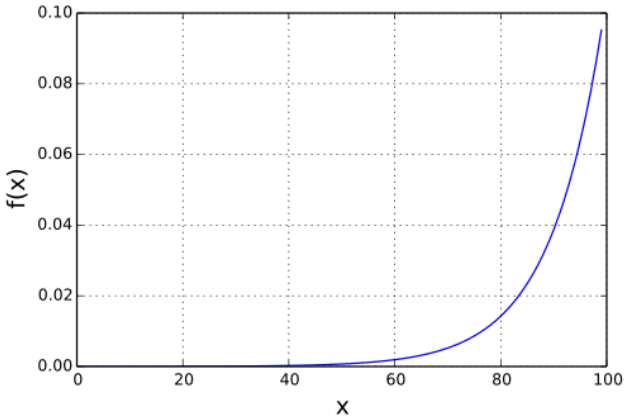


Figure 3.8: Softmax function [9]

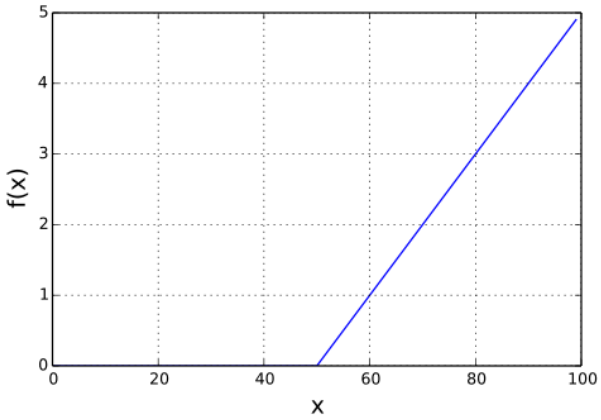


Figure 3.9: Relu function[9]

Relu

ReLU stands for rectified linear unit[9]. Mathematically, it is defined as y = max(0, x). It is described by :

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{3.22}$$

$$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases} \tag{3.23}$$

The characteristic of Relu function can be seen from figure 3.9

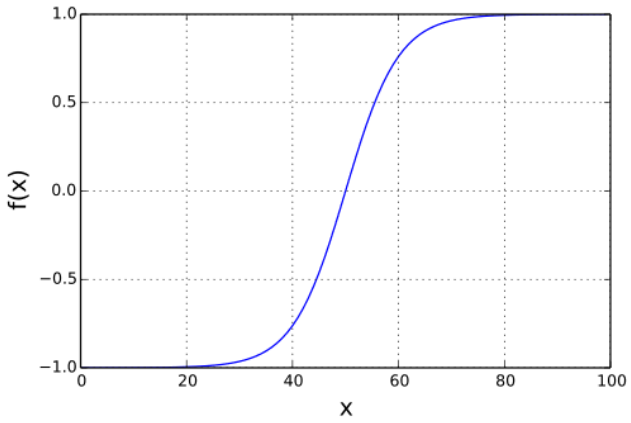


Figure 3.10: TanH function [9]

TanH

TanH is a hyperbolic function that ranges from -1 to 1[9]. The function can be described as

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1[45] \tag{3.24}$$

TanH function can be viewed from figure 3.10

3.5.6 Overfitting

There is a situation when the neural network is too much inclined toward the training data set rather than the general feature set of the object of interest, which can lead to poor performance for other testing datasets. The classifier should be aimed to learn the general feature by training, which can perform well during testing. However, the objective function is set to reduce the training loss and can often cause overfitting. One of the ways to reduce the overfitting is regularization. Regularization is the process that discourages learning a more flexible model (high variance and more fit toward noise) by reducing the magnitude of weights (adding penalty term). Another way to reduce the overfitting can be to train network with big and various datasets.

3.6 Dropout regularization

Dropout is one of the most effective methods to regularize network and prevent overfitting. Randomly neuron is chosen to stop being propagating. This makes the no weight update of the neuron through its incoming and outgoing connections. Dropout can also decrease training time cause some of the layers are dropped so less computation.

3.7 Deep learning

Deep learning is the subfield of machine learning. Deep learning can be supervised, unsupervised, semi-supervised, or even reinforcement learning. Making neural networks deep enough can produce great results like image classification, image segmentation, etc. [25]. Deep learning usually requires a large amount of data before it can use for the test due to a large number of a layer that needs to be trained. Though the deep network is often hard to train and requires too much data to train, the results produce are worthy cause it enables it to learn very complex and non-linear features.

3.8 Convolutional neural network

An image often contains a high volume of data in the form of color channels. It would be wasteful to have full connectivity of layers, and the massive number of the parameter to train may quickly lead to overfitting. The convolutional neural network takes advantage of the fact that the neighboring pixels are correlated in image, so its architecture is designed in a more sensible way. The convolutional neural network has neurons arranged in 3 dimensions: width(width of the image), height (height of the image), depth (color channel in image).

3.8.1 Convolutional layers

Convolutional in mathematics[46] is an operator, refers to the mathematical combination of two functions to produce the third function. It involves analyzing the sample of input signal contributes to the many points of output signals. It expresses the relation of how the shape of one signal is modified by the other.

In image processing, It is performed on the input data with the use of filter or kernel(which can be any or specific) to produce a feature map. The filter is a small matrix of numbers that are multiplied with the input to perform convolution. The filter is applied to different segments of the image sequentially, and this process can be viewed as filter sliding.

Sliding the filter all over input at every location with some interval, give the convolution and the results are put onto the feature map.

It is assumed that the features in images are found in nearby pixels locally rather than covering the whole image. Usually, the area of the filter is kept smaller than the size of the image to learn features from the relationship of neighborhood pixels. These local features can be found at any part of the image, which makes sliding a crucial process in creating a feature map. The receptive field is the input area which gets multiplied by the filter to produce one node in feature map.

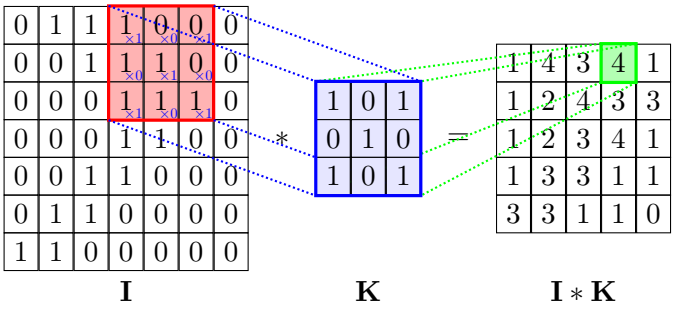


Figure 3.11: Convolution of filter or kernel K (center) blue matrix with the receptive field (red) of Image I (left) and its output (green) one node of feature map I*K (right)
Source: Created with LaTex package TikZ [43]

Figure 3.11, shows the image I with the kernel or filter K and output feature map (I*K). The red area displays the receptive field, blue matrix is the filter, and element in green is the one node of the feature map.

Stride

Stride is the step size the filter took each step during sliding of the filter. Stride size is usually one, which means the filter slides one pixel per each step. When the size of stride is increased the filter slides over the image with a larger interval and less overlap between the pixels.

Padding

It is not necessarily that for given image size, the filter size, and the stride are compatible. A zero value pixel can be introduced from the outer of the image to overcome this shortcoming. This layer of zero pixels surrounding the images is called padding.

3.8.2 Pooling layer

The output feature map can be sensitive to the location of the features in the input. This sensitivity can be addressed by downsampling the feature maps. The invariance of feature detection sensitive to the location is refereed by the technical phrase "local translation invariance "[47]. Hence making the feature maps a more robust to change in position of the feature in the image pooling is performed. Common usage of the pooling layer is to downsample; there are no trainable parameters associated with the pooling layer.

The pooling can be performed by averaging or taking the max of the features in the patch of the feature map. Some common methods of pooling are Average pooling and Max pooling. Max pooling can be seen by figure 3.12.

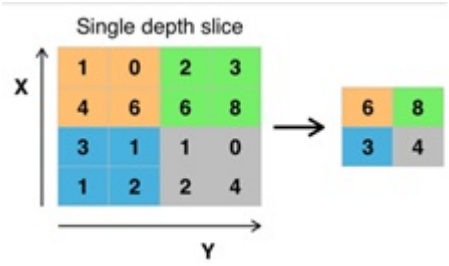


Figure 3.12: Example of max pooling where the max is taken over 4 number with stride 2 [10]

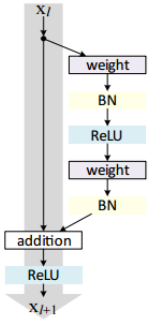


Figure 3.13: Micro architecture of Resnet 50 [11]

3.9 Neural network architectures for Image classification

3.9.1 VGG16 and VGG19

The 16 and 19 stand for a number of weight layers in the network[48]. Due to its depth and fully connected, it was hard to train the network. The training process can be made easier if the network is trained with less weight layer first and then after a smaller converged network can be used as the initializer for the larger deeper network, and the process was called Pre-training. It was the 1st runner up for image classification and winner of localization in ILSVRC[2] 2014.

3.9.2 ResNet50

In general convolution network, several layers are stacked and are trained to for feature filter layer by layer. In residual learning, the network will try to learn the residual. Residual [12] is the subtraction of features learned from the input of the layer. Its architecture is based on microarchitecture, which has small building blocks that can be used to construct the network. The collection of micro-architecture building blocks leads to macro architecture. The microarchitecture of Resnet 50 can be seen in figure 3.13.

3.9.3 Inceptionv3

The Inception [49] model make multi-level feature extractor by computing 1x1, 3x3 and 5x5 convolution within the same module of the network and output of these filters are piled on the channel space before being supplied into the next layer.

3.9.4 Xception

Xception [50] was proposed by Francois Chollet, the creator of Keras library. It is an extension of Inception architecture that replaces the standard Inception modules with depthwise separable convolution.

3.9.5 Mobilenet v2

The purpose of mobilenet [51] was to have a general-purpose computer vision neural network for mobile devices. Mobile net v2[52] introduces two new features to architecture — first, a linear bottleneck between layers and shortcut connections between the bottleneck.

3.9.6 Densenet

Densely convolution network connects each layer to every other layer in a feed-forward fashion. The proposed network model states if the connection between the layer close to the input and those close to the output is shorted, the neural network can be more considerably deeper, more accurate, and efficiently trained[53].

3.9.7 Nasnet

Google introduced "AutoML" that automates the design of the machine learning model; a neural controller network can suggest model architecture "child" who is trained and evaluate the task. The controller network is in the loop with the child network. The feedback from the "child" is used to inform the controller network on how to improve for the next iteration. This process gets repeated thousands of times — generating new architectures, testing them, and giving that feedback to the controller to learn from. Using this method, AutoML was able to determine the fittest layers on CIFAR-10, which performed well on ImageNet[2] image classification and COCO[27] object detection. This architecture of "NASNet" [54] has been formed by combing these two layers.

3.9.8 Mask R-CNN

Mask R-CNN[4] is state of the art Convolutional Neural network, which can do the instance segmentation described in 3.3. Mask R-CNN shares the same feature from Faster R-CNN[55] for object detection. It consists

of two stages the first Region proposal Network(RPN) scans the image and generates proposals areas where it is likely to contain an object. Another stage, which is the essence of fast R-CNN [48] classifies the proposals and generates bounding boxes and masks.

Anchors are the fixed bounding boxes of defined shape and sizes which are places into images and will be used for reference when localizing the object in the image.

A Region Proposal Network (RPN) takes an image and result in a set of rectangular object proposals, each with an object score with the help of anchors. The offsets of the image from the anchors are taken as input and propose an object location in the image.

The second stage of feature extraction extracts features using RoIPool[55] from each candidate box and performs classification and bounding-box regression. The output of regression determines the predicted bounding box in the form of x, y, w, h (x coordinate, y coordinate, width, height), and output of classification is a probability whether the predicted bounding box contains an object or not. Along with these two stages, in the second stage of Mask R-CNN in parallel to predicting the class and box offset, it also outputs a binary mask for each region of interest.

Bilinear interpolation is a resampling process that uses the average of the nearest pixel value to estimate new pixel value. It is an addition of linear interpolation for interpolating functions of two variables.[56].

RoIPool is an operation for obtaining a small feature map from each RoI. RoIPool first estimates a value for RoI to the separating granularity of the feature map. This estimated RoI is then subdivided into spatial bins, which are finally approximated by max-pooling. In each ROI bin, the value of the regularly sampled positions is calculated directly by bilinear interpolation. Thus, avoid the misaligned problem.

Network Architecture: Mask R-CNN have multiple architectures) Convolution backbone architecture used for feature extraction over an entire image) network head for bounding box recognition(classification and regression) .

Figure 3.14 shows the head architecture of Mask R-CNN with Resnet as Backbone.

Backbone: Its standard convolutional network that serves as feature extractor, convolutional network Resnet50 [57] with the introduction of Feature Pyramid Network [13].

Feature pyramid Network: The Feature Pyramid Network (FPN)[13] was introduced in Mask R-CNN with the purpose that it can properly render

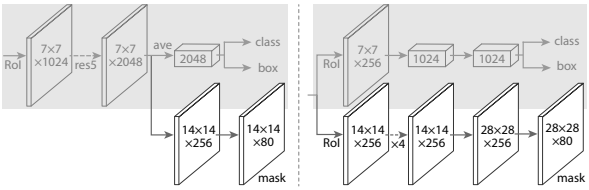


Figure 3.14: Head architecture of Mask R-CNN [4]. Left side of the architecture is extend version of Faster R-CNN with ResNet [12] and right side is extend version of Faster R-CNN with FPN [13]

the objects at various ranges. FPN improved the feature extraction pyramid by combining the other pyramid that takes the high-level features from the initial pyramid and carries them to lower layers. By doing so, it provides features at every level to have access to both, lower and higher-level features.

