BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

VISUAL CAMERA ORIENTATION ESTIMATION USING MACHINE LEARNING

ODHAD ORIENTACE KAMERY Z OBRAZU POMOCÍ METOD STROJOVÉHO UČENÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MARTIN KUBIČKA**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. MARTIN ČADÍK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

154459

Institut: Department of Computer Graphics and Multimedia (DCGM)
Student: **Kubička Martin**
Programme: Information Technology
Title: **Visual Camera Orientation Estimation using Machine Learning**
Category: Computer vision
Academic year: 2023/24

Assignment:

1. Familiarize yourself with the methods for estimating camera orientation from an image.
2. Select methods suitable for implementation and describe their properties, focusing especially on spherical convolutional network (SCNN) methods.
3. Design and implement a system for camera orientation estimation from an image.
4. Experiment with the system, measure the properties of the implemented method and discuss the possibilities of future development.
5. Present the achieved results in the form of a poster and a short video.

Literature:

- Baboud, L., Čadík, M., Eisemann, E., Seidel, H.P.: Automatic Photo-to-Terrain Alignment for the Annotation of Mountain Pictures. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011) (CVPR oral), 2011.
- Brejcha, J., Čadík, M.: Camera Orientation Estimation in Natural Scenes Using Semantic Cues. International Conference on 3D Vision (3DV) (oral presentation), Verona, Italy, 2018.
- Brejcha, J., Lukáč, M., Hold-Geoffroy, Y., Wang, O., Čadík, M.: LandscapeAR: Large Scale Outdoor Augmented Reality by Matching Photographs with Terrain Models Using Learned Descriptors. 16th European Conference on Computer Vision (ECCV), online, 2020. U.S. Patent No. 2022/0114365 A1.
- Taco S. Cohen, Mario Geiger, Jonas Koehler, Max Welling: Spherical CNNs. arXiv:1801.10130, 2018.
- Esteves, C., Allen-Blanchette, C., Makadia, A. et al. Learning SO(3) Equivariant Representations with Spherical CNNs. Int J Comput Vis 128, 588–600 (2020). https://doi.org/10.1007/s11263-019-01220-1.
- Yu W, Zha D, Mu N and Fu T. Hybrid CNNs. Proceedings of the 2019 3rd International Conference on Digital Signal Processing. (38-42). https://doi.org/10.1145/3316551.3316573.
- Zhan Gao, Fernando Gama, Alejandro Ribeiro: Spherical Convolutional Neural Networks: Stability to Perturbations in SO(3). arXiv:2010.05865. 2020.

Requirements for the semestral defence:
Points 1-3 of the assignment.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Čadík Martin, doc. Ing., Ph.D.**
Head of Department: Černocký Jan, prof. Dr. Ing.
Beginning of work: 1.11.2023
Submission deadline: 9.5.2024
Approval date: 9.11.2023

Abstract

The purpose of this work is to create a model using spherical convolutional neural networks that can estimate the orientation of a camera from two inputs, where the first input is a panorama and the second input is a photograph capturing a specific part of the panorama. In other words, the task is to find where in the panorama, which is the first input, is located the photo, which is the second input. In addition to three created models that address this problem, six new datasets have also been created, which expand the currently available number of datasets whose photos are in equirectangular or stereographic format.

Abstrakt

Účel tejto práce je vytvoriť model pomocou sférických konvolučných neuronových sietí, ktorý vie odhadnúť orientáciu kamery z dvoch vstupov, kde prvým vstupom je panoráma a druhým vstupom je fotka, ktorá zachytáva určitú časť panorámy. Inými slovami, úlohou je nájsť kde v panoráme, ktorá tvorí prvý vstup, sa nachádza fotka, ktorá tvorí druhý vstup. Mimo troch vytvorených modelov, ktoré riešia daný problém, vzniklo aj 6 nových datasetov, ktoré rozširujú momentálne dostupný počet datasetov, ktorých fotky sú v equirectangulárnom alebo stereografickom formáte.

Keywords

spherical convolutional neural networks, camera orientation estimation, camera pose estimation, semantic segmentation, probability, equirectangular projection, stereographic projection, panorama

Kľúčové slová

sférické konvolučné neurónové siete, odhad orientácie kamery, odhad pózy kamery, sémantická segmentácia, pravdepodobnosť, equirectangulárna projekcia, stereografická projekcia, panoráma

Reference

KUBIČKA, Martin. *Visual Camera Orientation Estimation using Machine Learning.* Brno, 2024. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Martin Čadík, Ph.D.

Rozšírený abstrakt

Problém, ktorý rieši táto práca, sa nazýva odhad orientácie kamery. Predstavme si, že máme dva vstupy: prvým je 360-stupňová panoráma, ktorá zachytáva celý priestor okolo nás, a druhým je fotografia, ktorá zachytáva časť tejto panorámy. Táto fotografia môže byť odfotená na tom istom mieste ako panoráma, ale v inom čase, a preto sa môže líšiť vo viacerých smeroch.

Orientácia kamery je základným prvkom počítačového videnia. Presné odhadnutie orientácie kamery je kľúčové pre mnohé aplikácie. Bežným príkladom využitia je rozšírená realita, kde môžeme do reálneho priestoru vkladať rôzne objekty, čím vieme skutočne rozšíriť realitu. Rozšírená realita má veľa aplikácií vo vzdelávaní, vojenskom výcviku a zábave. Ďalšími príkladmi môže byť autonómne riadenie, ktoré je čím ďalej, tým populárnejšie. Využitie nájdeme aj v robotike.

Túto úlohu je možné riešiť viacerými spôsobmi. Jednu skupinu spôsobov nazývame tradičné metódy a druhú skupinu metódy strojového učenia. Ako tradičnú metódu si môžeme predstaviť napríklad zarovnávanie vstupov pomocou hrán. Druhým typom sú metódy, ktoré využívajú neurónové siete. V tejto práci nás budú zaujímať metódy strojového učenia, konkrétne sférické konvolučné neurónové siete.

Sférické konvolúcie sú pre nás zaujímavé preto, lebo majú dve základné vlastnosti, ktoré sú kľúčové pri vstupe, ktorým je sférický objekt. Prvá vlastnosť je tá, že keď prevedieme sféru na 2D plochu, vzniknú deformácie, s ktorými si bežné konvolúcie nemusia vedieť poradiť. Druhou vlastnosťou je equivariancia voči rotáciám, čo bežné konvolúcie nemajú.

V tejto práci vznikli tri modely, kde každý má za sebou inú myšlienku, ako riešiť daný problém. Ďalším prínosom tejto práce je vytvorenie šiestich datasetov, ktoré obsahujú fotografie v equirectangulárnom a stereografickom formáte, pričom aj v súčasnosti je týchto datasetov nedostatok.

Visual Camera Orientation Estimation using Machine Learning

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. Ing. Martin Čadík Ph.D.. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Martin Kubička
May 9, 2024

Acknowledgements

I would like to thank my supervisor doc. Ing. Martin Čadík Ph.D. and my family for guiding me through the process of creating this thesis, giving me advice along the way and passing on their knowledge. I used ChatGPT 4 to check my grammar and improve some sentence formulations.

Contents

Chapter 1

Introduction

Knowledge of camera orientation is a fundamental element of computer vision. Precise estimation of the camera orientation is crucial for many applications. A common example is augmented reality, where we can insert various objects into real space, thus truly expanding reality. Augmented reality has many educational, military training, and entertainment applications [29]. Autonomous driving is a frequent topic, where it is absolutely crucial to obtain not only the orientation of the cameras, but also the overall pose. There are various sensors and cameras with the help of which we solve this problem, and with which the system subsequently decides on the next action. Also, a good example is robot applications, where, on the one hand, the position and orientation of objects are estimated, which is, however, a different task, but also the position and orientation of the robot itself, which is crucial for its movement in space.

In this work, I will focus on estimating camera orientation using machine learning methods, specifically using spherical convolutional neural networks. This task involves creating a dataset and acquiring other datasets for training, developing a program for transforming images into various spherical projections, more specifically into equirectangular projection and stereographic projection. Then it also involves creating a model for estimating the orientation of the camera. There are several ways to solve the problem using this type of convolutional neural networks, but in this work, we will look at the method where we use regression and the method for probability prediction. Finally, we will compare the created models and experiment with the model that has the best results.

Chapter 2

Camera orientation estimation

To fully understand the issue behind camera orientation estimation, it is important to explain terms like omnidirectional camera, camera orientation and finally camera orientation estimation. All these terms will be explained in subsections below.

