Poznan University of Technology
Faculty of Electrical Engineering
Institute of Control, Robotics and Information Engineering

Bachelor's thesis

APPEARANCE BASED LOCALIZATION EMPLOYING DEEP LEARNING

Tomasz Filanowicz
Filip Szymański

Supervisor
prof. Piotr Skrzypczyński PhD DSc

Poznań, 2019

Contents

Abstract

This bachelor's thesis describes project and implementation of an appearance based localization system based on deep learning. The introduction outlines directions of development and obstacles in modern robotics in particular in visual localization task. This topic is extended in the following chapter where existing solutions are discussed and a new approach is formulated. In the main part the developed solution is presented, as well as its theoretical support, external tools used and details of implementation. Experimental results obtained, are then described, summarized and possible improvements are proposed. The source code on which this thesis is based can be accessed at:
https://github.com/szymani/AppearanceBasedLocalization.

Streszczenie

Niniejsza praca inżynierska opisuje projekt oraz implementacje systemu do lokalizacji na podstawie widoków z wykorzystaniem głębokiego uczenia. Wstęp zarysowuje kierunki rozwoju jak i problemy obecnej robotyki, w szczególności zadania lokalizacji wizualnej. Ten temat rozwinięty jest w kolejnym rozdziale, gdzie omówione zostają istniejące obecnie rozwiązania oraz sformułowane zostaje podejście autorów. W głównej części zaprezentowane zostaje rozwiązanie wraz z podstawą teoretyczną, użytymi narzędziami zewnętrznymi oraz szczegółami implementacji. Otrzymane wyniki są następnie opisane i podsumowane po czym zaproponowane zostają możliwe udoskonalenia. Kod źródłowy na którym oparta jest niniejsza pracą dostępny jest pod adresem: https://github.com/szymani/AppearanceBasedLocalization.

This thesis is a teamwork of two students. Separation of duties was set as follows:

- Tomasz Filanowcz has written chapters 1, 5 and 8,
- Filip Szymański has written chapters 2 and 4.

The rest of the paper was completed collectively with a seamless assignment of tasks.

Chapter 1

Introduction

1.1 Aim

The aim of this work is to design and implement a solution enabling a mobile robot to perform unsupervised, yet efficient appearance based localization using deep neural networks. The proposed solution should be applicable in real word scenarios where environment conditions are different from those encountered at the learning stage.

1.2 Motivation

Nowadays, robot localization is a very fast advancing field of robotics. It is developed in many directions such as visual localization, indoor positioning system or GPS localization. Tendency for using visual localization lasts, and positively influence development of robotics systems. While more types of cameras and more computational power is available, the effectiveness of this approach increases and new architectures of neural networks are created.

There are several important aspects of appearance based localization to work properly. Main ones are surrounding detection and differentiation as well as analysis of collected views. However, main problem is successful localization of place in which the robot is currently located.

Popular vision systems are catadioptric cameras. Images obtained with use of it delivers broader spectrum and allow to analyse 360°view at one time. It leads to improvement of accuracy in visual localization.

There are several issues in visual localization systems. The main one is distorted image from catadioptric camera. Further, low resolution of obtained images and poor lighting conditions lead to reduction of accuracy in place recognition/classification. All algorithms mainly depends on quality and number of features of obtained images.

With progress in development of computing power and neural network architecture, the localization task can be performed with higher accuracy than before. Unfortunately, this technique requires more powerful hardware, than commonly used one to perform such complex tasks with use of convolutional neural networks. Similar problem applies to analysis of images, with increase of resolution, computing power has to increase as well. If computational resources are limited, the level of accuracy has to be sacirfied.

Chapter 2

Related Work

2.1 Visual Place Recognition

Traditionally, visual place recognition methods were based on hand-crafted descriptors. Application of such descriptors for topological place recognition spread videly after the publication of seminal paper by Cummins and Newman [8]. That hand-crafted descriptors worked well for specific problems. However, they have limitations. In [19] some of them were presented for life-long visual localization over datasets of more than 3000 km. More recent methods are often based on Convolutional Neural Networks. Here features are obtained by training. Results are usually surpassing those of hand-crafted solutions, especially if the task is not very specific. More attention was drawn to this approach after using it with great success for object classification by Krizhevsky in [2].

2.2 Appearance Based Localizaion

Subject of appearance based localization images was attempted by Roberto Arroyo in his work [7]. He proposed a solution based on hand crafted descriptors and Able for Binary-appearance Loop-closure Evaluation. Able especially, the ABLE-P version of this system is interesting, as it used panoramic images and Omnidirectional image can be transformed into panoramic picture. In ABLE-P each panoramic image is divided into subpanoramas, which are matched by using a cross-correlation technique, achieving more accurate similarity distances between a pair of panoramas. Later in the same paper Arroyo also tries to make use of powerful feature representations via CNN with his CNN-VTL (Convolutional Neural Network for Visual Topological Localization).

2.3 Re-identification and One-shot

In this work, the approach is different. The authors have noticed that this problem has many similarities to the person re-identification case. It is based not on classification, but on determining similarity between two images. Using results for each pair, a most probable answer is found. To achieve this, a vector of embeddings is calculated and then, by applying different methods, similarity is computed. This problem haas been taken on by Ejaz Ahmed in [3] who was focused on simultaneously finding an effective set of features and a corresponding similarity function. To knowledge of the authors, previously there have been two more papers concerning using deep learning for person re-identification: [9] and [22], but none about localization re-identification. If such a system with some modifications would be mounted on robot and used for lifelong localization, it could also be considered as a one-shot image recognition problem. One-shot learning is an aproach

to object categorization that aims to use only one or a few training images. In situation, when the robot leaves the area where images for training were taken, all new classes have only a single example. This increases complexity of the problem and usually leads to lower accuracy. There is a work [17] that uses Siamese neural networks for one-shot re-identification of letters of different alphabets. Solution presented in that paper was used by Tiago Freitas in [15] and is a starting point of this thesis.

Chapter 3

Artificial Neural Networks

To introduce Convolutional Neural Networks (CNN) a bit of historical background is required. Described networks are presented in sections 3.1 and 3.2. CNN is a subclass of Deep Neural Networks, which are Artificial Neural Networks, but with multiple layers between input and output layers, instead of just a single one.

3.1 Feedforward neural networks

First and simplest type of artificial neural network is Feedforward neural network. In this type information always moves in one directions, from the input, through hidden nodes and to the output. Artificial neurons are loosely inspired by biological neurons, i. e. brain cells, in that they receive input signals on multiple connections and only produce an output signal if a weighted sum of the inputs reaches a certain threshold [1].

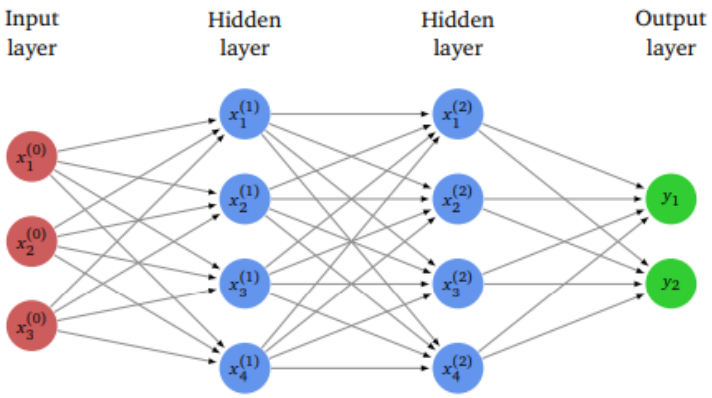


Figure 3.1: Feedforward neural network

3.2 Convolutional Neural Networks

3.2.1 Convolutional layers

Convolutional layers are basically filters moving over an image. In the past, values of these filters were hand picked by researchers depending on case and desired effect. Today, in convolutional layers, gradient descent algorithms are used for training, and achieving optimal weights. Using this approach proved to be much more effective. Four hyperparameters are available for configuration: number of filters, size of filter, stride and padding. Stride determines the spacing in width and height between consequent filter positions. Padding provides ability to increase, retain or decrease size of the input after application of convolutional layer, by padding the input with zeroes.

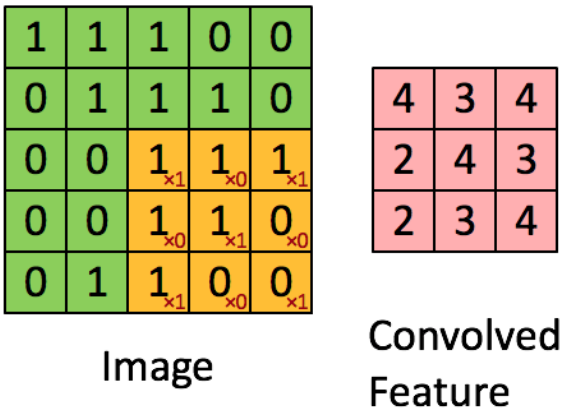


Figure 3.2: Principle of work of convolutional filter (in orange). Source: [23]

Polling is responsible for reducing the dimensions of input volume. Most common examples are *max pooling* or *average pooling*. Difference between those two is intuitive and can be easily understood from the graphic below (Fig 3.3).