3.10 Neural network for depth estimation

3.10.1 Monodepth

The depth estimation is done in the form of image reconstruction [14]. They present depth estimation as an image reconstruction problem during training. Assuming, given calibrated pair of binocular cameras, if the function can be learned to reconstruct one image from another, then some 3D information is learned about the scene. The two images (corresponding to left and right) from the calibrated stereo pair can be captured at the same moment in time. The attempt to find dense correspondence with the left image would allow reconstructing the right image. Similarly, the same can be done to reconstruct the left image, given the baseline distance been the camera and focal length depth can be recovered from predicted disparity.

Network estimate depth by inferring the disparities that warp the left image to match the right one or vice versa. The network generates the predicted image with backward mapping with a bi-linear sampler, which results in the fully differential model.

3.10.2 Monodepth2

Monodepth2 is improved version of monodepth [14]. New minimum reprojection loss function designed to handle occlusions, multi-scale sampling method to reduce visual artifacts, and auto masking loss to ignore training pixels have been proposed in the model.

The model can be seen in figure 3.16. (a) shows the depth network which performs the reconstruction task as described in [14]. (b) shows the pose network, which predicts the pair of frames at time steps. (c) Shows the proposed per-pixel reprojection loss, which shows during the time frame when the correspondence is good, the reprojection loss should be low rather than

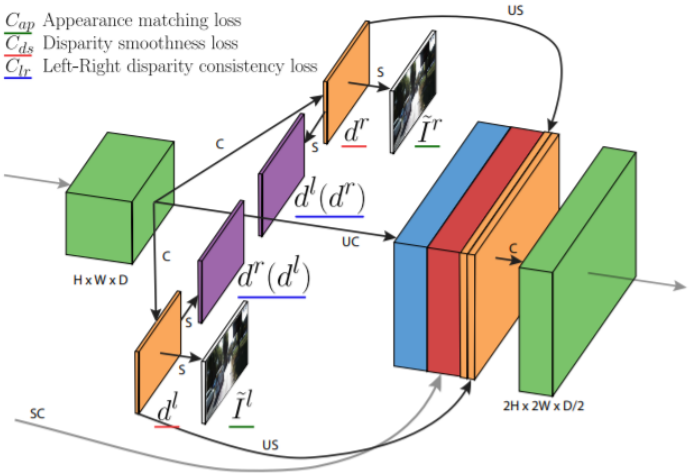


Figure 3.15: Loss model with left and right disparity maps ,d^l and d^r.The same module is input for four different output scales. C:Convolutional, UC: Up-Convolutional, S:Bilinear Sampling, US: Up-Sampling, SC:Skip Connection[14]

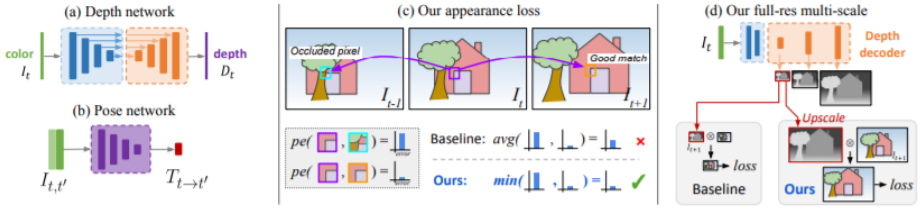


Figure 3.16: Overview of Monodepth2 Network [5]

using the average loss for matching the pixel when there are occlusions. Using minimum reprojection loss gives sharper results. (d) shows the proposed multi-scale sampling, which is performed in the intermediate layers, these layers upsample the depth predictions and compute all losses at the input resolution reducing visual artifacts.

Figure 3.16 shows the network architecture used in monodepth2.

Table 3.1 shows the comparison results among other state-of-the-art networks on KITTI dataset[30]. The results show monodepth2 was able to outperform other state-of-the-art network in self-supervised mono supervision, self-supervised stereo supervision and combined mono and stereo both and can be inferred from the scores.

3.11 Evalution of machine learning model

The trained model can be evaluated by testing. The metrics for evaluation depends on the machine learning task. Some commonly used metrics for evaluation are

Method	Train	Abs Rel	Sq Rel	RMSE	RMSE log	<1.25	$<1.25^2$	$<1.25^3$
Eigen[58]	D	0.203	1.548	6.307	0.282	0.702	0.890	0.890
DORN[59]	D	0.072	0.307	2.727	0.120	0.932	0.984	0.994
LEGO[60]	M	0.162	1.352	6.276	0.252	-	-	-
Ranjan[61]	M	0.148	1.149	5.464	0.226	0.815	0.935	0.973
Monodepth2[5]	M	0.115	0.882	4.701	0.190	0.879	0.961	0.982
Monodepth2 w/o pretraining[5]	M	0.132	1.044	5.142	0.210	0.845	0.948	0.977
Monodepth2(1024x320)[5]	M	0.115	0.882	4.701	0.190	0.879	0.961	0.982
Garg[62]	S	0.152	1.226	5.849	0.246	0.784	0.921	0.967
Monodepth R50[14]	S	0.133	1.142	5.533	0.230	0.830	0.936	0.970
Monodepth2 w/o pretraining [5]	S	0.130	1.144	5.485	0.232	0.831	0.932	0.968
Monodepth2[5]	S	0.109	0.873	4.960	0.209	0.864	0.948	0.975
Monodpeth (1024 x 320)[5]	S	0.107	0.849	4.764	0.201	0.874	0.953	0.977
UndeepVO	D*MS	0.183	1.730	6.57	0.268	-	-	-
Monodepth2 w/o pretraining[5]	MS	0.127	1.031	5.266	0.221	0.836	0.943	0.974
Monodepth2[5]	MS	0.106	0.818	4.750	0.196	0.874	0.957	0.979
Monodepth2(1024x320)[5]	MS	0.106	0.806	4.630	0.193	0.876	0.958	0.980

Table 3.1: Comparision of monodepth2 with the existing method on KITTI 2015 using Eigen split[5]
where, D - Depth supervision,
S - Self-supervised stereo supervison,
M - Self-supervised stereo supervison

		Predicted class	
		Positive(1)	Negative(0)
Actual class	Positive(1)	True positive (TP)	False negative (FN)
	Negative(0)	False positive (FP)	True Negative(TN)

Table 3.2: Confusion matrix for the binary classification[18]

3.11.1 Confusion matrix

The confusion matrix gives a detailed overview of correct and incorrect classification for each class. It can be considered as a table with prediction vs. ground truth.

Considering the example of binary classification in table 3.2. Here the row of the matrix represents the values from the actual class, while column represents the values in the predicted class.

True positive

True positive is the number of occurrences when the prediction is true, and the ground truth is also true.

False positive

False positive is the number of occurrences when the prediction is true, and the ground truth is false. It is also known as type 1 error.

True negative

True negative is the number of occurrences when the prediction value is false, and the ground truth is true.

False negative

False negative is the number of occurrences when the prediction value is false, and the ground truth is also false. It is also known as type 2 error.

3.11.2 Accuracy

Accuracy in the context of machine learning is the ratio of the number of correct predictions to the total number of predictions. It is given by

$$Accuracy = \frac{TP + TN}{Total} \tag{3.25}$$

It shows how often the trained model makes the correct prediction.

3.11.3 Misclassification rate

Miscalssification rate is given by the ratio of summation of false positive and false negative to the total number of predication.

$$Misclassification\ rate = \frac{FP + FN}{Total} \tag{3.26}$$

It shows how ofter the prediction of the trained model is wrong. It is also equivalent to unit minus accuracy.

3.11.4 True positive rate

True positive rate describes how often we make the correct prediction in case of the actual value is true. We want this to be as high as possible. It is also known as recall.

$$True\ positive\ rate = \frac{TP}{TP + FN} \tag{3.27}$$

3.11.5 True negative rate

True negative rate is the rate at which actual value was false, and our model predicts it as true. It is also known as specificity and given by the ration of true negative to the summation of a true negative and false positive.

$$True\ negative\ rate = \frac{TN}{TN + FP} \tag{3.28}$$

$$C = \text{Actual} \begin{matrix} & \text{Classifed} \\ & \begin{matrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \\ c_{n1} & & c_{nn} \end{matrix} \end{matrix}$$

Figure 3.17: Confusion matrix for multiclass classification [15]

3.11.6 Precision

Precision is the ration of true positive and summation of true positive and false positive. Precision describe about how much percent of the results are relevant.

$$Precision = \frac{TP}{TP + FP} \tag{3.29}$$

3.12 Multi class evaluation

Similarly the confusion matrix of multi class classification can be seen by figure 3.17 and the elements of confusion matrix can be found using equation 3.30

$$\begin{aligned} tp_i &= c_{ii} \\ fp_i &= \sum_{l=1}^{n} c_{li} - tp_i \\ fn_i &= \sum_{l=1}^{n} c_{il} - tp_i \\ tn_i &= \sum_{l=1}^{n} \sum_{k=1}^{n} c_{lk} - tp_i - fp_i - fn_i \end{aligned} \quad [15] \tag{3.30}$$

where,
tp_i is the true positve of class i,
fp_i is the false positive of class i,
fn_i is false negative of class i,
tn_i is true negative of class i

Chapter 4

Software tools

4.1 Google Direction API

The Directions API is a service that results in a route between locations. Directions API enables searching for directions for several modes of transportation, including: transit, driving, walking, or cycling[26]. The result gives the direction and route to the destination. The way is the path leading from start to the destination location in the form of the polyline. This polyline can be converted into other forms, such as a list of a waypoint for other applications.

4.2 Google Street View API

The Street View Static API lets to download the images available in the location passed in the request. The request is the URL parameters sent through a standard HTTP request, and results as a static image[3].

Working of street view API can be inferred from figure 4.1

4.3 Folium

Folium [63] is a Python package that visualizes the data on an interactive leaflet map. The library has several built-in tilesets from OpenStreetMaps[64] and other platforms such as mapbox. Moreover, it supports custom tilesets from different other API. Folium support both image, video, GeoJSON [20] and TopoJSON [65] overlay.

4.4 KERAS

Keras [23] is a high-level neural networks API, written in Python and capable of running on top of TensorFlow [24], CNTK [66] , or Theano [67].

```
Google Street View Image API      <-- API for Street View Images
            |
    google_streetview.api         <-- URL Request with query string
            |
         request                  <-- Download URLs and images
```

Figure 4.1: Street view work flow [16]

The architecture of the neural network Mask R-CNN is developed with the help of this package. It has support for CPU and GPU as well as TPU in case we use Google Colab. It provides an intuitive way of defining a neural network by using the functional API, which allows defining layer as functions. It is a powerful library with lots of options in tweaking the parameter of the layer during learing and other processes.

4.5 TensorFlow

TensorFlow[24] is a machine learning computational backend developed by Google LLC. It is a tool for expressing and implementing a machine learning algorithm. Tensorflow library consists of visualization tools like TensorBoard, which make debugging and optimization of the neural network easier. The computation developed in TensorFlow can be executed in a variety of systems with little or no change making it a flexible tool.

4.6 Pytorch

Pytorch[68] is an open-source machine learning tool developed by Facebook's AI research lab. It is used for applications such as computer vision[19] and natural language processing [69]. The architecture of CNN monodepth2 used in thesis is implemented in PyTorch. It has the capability to use GPU for parallel processing of data.

4.7 Google Colab

It is a free research tool for machine learning education and research tool which uses Jupyter [70] notebook environment from Google LLC, which can be run a maximum of 12 hours in single runtime[21].

Google Colab has support for deep learning application like Keras, TensorFlow, Pytorch [68] and OpenCV [71].Also, the other dependencies could be easily installed using pip installer. It also allows three different kinds of runtime hardware like GPU, CPU, and TPU, which can accelerate the process.

Chapter 5

Implementation

This chapter describes the implementation of a given task from the perspective of the software side. This section explains the technical specifications and parameters of each tool. A pipeline explains the flow of data and process from input to output. Moreover, its description shows the dependencies of one block on others. The data is processed to give a specific output and used in further blocks. The pipeline structure can be seen in figure 5.1 and arrangement of files can be seen in structure B.1

5.1 Input

Input is in the form of geojson[20] file, a list of tuples of coordinates with features consisting of geometry type polygon. Overpass API needs a pair of coordinates in the form of the bottom left and top right coordinate and assume to create a bounding box out of it. The bounding box is created from input geojson, which will be the input for the Overpass API query.

5.2 Creating a query

The input bounding box is split into small boxes of the area with some (say 0.005 equivalent of 55.66m [72]) minimum distance parameter of the side. These small bounding boxes are saved in a file in the form of a list. This list is the input to GSV API to download the available images for the coordinates provided. For each iteration of the list for the query, images, along with their metadata, are saved in a different directory(batch).

5.3 Downloading of Images from location

Street view images are in the form of 360 panoramas and divided into four parts of 640x640 image with a field of view 0, 90, 180, 270.