2.1 Omnidirectional camera

These days, there are many different types of cameras that could each be the subject of an entire thesis, but I have selected those that are closely related to this work - 360-degree or sometimes called omnidirectional cameras.

The characteristic of these cameras is that they cover almost the entire field of view of the surrounding sphere, or they cover 360 degrees of the horizon [3]. For this thesis these cameras or methods how to create 360 degree panoramas are must have for creating panoramas which are used in spherical convolution neural networks.

Equally important for us are the cameras used in our mobile phones or typical cameras as we know them. Images from these cameras are also crucial, because the task is to estimate what interests us in these photos, namely, the camera orientation.

As we know we also can create 360-degree panoramas without having omnidirectional cameras, because for example our phones allows us to make panoramas. Important question is in which projection are these panoramas stored and which projection we need for our task.

2.2 Camera orientation

What interests us in this work is the actual result of the camera. Specifically, photos and 360-degree panoramas, from which we can estimate the orientation of the camera. But first, let's explain what we mean when we say camera orientation.

The orientation of the camera in this work refers where in 360 degree panorama is located another photo (this is similar to image registration task but it is not the same). The actual orientation of the camera is described by the following three attributes:

- Pitch - It is the rotation of the camera around its horizontal axis (x). It determines whether the camera is tilted upward or downward.

- Yaw - It is the rotation of the camera around its vertical axis (y). It is similar to horizontal orientation and determines whether the camera is turned left or right.
- Roll - It is the rotation of the camera around its own axis (z).

Visualizations of angles you can see in figures 2.1 and 2.2.

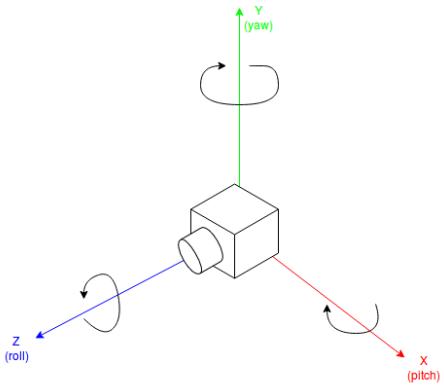

Figure 2.1: Pitch (red), yaw (green) and roll (blue) visualization on camera model. Image taken from [14].

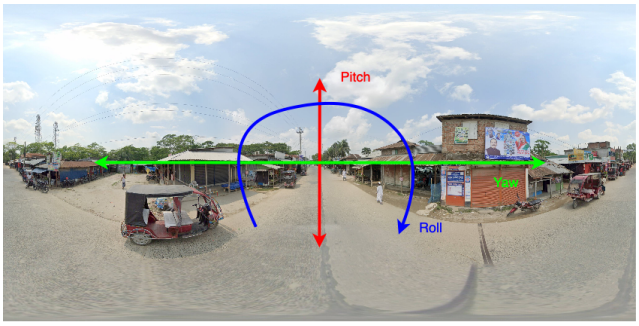

Figure 2.2: Pitch (red), yaw (green) and roll (blue) visualization on panorama.

Before moving on to the next chapter, it's important to clarify the terms, so camera orientation is often possible to confuse with the term camera pose, where both terms mean something different. When we talk about camera pose, we are talking about the position of the camera as well as its orientation, so camera orientation is a subset of camera pose [29].

2.3 Camera orientation estimation

Estimating the orientation of the camera is a complex problem with many challenges to consider. In my implementation, I have two inputs for estimating the camera's orientation. The first input is the actual 360-degree panorama, which is projected in a certain format onto a 2D canvas (more in section 2.3.2) and the second input is the actual photo, which is also projected into the same format as the panorama. One of the many problems, for example, is if the panorama is created in the summer months and the photo is taken on a day when the landscape was snow-covered. We will discuss more about such problems below.

2.3.1 Estimation challenges

There are many methods for estimating camera orientation. Each method has its advantages and disadvantages. Some methods better address some of the following problems, but on the other hand, may not be accurate for another type of problem (more about methods in section 2.3.3). Let's look at some of the problems we most commonly encounter in camera orientation estimation:

- Seasons and time - The time of year can also influence the estimation of the camera orientation. For example, in winter when landscape can be covered with snow, the

same scene may look completely different than in summer. Similarly, a problem may arise if the image is taken in the dark, which can cause noise.

- Location - The geographical location of the image can affect the visibility of various elements in the scene. For example, in mountainous areas, visibility may be limited due to the terrain, and mountainous areas are very variable. The presence of lakes, green areas, and other characteristics of the location can influence the estimation of the camera orientation.

- Image quality - Sometimes, we can see that images are noisy, which can be caused, for example, by poor lighting during shooting. Another problem that happens during photography is that we take a photo in a dynamic environment, meaning either we or the object are in motion, so the photo or object may be blurred. Both problems can distort the image, which can be a problem when estimating the camera orientation.

It means that, there are many problems in camera orientation estimation and we need to deal with it.

2.3.2 Panorama formats

In my implementation for estimating the camera's orientation, I use a 360-degree panorama in addition to a regular image. Its representation needs to be converted from a spherical form to a 2D surface. This conversion causes many problems, and therefore, there are multiple formats for representing a spherical object on a 2D surface, where each projection solves a different problem associated with projection, and each has its advantages and disadvantages, which we will discuss further.

To introduce the issue, I will explain the problem with geographical maps. Converting the spherical model of the Earth to a map means that, for example, some parts must be reduced compared to others, or stretched. For example Greenland often appears to be of similar size or even larger than Africa. However, this is a significant distortion of their actual sizes. In reality, Africa is much larger than Greenland.

Now let's take a closer look at some types of projections based on the type of surface onto which the sphere is projected, and discuss their individual advantages and disadvantages:

Cylindrical projection

Cylindrical projection can be simply imagined and visualized in such a way that we take a piece of paper, which we wrap around a sphere, where the paper and the sphere touch only at the equator [1]. This can be seen in Figure 2.3. If light is emitted from the center of the sphere, it projects the spherical surface onto the paper's surface. An important fact about this projection is that, similar to equirectangular projection, this type of projection has a problem representing the poles and the area around the poles, which are distorted and stretched to infinity. A characteristic is also that vertical lines remain preserved, but horizontal lines, except at the equator, tend to curve.

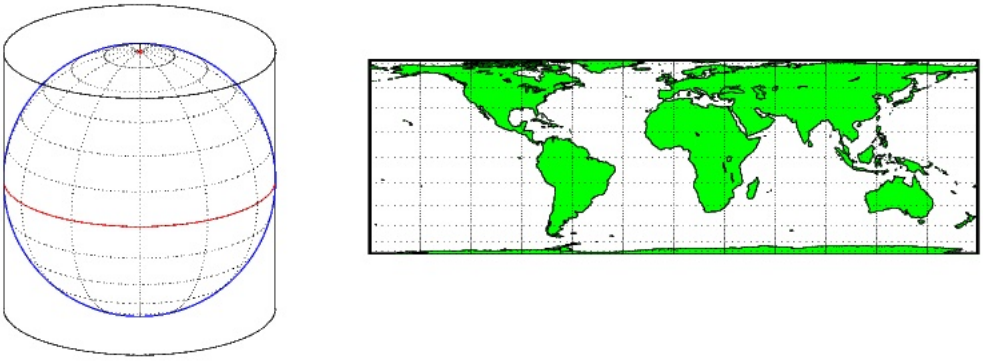

Figure 2.3: Cylindrical projection. Image taken from [4].

Equirectangular projection, also known as Equidistant cylindrical projection [2], is therefore a type of cylindrical projection. The difference is the aspect ratio of width to height, where the equirectangular projection has a ratio of 2:1. This difference can be seen if we compare Figures 2.3 and 2.4. This projection is important for us because it is one of the most common projections used as input for spherical convolutions. Examples can be found in section 5.1.

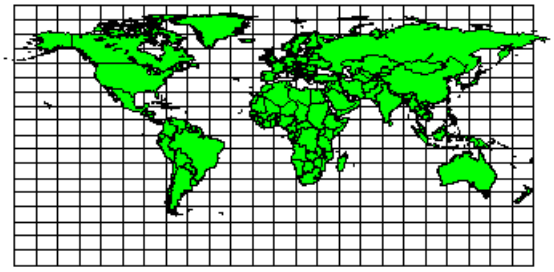

Figure 2.4: Equirectangular projection. Image taken from [2].