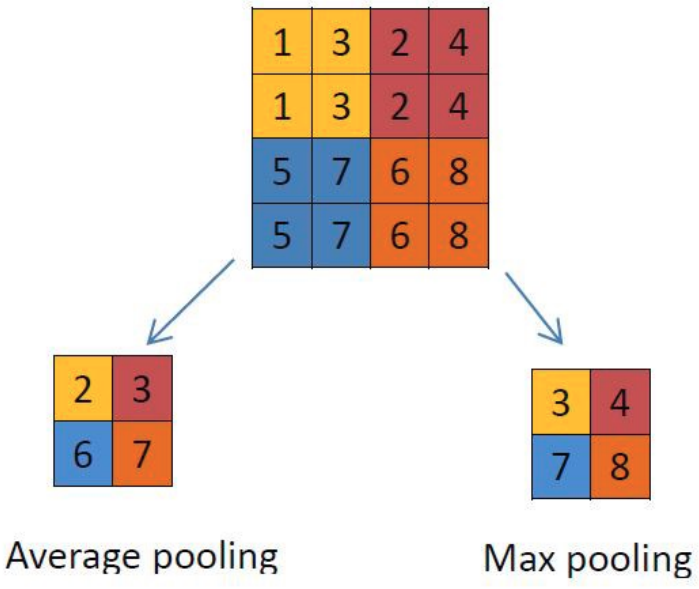


Figure 3.3: Comparison of max pooling and average polling. Source: [23]

Dense, or in other words fully connected layers are used for connecting each neuron with every value from previous layer. It is an extension of feedforward neural network briefly discussed in section 3.1. In practice, before a dense layer, a flattening layer is used to transform volume into vector.

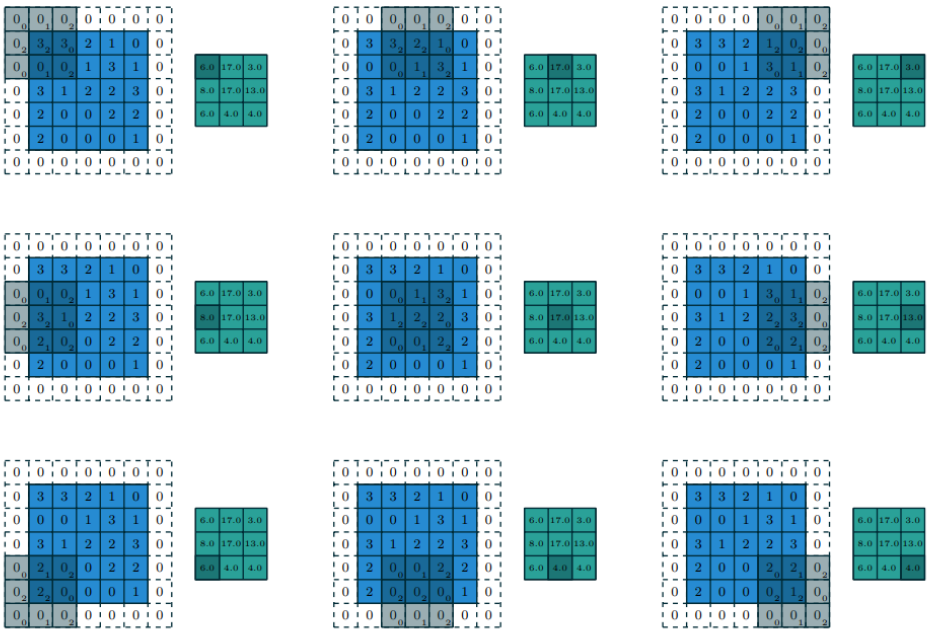


Figure 3.4: Input maps are blue, output maps are cyan. Filter position is denoted by shaded area. Kernel(filter) size is 3x3, input size is 5x5, stride value is 2x2, padding value is 1x1. Source: [23]

3.2.2 Loss functions

Loss function is used to train neural network. It measures inconsistency between predicted value and actual target. Value of function is always non-negative and strength of the model increase with decreasing loss.

Cross-Entropy

Cross-Entropy describes the performance in binary classification models, where output is assumed to take values 0 or 1. Multiclass Entropy is calculated as a sum of losses of every available class:

$$H(p,q) := -\sum_{x} p(x)log(q(x)), \tag{3.1}$$

where p and q are vectors.

Mean Squared Error

Mean Square Error commonly describe difference between two given vectors. It is widely used in linear regression as a performance measure. It is defined by:

$$H(p,q) := \frac{1}{n}\sum_{i=1}^{n}(p_i - q_i)^2, \tag{3.2}$$

where p and q are vectors of n values.

Triplet Loss

Triplet loss is a function that allows to learn output of similar embeddings (vectors) for data which belongs to the same class. In comparison to Mean Squared Error and Cross-Entropy, this method is not that popular due to difficulties in implementation. During loss computation, three vectors are created: first for anchor sample, second for positive sample and third for negative sample. Triplet loss is defined by:

$$H(a, p, n) := max(d(a, p) - a(a, n) + margin, 0), \tag{3.3}$$

where d is a distance measure between embeddings and margin denotes a desired minimal distance between *d(a,p)* and *d(a,n)*. Margin was implemented for purpose of dividing positive and negative clusters from each other in some space and for avoiding already correct vectors.

3.2.3 Optimizers

Optimization algorithms are very helpful during minimization or maximization of loss function which is function dependent on model parameters used to compute target values from given predictors. Optimizers are also vital when it comes to training of NN model. Two basic optimizers used nowadays in machine learning are Adaptive Moment Estimation (Adam) and Stochastic Gradient Descent (SGD).

Adam

Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments [14].

The Adam algorithm, is a method for efficient stochastic optimization where only first-order gradients are required without huge amount of memory. In this method, for different parameters individual learning rates are computed using estimates of gradients of first and second moments. Main advantages of Adam optimization are: magnitude of updates of parameter are unchangeable during rescaling of the gradient, stepsize hyperparameter bounds approximately step sizes, stationary objective is not required, it is able to work with sparse gradients.

SGD

Stochastic gradient-based optimization is important part of many fields of science and engineering. A lot of problems can be treated as parametrized objective function optimization involving optimization or minimization of its parameters. Gradient descent is an efficient method of optimization if the function is differentiable with respect to parameters.

In stochastic gradient descent (SGD) parameter update for each training sample is performed. In contrast to batch gradient descent where computations for large datasets are performed at every recomputation of gradients for similar examples, SGD performs one update at the time. This method is usually much faster. Mentioned optimization algorithm performs frequent updates with a high variance that cause heavily fluctuation of the objective function.

On the one hand, fluctuation allows to reach new potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting. Nevertheless, when the learning rate is decreased slowly, the same convergence behaviour as bath gradient is shown by SGD, it converges mostly to a local or global minimum for convex or non-convex optimization.

Modified SGD

Modified SGD algorithm is an adaptation of the original stochastic gradient descent, but it is modified specially to allow layer-wise learning rate and momentums as described in [17]

In this method, every layer is trained separately, one at the time. This allows selection of parameters differently for every layer, what leads to better performance in learning.

3.2.4 Activation functions

Artificial neurons, calculate a weighted sum of the input, add bias and after that decision if it should be activated or not is made. For the purpose of making this decision activation functions are implemented. To check value of the output an decide if the connections of given neuron should be activated.

Sigmoid

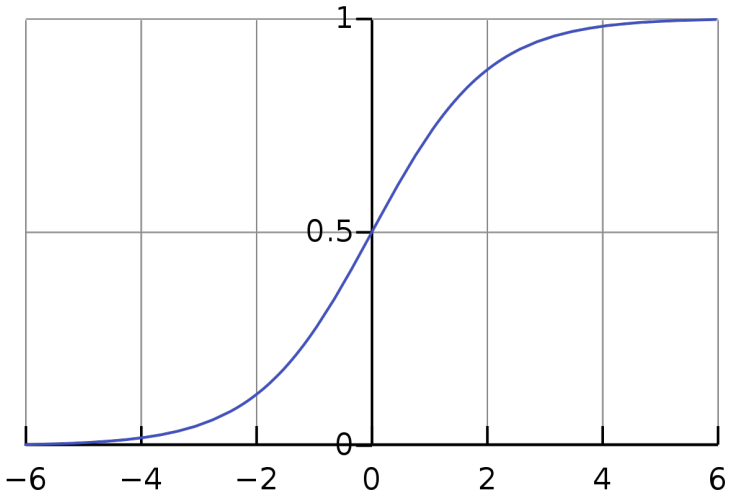


Figure 3.5: Sigmoid function

Sigmoid function is one of most commonly used type, as a activation function is very beneficial in terms of machine learning. Its nonlinear nature, makes combinations of its function also nonlinear what leads to stacking layers, and binary activations in comparison to step function.

It can be noticed in Fig. 3.5 that in range of input X between -2 and 2, output values are very steep. Very small changes in values of input, cause major changes in values of the output. This function has a tendency to bring the Y values to either end of the curve what makes clear distinctions while predicting. Next advantage is that, in contrast to linear functions, output value is bound in range (0,1).

Main problem, of sigmoid function is that in ends of functions, the output values respond poorly to changes in input. What causes the small gradient at this region. The problem of vanishing gradients is rised. The neural network do not learn more or learn extremely slow.