GSV API takes query: parameter location, (in the form of a list of coordinates (output from Overpass API)), size of the image, heading, field of view, etc. and return the pictures available at the location along with their

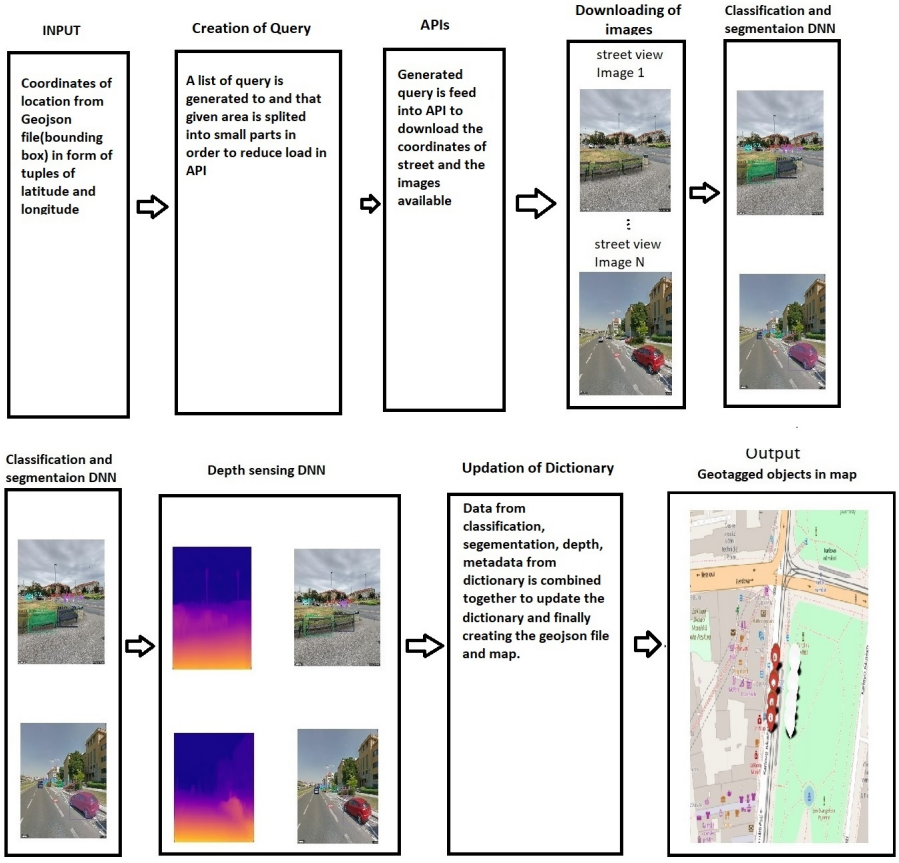


Figure 5.1: The proposed pipeline for city mapping

metadata. These outputs are RGB images and used as the input for the deep learning network(s).

5.3.1 Structure of metadata

Figure 5.3 is the example of metadata where the first query results in an image with a given pano_id and second query result with no image available for that coordinates. The metadata is a list that consists of a dictionary for each request. The dictionary consists of "copyright", "data", location (in the form of latitude and longitude), panorama id, status, and file name. This metadata is used to create a database to match the corresponding image to information during the later phase.

The structure of download dictionary with batches are as follows B.2

5.4 File Handling

A Python dictionary combines the metadata of each downloaded batch as a database. Duplicate entries and "NOT FOUND" queries are deleted to

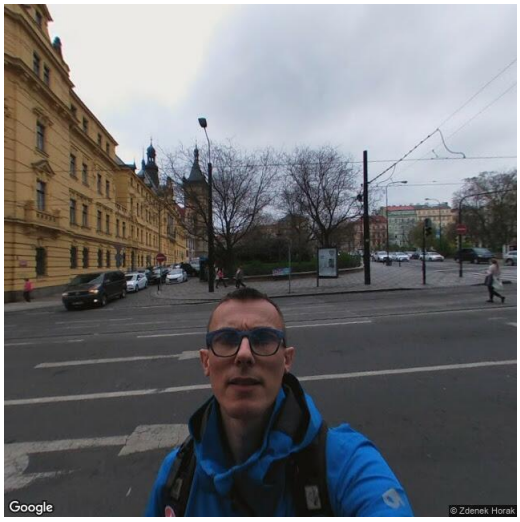


Figure 5.2: Example of downloaded image(640x640)from GSV API


```
[{"copyright": "\u00a9 Google", "date": "2017-06", "location": {"lat": 50.0899134, "lng": 14.394997},
 "pano_id": "S7CJxO3k-F86ln9ELGRusg", "status": "OK", "_file": "gsv_0.jpg"},
  {"status": "NOT_FOUND"}]
```


Figure 5.3: Structure of metadata from GSV API

optimize the dictionary. The name of the files is renamed based on their panorama id along with angle.

Note - There was no way to differentiate the images from the panorama id with a different heading. Cause the localization of the object in space needs the heading information; the photos are renamed with the addition of there heading angle at the end. It is assumed and tested that the 1st image with the same pano_id is the angle which is input first argument in GSV API during downloads.

The renamed images are stored in directory "imgdataset" and its structure can be seen from B.5

5.5 Classification and segmentation

Images are taken from "imgdataset" and checked with the "img_db.json" to match the information with metadata are iterated with classification and segmentation. A dictionary (det_dic.json) stores the detected object along with the location of the image and is used in a later phase to create a statistical graph.

Each detection creates the mask and the string of class names and the bounding box. A separate directory store the mask files of detection and organized based on the file name. The corresponding information with masks and objects is stored in the dictionary mask_dict.json. The structure of the mask dictionary can be seen from B.3.

{"OhYfPJDXgdbRfXCTY7PnEw-90": {"date": "2017-06", "location": {"lat": 50.0898872, "lng": 14.3950413},
"person": {"mask0": 611.0, "mask1": 576.0, "mask2": 553.5}, "car": {"mask3": 385.0}},
"4Iuxzz51jKCoQad52ZYYJQ-270": {"date": "2009-05", "location": {"lat": 50.09084038021513,
"lng": 14.39316123784426}, "car": {"mask0": 471.5}, "person": {"mask1": 441.5}, "umbrella":
{"mask2": 264.0, "mask3": 252.0}}}

Figure 5.4: Structure of mask dictionary

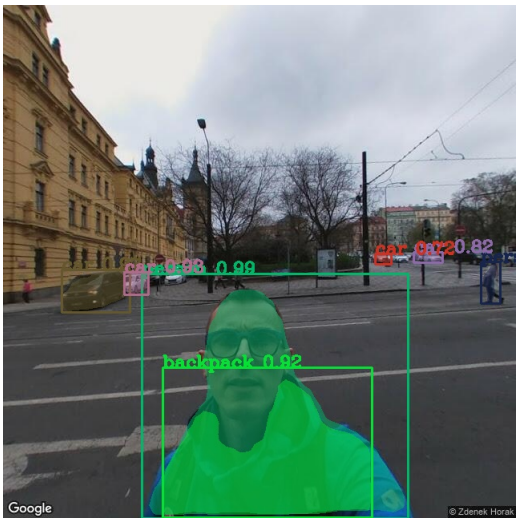


Figure 5.5: Example of the resulted image processed with Mask R-CNN with different color mask and confidence score of the detected class

5.5.1 Mask dictionary

An example of a mask dictionary data structure can be seen in figure 5.4. It consists of two images and the detected object with mask number:

The filename (pano_id) is the key for the Python dictionary, which consists of another nested dictionary with keys (like date, location, detected class). Each detect class in the image has a dictionary with a key mask(some number), and the value associated with key mask(some number) is the centroid of the bounding box generated by Mask R-CNN.

5.6 Depth estimation

Depth is estimated by another state of the art deep neural network on images inside the directory "imgdataset." The result is a depth value for each pixel in the image.

The images from this step are stored in the directory "Depth" with grayscale .jpeg image and .npy array so it can be used for visualization. The structure of the depth mask can be viewed from B.4.

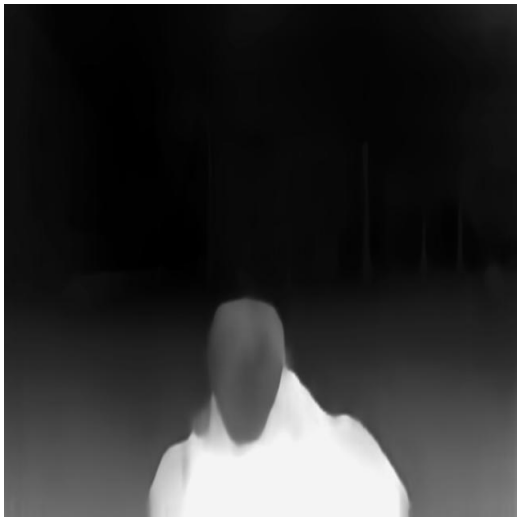


Figure 5.6: Example of resulted depth image processed with monodepth2 when converted to grayscale


```
{"OhYfPJDXgdbRfXCTY7PnEw-90": {"date": "2017-06", "location": {"lat": 50.0898872,
"lng": 14.3950413}, "person": [50.0898581, 14.395041439972458, 50.0898616, 14.395041476151588,
50.08986385, 14.395041635809271], "car": [50.0898807, 14.395049979805044]},
"4Iuxzz51jKCoQad52ZYYJQ-270": {"date": "2009-05", "location": {"lat": 50.09084038021513,
"lng": 14.39316123784426}, "car": [50.09085553021513, 14.393156106941092], "person":
[50.09085253021513, 14.393154604645881], "umbrella": [50.090834780215125, 14.393160499107472,
50.090833580215126, 14.393160220052788]}}
```


Figure 5.7: Structure of final dictionary

5.7 Depth Analysis

The segmented masks saved in the previous step is applied over corresponding depth image using the information form (mask_dict.json) dictionary. The depth with the pixels where the mask value is true is averaged and taken as the depth of the object. The centroid of the bounding box in the image gives information about the position of the image in other axes. The depth images are cropped within masks and are stored in the separate directory(mask_depth) to visualize the results.

A dictionary file "final_dict.json" stores the organized dictionary for each image with features and detected object location. Below is the structure of "final_dict.json". Figure 5.7 is the example of the final dictionary.

The coordinates of the object in each image in the final dictionary are the result of the bounding box position from the mask dictionary, and the depth calculated when an average of per-pixel depth is estimated with the mask over depth image.

Figure 5.8 show the depth of the six detected objects with their average depth value.

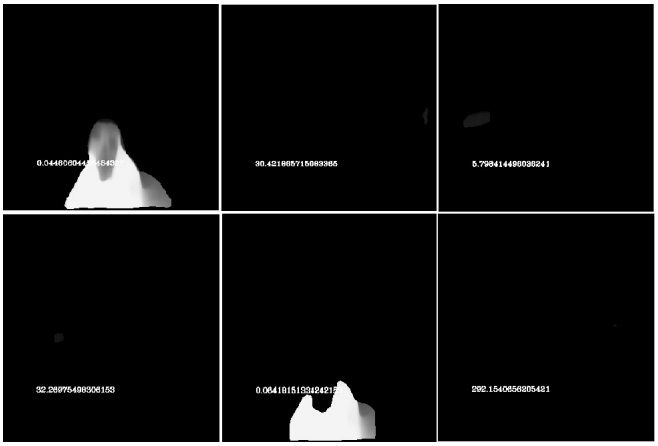


Figure 5.8: Individual depth masks of detected class with their estimated depth value


```
{"type": "FeatureCollection", "features": [{"type": "Feature", "geometry": {"type": "Point",
"coordinates": [14.397847, 50.089921]}, "properties": {"marker-color": "#FAEBD7", "class": "person",
"date": "2018-07", "file_name": "CAoSLEFGMVFpcE1SYTFQVWg4MXJRZ3NwLVRkSmF4T0dyMzdTUmR6LU4xMHR4T0tF-0"}},
{"type": "Feature", "geometry": {"type": "Point", "coordinates": [14.397832, 50.089921]}, "properties":
{"marker-color": "#FAEBD7", "class": "person", "date": "2018-07",
"file_name": "CAoSLEFGMVFpcE1SYTFQVWg4MXJRZ3NwLVRkSmF4T0dyMzdTUmR6LU4xMHR4T0tF-0"}}}
```


Figure 5.9: Structure of output geojson file

5.8 Creating Geojson file and map

The data from the final dictionary is used to create the geojson file. Geojson file consists of information on the name of the image, date of origin, location, and the name of the class. The information is stored as a feature with geometry "point" and properties as a class, date, marker color, and file name. The structure of output geojson file can be seen in figure 5.9

Python package Folium has been used to bind the data on a map for visualizations as markers on the map. The directory "maps" store the individual file for each class with a visualization marker on the map.

5.9 Statistics

The number of detection is calculated and written to file with the class name and the number of detection. The same information is visualized with a bar graph(number of detection vs. classes).

With each run of the whole pipeline, a separate database is generated, each database results with individual output of visualization file from Folium and geojson file. The geojson file and the statistics are combined to merge the overall data.

The database structure is as follows B.5

Chapter 6

Methodology

This chapter describes the methods and approach used in blocks of pipeline mentioned in the chapter Implementation. This chapter also describes the purpose of using specific parameters.

6.1 Building query

Due to the limited load capacity of API (total capacity of overpass API server is 1,000,000 requests per day)[37], query and downloading done are small parts to reduce the server loads and prevent timeout (server busy) errors. These bounding boxes are equal in the area given the minimum distance say (0.005 equivalent of 55.66m [72]) between the side of the coordinate. The query is generated in the form of a list of the coordinates (top right and bottom left), which is further used by the Overpass API to find the available coordinates on the streets. Points between two coordinates are divided equally using linespace[73]. These points are in the form of lists and saved in the text file, which serve as the input for the next step.

6.2 Overpass API

Overpass API is used to get the coordinates available on the streets. Overpass API gets the argument in the form of the bounding box one at a time (list of two coordinates top right and bottom left) and results with the coordinate available on the streets. In overpass API, adding parameter "way" and selecting the type of way which in case "highway" for example gives the coordinates of highways within the area of interest. The resulting query gives output in the form of the nodes, a tuple of coordinates(latitude and longitude). These coordinates are used in GSV API to download the images.

```
# 50.10291748018805, 14.39132777985096              dejvice
# 50.0795436,14.3907308                             Strahov
# 50.0746767,14.418974                              Karlovo namesti
apiargs = {
  'location': '50.0753397,14.4189888 ; '
              '50.0795436,14.3907308 ;'
              '50.10291748018805, 14.39132777985096 ;'
              '50.0753575,14.4060179',
  'size': '640x640',
  'heading': '0;45;90;135;180;225;270',
  'fov': '90',
  'key': '[redacted]',
  'pitch': '0'
}
```

Figure 6.1: Parameters of GSV query

6.3 Downloading the Images with Google Street View API

Street View is one of the features in Google Maps; It is a virtual reproduction of street surroundings on Google Maps. It consists of millions of panoramic images that come from two sources - Google and contributors[3].