Conic projection

As the name implies, a conic projection is created by projecting a sphere onto a cone that is positioned over the sphere [4]. As we can see in Figure 2.5, the idea is that the apex of the cone is aligned with the polar axis of the sphere. A cone that contacts the Earth at only one latitude line is called a tangent cone. Conversely, a smaller cone that cuts through the Earth at two separate points is known as a secant cone. Conic projections are known for causing less distortion in mid and high latitude regions compared to cylindrical projections.

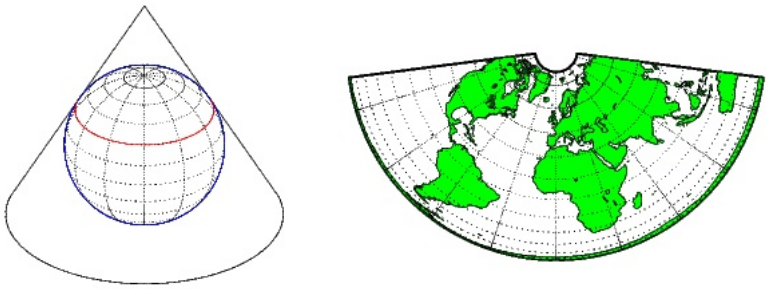

Figure 2.5: Conic projection. Image taken from [4].

Azimuthal projection

The idea behind the azimuthal projection is to map the sphere onto a flat surface, which differs from the previous two projections where the sphere is mapped onto a cylinder or a cone [4]. As we can see in Figure 2.6, it works by aligning the plane with one of the sphere's poles. The meridians appear as straight lines emanating from the pole, while the parallels are depicted as complete circles centered around the pole.

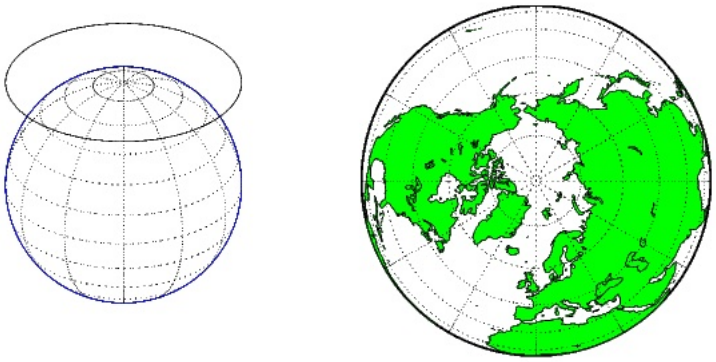

Figure 2.6: Azimuthal projection. Image taken from [4].

Stereographic projection is a type of azimuthal projection [16]. This projection is also known as little planet projection. This projections is also important for us same as equirectangular projection because it is commonly used as input for spherical convolutions. Examples can be found in section 5.1.

2.3.3 Methods

Over time, many different methods have been invented to estimate camera orientation, and therefore, in the following section, I will introduce a classification with a description of important estimation methods.

Based on different approaches, we can divide camera orientation estimation methods into these categories:

- Traditional methods - The first category I would call traditional or old-school methods, not to suggest anything negative about these methods, but recently there has been

a huge boom in deep learning. This category includes all methods that do not use neural networks to estimate the orientation of the camera.

- Deep learning methods - In these methods, we use the mentioned neural networks, where so far it has mainly been about convolutional neural networks and spherical convolutional neural networks.

Traditional methods

Image aligning with terrain model

In the paper [6], the authors focus on mountainous regions, where the environment and weather often change throughout the year, and where other implementations frequently fail. This method assumes a known field of view and an estimated viewpoint position. It operates by acquiring silhouette edges, which are the best option for Photo-to-Terrain Alignment under these conditions. After employing an edge detection algorithm, the technique finds the best match between the silhouette edges and a synthetic model. The introduced Robust silhouette map matching metric works on the principle that silhouette maps contain T-junctions, which correspond to points where two edges meet but do not cross. This approach not only aligns the photo with the model but also, for example, allows for easy retrieval of peak names etc. Similar to this work, there are other papers like [5] and [24], which are also based on the principle of aligning photos with terrain models.

Using Semantic Cues

The method mentioned above often aligns using either single or multiple modalities. The method introduced in the paper [8] aligns an image and a terrain model based on edges and also semantic segments (more about semantic segmentation in section 3.3.1). Examples of semantic segments in this case might include lakes or forested areas. For the input, which is a photograph, it is assumed that latitude, longitude, elevation, and the horizontal field of view are known. Semantic segmentation was performed by fine-tuning models such as `FCN` and `Deeplabv2-VGG-16` and one non-CNN method. The matching was done using cross-correlation between the query and panorama on SO(3) (SO(3) is explained in 3.5).

Orientation estimation using Vanishing Point

This method focuses on understanding the camera's orientation through the concept of vanishing points. Vanishing points are where parallel lines in the real world appear to converge in an image due to perspective effects. An example of a vanishing point is railroad tracks, which, when viewed from the right angle, seem to converge. By identifying these points and analyzing the geometric relationship between them and the layout of the scene, the algorithm can estimate the camera's orientation. Thus, it relies on the geometry of the environment, such as the arrangement of walls, as mentioned in the work [19]. In work [18], the authors utilize this concept for motion detection and line classification, which then allows them to estimate camera orientation.

Hybrid Camera Pose Estimation

Most methods are measured either by a structure-based principle or a structure-less principle. Structure-based methods utilize 2D-3D correspondences, where 2D points in the

image are matched with 3D points in the model, and structure-less methods rely on 2D-2D correspondences between two images. The hybrid approach presented in the work [9] combines these methods. The selection of the most suitable solver is data-driven, meaning the algorithm evaluates the quality and type of data available. By solvers in this context, we mean either a structure-based method or a structure-less method. What makes this method unique is that it uses the RANSAC algorithm (Random sample consensus) to dynamically (in real-time) select the solver based on the data being processed.

Deep learning methods

Using convolutional neural networks

The use of convolutional neural networks for this task can also be beneficial, as with various other tasks. There are multiple convolutional neural networks, each using a slightly different approach to the task, which are used for estimating the orientation of the camera or for overall camera pose estimation.

The paper [26] provides an overview of deep learning approaches for camera pose estimation. The reason why I mention this work is that it is inspiring for finding the best way to address our problem. It reviews key deep learning methods for pose estimation, discusses their evolution, improvements, variants and compares their performance. The work initially focuses on PoseNet, which, however, performed poorly in generalizing to new, unseen scenes and had significant localization errors compared to traditional methods. Then, hybrid methods are mentioned, which combine deep learning with traditional feature based methods. These methods improve both the speed and accuracy of pose estimation compared to using deep learning or feature based methods alone. Local learning approaches focus on improving specific subtasks within the pose estimation pipeline, like feature matching. This approach seems to be more flexible and robust.

One specific example using deep learning is `UprightNet`, which is designed for estimating the 2DoF camera orientation from a single RGB image, especially within indoor scenes. The 2DoF stands for two degrees of freedom, which in terms of camera orientation estimation, means that the work estimates the pitch and yaw angles. The input is a single RGB image. From this image, it predicts surface geometry for the local camera and global upright coordinate systems. The camera orientation is then calculated as an alignment between the mentioned predictions plus with the help of computed weight maps for the input.

In the paper [21], the authors solve the problem of estimating camera pose from a different perspective, specifically estimating the relative pose of one camera to another. The input consists of two RGB images, meaning that they estimate the pose of one image relative to the other. The authors decided to use a Siamese network architecture, meaning that each input is processed in its own identical CNN branches, which have the same architecture and share weights. The outputs from these branches then go into two fully connected layers. In their work, the authors utilized a pretrained AlexNet model. They managed to achieve better accuracy compared to traditional methods, which struggled with various types of problems such as extremely large viewpoint changes, and overall, their approach found more correspondences among the inputs.

Using spherical convolutional neural networks

The use of spherical neural networks brings key features to this task, which we will discuss in the following chapters. The approach is largely the same as when using ordinary convolutions.

So far, there has been only one work that uses spherical convolutional neural networks specifically for camera pose estimation [30]. It introduces a method for estimating a camera's 6DoF pose from panoramas using these spherical convolutional neural networks. In this case, 6DoF means that three values are determined for the position, which represent translations in the x, y and z axes. Additionally, the remaining three values determine pitch, yaw, and roll. Again, in this case, the input is a single image. The work utilizes an implementation [11], introduced in 3.5.1. The entire architecture thus consists of spherical convolutions implemented in the mentioned article. Specifically, the input goes to an Encoder, from which the output goes to two decoders, where one is used for orientation estimation and the other for position estimation. The main difference between the decoders is that the orientation decoder does not use MLP layers (more about MLP's in section 3.2).