Tanh

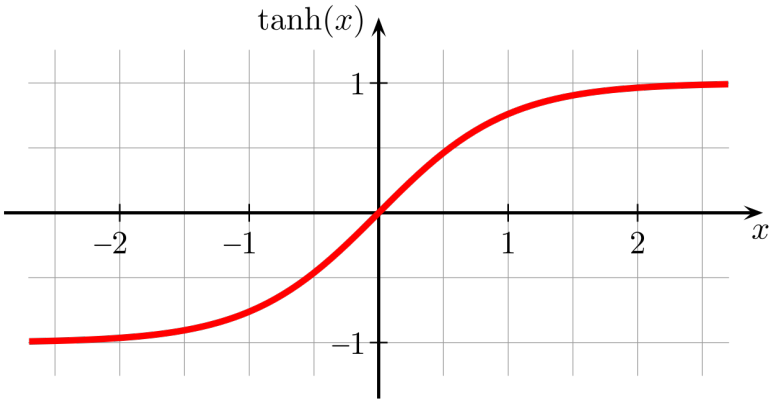


Figure 3.6: TanH function

Tanh as well as Sigmoid function is widely used. It is also naturally nonlinear what allows to stack layers. Its bounded in range (-1,1), like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered [1]. Significant difference is that gradient is stronger for tanh than sigmoid, it means that derivatives are steeper. In mentioned activation function shown in Fig. 3.6 also problem of vanishing gradients occurs.

ReLu

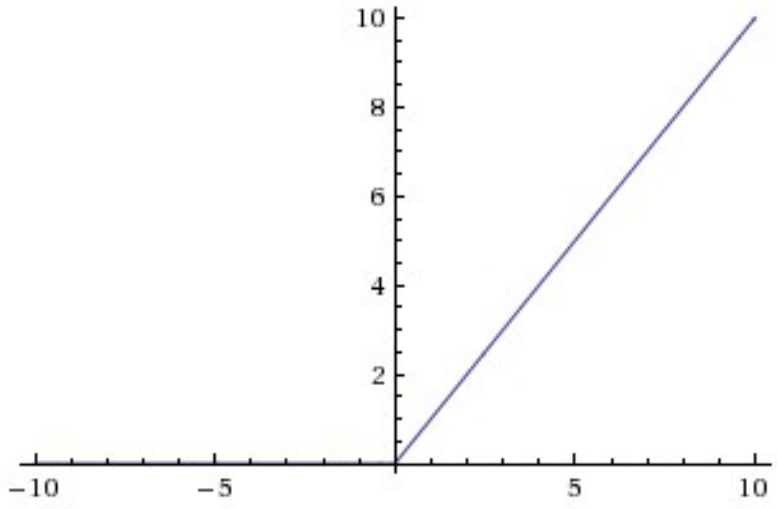


Figure 3.7: ReLu function

The ReLu function as it is shown in Fig. 3.7, gives an output x if x is positive and 0 otherwise. Whole function is nonlinear in nature, only positive axis has linear characteristics. Stacking of layers is possible. Strong benefit of ReLu function is activation sparsity. Because of its characteristics, for network with random initialized or normalized weights about 50 % of the network yields 0 activation. It means that the fewer neurons are acivated and the network is lighter.

Biggest disadvantage of ReLu is "dying ReLu problem". Because of horizontal line for negative values of the input, the gradient can go towards 0. The gradients for negative activations, obtain

0 as well, what causes neurons to stop responding to input in the given state. This problem can cause several neurons to just die and not respond making a substantial part of the network passive. One of solution of this problem is Leaky ReLu.

ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.

3.2.5 Overfitting countermeasures

Pooling

It is common to periodically insert a Pooling layer in-between Convolutional layers in a CNN architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

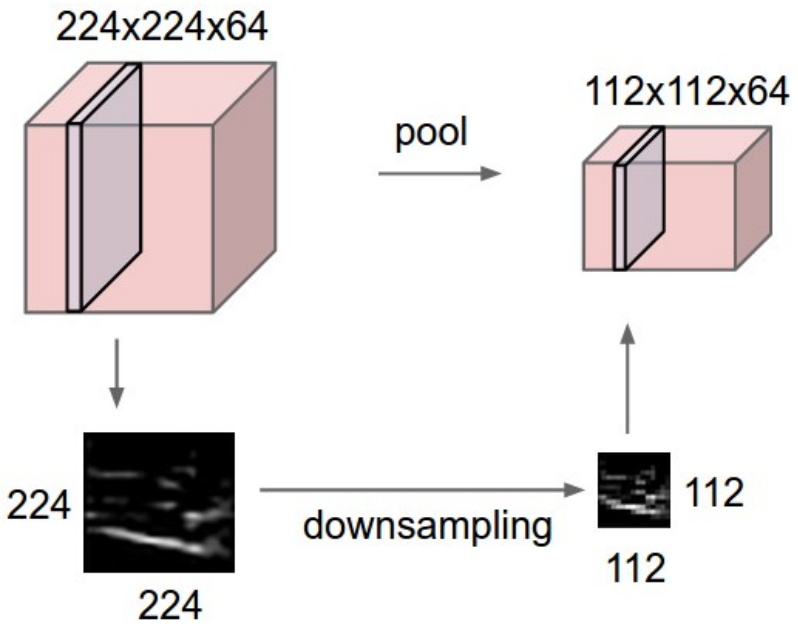


Figure 3.8: Pooling

Augmentation

To prevent overfitting and improve accuracy, augumentation technique can be implemented. It is the technique which increase size of the training dataset. Most augmentation techniques perform a 2D local transformation on the image like rotation and translation or color changes to generate multiple variations of one input image. With this technique, network learns bigger amount of different features from one image, what increases efficiency of the network [5].

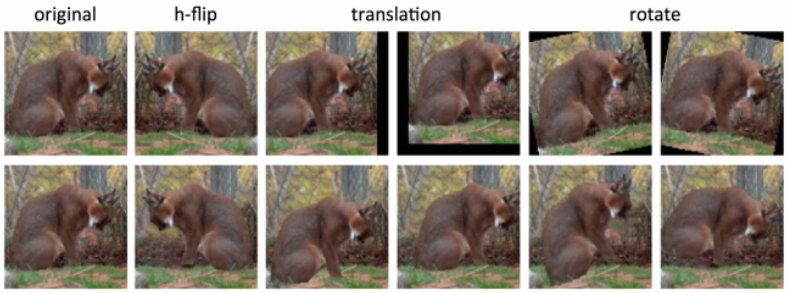


Figure 3.9: Augmentation

Dataset division

Additionally, the dataset is divided into three parts: one for training, one for validation and one for evaluation. With this procedure the risk of overfitting only to training set and inability to work well for new data is minimized. That topic is further described in chapter 5.

3.3 Siamese Neural Network

A siamese neural network is composed of two identical networks with shared weights that accept different inputs. Those two networks compute so-called embeddings, vectors that identify each image. Size of those resulting vectors is much smaller than size of input image. This makes later comparisons faster and saves computational resources. At the output of these sisterly networks is an energy function that calculates a difference function of choice. Some of the more popular distance functions:

- Manhattan distance (L1)

$$\sum_{i=1}^{n} |p_i - q_i| \tag{3.4}$$

- Euclidean distance (L2)

$$\sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{3.5}$$

The result of those function describes how similar are those two input images.

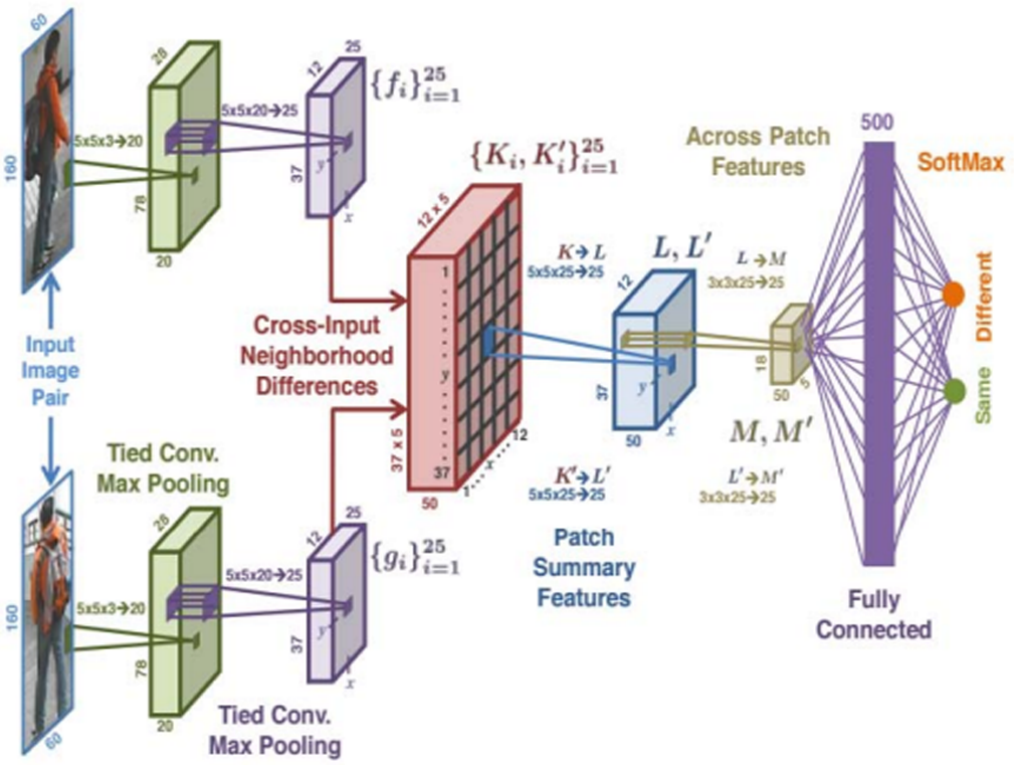


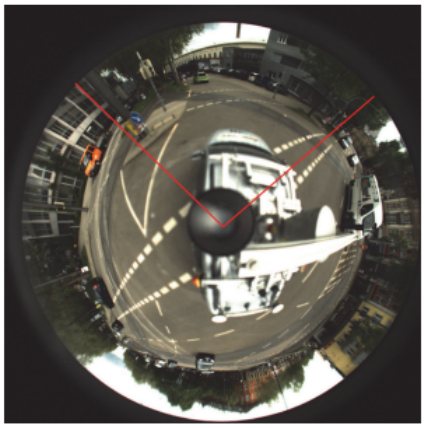
Figure 3.10: Exemplary usage of siamese neural network architecture for person re-.. identification. Source: [4]