Google street view API python sends a URL request to the server to download the images. The requests are in the form of string made up of parameters needed to fulfill the query; the server returns with the metadata with the information of images such as date, location in the form of latitude and latitude, panorama id, status, and file name.

The parameter string is in the form of a Python dictionary with keys like location, size, heading, the field of view, pitch, and developer key. A developer key is one of the most critical parameters to make Google Street API work.

Figure 6.1 is an example of Street View API parameters to showing the "apiargs" dictionary with key and values.

The result is in the form of a dictionary with metadata. RGB images can be downloaded with helper function "download_ links" specifying the parameter with the name of the folder or location of the directory in the form of string.

6.3.1 Selecting the parameter for Google street view API

Argument 'location' takes the coordinates of the location in the form of latitude, longitude. Multiple locations can be fitted in arguments. If no image is available, then API will return a generic image with the text "Sorry, we have no imagery here."

The highest resolution that can be downloaded is 2048x2048, which is available with google premium account services. In the thesis, a standard 640x640 image is used for the whole process.

The heading indicates the compass heading of the camera; accepted values

(a) : Example of downloaded image with pitch 90

(b) : Example of downloaded image with pitch -90

Figure 6.2: Downloaded GSV image with the different pitch angle

are 0 to 360. North is indicated by 0. Heading of 0, 90, 180, and 270 is chosen with the aim to extract all information from the 360 panoramic images.

FOV shows the horizontal field of view and is set to set to maximum (120) to get all possible details from the image.

Pitch is the camera up and down angle. For example, having a pitch to -90 or 90 gives the images headed up to the sky and bottom to the floor.

Figure 6.2 shows a downloaded image with 6.2a with 90 and 6.2b with -90 angle

6.4 Maintenance of dictionary

GSV API tries to grab the panoramic images close to the coordinate provided. GSV API also downloads the metadata in the form of a dictionary that consists of information like the name of the image, location, date, etc. There are cases when it is possible to download the same image multiple times. A dictionary is maintained to avoid the processing of the same multiple times. Dictionary consist of all metadata of all the images, the same images (which is checked on the basis of the panorama id) is deleted from the list. This dictionary is further used in file handling to rename the images.

6.5 File handling

GSV API allows the user to download the panoramic images in the form of a 2D image on the basis of the angle of heading. The whole panoramic image can be downloaded using images providing the different angles of heading. But unfortunately, API doesn't provide any information about the angle of heading. The images download sequentially on the basis of the heading angle provided during the API query. In this step, images are renamed on the basis of there panoramic id along with there heading angle. This step reduces the

duplicate images and adds information about the heading angle, which is crucial when estimating the position of the marker in space.

6.6 Classification and segmentation

Mask R-CNN[4] is state of the art deep neural network proposed by Facebook research scientist Kaiming HE in 2017 which can perform instance segmentation and object detection together. It can have several different backbone architectures like Inception V2, ResNet 50, Resnet101, and Inception-Rennet2. Mask R-CNN with Resnet-101-FPN backbone was able to outperform another state of the art network like MNC and FCIS [74] winner of COCO 2015 and 2016 segmentation challenge. Pre-trained weights on the COCO dataset with Resnet 101 architecture were used to classify data on GSV images.

Images from GSV are taken as input and processed with Mask R-CNN for classification and segmentation of objects. These images are processed with Mask R-CNN, and with the pre-trained weight of 81 classes, the output is the images with classification with labels and confidence score, bounding boxes around the detected object, colored pixel-wise mask on detected objects.

Implementation of the architecture of Mask R-CNN was done in Keras framework and TensorFlow computational backend and is taken from the open-source repository from Matterport [75]. Model implementation and architecture of Mask R-CNN is done in Python and can be seen in script model.py. It consists definition of all the layer and architecture of Mask R-CNN in TensorFlow.

Implementation of the creation of masks and image processing is done with the OpenCV [71]. The output from the results are in form of image with overlay of text(class name) with number(confidence of being of that class, from zero to one, being one means full confidence) bounding box to localize the classified object in image and mask with color to cover pixel-wise segmentation of detected object in image.

With ResNet 101 as backbone architecture, the model provides several changeable parameters to make it fit for a variety of the application. Some of the easily configurable parameters are minimum confidence detection, learning momentum, learning rate, etc.

Table 6.1, shows some of the configurable parameters of Mask R-CNN [75]. The number of classes depends on the pre-trained model. Minimum confidence decides the creation of a bounding box. If the confidence of the classified object is less than the minimum confidence, no bounding box is formed. The minimum confidence score is set to a threshold of 0.7, which can reduce false detection (False Negatives).

BACKBONE	resnet101
BACKBONE_STRIDES	[4, 8, 16, 32, 64]
BATCH_SIZE	1
BBOX_STD_DEV	[0.1 0.1 0.2 0.2]
DETECTION_MIN_CONFIDENCE	0.7
DETECTION_NMS_THRESHOLD	0.3
GPU_COUNT	1
IMAGES_PER_GPU	1
IMAGE_CHANNEL_COUNT	3
IMAGE_MAX_DIM	512
IMAGE_META_SIZE	93
IMAGE_MIN_DIM	400
IMAGE_MIN_SCALE	0
IMAGE_RESIZE_MODE	square
IMAGE_SHAPE	[512 512,3]
LEARNING_MOMENTUM	0.9
LEARNING_RATE	0.001
MASK_POOL_SIZE	14
MASK_SHAPE	[28, 28]
MAX_GT_INSTANCES	100
VALIDATION_STEPS	50
WEIGHT_DECAY	0.0001

Table 6.1: Configurable parameters of Mask R-CNN

Figure 6.3: Wrong classification of class knife with good confidence score of 0.76

Figure 6.3, shows an example of the class knife, which is classified wrong and makes less significance in street view images.

With each detection of an object, a mask is saved with the use of Python package Numpy. The corresponding dictionary is updated with the name of the image, class of the object, mask number, and the coordinate of the centroid of the detected object.

6.7 Depth of the detected object

Classified and segmented images are in the form of RGB and do not describe any depth information. In order to localize and place under the map, another deep neural network developed by the author and his team monodepth2[5] is used. This convolutional neural network architecture is implemented in Pytorch and uses an unsupervised method to predict depth from a single image. The model was trained on cityscapes with Resnet 50 [57] architecture

Figure 6.4: An estimated depth value(rounded off to whole number) of objects inside bounding boxes

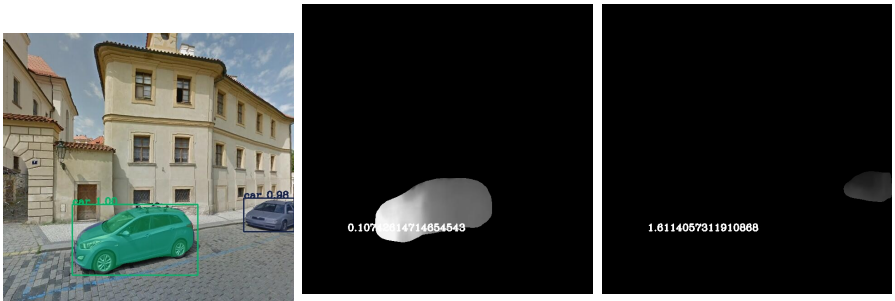
on Cityscapes [76] training data set, which consists of 22973 training images. For the testing "mono+stereo_1024x320" model was chosen for processing as it shows the best results during evaluation [5].

The purpose of finding the depth of the object is to place it as close to the real position on the map. The depth value of the detected objects is estimated with the help of the mask of the object generated during the segmentation step. The average value of depth per pixel within the masks of objects is taken. Further, this depth value is rescaled and added with latitude and longitude of the image to create the location of the detected object in space considering the formula and calculation from figure 6.9.

Example of the estimated depth of objects rounded off and converted to real number can be seen in figure 6.4

6.8 Depth Analysis

The mask from the classification and segmentation step is used with the depth image. The average depth from the images within the mask is used as the distance of the object with respect to the image. These masks are cropped from the depth images, and the estimated depth value is written in the image with the text string. The image is color-coded to grayscale shown in figure 6.5, and these images are saved under directory "Maskdepth" for



(a) : Detected objects in image

(b) : Converted grayscale image of mask of first detected object

(c) : Converted grayscale mask of second detected object

Figure 6.5: Detected objects and their converted gray scale image

visualization. A text string "depth" using OpenCV is overlayed on the image to show the estimated depth of the object respect to the image.

A dictionary is created using all the corresponding data to the image and will be used to create the final geojson file and other outputs.

6.9 The scheme of creating the map

Data from the above sections have been used in the following way to locate the marker in the map.

A panoramic image is divided into four parts with 0, 90, 180, and 270 degrees. There is a different formula to calculate the location of the object for each angle. Figure 6.6, shows the axis each object should take for an image with a given heading angle.

For an example of heading angle 270, the position of the bounding box adds to the latitude(horizontal) of images and depth to the negative of longitude. Similarly for 0, depth is taken positive of latitude and position of the bounding box is taken in the direction of longitude.

XX represents the bounding box location in the image of 640x640; depth represents the estimated depth of object using DNN.

Formula to finding the latitude and longitude is as follows.
For 0 degree:

$$Lat = latofimage + (0.0000001 * Depth) \tag{6.1}$$

$$Lon = lonofimage + (0.0000001 * (XX - 320) \tag{6.2}$$

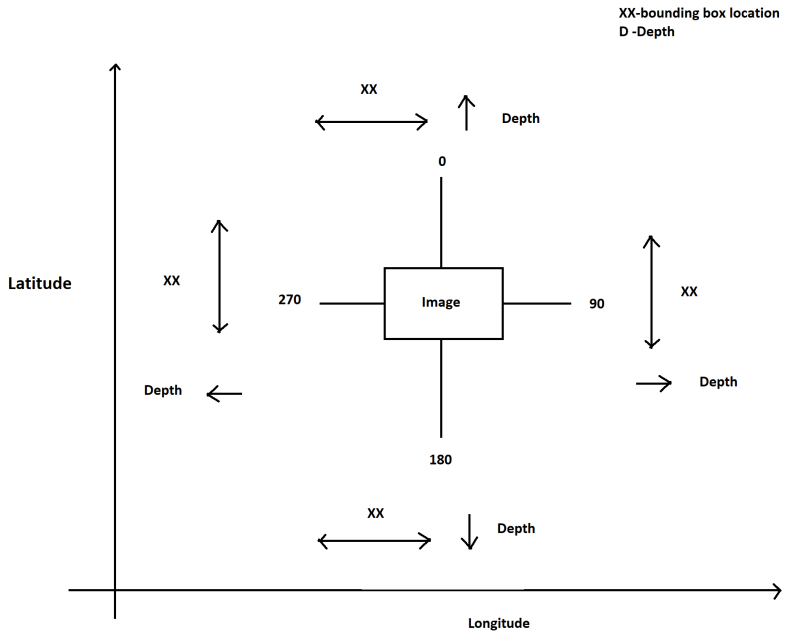


Figure 6.6: Scheme for estimating the location of an object, with different heading angles.

For 90 degree:

$$Lat = latofimage - (0.0000001 * (XX - 320)) \tag{6.3}$$
$$Lon = lonofimage + (0.0000001 * Depth) \tag{6.4}$$

here xx get flipped For 180 degree:

$$Lat = latofimage - (0.0000001 * Depth) \tag{6.5}$$
$$Lon = lonofimage - (0.0000001 * (XX - 320) \tag{6.6}$$

For 270 degree:

$$Lat = latofimage + (0.0000001 * (XX - 320) \tag{6.7}$$
$$Lat = lonofimage - (0.0000001 * Depth) \tag{6.8}$$

For the source [72] , 0.000001°= 0.11 m

6.10 Marker visualization in the map

The final dictionary from the previous step is used to generate a geojson file, and the visualization of the marker in the map using Folium. A separate file which consists of visualization for the same class is also generated. The marker shows the name of the images as an overlay and the name of the class

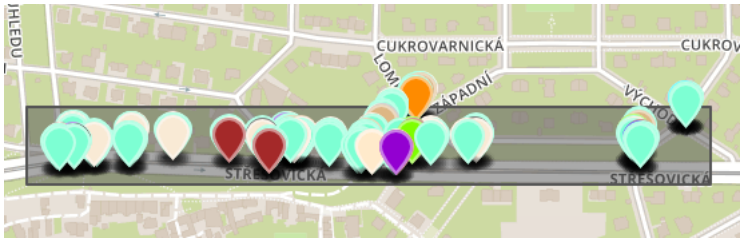


Figure 6.7: Resulting map from the geojson file

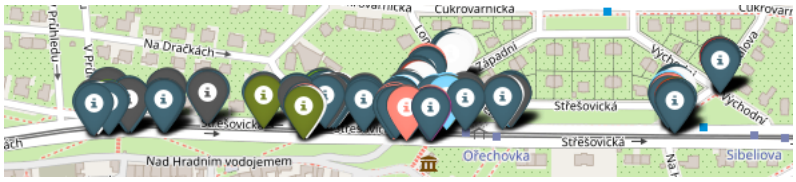


Figure 6.8: Resulting map from the folium

when selected. Moreover, these makers have a separate color for different classes.