We will also use this method for estimating the orientation of the camera. However, what will be different compared to the work mentioned above is that we estimate only the orientation of the camera and not the pose, but the biggest difference is that we have two high-resolution photos as input, which makes the resulting model completely different, and thus we must also consider other factors, which are described in section 5.3.

There are more works such as [19], [23], [22], [10] where similar techniques for estimating camera orientation are presented.

Chapter 3

Spherical convolutional neural networks

In this chapter, let's introduce spherical convolutional neural networks, which offer significant advantages over ordinary convolutional neural networks. For a complete understanding, let's first recap what neural networks are and the types of neural networks used for computer vision, eventually leading to spherical convolutional neural networks themselves.

3.1 Neural networks

Neural networks consist of layers, each layer being made up of nodes [15]. These are divided into an input layer, hidden layer, and an output layer. Each node has its own weight and threshold. The weight of a node represents the strength of the connection between nodes, or we can imagine it as the importance of a given input. The threshold represents the value that, when exceeded, activates the node, causing the data to move to the next layer.

An important part of neural networks are data, which allows us to train the network to make the most accurate decisions possible at the end.

There are many types of neural networks used in computer vision, but the most well-known are convolutional neural networks. However, in this work, we are interested in spherical neural networks, which are derived from convolutional neural networks.

3.2 Multi-layer perceptrons

To understand the motivation for moving to convolutional neural networks, it's important to briefly introduce multi-layer perceptrons.

Multi-layer perceptrons are characterized by each perceptron being connected to every other perceptron in fully connected layers [28]. A disadvantage of this type of network is that the number of parameters can grow to a high number quite quickly, often resulting in inefficiency due to redundancy.

What I want to point out is that they accept a 1D vector as input. This type of network was used in computer vision, but now it has been replaced by convolutional neural networks. At first glance, it might seems weird to input 2D images as 1D vectors, but for example, training on the `MNIST` dataset showed that this type of network can achieve high accuracy.

3.3 Convolutional neural networks

Convolutional neural networks compared to multi-layer perceptrons, are more efficient in many ways and can be deeper. The main difference I want to highlight in this section is that with convolutional neural networks, we are not talking about vectors, but about 2D photos, where filters sequentially pass through the input image.

Another key feature of convolutional neural networks is their design for translational equivariance, where some convolutions can also be translationally invariant [7]. As can be seen in Figure 3.1 , if a convolution is translationally equivariant, it means that, for example, if we have a first photo with the number 4 in the center and a second photo with the same number but in the top left corner, the result after the convolution will look the same, but the numbers will be displayed in the center in the case of the first photo and in the top left corner in the case of the second photo. If they were not translationally equivariant, these outputs would not look similar at all. Sometimes we encounter translationally invariant convolutions, where for our two inputs, as can be seen in Figure 3.2, the difference in the output would be that the output after convolution from the first photo would be exactly the same as the output after convolution from the second photo.

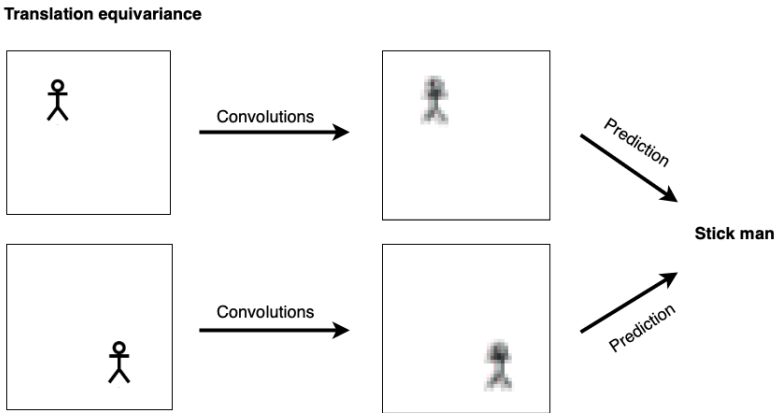

Figure 3.1: Translation equivariance visualization

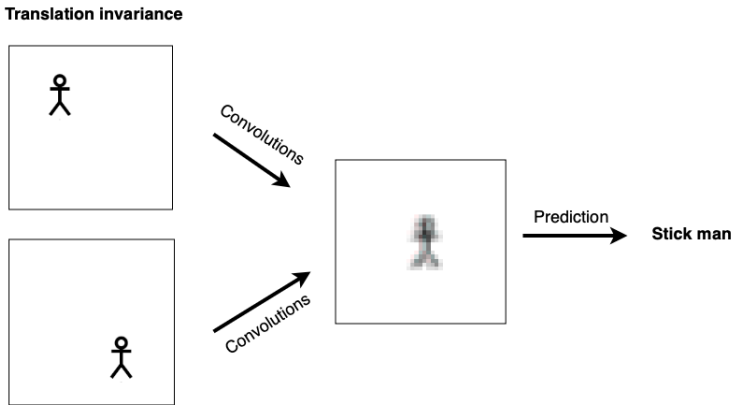

Figure 3.2: Translation invariance visualization

A common problem is the fact that sometimes we need a high-resolution photo as input, which can cause several problems, including memory issues. However, there are so-called Vision transformers, which can handle high-resolution photos. In the following part, we will briefly introduce Vision transformers.

3.3.1 Semantic segmentation

One of the methods where convolutional neural networks are used is semantic segmentation. Simply described, semantic segmentation works by predicting the class of each pixel [20]. This together creates an image that contains areas divided by colors, where each color represents a different class. One common example is the segmentation of traffic signs, but currently, it is widely used in healthcare.

3.4 Vision transformers

Using Vision transformers brings certain benefits that can be utilized, for example, in our task. To be correct, Vision transformers were not directly designed for processing high-resolution photos, but they were designed as a different method of looking at the input compared to convolutional neural networks.

Unlike convolutional neural networks that process the input directly, Vision transformers divide the input into so-called patches [17]. Vision transformers contain self-attention mechanisms that process relationships between all patches.

As mentioned above, there are Vision transformers that efficiently process high-resolution inputs based on their operation on the input described above.

3.5 Spherical convolutional neural networks

Spherical convolutional neural networks are becoming increasingly potential, as evidenced by examples mentioned in the Introduction of this work. This type of network retains the properties of ordinary convolutional neural networks while adding unique features.

Spherical convolutions add a different look on rotations compared to ordinary convolutions. This also leads to much greater memory demand, which we should consider when wanting to use spherical convolutional neural networks.

In this type of convolution, we often encounter the terms S^2 and $SO(3)$, each meaning something different. S^2 describes the surface of a sphere, and simply said, $SO(3)$ describes all possible rotations in three-dimensional space.

3.5.1 Overview

Spherical CNNs

The paper [11] presents spherical convolutional neural networks. Specifically, it discusses their advantages, which include equivariance to rotations (further explained) and handling of distortions that occur when transforming a spherical object into a planar object. The input images are processed on S^2. The basis of this method is that it operates with 3D

rotations instead of translations. Therefore, the filters move not only in this, but also in other implementations, on $SO(3)$. A spherical correlation defined in this work involves calculating the inner product of the input feature map with a rotated version of the filter for various rotations, so spherical correlation can handle rotations and distortions. It uses Fast Fourier Transform for efficiency in computation.

Learning SO(3) Equivariant Representations with Spherical CNNs

The paper [13] addresses the problem of equivariance to rotations. It introduces a spherical convolutional network that performs convolutions in the spherical harmonic domain. Inputs to the network are 3D shapes represented as functions on the sphere. Spherical convolution is implemented by transforming 3D data into the spherical harmonic domain, where convolutions are executed as pointwise multiplications. Moving in the spherical harmonic domain then ensures equivariance to rotations.

SphereNet: Learning Spherical Representations for Detection and Classification in Omnidirectional Images

The paper [12] introduces spherical convolution designed for processing panoramas by adapting convolutional neural networks to work with the spherical geometry. It achieves this by presenting a kernel sampling pattern that handles the distortion by projecting the regular grid points used in convolutions onto a tangent plane of the sphere at different locations. This projection adjusts the positions of the kernel to match the expected locations of features on the sphere. This convolution is not designed to be rotation equivariant.