Chapter 4

Software and hardware tools

4.1 Catadioptric vision system

In early optic systems catadioptric combinations were often used. Starting with catadioptric lighthouse reflectors constructed in 1820 followed by catadiopitric microscope in 1859 and many more to come. Today there exist another field for which they fit perfectly. They are heavily used in autonomous driving and robotics perception industry. Catadioptric camera at relatively cheap cost, overcomes the problem of a limited field provided by common perspective cameras and captures a panoramic image. Additional, but obvious, advantage is the ability of providing 360 degree field of vision.



(a) Catadioptric Image (Resolution 1400×1400, FOV $= 360°$)

(b) Perspective Image (Resolution 1392×512, FOV $\sim 90°$)

(c) Panoramic view of the same intersection scene

Figure 4.1: Comparison of images taken in the same place by different types of camera. Source: [20]

The structure of catadioptric camera is quite simple. The whole system consist of camera pointing the center of a curved mirror. By selecting appropriate distance between this two objects, omnidirectional image is attained. Camera used for obtaining dataset for this thesis was a project presented in [18] so architecture and software is thoroughly described inside. One thing worth mentioning is the fact that final version of this device was accomplished after the thesis was finished, due to the deadlines. The webcam on top of it is a part of another project, and was unused in this project.

Figure 4.2: Camera used for obtaining the dataset

4.2 Python

Python is a programing language first introduced in 1991 by Guido von Rossum. Philosophy behind it are summerized in *The Zen of Python* [21] with aphorisms like:

- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.

As shown above, readability was important for the developers and is forced by using significant whitespaces in syntax for managing program structure. It is interpreted and object-oriented, which makes it ideal for fast development, modularity and reusability. The debbuger is built into Python itself and raises exception when discovers an error. Another aspect is the Open Source. That favors creation of third party software. Libraries for various functionality such as Databases, Networking, Image proceessing and many more are developed. Easy syntax and dynamic type system encourage new programmers to use it and earns Python third place in latest TIOBE Index.

Dec 2018	Dec 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.932%	+2.66%
2	2		C	14.282%	+4.12%
3	4	▲	Python	8.376%	+4.60%
4	3	▼	C++	7.562%	+2.84%
5	7	▲	Visual Basic .NET	7.127%	+4.66%
6	5	▼	C#	3.455%	+0.63%
7	6	▼	JavaScript	3.063%	+0.59%
8	9	▲	PHP	2.442%	+0.85%
9	-	⇈	SQL	2.184%	+2.18%
10	12	▲	Objective-C	1.477%	-0.02%
11	16	⇈	Delphi/Object Pascal	1.396%	+0.00%
12	13	▲	Assembly language	1.371%	-0.10%
13	10	▼	MATLAB	1.283%	-0.29%
14	11	▼	Swift	1.220%	-0.35%
15	17	▲	Go	1.189%	-0.20%

Figure 4.3: TIOBE Programming Community Index [16]

4.3 Libraries

4.3.1 Keras

Keras is a high-level neural networks API written in Python. Main goals of the developers was to provide ability to make quick changes and fast experimentation. It is also very user friendly and easy to learn. That is one of the reasons why Keras had over 250,000 users as for mid 2018 according to [11]. Option to use one of three most popular backends: TensorFlow, Theano or CNTK, gives a developer great flexibilty yet he does not have to go low-level. It is backed by many leading companies, such as: Google (TensorFlow), Microsoft (CNTK), Amazon AWS (MXNet), NVIDIA, Uber and Apple (with CoreML).

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_1 (InputLayer)         (None, 120, 160, 1)       0
__________________________________________________________________________________________________
input_2 (InputLayer)         (None, 120, 160, 1)       0
__________________________________________________________________________________________________
sequential_1 (Sequential)    (None, 4096)              39598528    input_1[0][0]
                                                                   input_2[0][0]
__________________________________________________________________________________________________
lambda_1 (Lambda)            (None, 4096)              0           sequential_1[1][0]
                                                                   sequential_1[2][0]
__________________________________________________________________________________________________
dense_1 (Dense)              (None, 1)                 4097        lambda_1[0][0]
==================================================================================================
Total params: 39,602,625
Trainable params: 39,602,625
Non-trainable params: 0
```

Figure 4.4: Model summary generated with keras

Keras is definitely the backbone of this project. It was responsible for constructing neural network model and performing all training and evaluation operations (Listing 4.1).

```
import keras
convolutional_net = Sequential()
convolutional_net.add(Conv2D(filters=64, kernel_size=(10, 10),
                             activation='relu',
                             input_shape=self.input_shape,
                             kernel_regularizer=l2(
                                 l2_regularization_penalization['Conv1']),
                             name='Conv1'))
convolutional_net.add(MaxPool2D())

convolutional_net.add(Conv2D(filters=64, kernel_size=(7, 7),
                             activation='relu',
                             kernel_regularizer=l2(
                                 l2_regularization_penalization['Conv2']),
                             name='Conv2'))
convolutional_net.add(MaxPool2D())

convolutional_net.add(Conv2D(filters=64, kernel_size=(4, 4),
                             activation='relu',
                             kernel_regularizer=l2(
                                 l2_regularization_penalization['Conv3']),
                             name='Conv3'))
convolutional_net.add(MaxPool2D())

convolutional_net.add(Flatten())
convolutional_net.add(Dense(units=4096, activation='sigmoid',
                              kernel_regularizer=l2(
                                  l2_regularization_penalization['Dense1']),
                              name='Dense1'))

input_image_1 = Input(self.input_shape)
input_image_2 = Input(self.input_shape)
encoded_image_1 = convolutional_net(input_image_1)
encoded_image_2 = convolutional_net(input_image_2)

l1_distance_layer = Lambda(
    lambda tensors: K.abs(tensors[0] - tensors[1]))
l1_distance = l1_distance_layer([encoded_image_1, encoded_image_2])

prediction = Dense(units=1, activation='sigmoid')(l1_distance)

self.model.Summary()
self.model = Model(inputs=[input_image_1, input_image_2], outputs=prediction)
```

Listing 4.1: Fragment of code for resposible creating a model.

4.3.2 OpenCV

According to [12] OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learining software library. It is written in optimized C++ which enables multi-core processing. Developers have put focus on computational efficiency and real-time applications. OpenCV offers C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android.
Authors have chosen it because of previous exeperience as well as its popularity and availability of additional learning materials. This tool was used in scripts for capturing images taken by catadioptric camera that made the dataset.

```
        cam = cv2.VideoCapture(1)
        cv2.namedWindow('Live view')

        while rep != 0:
            ret_val, img = cam.read()
            img = cv2.flip(img, 1)
            cv2.imshow('Live view', img)
            pathh = "{}{}.png".format(file_path, (last + (total_num-rep)))
            cv2.imwrite(pathh, img)
            time.sleep(delay)
            k = cv2.waitKey(1)
```

Listing 4.2: Fragment of code resposible for capturing images

4.3.3 Matplotlib

According to [13] Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

fig, ax = plt.subplots()
ax.plot(t, s)

ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()

fig.savefig("test.png")
plt.show()
```

Listing 4.3: Sample code for basic ploting in Matpotlib. Source: [13]

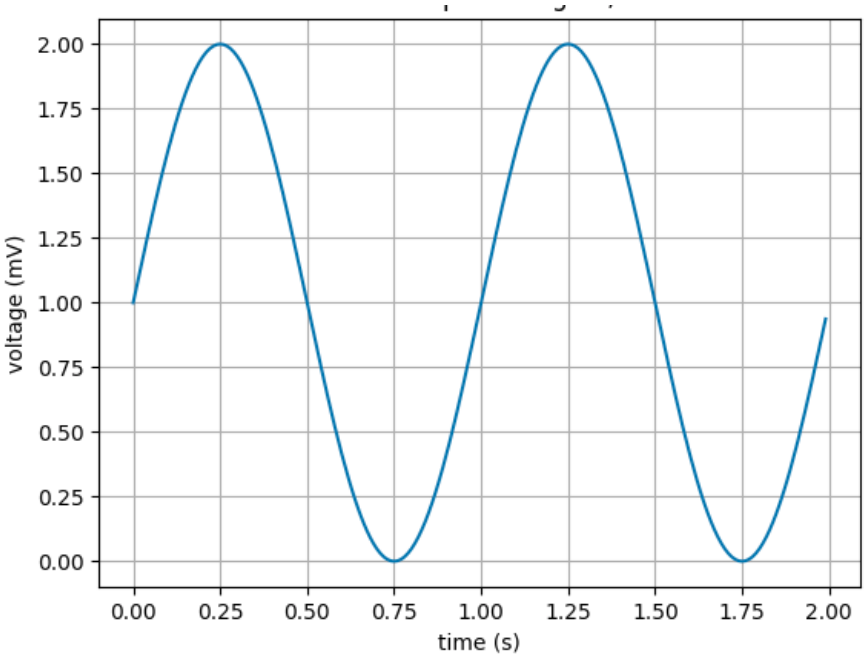


Figure 4.5: Result of code above. Source: [13]

This useful library was used to verify correctness of loader script as well as rough evalution of trends during training and experimentation with hyperparameters. In later stages it was also applied for presenting results of previous work.