Geojson file is created using Python package geojson. Markers have feature geometry "point" and have properties like marker color, class name, date of the image taken from the final dictionary. Colors are picked from python package Matplotlib[77]. These colors are in the form of a hex code, which is HTML compatible. The resulting map can be seen from figure 6.8 and 6.7

6.11 Implementation in Google Colab

Implementation of the whole process work in Google Colab is done in the same way with some extra setups.

6.11.1 Building the environment

There are some dependencies and packages which need to be installed into the instance to run the whole setup. The first block is to start with downloading and installing all the dependencies.

6.11.2 Downloading and running of scripts

The next block downloads the repositories consisting of the scripts. After downloading files from the repository, the input is given by uploading a geojson file with a polygon feature(rectangle).

6.11.3 Visualization

The statistics of the data are visualized with the bar graph, and the generated geojson file can be used to view the markers.

6.12 Experimentation

Below is a list of supplementary experiments performed during the thesis for improvement

6.12.1 Downloading images from the status of GSV API

All available images can be downloaded with a simple approach by sampling the space. The latitude with two endpoints can be sampled easily and can be iterated over the vertical plane. Google street view API returns metadata with each request sent with an "apiarg" string. A result is a dictionary, which contains location, panorama id, date, status. Using the value from key "status" as "OK" implies the image at the location is available and "NOT FOUND" for no image found for location. The idea was to use the status string and build the algorithm which can download images sequentially. The proposed algorithm can be seen with the following flowchart 6.9.

It was necessary to have images sequentially in order to avoid the same objected detected multiple times in a nearby location. A script was developed with logic to download the images sequentially.

The approach was first to find the point where street view is available. For that, random points are sampled in space under given coordinates. The query is sent with that location, location with status "OK" is put in a separate list and used afterward. A list of points is found by making a circle around those points; this list is passed again in query and searches for status "OK" if found add to list and repeat the previous step until a point is out of the region under coordinates specified.

This method reduces the loss of available image under the area of interest. This method would allow almost downloading of all the images available from GSV.

Figure 6.9 shows the algorithm to download images.

6.12.2 Mapping without depth Image

An attempt to create and place the object in the map without using depth images. Simple geometry was used to do such, 6.10 shows the approach.

Considering the image to be 640x640, Assuming that the bottom center(320,640) as zero and the origin of the axes. The values in x - y coordinate is scaled and summed up with the geographic coordinate of the image and used as the location of the object. This naive approach gives the results as the object at the top portion of the image as more depth and the object

located at the bottom of the image less depth. It might be correct and fast for some cases, but it fails to give optimum results.

6.12.3 Changing parameters of Mask R-CNN

Mask R-CNN architecture allows changing many parameters during the training of the data set as well as testing of data sets. The parameter changed during testing was the minimum confidence score for the classification. Changing the minimum confidence score can eliminate the problem like 6.3.

6.13 Clustering

There might be the case when the same object is detected multiple times. For example, the car following the camera on the street. Clusters can be created with the same class and finding the centroid of the cluster. This centroid is then used as the location of the objects. Kmeans clustering algorithm gives the centroid of clusters, which is used for the location of the marker in the map. The number of clusters is estimated with the maximum distance between two pictures with the intuition to form cluster on the basis of the density of images in area, rather using the elbow graph to find the optimum value [78]. With the centroid with some cluster size, the same object detected in different images with depth will gather together and make dense location points in the correct location while others with incorrect points will be considered with less weight.

Clustering was implemented with the Python package Sci-Kit learn [79]. The clustering algorithm is iterated over keys from the dictionary to find the centroid to make a certain number of the cluster which is a function of maximum distance stated above.

However, the clustering algorithm did not converge to the right solution. The localization was poor, and there are often no multiple detections of the same object. So this implementation is not used during the final program.

6.14 Waypoint coordinates

Approach to download the sequential image in the street. Google direction API takes the input argument like starting and destination location and makes the request to the server. The server responds with a dictionary consisting of the information like all the ways possible and their routes, time to arrive, time for depart. Out of which key "polyline" is value consist of polyline points in the form of string. The string is encoded with an encoded polyline algorithm format [80]. Polyline string is then decoded back to latitude, longitude coordinate system. A python package polyline [81] is used to decode the polyline

string into a list of tuples, which consist of latitude and longitude of waypoints.

Figure 6.11 shows the example of a polyline with its corresponding coordinates when decoded in the form of tuples.

Even though decoding coordinates of waypoint were found, but they are not close enough to have efficient mapping. Hence, the next task was to make the waypoint close enough to each other so that there will not be much difference between adjacent GSV images. A significant difference can result in loss of information and no objects in space. An example of such a waypoint before sampling can be in figure 6.12.

To make the waypoints close enough and to have GSV images, points were sampled with minimum distance parameter and sampled if their distance is more than the minimum distance. Sampling is done with Numpy linspace function. The output of such can be visualized in figure 6.13.

This approach does not allow to cover the whole area of interest; instead, it gives the results just for a specific street. So the approach has not been used in the final program.

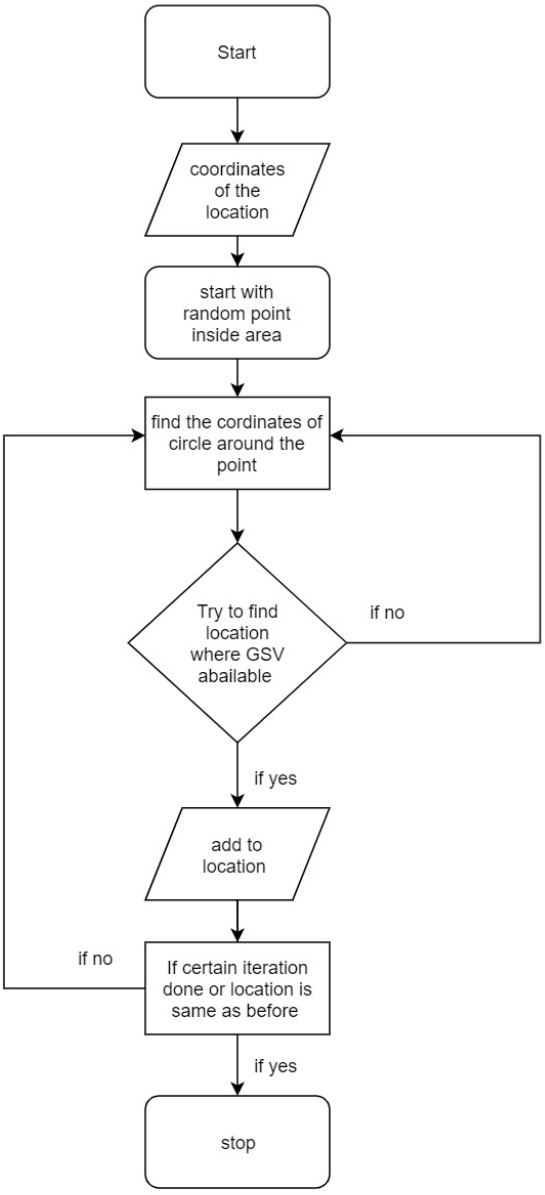


Figure 6.9: Flow chart for downloading the image with the status result

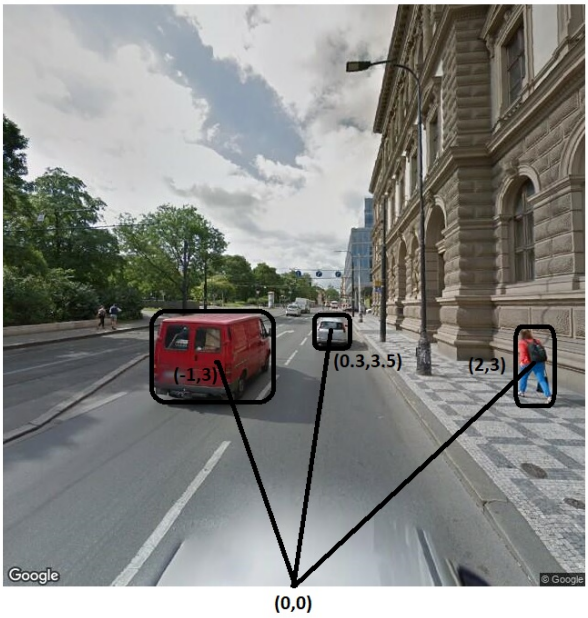


Figure 6.10: Geometric approach of finding location of object from the image considering (320,640) as the center of coordinate system

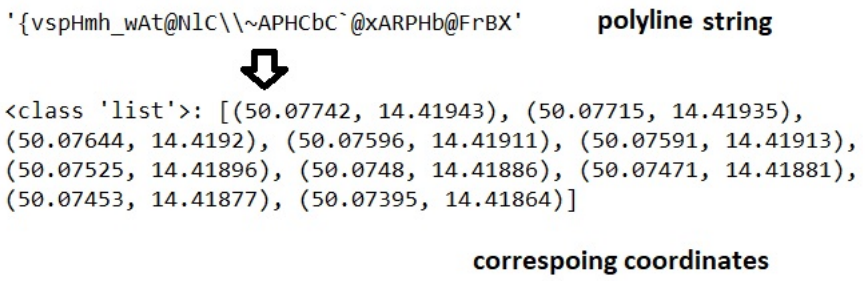


Figure 6.11: Polyline with corresponding coordinates

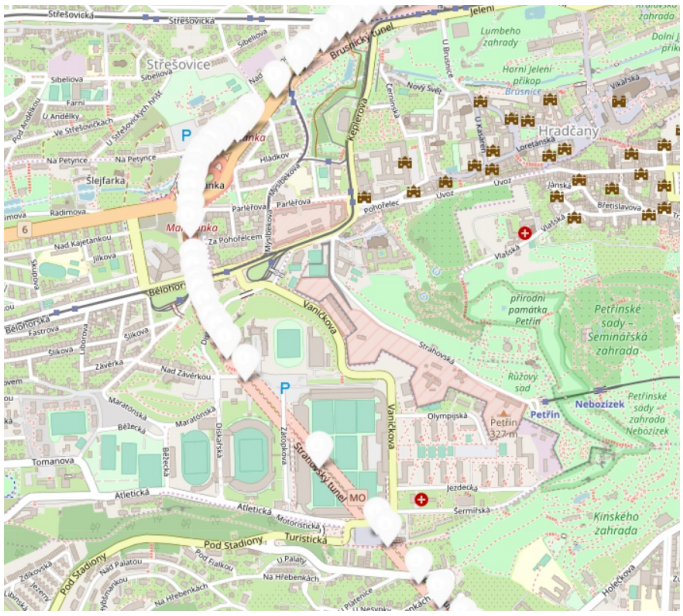


Figure 6.12: Resulting waypoints from polyline from Google Direction API. The points have been decoded and placed in form of White marker

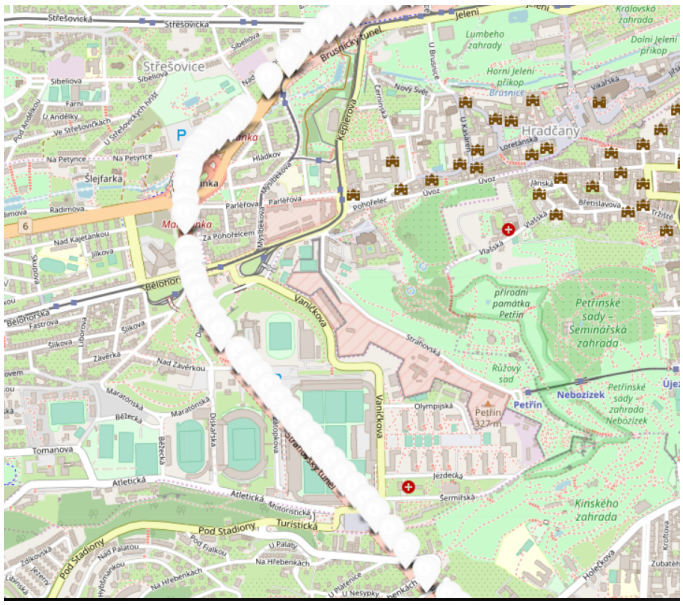


Figure 6.13: Sampled waypoints when the distance between two waypoints is greater than minimum distance

Chapter 7

Experimentation evaluation

The final results of the projects can be seen under the appendix with some of the images and their marker in map with the location with the statistics also with the availability of GSV images in that area.

7.1 Resulting map from the Setup

The end result is the geojson file, which is a form of database with store geological information of feature with its properties. The map visualization generated by Folium, which uses service from Openstreet maps. There are individual maps visualization for each class and a combined map visualization with different marker colors.

7.2 Result of Map generation with overpass API nodes

One of the problems faced with Overpass API is that it gives a run time error when there are too many requests. Fail cases needed to be developed for the issues. Though overpass API gives results with coordinates of the node, because the coordinates of nodes were not in the proper sequence of the waypoint, i.e., does not able to catch adjacent image for feature extraction. The other finding with the API is the resulting nodes do not give the coordinates distributed on the street. The resulting nodes are starting and the endpoint of the streets. Several combinations of the highway and other ways mixed together to get the more sampled points within the area of interest.

7.3 Mapping with the geometric method and Kmeans clustering

As mentioned in section methodology, the idea of tagging objects with geometry does not provide good results, also as mentioned before, the heading of the downloaded images is not fixed and gets changed from time to time.

Figure 7.1: Resultant marker when using the geometric mean and clustering approch

The results are quite wrong, and duplicate images make a huge difference in it. Also, due to the lack of heading in the picture, it was no reason to put the marker in the location of the image + the geometric parameter in the image (in x and y). One of such resulting example of mapping can be seen in 7.1.