Learning Spherical Convolution for Fast Features from 360° Imagery

The paper [27] presents an approach for processing 360° images in equirectangular projection. The convolution kernels are adjusted to match the distortion at different parts of the equirectangular projection. This means that for each row (latitude) of the photo, there is a separate kernel size determined to accommodate the distortions of the equirectangular projection.

With this type of convolution, we talk mainly about two of their advantages. The first advantage is equivariance, or invariance to rotations [13], and the second advantage is the significant deformation that occurs when we want to project a sphere onto a 2D surface [12]. These properties are typically not found in ordinary convolutions.

As mentioned, we talk about equivariance or invariance to rotations. The principle is similar to ordinary convolutions, which are equivariant or invariant to translations. In practice, this means that if a convolution is not equivariant or invariant to rotations, then when we display the output of some photo after convolution and then display the output of the same photo but rotated around the Z-axis, the outputs will not resemble each other, and thus will look completely different, this can be seen in Figure 3.3. Therefore, if a spherical convolution ensures rotation equivariance, the outputs will be the same, where the second output will be rotated around the Z-axis by a certain number of degrees, so if we rotate the second output back by the same number of degrees, both outputs will match, as visualized in Figure 3.4. We want equivariance to rotations when we want to

preserve information about this angle. Rotation invariance, visualized in Figure 3.5, means that both outputs will look the same, and in practice, it works in such a way that the filters rotate, and that is the reason why the outputs will look the same. As we might guess, invariance loses information about this angle. What we must be careful about is that not every implementation of spherical convolution ensures equivariance and invariance to rotations. For a better understanding and visualization, I include an image below.

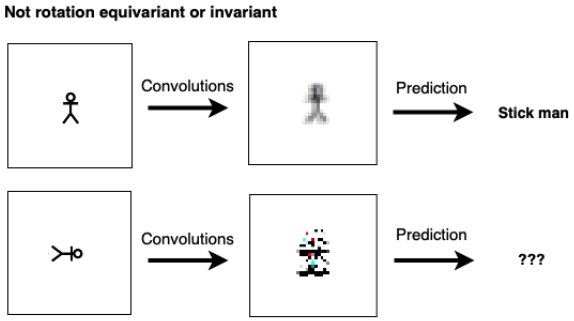

Figure 3.3: Not rotation equivariance or invariance visualization

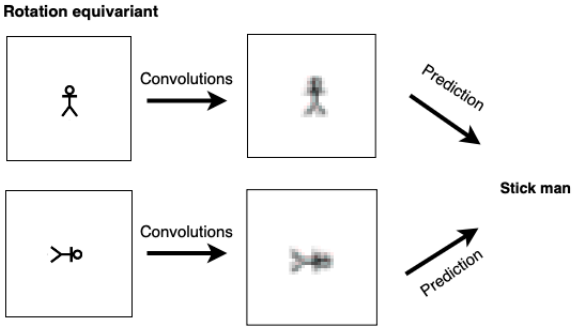

Figure 3.4: Rotation equivarance visualization

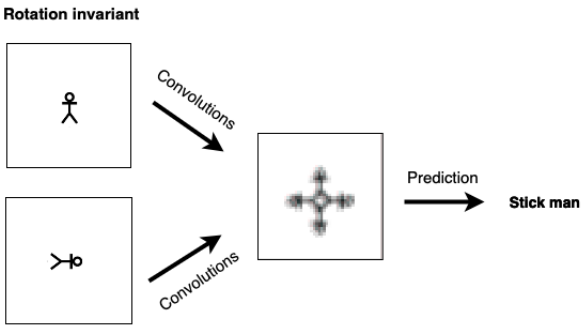

Figure 3.5: Rotation invariance visualization

In the article [11], there's a comparison of training number classification model on images using ordinary convolutional networks and their implementation of spherical convolutional

neural networks. The results are precisely as expected, thus during training, for example, on rotated inputs of the MNIST dataset and subsequent testing, the ordinary convolutional neural network had an accuracy of 23%, and the spherical one had an accuracy of 95%. Similarly, this was the case when they trained on non-rotated photos and then tested on rotated photos.

Another mentioned advantage is that if we want to project a sphere onto a 2D surface, deformations will occur, which can be a problem when training using ordinary convolutions. An example of deformation is shown in Figure 3.6. In reality, the input for these convolutions is a 2D photo onto which a sphere is projected using the formats mentioned in section 2.3.2. The trick is that, for example, in the implementation [11], the input of the first convolution S^2 (`s2conv`) is a 2D photo in one of the mentioned formats, and then $SO(3)$ convolutions (`so3conv`) follow. We can also see the difference in the way how the filters pass through the photo in Figure 3.7.

Figure 3.6: Distorted car after transformation from sphere to plane

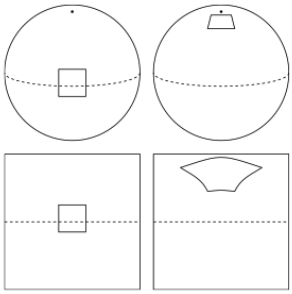

Figure 3.7: Visualization how spherical convolutions pass through photo. Image taken from [11].

When training spherical convolutional neural networks, one must consider certain challenges. The first challenge is the lack of many pretrained models. Similarly, the second challenge is the lack of datasets containing images in formats that project a sphere onto a 2D surface. Another issue is the memory demand of these computations, simply due to the fact that we are in 3D space and not in 2D surface as with ordinary convolutional neural networks. With the use of high-resolution photos, this problem quickly increases, so Vision transformers, as mentioned above, can be used. Currently, there are Vision transformers that operate in the spherical domain, but they do not ensure equivariance or invariance to rotations.

The main motivation for using these networks is the spherical input, but often the fact that the input in the spherical domain is rotated, thus ordinary convolutions are not sufficient.

Chapter 4

Camera orientation estimation using SCNNs

In the previous section 3.5, we introduced the advantages of spherical convolutional neural networks, and in this chapter, we will discuss how we can use spherical convolutional neural networks to estimate the orientation of the camera.

We have partially introduced the goal and task of this work in the previous chapters, but let's go deeper into more details. The goal is to estimate the orientation of the second input in the first from two photos, where one photo is a 360-degree panorama and the other is generally taken at the same place or captures the same place as part of the panorama, but at different times (examples of inputs are shown in one of the following sections 5.1). Thus, our goal is to estimate the orientation of the second input in the first (for simplicity, we can imagine that we want to find where the second input is located in the first), where the orientation is determined by three angles pitch, yaw, and roll (a closer explanation of the angles is found in section 2.2), where pitch has a range of 180 degrees and the remaining two angles have a range of 360 degrees. Task pipeline can be seen in Figure 4.1.

The reason for using spherical convolution is its advantages. As we learned in the previous section 3.5, it's mainly about two advantages. The first advantage is that it processes the input as a spherical object, so for our 360-degree panorama, which is originally a sphere, there are no distortions, where with ordinary convolutional neural networks, where we process the input as a 2D surface, these distortions could cause problems. The second advantage is that spherical convolutional neural networks offer equivariance or invariance to rotations, which ordinary convolutional neural networks do not offer. For us, it's important to choose a convolution that is equivariant and not invariant because otherwise, we would lose information about the angle around the Z-axis (roll).