4.3.4 NumPy

NumPy is a crucial package for scientific computing, early vesions were created by members of matrix-sig. It was initially meant for array computing on its N-dimensional array object. However the latest version provides much more funcionality. It includes basic linear algebra, random number capabilities, Fourier transforms as well as tools for integrating C/C++ Fortran code.

NumPy was used mainly during creating batches from loaded dataset. Its matrix operations and randomization capabilities proven essential.

```
1        import numpy as np
2        number_of_pairs = int(len(path_list) / 2)
3        pairs_of_images = [np.zeros((number_of_pairs, self.image_height,
4        self.image_width, 1)) for i in range(2)]
5        labels = np.zeros((number_of_pairs, 1))
```

Listing 4.4: Examples usage of NumPy for creating empty matricies of given size

Chapter 5

Dataset

Dataset is widely seen as collection of data. In machine learning development it is responsible for containing data used for training and evaluating of neural network models. Usually they are splitted into subsets training, validation and test set. The reason of using three subsets instead of two is that developing a model always involves tuning its configuration: for example, choosing the number of layers or the size of the layers (called the hyperparameters of the model, to distinguish them from the parameters, which are the network's weights). You do this tuning by using as a feedback signal the performance of the model on the validation data. In essence, this tuning is a form of learning: search for a good configuration in some parameter space. As a result, tuning the configuration of the model based on its performance on the validation set can quickly result in overfitting to the validation set, even though your model is never directly trained on it [6]. Additionally, with every tunning of parameter some part of validation data leaks to the model and as a result model performs only well on validation data instead of completely new one.

Figure 5.1: Basement

Dataset was specially created for purpose of this project. It contains about 1000 not interfered catadioptrical images. Resolution was decreased because of limited computing power from 640x480 to 160x120. All pictures were taken inside building of the Mechatronics Center at PUT. Due to crowd problem, dataset was mainly taken in evening hours, so mainly light in these photos comes from lamps from building. For easier differentiation by features, dataset includes 3 different floors

of the building: Basement Fig. 5.1, Ground floor Fig. 5.2. and Third Floor Fig. 5.3.. Every floor is of different colours of walls and of different arrangement.

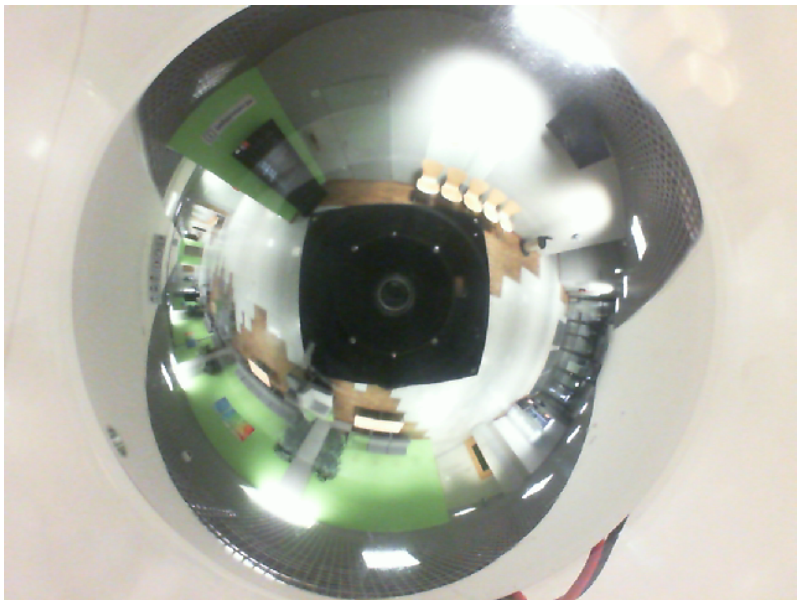


Figure 5.2: Ground Floor

Collected photos were splitted into three subsets, Ground and Third floor are assigned to training and validation set which ratio is 80/20. Whole Basement was taken as a Evaluation set.

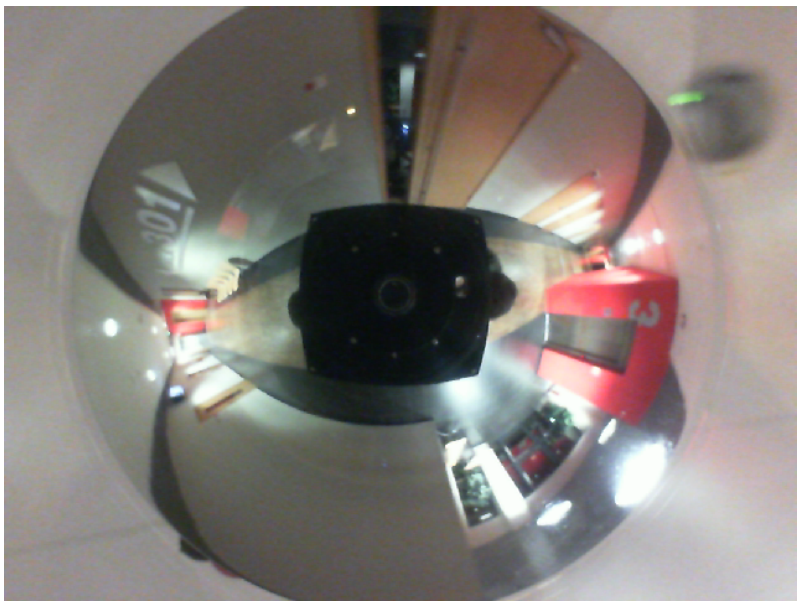


Figure 5.3: Third Floor

Chapter 6

Implementation

6.1 Neural network model

Siamese neural network is a perfect choice for the task of matching two images. It was used in this thesis with quite good outcome. Exact results are described in chapter 7. Treating the problem as finding similarity between images in opposition to classification of images has proved effective. This approach would be even more favoured if the system would be mounted on moving robot, and dataset would keep increasing. Standard classification would need more and more time for categorization with each additional class.

After long experimentation and testing different variants, this one was chosen (Fig 6.1). Three blocks composed of convolutional layer activated with RELU plus max pooling, are followed by a fully-connected dense layer with sigmoid activation. Next embeddings of both images are compared with L1 or L2 distance function and the result is fed to dense layer responsible for prediction if it is the same place or not.

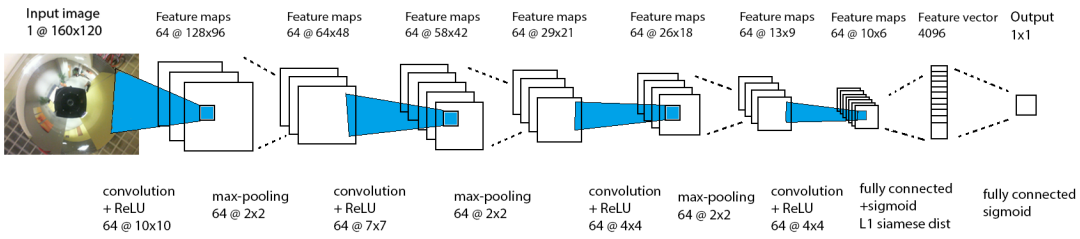


Figure 6.1: Architecture of CNN. Second branch of siamese neural network is omitted.