In the marker, the white marker represents the person, and the red one represents the car. It is hard to distinguish any of them due to high density. This example can also be seen as the same problem when the same object is placed four times and rectified with a clustering algorithm during convergence.

7.4 Mapping using depth and metadata

The updated object's location from the metadata and the depth with some normalization and scaling gives the approximate results. However, there are still some fail cases when the depth of images is at the extreme. Some predefined constant is set for such situations. These constants will interchange with depth if it passes the threshold.

Figure 7.2: Misclassification of the grill (purple mask) as the bench with confidence score of 0.97

7.5 Use of GSV API to download the sequence of image

As mentioned in the methodology section 6.9, though, the algorithm was able to grab most of the images available at the location. However, the number of requests to the server is significantly increased. The more straightforward solution is to use the polyline. However, this algorithm performs and returns only the access points, there might be some cases where polyline coordinates can be used, but no GSV image found at those locations.

7.6 Performance of Mask R-CNN on GSV dataset

Experimentation with the Mask R-CNN at 3 different locations in Prague.

1) 50.10291748018805, 14.39132777985096 Dejvice
2) 50.0795436,14.3907308 Strahov
3) 50.0746767,14.418974 Karlovo namesti

With the specified parameters of Google Streetview API as mentioned in 6.3 in Implementation section total 27 images were downloaded.

All images were processed with Mask R-CNN with Resnet101 as the backbone and pre-trained weight available with 81 classes. Among all 27 pictures, there was correct classification except eight misclassified objects, examples of such can be seen below.

In figure 7.2, Grill is misclassified as the bench.
In figure 7.3, street name text is classified as the car.
In figure 7.4, person is detected in between the trees with confidence 0.95.
In figure 7.5, there is no classification of the car at the instance.

Figure 7.3: Street name text (red mask) is classified as the car

Figure 7.4: Person is detected in between the trees with confidence 0.95

In an experiment of 1.5 sq mile area, 1000 downloaded images, and 2475 objects were detected. The evaluation of the detection was as follows.

The confusion matrix figure A.7 shows predicted vs actual value for each class. The objects which were classified wrong and did not belong to any of the class are put into background class (actual value) and add up as False positive. Similarly, the objects which were not detected but present in the image is put into the background class (predication value) add up as false negative.

The average precision of the Mask R-CNN on GSV images was 0.7644, and an average recall was 0.8591, which is calculated by taking the average of precision and recall from all the classes Considering no ground truth available and evaluation sets created manually.

7.7 Performance of GSV Images

Images from GSV are captured with a 360 camera system mounted on Google's vehicle. Also, Street View images can be in the form of continuous images at street or 360 images at some viewpoints. There are some cases in continuous street view images when the car changes its lanes. Also due to imagery captured with the vehicle, there might be some glitches at some location due to uneven speed, or the image stitching algorithm does not work correctly.

Figure 7.5: No detection(false negative) of class car (red car)

Figure 7.6: Shift of the lanes from left to right on straight road though the coordinates of image is at center of road

Figure 7.6 shows the shifting of lanes from left to right, but the coordinates of image location remain in the center of the street.

Glitches in GSV images can be seen in figure 7.7 The shift of lanes can cause loss of the region in which we are more interested in; the anomaly can be seen in figure 7.6

7.8 Performance of Depth Images

The depth image is stored in the ".npy" file, which is a form of an array, and it ranges from 0 - 255 per pixel. The depth value is rounded off to its whole number and can be seen in figure 7.8. The estimated depth value is proportional to the distance of the object respect to the camera. The results have significant variance through the experiment.

7.9 Performance of Folium

Folium is a python package that uses OpenStreetMap [64] maps to render and bind data to the map. Detected objects are placed in the overlying map as a marker. In a personal experiment, when the number of markers reaches more than 40,000, the map becomes unresponsive and shows no results. From section 8.2, It can be seen 0.01 square miles can hold 1200 + feature.

Figure 7.7: Glitchs in GSV

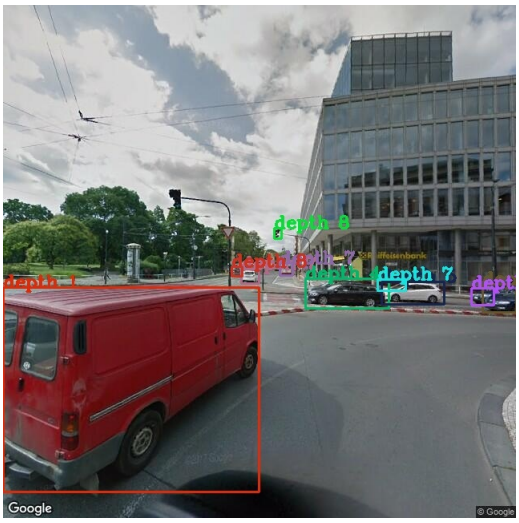


Figure 7.8: Visual reference of the estimated depth

Chapter 8

Results

Altogether thesis consists of data mining, usage of the machine learning algorithm, use of DNN, manipulation of the database, scripting in Python, computer vision, API integration problems. The work done converged into adequate usable solution with an exploration of the state of the art and usage of them to prepare the application.

The software outputs a vectorized description projected and visualized over an underlying map. The vectorized description is in the form of a marker placed on the map.

8.1 Performance of localization

Some examples of the objects which are localized by the application with the maker on the map.

Figure 8.1, 8.2 shows there position in the map 8.3 Purple marker shows traffic light and light Orange shows potted plant.

8.2 Performance on large data set

An example of showing detection done in various locations covering 17.47 sq. Miles area shown in figure 8.4. There were 15,877 downloaded images. The bounding box shows the area of interest, and the markers within the box are the results generated after the tests.

Table 8.1 showing the number of detections per class. Top 5 classes during an test were

Cars with 60972 detections
Person with 15196 detections
Truck with 1974 detections
Bench with 1285 detections

Figure 8.1: Downloaded images at 0 and 90 heading

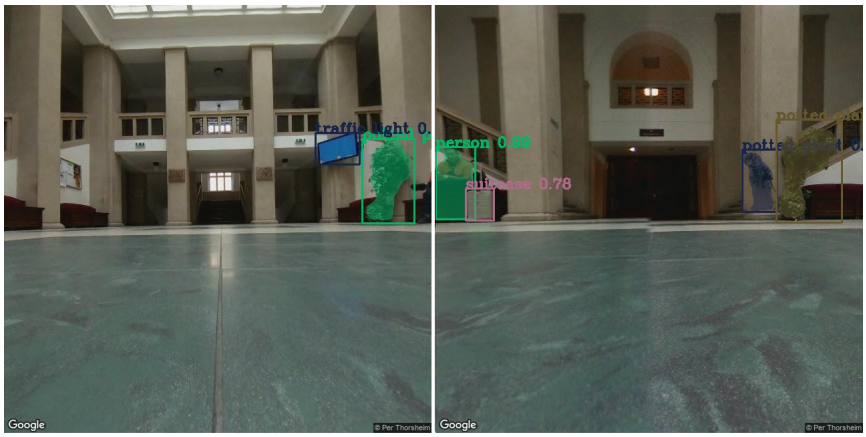


Figure 8.2: Downloaded images at 180 and 270 heading

potted plant with 1282 detections

There were several other classes with few detections, which can be inferred for the table 8.1.

The Mask R-CNN pre-trained model was trained with 81 classes. Out of them, there are some classes like elephant and giraffe where the expectancy is low on the streets. These classes often show misclassification. Example of the detection of two giraffes in the area shown in figure 8.4, and both were misclassified and can be seen in figure 8.5.

On an average of the test area of 0.01 sq. miles, it took approximately 200 sec on the Google Colab notebook to complete the whole process. There were 192 images, and the number of detected features were 1263 for the following test.

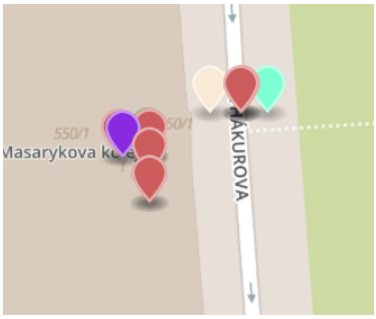


Figure 8.3: Markers of the detected class on the map , purple marker shows the traffic light light orange shows potted plant

Class name	Number of detections
BG	0
person	15196
bicycle	173
car	60972
motorcycle	201
airplane	19
bus	889
train	459
truck	1974
boat	1096
traffic light	1261
fire hydrant	287
stop sign	1066
parking meter	46
bench	1285
bird	38
cat	0
dog	16
horse	11
sheep	3
cow	18
elephant	7
bear	1
zebra	0
giraffe	10
backpack	91
umbrella	195
handbag	142
tie	2
suitcase	94
frisbee	11
skis	17
snowboard	4
sports ball	9
kite	19
baseball bat	0
baseball glove	0
skateboard	14
surfboard	15
tennis racket	4
bottle	181

Class name	Number of detections
wine glass	5
cup	37
fork	10
knife	3
spoon	1
bowl	34
banana	0
apple	0
sandwich	0
orange	0
broccoli	8
carrot	0
hot dog	0
pizza	0
donut	0
cake	3
chair	1030
couch	22
potted plant	1282
bed	14
dining table	208
toilet	52
tv	68
laptop	15
mouse	4
remote	5
keyboard	17
cell phone	2
microwave	1
oven	5
toaster	0
sink	62
refrigerator	42
book	133
clock	132
vase	39
scissors	1
teddy bear	2
hair drier	0
toothbrush	0

Table 8.1: Number of detection with their classes

8.3 Drawback

This section describes the shortcoming of the methods and algorithms used in the thesis.

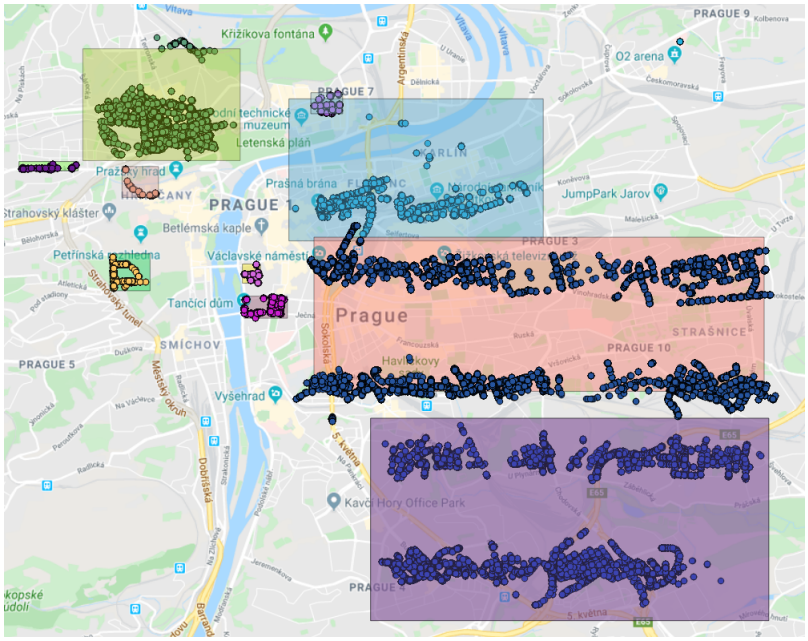


Figure 8.4: Combination of various database together with bounding box as given input and marker points as the output

8.3.1 Location of the markers are not correct

The depth of images is estimated with the state-of-the-art DNN. Still, there was a difference in the ground truth and the estimated value. Keeping in mind that the information is purely based on a 2D RGB image. The measured value doesn't always converge to have closer or expected value. There was variance with different scenarios according to the image. A panoramic 360 image is divided into four parts according to their heading angle. An image can consist of many objects with a different perspective and orientation. Finding the angle of the object with the 2D image can be tricky and out of the scope of the thesis topic.

8.3.2 Google Colab

Though the final software is implemented and visualized in the Google Colab notebook, the development was done with the local computer. The reason was due to limitations in the period of run time. Google colab allows only 12 hours of run time in a single run, which can disturb the progress when the test is performed on a large area. Also, during the period of 2018 on individual experience, the colab notebook crashed 2 out of 10 times during development.

(a) : Misclassification of the giraffe with confidence score of 0.95

(b) : Misclassification of the giraffe with confidence score of 0.75

Figure 8.5: Giraffes detected during test

8.4 Future work

8.4.1 Training own dataset

Training can be done for both classification, segmentation, and depth model. The pre-trained model used for classification and segmentation consists of 81 classes for detection. The images from GSV can be labeled for training the model with the classes of interest.

8.4.2 Downloading sequence of images

In order to eliminate cases of multiple detections, images can be downloaded, which are closer to each other in terms of their coordinates. The images of the adjacent coordinate can be processed with feature extraction, to find the same objects in the images. If the same object is found multiple times, the mean position of the object can be used as its original position.

One approach could be to download the images sequentially by using google direction API to get the location.

The output from GSV results with sequential images like figure 8.6.

Direction API and Google street view API

There might be some cases when direction API would result in the polyline, which gives the sequence of waypoint image from our starting point to endpoint. In the series of the image, the same object can be detected using feature extraction and can be eliminated.

Figure 8.6: Sequence of images downloaded from left to right

Chapter 9

Conclusion

This thesis was aimed to design, implement, and evaluate deep neural networks solution for city mapping using Google Street View imagery. Design and implementation of software have been discussed in chapter 5 along with methodology in chapter 6. Evaluation of state-of-the-art network was done in chapter 7 with results shown in chapter 8. The solution uses two state-of-the-art networks to develop a solution. The implementation is done in Python with Keras, TensorFlow, and Pytorch frameworks on existing pre-trained models. The user interface for the application execution, processing of the input images, and visualization of the results were realized using Google Colab. Comparison with previous state-of-the-art was analyzed and described in chapter 3.