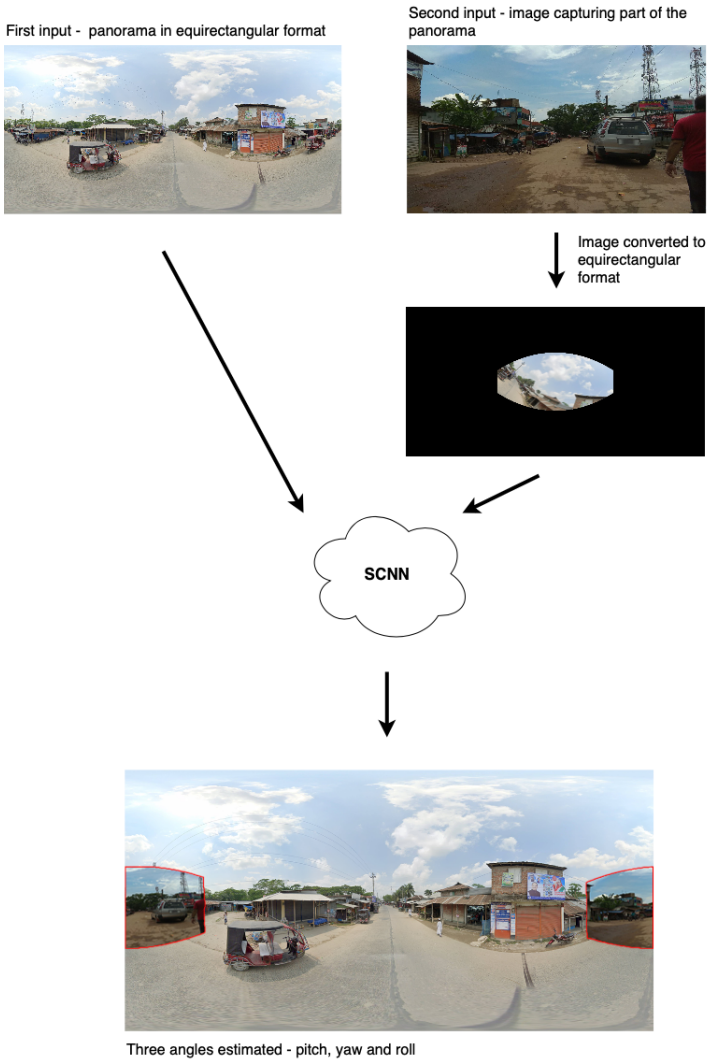

Figure 4.1: Task pipeline

When choosing a spherical convolution, it's necessary to be careful about which format of projecting the sphere onto a 2D surface we use. Generally, two formats are mainly used: equirectangular and stereographic (their description is in section 2.3.2). Another question is whether to solve the task using regression or classification, where so far there is no definitive answer. In regression tasks, we try to predict a continuous value based on inputs. A typical example is estimating the price of a house. In our case, the objective is to estimate three angles, represented as decimal numbers, based on two inputs. On the other hand, classification tasks involve categorizing the content of an image into classes based on the input. An example could be a network that identifies whether a photo contains a dog or a cat. In our scenario, this approach could be utilized either by creating a separate class for each angle or by classes that define the range of angles an input belongs to, such as dividing pitch, yaw and roll into 60 degree intervals. These classes could then be used as information in further models, which might, for example, be based on regression.

The use of multi-layer perceptron layers can be very memory-intensive and make learning challenging for the network. A possible solution could be using a method inspired by

semantic segmentation, where the output is a probability map. This map gives the probability for each pixel that says if answer could be on this pixel. In this method, the inputs remain the same, but the ground truth is now a photo, not three angles. Specifically, the ground truth is a probability map, typically created by binary marking the area where the photo is located as 1 and where it is not located as 0. Then, a Gaussian blur is applied to this photo to create the probability map as can be seen in Figure 5.6. The motivation for using this method is to avoid the use of multi-layer perceptron layers.

This convolution also carries certain challenges that need to be mentioned. Since we are working in with Fourier transforms on $SO(3)$, computations are time-consuming and memory-intensive. This is compounded by the fact that the inputs are high-resolution photos, where we are talking about resolutions from 1000x1000 pixels. When we add a large dataset, for example, with 20,000 training and testing data, we find that one epoch can take up to 2 hours. Another problem with current implementations is inaccurate equivariance, where the feature map from a rotated input should be the same but only rotated by a certain angle. However, this is not the reality, and as a result, distortion is found. Since there are not pretrained models which use this type of network, I decided for training from scratch.

Chapter 5

Implementation

In this chapter I will show you implementation of the key parts of this work, specifically consisting of creating datasets, preprocessing, and an overview of the developed models.

The implementation of the work is in `Python`, which offers many libraries for solving the given problem, especially using `Pytorch` for selected convolution in model implementation.

5.1 Datasets

Before creating model, it is necessary to have dataset on which the model will be trained and then tested. The data for my implementation of the spherical convolutional neural network model consists of 2 photos. The first photo is a 360-degree panorama in equirectangular or stereographic format, for example, of a landscape, and the second photo is taken at the given geographical longitude and latitude, capturing a certain part of the panorama, but our goal is not to have photo directly extracted from the panorama (also there is an initial dataset where second photo is cutout for panorama), because that is not our main goal. The second photo is then in preprocessing converted into equirectangular or stereographic format. Furthermore, it is necessary to obtain the camera's orientation angles and field of view for the second photo in the panorama.

5.1.1 SPPAI dataset

In my implementation of data acquisition, the first thing I implemented was obtaining a photo that forms part of the panorama, see Figures 5.1 and 5.2, at a randomly generated geographical latitude and longitude. For acquiring these photos, I used the `Mapillary API` service, where, after generating random geographical latitude and longitude, I verified whether a photo exists at the given coordinates, and if so, then it was necessary to determine whether a panorama also exists at those coordinates. That is why I have named this dataset Same Place Panoramas and Images Dataset (SPPAI). For this information, I used the `Google Street View Static API`, which also provides API for panorama downloading. Therefore, if a photo and a panorama exist at the given coordinates, then data begin to be downloaded and stored otherwise, coordinates are generated again until a photo and a panorama are found on same coordinates. It is important to mention again that I used the equirectangular format for projecting the 360-degree panorama onto a 2D surface. The obtained perspectives also include information about pitch, yaw, and roll angles, but what is missing in this dataset is the field of view information. However, this could be a topic

for extending this work. This dataset contains 393 of these pairs and using implemented program described below, there can be generated much more. This problem allowed us to use this dataset only partially, specifically for creating additional datasets as will be mentioned later. Therefore, it was not necessary to expand this dataset, but it can be expanded by running the scripts described below.

The actual program for downloading data is implemented in the file `src/datasetGenerator/datasetGenerator.py`. Functions for downloading random photos are implemented in the file `src/datasetGenerator/mapillaryImageGenerator.py`, and functions for downloading panoramas are implemented in the files `src/datasetGenerator/randomPanoramaIDGenerator.py` and in the file taken from GitHub [1] `src/datasetGenerator/download.py`. Dataset can be found on this link [2].

Figure 5.1: Downloaded panorama in equirectangular format

Figure 5.2: Downloaded perspective on same location as panorama 5.1

[1] https://github.com/robolyst/streetview
[2] https://nextcloud.fit.vutbr.cz/s/9QyZ7AiFpBSY2sM

5.1.2 PAC datasets

PAC

From the previous dataset, a simpler Panoramas and cutouts (PAC) dataset was created for the initial testing of the developed model. In this dataset second image is cut out from panoramas at random angles and random field of view, this can be seen in Figure 5.3. The implementation of this dataset is located in `src/preprocessing/PAC/preprocessing.py`, and for creating cut outs I used `Python` library `convert360` [3], which can create cutouts at given pitch, yaw, roll and field of view. Each cut out undergoes augmentation, where augmentation is randomly selected from contrast, blur, or noise. This dataset contains around 1975 of these pairs but also dataset with around 20k of these pairs was generated and can be created by using script above. Dataset with 1975 pairs can be found on this link [4].

Figure 5.3: Augmented panorama 5.1 cutout transformed to equirectangular format

StereoPAC

Because I tried two different implementations of spherical convolutions and the second one required input in the stereographic format, it was necessary to transform the PACD dataset to stereographic projection, this can be seen in Figures 5.4 and 5.5. For this, I used an existing implementation in `C++` [5], which can convert equirectangular format to stereographic. The script for creating the dataset can be inspired by `src/preprocessing/equir2stereo/src/main.py`. Similarly, this dataset can be found in a version with 1975 pairs at this link [6].

[3] https://github.com/sunset1995/py360convert
[4] https://nextcloud.fit.vutbr.cz/s/oce47DbP7dzerCt
[5] https://github.com/chinhsuanwu/360-converter
[6] https://nextcloud.fit.vutbr.cz/s/roLtwWwDJ2iq6je

Figure 5.4: Panorama in stereographic format

Figure 5.5: Augmented cutout in stereographic format

PACNoRoll

When training the angles pitch and yaw without the roll angle, with one of the implementations of spherical convolution that does not ensure equivariance to rotations, it was necessary to create a dataset with cutouts but without the third angle, roll. Often, this dataset may be sufficient because people tend to take photos aligned and not tilted. The script for creating the dataset can be inspired by `src/preprocessing/PAC/preprocessing.py`. This dataset contains 5910 pairs and can be found on this link [7].