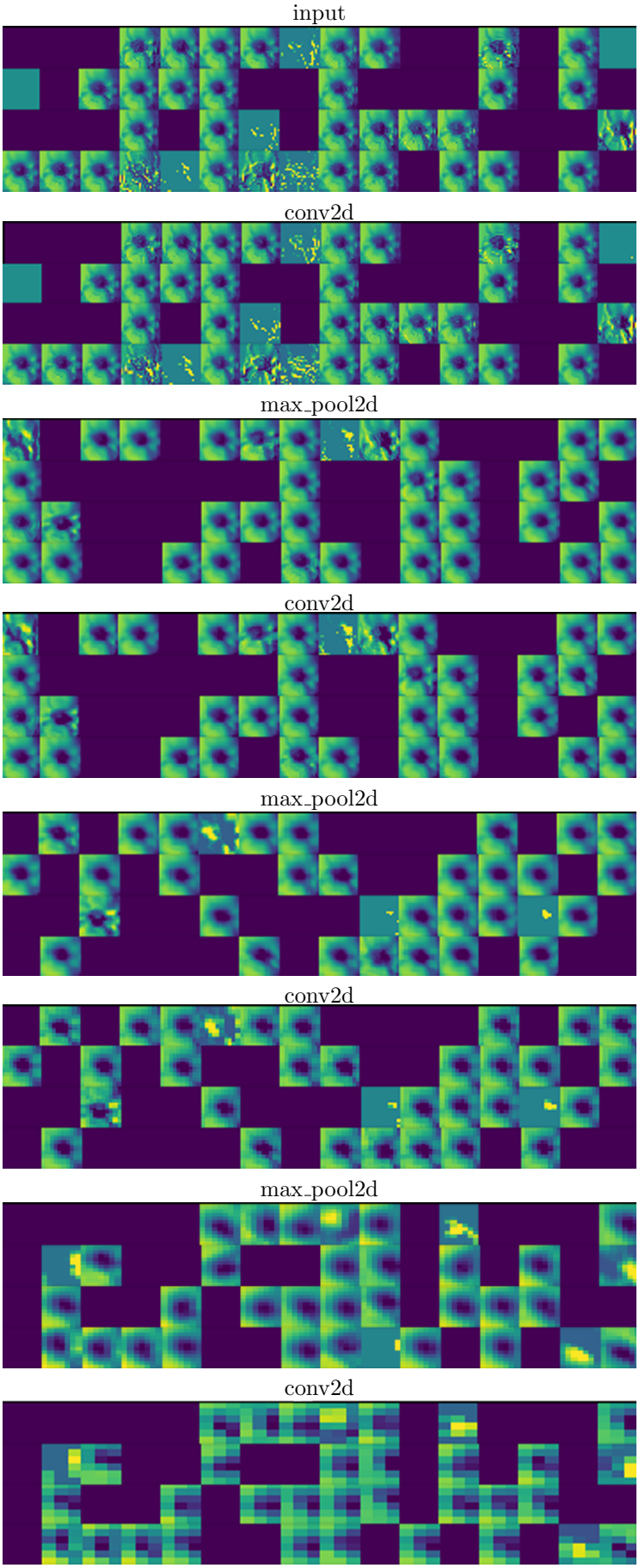


Figure 6.2: Visualization of every channel in every intermediate activation.

The first layer is arguably retaining the full view in training photo, although there are several filters that are not activated and are left blank. At that stage, the activations retain almost all of the information present in the initial picture.

As training goes deeper in the layers, the activations become increasingly abstract and less visually interpretable. They begin to encode higher-level concepts such as single borders, corners and angles. Higher presentations carry increasingly less information about the visual contents of the image, and more information related to the class of the image.

On the last layer, it cannot be interpreted what is in the picture and more filters are not activated. It means that there is less informations for humans to learn at this point.

6.2 Training

In order not to corrupt results, each and every configuration used later for comparisons was trained for the same number of iterations. That includes different hyperparameters, optimizers options, distance functions and architecture complexity. Results for each variation are analyzed and described in detail in subsequent chapter. Due to limited time and computing resources 20000 iterations per setting was chosen. That made it possible to prefigure trends and pick best configuration. After that the final structure was trained for 30000 iterations to learn its highest capabilities achievable on available hardware.

Changeable hyperparameters that are available:

- Learning rate
- Learning rate multipliers and regularization penalization for each layer (Only for modified SGD optimizer)
- Batch size
- Momentum and momentum slope
- Iterations
- Filters number and size

Additionally, after every 500 iterations learning rate is decreased by 1%. With the aim of preventing waste of time, a limit is implemented that stops training after 10000 iterations without improvement of accuracy. Following each 1000 repetitions, weights of the model are written to .h5 and .json files, but only if the accuracy has progressed comparing to those already saved. Described funcionality is presented in Listing 6.1.

```
        for iteration in range(number_of_iterations):
            # train set
            images, labels = self.omniglot_loader.get_train_batch()
            train_loss, train_accuracy = self.model.train_on_batch(
                images, labels)
            if (iteration + 1) % 500 == 0:
                K.set_value(self.model.optimizer.lr, K.get_value(
                    self.model.optimizer.lr) * 0.99)
            if K.get_value(self.model.optimizer.momentum) < final_momentum:
                K.set_value(self.model.optimizer.momentum, K.get_value(
                    self.model.optimizer.momentum) + momentum_slope)
            train_losses[count] = train_loss
            train_accuracies[count] = train_accuracy

            # validation set
            count += 1
            print('Iteration %d/%d: Train loss: %f, Train Accuracy: %f, lr = %f' %
                  (iteration + 1, number_of_iterations, train_loss, train_accuracy,
                  K.get_value(self.model.optimizer.lr)))

            if (iteration + 1) % evaluate_each == 0:
                number_of_runs_per_alphabet = 40
                validation_accuracy = self.omniglot_loader.one_shot_test(
                    self.model, support_set_size, number_of_runs_per_alphabet,
                     is_validation=True)
```

```
26
27                  self.__write_logs_to_tensorboard(
28                      iteration, train_losses, train_accuracies,
29                      validation_accuracy, evaluate_each)
30                  count = 0
31
32                  if (validation_accuracy == 1.0 and train_accuracy == 0.5):
33                      print('Early Stopping: Gradient Explosion')
34                      print('Validation Accuracy = ' +
35                            str(best_validation_accuracy))
36                      return 0
37                  elif train_accuracy == 0.0:
38                      return 0
39                  else:
40                      # Save the model
41                      if validation_accuracy > best_validation_accuracy:
42                          best_validation_accuracy = validation_accuracy
43                          best_accuracy_iteration = iteration
44                          model_json = self.model.to_json()
45                          if not os.path.exists('./models'):
46                              os.makedirs('./models')
47                          with open('models/' + model_name + '.json', "w") as json_file:
48                              json_file.write(model_json)
49                          self.model.save_weights('models/' + model_name + '.h5')
50
51              if iteration - best_accuracy_iteration > 10000:
52                  print('Early Stopping: validation accuracy
53                      did not increase for 10000 iterations')
54                  print('Best Validation Accuracy = ' +
55                        str(best_validation_accuracy))
56                  print('Validation Accuracy = ' + str(best_validation_accuracy))
57                  break
```

Listing 6.1: Main training loop

6.3 Encountered problems

During the course of writing this thesis a number of problems occurred. Some of them were related to restricted computational resources. No dedicated tensor processing unit (TPU) or high end GPU was available. Other obstacles resulted from lack of time or specific nature and limited size dataset.

6.3.1 Resources shortage

Limited computing capabilities lead to significant increase of training time (Fig 6.1). This in turn forced on authors reduced iterations for each tested configuration. For example when checking influence of batch size, 5000 iterations had to suffice to determine trends and chose batch size used in later experiments. There is a possibility that batch size that seemed to be best would be outclassed by other configuration at higher iterations. Those risks had to be taken to meet the deadline of bachelor's thesis.

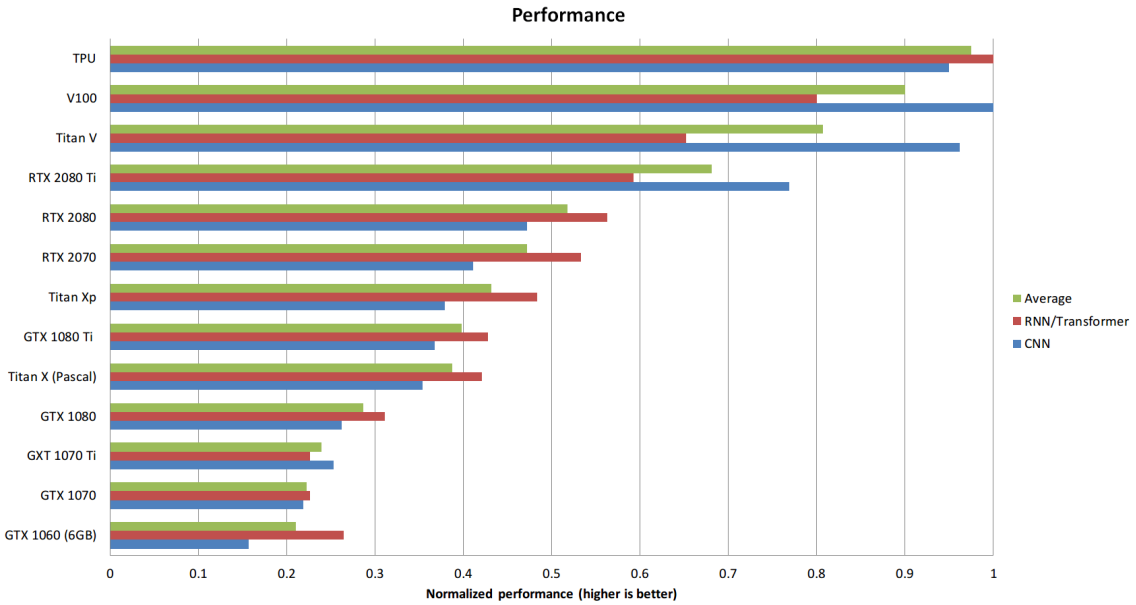


Figure 6.3: Comparison of performance of GPUs and TPU. Experiments described in this thesis were performed on GTX 1060 (4GB). This version is not even listed in the chart of best suited hardware for machine learning. Source: [10]

6.3.2 Overfitting

Due to specific requirements, no ready dataset was found. It had to be prepared especially for this considerations. That in turn consumed time. Dataset is not big in comparison to standard dataset sizes used for trainging in machine learning. To deal with this problem image augumentor was used. This solution increased number of training images available. It randomly performed operations such as:

- Shift
- Rotation - in range -15°to 15°
- Shear - in range -20°to 20°
- Zoom

To further lower risk of overfitting, other countermeasures were taken. Implementation of three separate sets: training set, validation set and evaluation set was already discussed in chapter 2. In addition, architecture of neural network was altered. Number and size of convolutional layers has been reduced. Another benefit of this maneuver was increasing the speed of training.