The necessary proof of concept is demonstrated with the output in Google Colab. Different approaches have been discussed with their pros and cons for the individual block of the pipeline with scope of improvement in future work. The software outputs a vectorized description projected and visualized over an underlying map. The vectorized description is in the form of a marker placed on the map. The evaluation of Mask R-CNN on GSV images gives the average precision value of 0.7191 and an average recall of 0.7684 when tested over more than 1000 images manually.

The software solution can result in an application that can be used to find different objects located in space. A neural network could be trained to find a particular object and find it in the area. Further, it can be trained to detect an anomaly in the vast data set, which is quite dull and tedious work for a human. It can be used used to analyze the scenario with a different GSV timeline. There comes an infinite possibility to use this a base to create a specific application. Some of the example like to detect the number of trash and cans in the particular area to implement the optimum amount of trash bin on streets or to find the density of plants in a area or maybe find number of human in streets to find out the busiest streets in the city or finding free space to put more ATM or finding the nearest fire hydrant in case of fire or vice versa.

Appendix A

Pictures

Class name	Number of detections
BG	0
person	77
bicycle	13
car	762
motorcycle	4
airplane	0
bus	1
train	0
truck	19
boat	24
traffic light	9
fire hydrant	0
stop sign	10
parking meter	0
bench	8
bird	0
cat	0
dog	0
horse	0
sheep	0
cow	0
elephant	0
bear	0
zebra	0
giraffe	0
backpack	0
umbrella	2
handbag	1
tie	0
suitcase	0
frisbee	0
skis	0
snowboard	0
sports ball	0
kite	0
baseball bat	0
baseball glove	0
skateboard	0
surfboard	1
tennis racket	0
bottle	0

Class name	Number of detections
wine glass	0
cup	0
fork	1
knife	1
spoon	1
bowl	0
banana	0
apple	0
sandwich	0
orange	0
broccoli	0
carrot	0
hot dog	0
pizza	0
donut	0
cake	0
chair	10
couch	0
potted plant	55
bed	0
dining table	6
toilet	0
tv	0
laptop	3
mouse	1
remote	1
keyboard	0
cell phone	0
microwave	0
oven	0
toaster	0
sink	1
refrigerator	0
book	5
clock	2
vase	2
scissors	0
teddy bear	0
hair drier	0
toothbrush	0

Table A.1: Number of detected objects per class within area of interest(Munich)

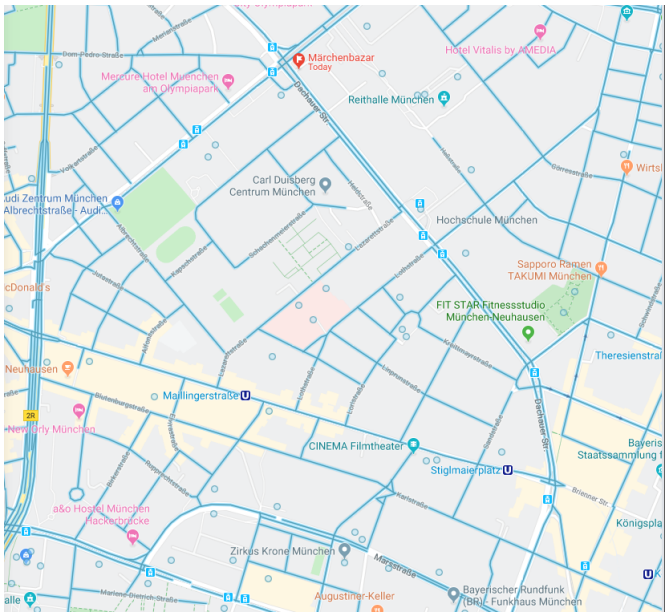


Figure A.1: Availability of GSV images in Munich

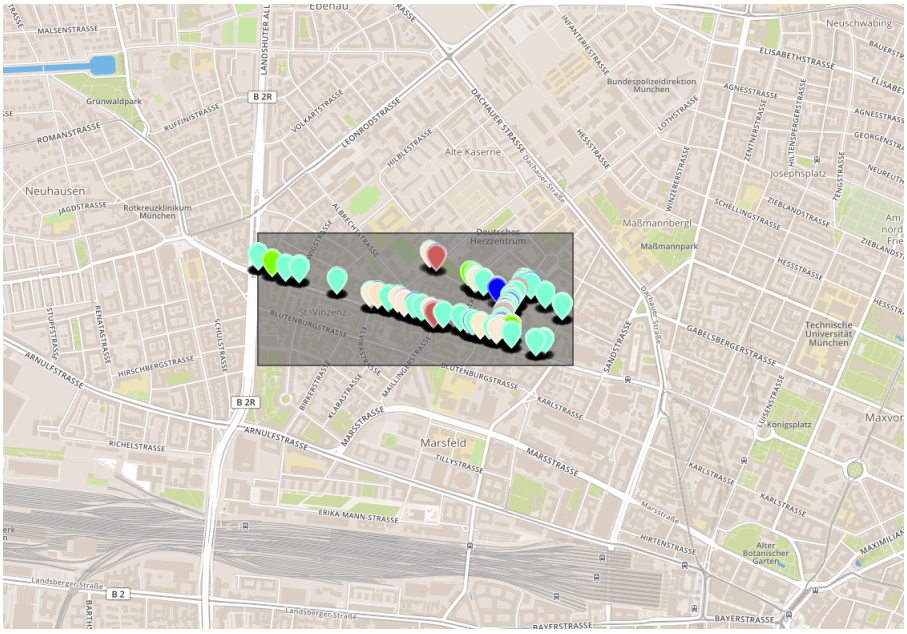


Figure A.2: Munich street with input bounding box (black) with the resulting marker

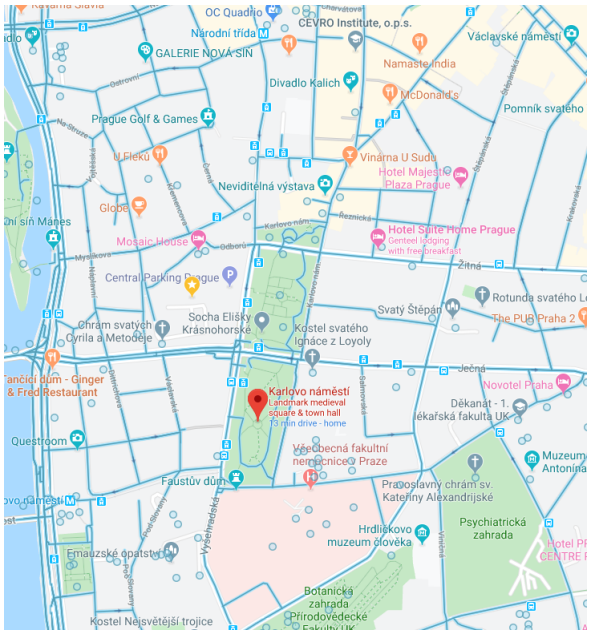


Figure A.3: Availability of GSV images in Prague street

Figure A.4: Street in Prague with area of interest as bounding box (black) with the resulting marker

Class name	Number of detections
BG	0
person	77
bicycle	13
car	762
motorcycle	4
airplane	0
bus	1
train	0
truck	19
boat	24
traffic light	9
fire hydrant	0
stop sign	10
parking meter	0
bench	8
bird	0
cat	0
dog	0
horse	0
sheep	0
cow	0
elephant	0
bear	0
zebra	0
giraffe	0
backpack	0
umbrella	2
handbag	1
tie	0
suitcase	0
frisbee	0
skis	0
snowboard	0
sports ball	0
kite	0
baseball bat	0
baseball glove	0
skateboard	0
surfboard	1
tennis racket	0
bottle	0

Class name	Number of detections
wine glass	0
cup	0
fork	1
knife	1
spoon	1
bowl	0
banana	0
apple	0
sandwich	0
orange	0
broccoli	0
carrot	0
hot dog	0
pizza	0
donut	0
cake	0
chair	10
couch	0
potted plant	55
bed	0
dining table	6
toilet	0
tv	0
laptop	3
mouse	1
remote	1
keyboard	0
cell phone	0
microwave	0
oven	0
toaster	0
sink	1
refrigerator	0
book	5
clock	2
vase	2
scissors	0
teddy bear	0
hair drier	0
toothbrush	0

Table A.2: Number of detected object per class within area of interest(Prague)

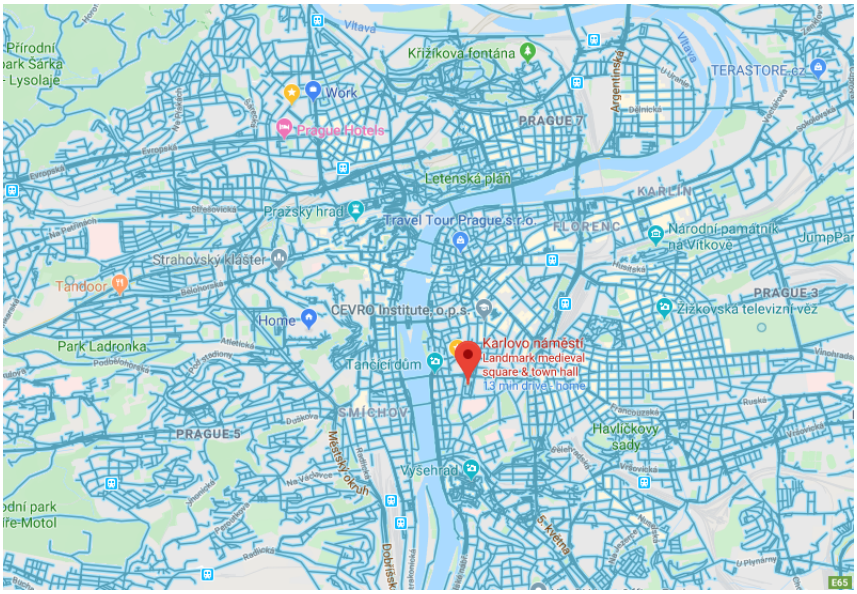


Figure A.5: Availability of GSV images at area of interest

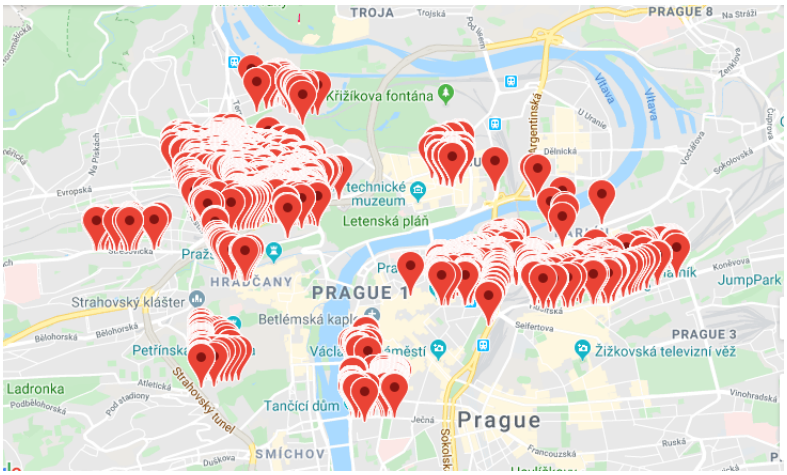


Figure A.6: Resulting visualization of large scale map with 17336 detections

Class name	Number of detections
BG	0
person	3121
bicycle	46
car	11717
motorcycle	73
airplane	2
bus	226
train	112
truck	322
boat	264
traffic light	313
fire hydrant	53
stop sign	193
parking meter	9
bench	229
bird	6
cat	0
dog	3
horse	1
sheep	2
cow	1
elephant	2
bear	0
zebra	0
giraffe	2
backpack	29
umbrella	32
handbag	24
tie	2
suitcase	13
frisbee	3
skis	0
snowboard	1
sports ball	2
kite	1
baseball bat	0
baseball glove	0
skateboard	4
surfboard	4
tennis racket	1
bottle	21

Class name	Number of detections
wine glass	2
cup	9
fork	1
knife	1
spoon	0
bowl	16
banana	0
apple	0
sandwich	0
orange	0
broccoli	0
carrot	0
hot dog	0
pizza	0
donut	0
cake	0
chair	178
couch	3
potted plant	155
bed	0
dining table	28
toilet	25
tv	11
laptop	2
mouse	0
remote	0
keyboard	1
cell phone	1
microwave	1
oven	1
toaster	0
sink	18
refrigerator	10
book	18
clock	20
vase	1
scissors	0
teddy bear	0
hair drier	0
toothbrush	0