PACNoRollProb

In the search for a successful solution, where one of the solutions involved the use of probability prediction, it was necessary to create a dataset where the panorama and photo are the same as in the PACNoRoll dataset, but the ground truth this time are not float numbers but a photo, where the correct orientation is marked in white and the background is black, a Gaussian blur is then applied to represent the probability map, as can be seen in the Figure 5.6. The script for creating the dataset can be found at `src/preprocessing/Probability/PitchYaw/preprocessing.py`. This dataset contains 5910 pairs and can be found on this link [8].

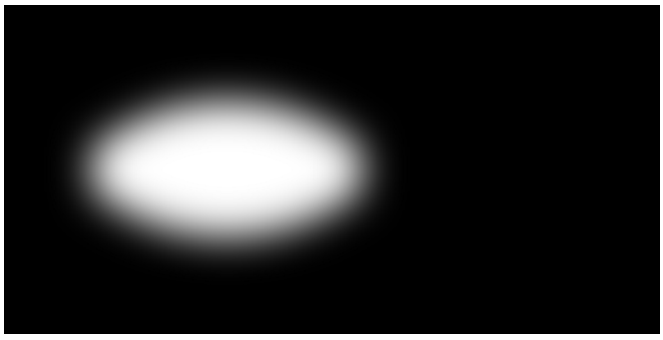

Figure 5.6: Ground truth visualization for PACNoRollProb dataset

[7] https://nextcloud.fit.vutbr.cz/s/bgoA3C3jssfoDcg
[8] https://nextcloud.fit.vutbr.cz/s/ZqYGBwQH9WtcZTH

PACNoRollYawProb

For similar reasons as in the previous section 5.1.2, a dataset was created for use also for probability prediction, where the difference is that this dataset is designed for the network to learn to estimate the yaw angle, and thus, in other words, the cutout is positioned in the photo at the correct pitch angle. The script for creating the dataset can be found at `src/preprocessing/Probability/Yaw/preprocessing.py`. This dataset contains 3000 pairs and can be found on this link [9].

5.1.3 GeoPose datasets

GeoPose3K

An important dataset is the GeoPose3K dataset [6], which contains rendered models of panoramas and photos which are always some part of this panoramas in mountainous areas, where similarly, the necessary data such as field of view, pitch, yaw, and roll are available. This dataset contains panoramas in cylindrical fromat.

EquiGeoPose3K

Because GeoPose3K contains panoramas in cylindrical format, it was necessary to convert these panoramas to the equirectangular format. The script for creating the dataset can be found at: `src/preprocessing/geoPose/preprocessing.py`. The dataset can be found at this link [10].

StereoGeoPose3K

Since one of models uses the stereographic format as input, it was necessary to convert the initial dataset to the required format. The script for creating the dataset can be inspired by: `src/preprocessing/equir2stereo/src/main.cpp`. The dataset can be found at this link: [11].

It's important to clarify the contents of the .csv or .txt files that are available for each pair of photos. In every dataset, they contain 3 basic pieces of information about the second photo/input - the angles pitch, yaw, roll.

For the SPPAI dataset, this file additionally contains information in the following order: Google Street View panorama ID, Mapillary photo ID, latitude, longitude, pitch, yaw, and roll, where the orientation angles are described in the Mapillary documentation [12].

The PAC and StereoPAC datasets contain information in the following order: pitch, yaw, roll, and field of view. For simplicity, the angles are in degrees, and the pitch angle starts from the south, where the angle is 0 degrees, and at the north, it is 180 degrees. The yaw angle starts from the west, where the angle is 0 degrees, and in the east, the angle is 360 degrees. The roll angle has 0 degrees in the aligned position, and according to the direction of the clock hands, degrees increase up to 360 degrees.

[9] https://nextcloud.fit.vutbr.cz/s/SjdF6tSJSHN73fP
[10] https://nextcloud.fit.vutbr.cz/s/Pnwz6r9sbWABwPr
[11] https://nextcloud.fit.vutbr.cz/s/JNwyTKz3kec6wjb
[12] https://www.mapillary.com/developer/api-documentation

In the EquiGeoPose3K and StereoGeoPose3K datasets, the information file contains in the reading order: pitch, yaw, and roll, where angles works exactly the same as in the previous datasets.

5.2 Preprocessing

For this task, a significant part of the preprocessing was done during the creation of the SPPAI dataset, where the preprocessing of the mentioned dataset is implemented in the file `src/datasetGenerator/datasetGenerator.py`. The task of preprocessing is, first, to check the input data, as sometimes downloaded panoramas or perspectives are entirely black or have black edges. Images that are entirely black are discarded, and images with black edges are cropped so that the edges are not present. Furthermore, in this part, the size of the panoramas is adjusted to the same size. The preprocessing and creation of other datasets consisted of their very creation, where the implementation can be found at the individual datasets mentioned in section 5.1.

5.3 Models

In this section, I will introduce the models that were developed for our task. After development, testing, and experimentation, three main models were created, each with a different idea and a different perspective how to solve the given problem. The third model is extended by an additional auxiliary model.

At the beginning of this section, it's important to mention again that for this type of convolution, there are no pretrained models available, and thus it's necessary to train the network from scratch, which, along with other challenges mentioned in chapter 4, brought several challenges and thus the need for a lot of experimentation and creation of several models, each with different idea how to solve the problem. The models can be compared according to their results and accuracy, and thus deduce the best direction for solving this problem.

Training initially took place on datasets from the PAC family described in section 5.1.2, which, for repetition, consist of a panorama as the first input and an augmented cutout from that panorama as the second input. I chose this because if we can't train the network on this type of dataset, then we most likely won't be able to train the network on another dataset where the inputs do not have such a direct relationship as we can see, for example, in the GeoPose3K dataset described in section 5.1.3.

5.3.1 S2CNN model

Currently, there are 2 implementations of spherical convolutions. The first is `s2cnn` [11], which ensures all of the benefits mentioned in section 3.5. Then there is the second implementation, `SphereNet` [12], which does not ensure rotation equivariance, and therefore, for this model, I chose the first implementation of mentioned convolutions.

This implementation is based on SO3 Fourier transformations. It includes 2 types of convolutions, where first, for the input, we need `S2Convolution` and then we apply `SO3Convolution`, this is because we initially move on the surface of a sphere and then we

need to apply $SO(3)$ convolutions due to rotations. According to [11], the input for this convolution are images in stereographic format.

As I mentioned above, the implementation requires input in stereographic format, so I used the StereoPAC 5.1.2 training dataset. Again, as I mentioned, it's important to first use a simpler dataset for training and then do fine-tuning on another dataset, so we can verify the functionality of the architecture.

However, what proved to be the best architecture, given the fact that we are limited by GPU memory, is an architecture made of one S^2 convolution layer and three $SO(3)$ convolution layers, where with this type of convolutions, pooling can also occur simultaneously, followed by batch normalization after which the activation function `ReLU` is used. Convolutions are applied separately to each of the two inputs, where after converting the outputs of the last convolution to 1D, the inputs are concatenated across the first dimension. There are 2 fully connected layers, with an activation function `ReLU` following the first. The `features out` parameters are set to 16, 32, and 64 for convolutions and for fully connected layers are set to 64 and then 3, because as output we want three angles: pitch, yaw, and roll. Architecture can be seen in Figure 5.7. The best learning rate was found to be 0.0001 with the use of `Adam` optimizer, and a polynomial learning rate scheduler was used. A resolution of 1000x1000 pixels and a batch size of 16 were used. During testing, normalization of the model output also helped, because an angle of 0 degrees is the same as 360 degrees, etc. Since I decided to solve the task using regression, `MSELoss` was used.

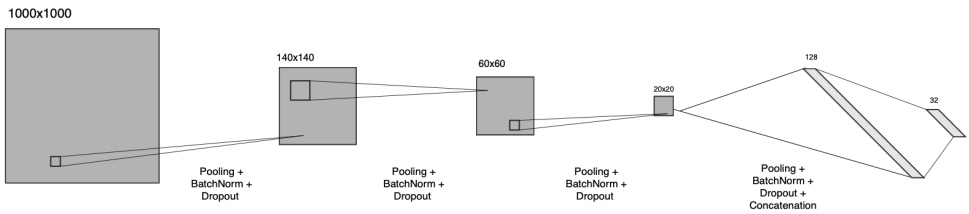

Figure 5.7: Model architecture.