Chapter 7

Experimental Results

7.1 Comparison of configurations

7.1.1 Architecture complexity

The accuracy was tested for various layer numbers and sizes. To ensure correctness of trials, no other hyperparameter was changed. The purpose of this operation was determining trends, not calculating best achievable accuracy, so only 5000 iterations were performed to save time. Modified stochastic gradient decent (SGD) optimizer was used in all attempts. Outcome is presented in Fig 7.1. Exact values are concluded in Table 7.1. As for description, 64x64x64x4096 indicate three convolutional layers (each followed by max pooling) with 64 filters, after which comes fully connected layer of size 4096.

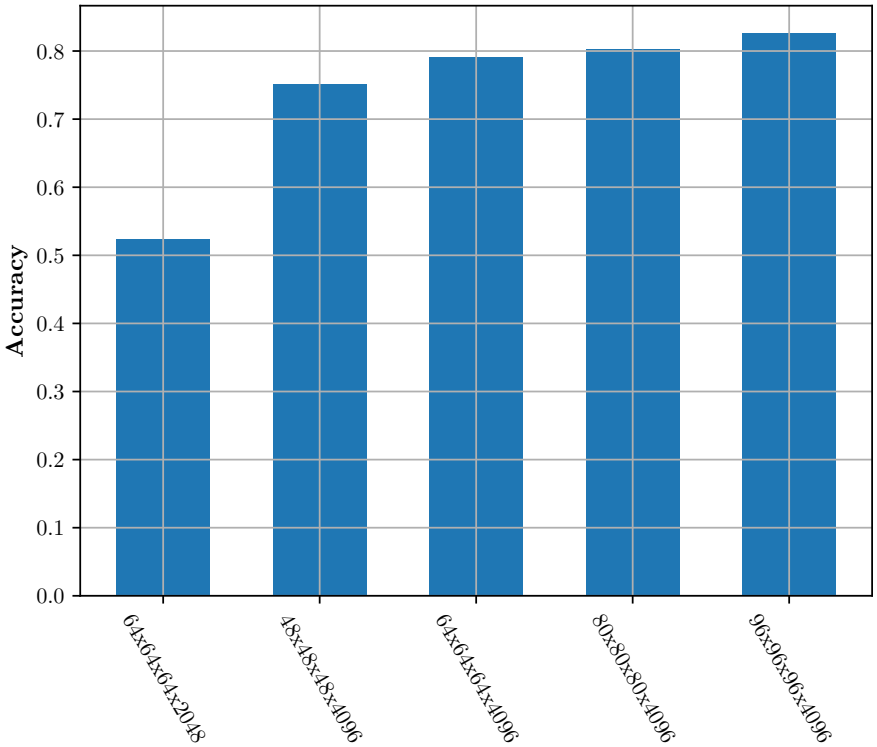


Figure 7.1: Relation between model size and accuracy.

	Accuracy
64x64x64x2048	0.5242
48x48x48x4096	0.7511
64x64x64x4096	0.7902
80x80x80x4096	0.8026
96x96x96x4096	0.8253

Table 7.1: Accuracy for various model structures.

From visualization Fig 7.1 it can be observed that although accuracy kept increasing along with number of filters, rate of this growth diminished. It is possible that for higher number of filters that rise would be insignificant. On the other hand increase of training time was substantial, therefore due to limited resources 64x64x64x4096 was chosen as optimal.

7.1.2 Batch size

Next performance tests were conducted with focus on influence of batch size. Similarly to trials described in previous section, all other hyperparamters remained unchanged and 5000 iterations per setting was performed. Results are presented in Table 7.2 and visuailzed in Fig 7.2.

	BS = 1	BS = 2	BS = 4	BS = 8	BS = 10	BS = 16	BS = 32
Modified SGD	0.5469	0.5226	0.5473	0.7438	0.7511	0.8663	0.3402
Adam	0.3214	0.3367	0.4682	0.4977	0.6374	0.6592	0.2875

Table 7.2: Relation between batch size and accuracy for different optimizers.

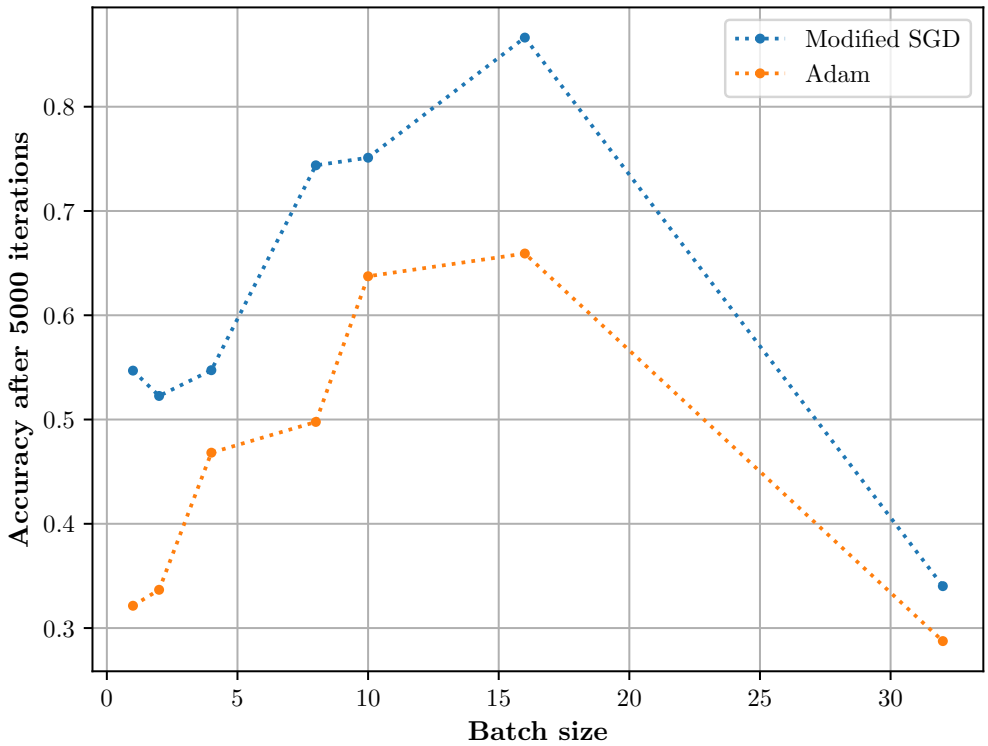
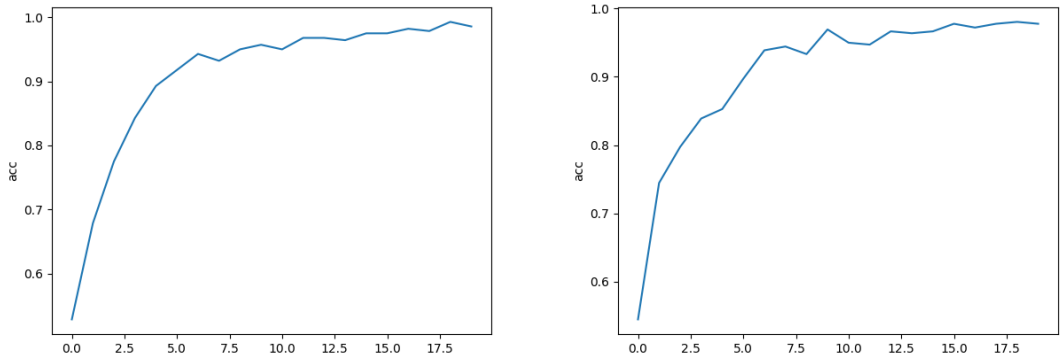


Figure 7.2: Relation between batch size and accuracy for different optimizers.

More time was devoted to fine tuning hyperparameters for modified SGD optimizer. This is reflected on its better results when compared to Adam. Nevertheless trends are similar for both Adam and SGD optimizers. The analysis shows that system with to low batch size does not handle the task well. Higher sizes are advised but only up to a point. Too big batches cause deterioration of accuracy. Furthermore, training time substantially increases. For that reason batch size equal to 16 was selected for further evaluations.

7.1.3 Separate floor for evaluation vs mixed evaluation

The neural network was tested on two types of datasets. The first one with two floors dedicated for training and validation and third, completely different one for evaluation and the second with mixed floors for all subsets (training,validation and evaluation). At the beginning dataset with separate floor for evaluation was tested, but results were not satisfying because of too low accuracy. The problem was triggered by too big differences in visual appearances of each floor. To solve this, sections were mixed and evaluation set was created from images from three different floors. Results improved drastically.