Table A.3: Number of detected object per class

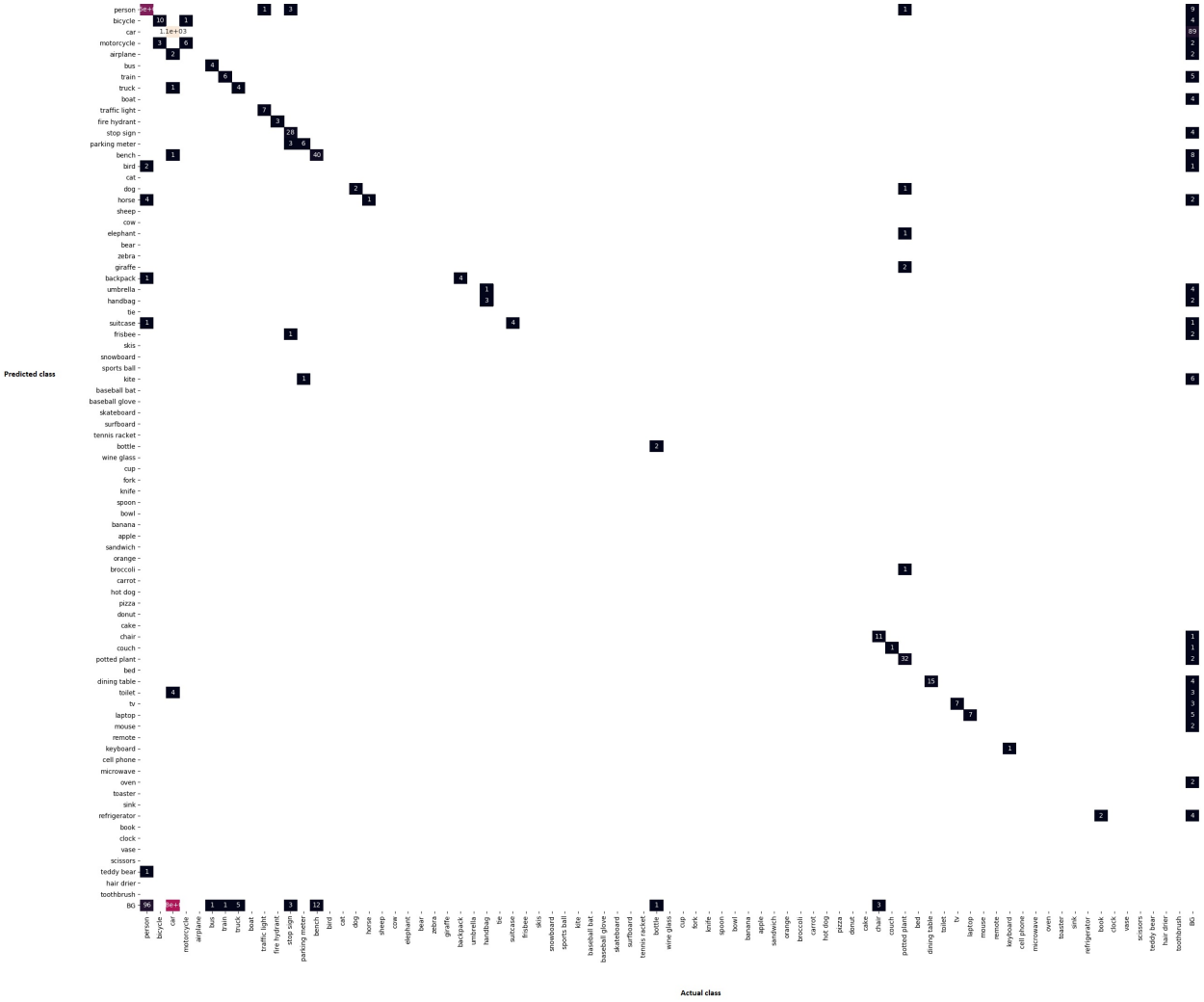


Figure A.7: Confusion matrix of 81 classes

Appendix B

File structure

B.1 Structure of Files

- Root directory
  - monodepth2 - CNN package for depth estimation
    - ...
  - Mask RCNN - CNN package for object classification and segmentation
    - ...
  - box.txt - bounding box coordinates from geojson file
  - input.geojson - input geojson file
  - run_all.sh - bash script to run program sequentially
  - requirement.txt - requirements for the environment
  - workspace - directory of python scripts
    - database - directory for all the database created
      - ...
    - readgeojson.py - read input.geojson file
    - build_query.py - python script to build query
    - gsv_down.py - script to download GSV images
    - filehan.py - script for file rename and editing
    - CNNdet.py - script for classification and segementation(MaskRCNN)
    - CNNdepth.py - script to depth CNN(Monodepth2)
    - anadepth_depth.py - script to analyze depth
    - cre_map.py - script for creating maps
    - mergemap.py - script for mergring maps and outputs
    - finaldetdic.json - dictionary consist of all detection from all database t
    - finalmap.geojson - combined geojson file of all detections
    - overallstatus.png - graph number of detection vs class
    - statusinnumber.txt - statistic number of detection for each class

B.2 Structure of Downloads

```
Downloads - download images from GSV API
├─0 - batch 1
│  ├─gsv1.jpeg - images
│  ├─metadata.json - metadata from API
│  └─...
├─1 - batch 2
│  ├─gsv1.jpeg - images
│  ├─metadata.json - metadata from API
│  └─...
└─...
```

B.3 Structure of Masks

```
Mask - numpy mask file which should be applied on depth image
├─filename1(pano_id)
│  ├─Mask0 - numpy mask file
│  └─...
├─filename2(pano_id)
│  ├─Mask0 - numpy mask file
│  └─...
└─...
```

B.4 Structure of Mask_depth

```
Mask_depth - visualization of resulting depth when mask is used
├─filename1(pano_id)
│  ├─Mask0.jpeg
│  └─...
├─filename2(pano_id)
│  ├─Mask0.jpeg
│  └─...
└─...
```

B.5 Structure of database

- Database
  - 0 - Database1
    - Downloads - download images from GSV API
      - ...
    - imagedataset - Renamed images wiht pano_id
      - ...
    - Detection - Processed images with segmentation and classification
      - ...
    - Depth - Processed images with depth estimation in form of gray scale and r
      - ...
    - Mask_depth - visualization of resulting depth when mask is used
      - ...
    - Mask - numpy mask file
      - ...
    - Maps - Map directory for each class
      - class1.html
      - ...
    - det_dict.json
    - map.html
    - hmap.html - heat map of the database
    - img_db.json -combined dictionary from all metadata from each batch
    - query_points.txt - list of query points for each batch
    - myfile.geojson - out put geojson file for each batch
    - final_dict.json - dictionary which get updated during the pipeline
    - mask_dict.json - information consisting mask file name and depth
    - status.png - graph number of detection vs class
    - statusinnumber.txt - statistic number of detection for each class
  - 1 - Database2
    - ...
  - ...

Appendix C

Bibliography

[1] Stanford Vision Lab, "Large Scale Visual Recognition Challenge (ILSVRC)," 2015.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[3] Developers.google.com, "Developer Guide," 2018.

[4] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 2980–2988, 2017.

[5] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth prediction," October 2019.

[6] A. Chandra, "How do you decide on what filters to use in CNN?," 2018.

[7] P. Pošík and P. Pošík, "CZECH TECHNICAL UNIVERSITY IN PRAGUE Faculty of Electrical Engineering Department of Cybernetics Linear Methods for Regression and Classification," tech. rep.

[8] P. Pošík and P. Pošík, "CZECH TECHNICAL UNIVERSITY IN PRAGUE Faculty of Electrical Engineering Department of Cybernetics Neural Networks. Introduction and Rehearsal," tech. rep.

[9] Juan Miguel Valverde, "Activation Functions in Deep Learning (Sigmoid, ReLU, LReLU, PReLU, RReLU, ELU, Softmax) – Lipman's Artificial Intelligence Directory."

[10] Http://cs231n.stanford.edu/, "Convolutional Neural Networks (CNNs / ConvNets)," 2019.

[11] E. Villa, "Convolutional Neural Network," 2019.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Enzyme and Microbial Technology*, vol. 19, pp. 107–117, dec 2015.

[13] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 936–944, 2017.

[14] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *CVPR*, 2017.

[15] J. A. (https://stats.stackexchange.com/users/153844/josh albert), "How to build a confusion matrix for a multiclass classifier?." Cross Validated. URL:https://stats.stackexchange.com/q/338240 (version: 2019-07-31).

[16] G. Develper, "google-streetview 1.2.9," 2019.

[17] T. Zhang and R. Ma, "Identify the cells' nuclei based on the deep learning neural network," *arXiv e-prints*, p. arXiv:1911.09830, nov 2019.

[18] "Decoding the Confusion Matrix - Towards Data Science."

[19] G. Stockman and L. G. Shapiro, *Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2001.

[20] Geojson, "GEOJSON," 2016.

[21] Colab.research.google.com, "Welcome to Colaboratory!."

[22] Python Software Foundation, "Python," 2019.

[23] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[24] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] Developer.google.com, "Direction API," 2019.

[27] T.-Y. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft {COCO:} Common Objects in Context," *CoRR*, vol. abs/1405.0, 2014.

[28] K. Kita-wojciechowska and E. Sciences, "Google Street View image of a house predicts car accident risk of its resident,"

[29] S. Law, B. Paige, and C. Russell, "Take a Look Around: Using Street View and Satellite Images to Estimate House Prices," 2018.

[30] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: the kitti dataset," *The International Journal of Robotics Research*, vol. 32, pp. 1231–1237, 09 2013.

[31] M.-Y. Liu, S. Lin, S. Ramalingam, and O. Tuzel, "Layered Interpretation of Street View Images," 2015.

[32] J. Kang, M. Körner, Y. Wang, H. Taubenböck, and X. X. Zhu, "Building instance classification using street view images," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, pp. 44–59, 2018.

[33] V. A. Krylov, E. Kenny, and R. Dahyot, "Automatic discovery and geotagging of objects from street view imagery," *Remote Sensing*, vol. 10, no. 5, 2018.

[34] S. Suresh, N. Chodosh, and M. Abello, "Deepgeo: Photo localization with deep neural network," *CoRR*, vol. abs/1810.03077, 2018.

[35] M. A. Sudharshan Suresh, Nathaniel Chodosh, "50States10K," 2018.

[36] O. Samano Abonce, M. Zhou, and A. Calway, "You Are Here: Geolocation by Embedding Maps and Images," *arXiv e-prints*, p. arXiv:1911.08797, Nov 2019.

[37] Osmlab, "Overpass API Documentation," 2015.

[38] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," *arXiv e-prints*, p. arXiv:1511.07247, Nov 2015.

[39] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *arXiv e-prints*, p. arXiv:1505.04597, may 2015.

[40] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," *arXiv e-prints*, p. arXiv:1608.06993, aug 2016.

[41] X. Li, H. Chen, X. Qi, Q. Dou, C.-W. Fu, and P. A. Heng, "H-DenseUNet: Hybrid Densely Connected UNet for Liver and Tumor Segmentation from CT Volumes," *arXiv e-prints*, p. arXiv:1709.07330, sep 2017.

[42] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss Functions for Neural Networks for Image Processing," *arXiv e-prints*, p. arXiv:1511.08861, nov 2015.

[43] "Package/Tikz." \url{http:https://github.com/PetarV-/TikZ/}, 2018.

[44] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[45] "Complete Guide of Activation Functions - Towards Data Science," 2019.

[46] Wikipedia, "Convolution," 2019.

[47] J. Brownlee, "A Gentle Introduction to Pooling Layers for Convolutional Neural Networks," 2019.

[48] R. Girshick, "Fast R-CNN," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 1440–1448, 2015.

[49] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," 2015.

[50] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 1800–1807, 2017.

[51] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017.

[52] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2018.

[53] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 2261–2269, 2017.

[54] B. Zoph and J. Shlens, "Learning Transferable Architectures for Scalable Image Recognition,"

[55] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster {R-CNN:} Towards Real-Time Object Detection with Region Proposal Networks," *CoRR*, vol. abs/1506.0, 2015.

[56] En.wikipedia.org, "Bilinear interpolation," 2019.

[57] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *Journal of Urology*, vol. 131, pp. 262–263, feb 2016.

[58] J.-W. Bian, Z. Li, N. Wang, H. Zhan, C. Shen, M.-M. Cheng, and I. Reid, "Unsupervised Scale-consistent Depth and Ego-motion Learning from Monocular Video," tech. rep.

[59] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep Ordinal Regression Network for Monocular Depth Estimation," *CoRR*, vol. abs/1806.02446, 2018.

[60] Z. Yang, P. Wang, Y. Wang, W. Xu, and R. Nevatia, "{LEGO:} Learning Edge with Geometry all at Once by Watching Videos," *CoRR*, vol. abs/1803.05648, 2018.

[61] A. Ranjan, V. Jampani, K. Kim, D. Sun, J. Wulff, and M. J. Black, "Adversarial Collaboration: Joint Unsupervised Learning of Depth, Camera Motion, Optical Flow and Motion Segmentation," *CoRR*, vol. abs/1805.09806, 2018.

[62] R. Garg, V. K. B. G, and I. D. Reid, "Unsupervised {CNN} for Single View Depth Estimation: Geometry to the Rescue," *CoRR*, vol. abs/1603.04992, 2016.

[63] R. Story, "Folium," 2019.

[64] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ." `https://www.openstreetmap.org`, 2017.

[65] M. Bostock, "TopoJSON," 2016.

[66] F. Seide and A. Agarwal, "Cntk: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 2135–2135, ACM, 2016.

[67] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Bleecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. Ebrahimi Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[68] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS-W*, 2017.

[69] A. Gelbukh, "Natural language processing," in *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, pp. 1 pp.–, nov 2005.

[70] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter Notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (F. Loizides and B. Schmidt, eds.), pp. 87–90, IOS Press, 2016.

[71] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A brief introduction to OpenCV," *MIPRO, 2012 Proceedings of the 35th International Convention*, pp. 1725–1730, 2012.

[72] "Approximate Metric Equivalents for Degrees." ://www.usna.edu/Users/oceano/pguth/md_help/html/approx_equivalents.htm.

[73] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science Engineering*, vol. 13, pp. 22–30, mar 2011.

[74] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully Convolutional Instance-aware Semantic Segmentation," *arXiv e-prints*, p. arXiv:1611.07709, nov 2016.

[75] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow." `https://github.com/matterport/Mask_RCNN`, 2017.

[76] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[77] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[78] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 881–892, July 2002.

[79] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "{API} design for machine learning software: experiences from the scikit-learn project," in *ECML*

PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122, 2013.

[80] Developers.google.com, "Encoded Polyline Algorithm Format," 2018.

[81] "GitHub - hicsail/polyline: A Python implementation of Google's Encoded Polyline Algorithm Format.."