(a) Separate floor as a evaluation set.

(b) Evaluation set composed of all floors.

Figure 7.3: Comparison of results for both variants.

In Fig 7.3 the described problem is not clearly visible. That is because plots present not evaluation, but validation accuracy. Final value reached is 0.9875 for (a) and 0.9706 for (b). However when final evaluation accuracy is compared, findings are totally different: 0.4144 for (a) and 0.8932 for (b). Of course this shows that variant with mixed datasets (b) works much better. Described problem could be potentially solved with increase of variate of training set, for example with adding photos of even more different floors.

7.1.4 L1 distance vs L2 distance

When testing distance losses L1 and L2, it is interesting to consider differences between those two metrics. The L2 distance prefers many medium disagreements to one big one. Particulary, the L2 distance is much more unforgiving than L1 distance when it comes to differences between two vectors. Because of that tests went unsuccessfully. Evaluation of L1 went well, but L2 was more problematic. There was a problem with achieving satisfactory outcome. This could be caused by too much overfitting in a model when combined with L2. Another possible explanation is that test were too short. It may very well be, that after higher amount of iterations L2 would start to show some promise.

7.2 Best result

After all previous tests an optimal configuration was derived. Using this settings a training session of 30000 iterations was preformed. The final accuracy on evaluation set reached **0.9125**. Such value is a very good result taking into consideration that it can be improved by solving problems described previously in this thesis. Particularly more computational power could have considerable impact. The accuracy is high enough for the system to be potentially usefully in real world application. Training curve of this specific model is presented in Fig 7.4.

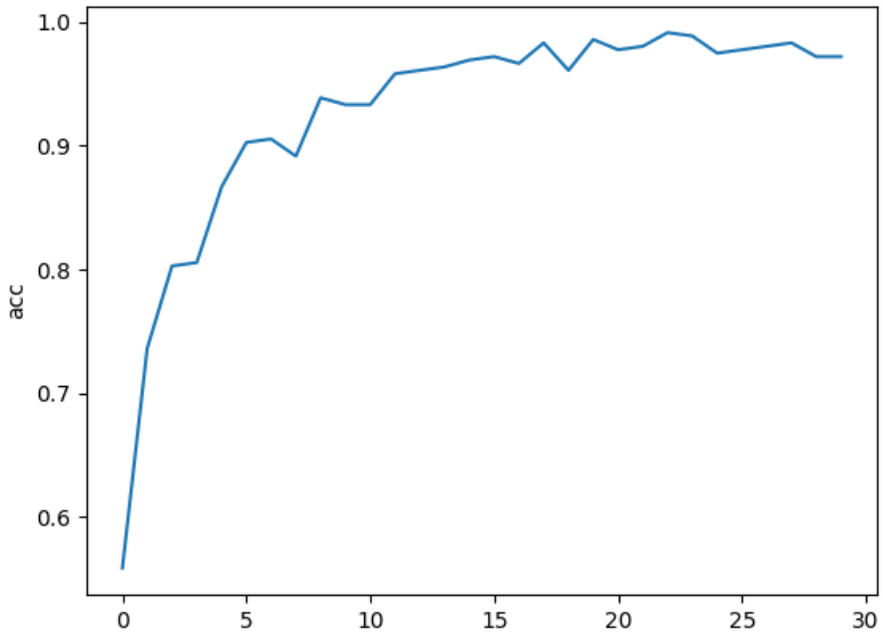


Figure 7.4: Training curve of best configuration found (Validation accuracy).

Chapter 8

Summary and future work

Appearance based localization employing deep neural networks can be used in many fields of robotics science. This issue is very complex and quite crucial. One of the methods of obtaining pictures to localize is to use catadioptric camera. In above bachelor thesis, implementation of this method was realized and research examining effectiveness was carried out.

First of analysed factor which influence work of implemented algorithm was architecture complexity. Comparing results of different amount of filters and convolutional layers showed advantage of 96x96x96x4096 architecture but vital factor was not only obtained accuracy but also time of training. Taking everything into consideration, chosen was 64x64x64x4096 architecture due to significant difference in time and little difference in accuracy.

Performed were also tests concerning batch sizes with different types of optimizers. With use of chosen architecture in previous test, changed were sizes of batches for Adam and Modified SGD optimizer. Best value of training was obtained with batch of size 16.

The assumption of next experiment was to compare the influence of the appropriate creation of the datasets on evaluation of training. Two datasets were build, one with mixed sections of floors and second with whole different sections from separate floor. It turned out to be more beneficial to use mixed set. Despite, that the results did not live up to expectations, it can be improved by increasing variety of images included in training set as well as making training set and evaluation set more similar.

Lastly, tested were distance losses. Comparing L1 and L2 types went unsuccessfully. Due to encountered problems achieving results with L2 loss was not possible. Although, it may be possible to overcome this issue, after appropriate tunning of hyper parameters and number of iterations, promising results can be obtained.

After performing all experiments, last test was executed. With best obtained parameters, accuracy of final training had very optimistic value of the order of **0.9125**. Result on that level can be used in real world application and can be slightly improved by solving encountered problems.

There are further ways to improve the results. Many extensions to dataset can be applied. Adding cases with people on pictures, different light conditions, artificially erased spots or pasted people can be beneficial in obtaining higher value of accuracy. Some of improvements were already mentioned in chapter 7. Despite the fact that current results are promising, there is huge space for improvement. Collaterally, by the growth of computational power and future development of neural networks, the accuracy and performance will increase.

Bibliography

[1] J. Johnson A. Karpathy and F.-F. Li. Course notes on cs231n: Convolutional neural networks for visual recognition url: https://cs231n.github.io, 2016.

[2] I. Sutskever A. Krizhevsky and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NIPS), 1106–1114.*, 2012.

[3] Ejaz Ahmed. *An Improved Deep Learning Architecture for Person Re-Identification.* University of Maryland 3364 A.V. Williams, College Park, MD 20740.

[4] Jones M. Ahmed, E. and T.K. Marks. An improved deep learning architecture for person re-identification. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3908-3916)*, 2015.

[5] Ignacio Garcia-Dorado Aysegul Dundar. Context augmentation for convolutional neural networks. *https://arxiv.org/pdf/1712.01653.pdf*, 2017.

[6] Francois Chollet. *Deep learning with python.* Manning Publications Co., 2017.

[7] Roberto Arroyo Contera. *Topological Place Recognition for Life-Long Visual Localization, PhD. Thesis.* Universidad de Alcalá, 2017.

[8] M. Cummins and P. Newman. Fab-map: Prob-abilistic localization and mapping in the space of appearance. *International Journal of Robotics Research (IJRR), 27(6):647–665.*, 2008b.

[9] Z. Lei D. Yi and S. Z. Li. *Deep metric learning for practical person re-identification.* ICPR, 2014.

[10] Tim Dettmers. Which gpu(s) to get for deep learning: My experience and advice for using gpus in deep learning. *http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/*, 2018-11-05.

[11] Keras Developers. https://keras.io/.

[12] OpenCV Developers. https://matplotlib.org/.

[13] The Matplotlib development team. https://matplotlib.org/.

[14] Jimmy Lei Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *Published as a conference paper at ICLR 2015*, 2015.

[15] Tiago Freitas. *https://github.com/Goldesel23/Siamese-Networks-for-One-Shot-Learning.*

[16] Tiobe Index. https://www.tiobe.com/tiobe-index/.

[17] Richard Zemel Koch, Gregory and Ruslan Salakhutdinov. *Siamese Neural Networks for One-shot Image Recognition.* University of Toronto, 2015.

[18] Paweł Kozłowski and Wojciech Drankiewicz. *System wizyjny o hybrydowym polu widzenia dla robota mobilnego, Bachelor Thesis.* Poznań University of Technology, 2016.

[19] P. Neubert N. Sünderhauf and P. Protzel. Are we there yet? challenging seqslam on a 3000 km journey across all four. *Workshop on Long-Term Autonomy at the IEEE International Conference on Robotics and Automation (W-ICRA).*, 2013a.

[20] Miriam Schönbein. *Omnidirectional Stereo Vision for Autonomous Vehicles.* Dissertation, Institut für Mess- und Regelungstechnik, Karlsruhe Institut für Technologie (KIT), Karlsruhe„ 2014.

[21] tim.peters@gmail.com (Tim Peters). The zen of python, url: https://github.com/python/peps/blob/master/pep-0020.txt.

[22] T. Xiao W. Li, R. Zhao and X. Wang. *DeepReID: Deep Filter Pairing Neural Network for Person Re-Identification.* CVPR, 2014.

[23] Pranjal Yadav. A deeper understanding of nnets (part 1) — cnns. *https://towardsdatascience.com/a-deeper-understanding-of-nnets-part-1-cnns-263a6e3ac61*, 2018.

Poznan University of Technology
Faculty of Electrical Engineering
Institute of Control and Information Engineering

Typeset using LaTeX in Computer Modern.

BibTeX:

```
@mastersthesis{ key,
    author = "Tomasz Filanowicz \and Filip Szymański \and ",
    title = "{Appearance based localization employing
deep learning }",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2019",
}
```