It's necessary to clarify how these convolutions work because their functionality and parameters differ from ordinary convolutions. As I mentioned, it's always necessary at first to use `S2Convolution` and then `SO3Convolutions`. Each convolution takes `nfeature_in` and `nfeature_out` as input parameters, which are same as in ordinary convolutions. However, new different parameters include `b_in` and `b_out`, representing the bandwidth of the FFT, and in practice, represent the input resolution divided by 2, so if we have a 1000x1000 input then the input bandwidth will be 500 and if we set the output bandwidth, for example, to 150, then the output will be in a resolution of 300x300. The number 150 is not entirely random because this number limits the output bandwidth, which is restricted due to the high computational and memory demands. The last parameter is `grid`, which represents the grid, where we have 2 types to choose from. The first type is `near_identity_grid`, which is preferred choices because they represent kernels defined at the north pole and perform SO(3) rotations. The second type, `equatorial_grid`, represents ring kernels around the

equator. Each grid contains further parameters. The parameter `max_beta` adapts the size of the kernel as an angle measured from the north pole. With this parameter, we are able to perform the mentioned pooling. For example, if we want to do pooling by a factor of 2, we need to divide `bandwidth` by 2 and multiply `max_beta` by 2. The parameter `n_beta` represents the number of rings of the kernel around the equator, which in practice means that setting `n_beta` to 1 would represent a 3x3 kernel size in ordinary convolutions, but it's important to remember that we are on a sphere and the kernels look like rings. Other parameters are usually not set and default values are used.

Training results

With this model, we didn't achieve an accuracy of even 1 percent, where there are several reasons why. As I've already mentioned, there are no pretrained models for this type of convolution, which complicates the work and the results. Since we are working at a high resolution and even moving in 3D space with this type of convolution, we are limited by the size of GPU memory, where ideally, we would need over 100 GB of memory. Added to all of mentioned there is the fact that for testing the model, I used a dataset with 20K images 5.1.2, where one epoch lasts over 2 hours. Specific problems associated with GPU memory size include the fact that we cannot experiment with higher input resolutions, where, on the one hand, we run out of memory, and on the other hand, the output `bandwidth` is limited to 150, meaning the output resolution can be imagined as 300x300 pixels, thus a significant jump and loss of a lot of information. We would prefer to avoid pooling to some extent, during which we also lose many details that are key for this task. The prediction itself occurs in the fully connected layers, which have a large number of parameters and are computationally and memory demanding, therefore, we can only have two fully connected layers.

5.3.2 SphereNet model

After the failure of the previous model, I decided to implement a model with a different implementation of spherical convolution. Currently, there are two implementations implemented: the spherical convolution implementation used in the previous model and the `SphereNet` implementation [12], which takes photos in equirectangular format as input and moves in 2D space during all convolutions, unlike the previous convolutions. However, the disadvantage of this convolution and the reason why I did not choose this convolution from the beginning is the fact that the implementation does not ensure equivariance to rotations. This implies that with this model, we will need to train inputs that are not rotated around the Z-axis, and thus at the roll angle. In practice and for our assignment, this may not be such a big problem since people typically try to take photos that are aligned with horizon, but in other types of tasks, this could be a problem.

For this model, I used the PACNoRoll dataset 5.1.2 for training. Like in the previous model, I decided to address this problem using regression.

Considering the available GPU memory size in the creation of the model, the best possible architecture consists of three layers, where each layer is created by a `SphereConv2D` convolution followed by `SphereMaxPool2D` pooling. After each convolution, there is a `ReLU` activation function. The input is again an RGB photo. The features out parameters are set to 8, 16, 32. Each of the two inputs passes through the layers separately, and after the last

layer, concatenation is performed. There are 2 fully connected layers where the number of input features are over a million parameters, and the output is 2048. In the second fully connected layer, the output is set to 2, because in this case, we predict only 2 angles: pitch and yaw. Inputs are in a resolution of 1536 in width and 768 in height. Visualization of architecture can be seen in Figure 5.8. `MSELoss` is used. The most optimal learning rate turned out to be 0.0000015, using the `Adam` optimizer. A polynomial learning rate scheduler was used. The batch size was set to 16.

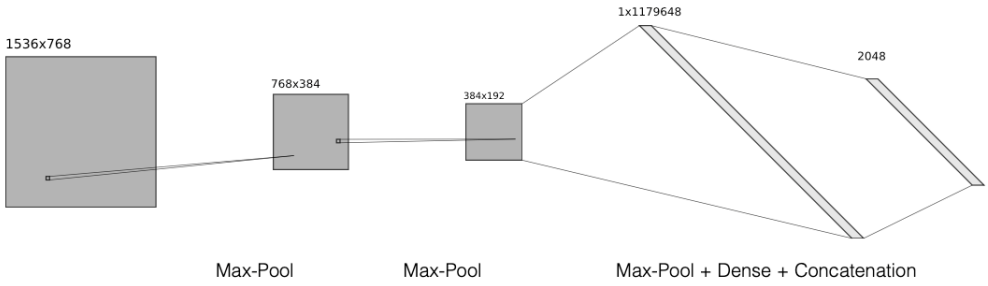

Figure 5.8: Model architecture.

Training results

After 120 epochs, as we can see in Figure 5.9, overfitting was present, but we achieved a validation loss of 1.67 and the best accuracy of 2 percent with a tolerance of 0.15 radians (8.5 degrees), which is not sufficient for our task.

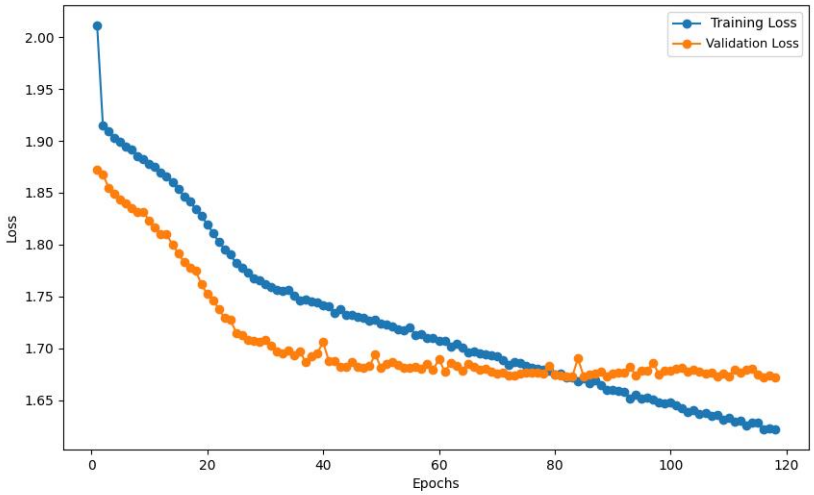

Figure 5.9: Test and validation loss graph.

There are several reasons for the low accuracy achieved. First and main reason is that we are training from scratch because, again, there are no pretrained models for this type of convolution. Another problem is that we are limited by the availability of GPU memory size, which means we cannot, for example, increase the resolution of the photo which is something that seems very effective in decrease of an accuracy and loss. Testing showed that 3 convolutional layers are enough, and if more GPU memory will be available, it would be beneficial to add more fully connected layers. This is another problem since we have a high resolution and, for this task, insufficient size of GPU memory, so we would prefer not to perform pooling so often, which is not possible, and thus we lose detail for the fully connected layers. We would also need more fully connected layers because they play the most crucial role, and large jumps in parameters are not great.

5.3.3 SphereNet probability prediction model

As was mentioned in the previous section 5.3.2, one of the main problems of previous models was the prediction itself, which occurs in the last fully connected layers consisting of multi-layer perceptrons. For this reason, it was necessary to develop a model that does not include multi-layer perceptron layers. Therefore, I decided to use the probability prediction method inspired by the article [25], where the `U-Net` architecture was introduced. This architecture is popular and is often used in solving various problems.

For this model, I again decided to use spherical `SphereNet` convolution (`SphereConv2D`) and spherical max pooling (`SphereMaxPool2D`) [12]. The `U-Net` architecture was thus adapted for our problem, where the type of convolution was changed from ordinary 2D convolution to spherical convolution. The same applies to pooling layers.

As a reminder, the `SphereNet` implementation does not ensure rotation equivariance, and thus the training data are inputs that are not rotated around the Z-axis (roll angle), and thus the dataset 5.1.2, which is created to solve the problem using probability prediction. The input was the same as in previous models, but the difference was in the ground truth, where it did not consist of three numbers, but of a photo composed of probability map, indicating the correct orientation. It means that the model predicts only one class. An example of ground truth can be seen in section 5.1.2. This implies that the prediction also does not consist of, in this case, 2 numbers, but of so-called probability maps, indicating where the result is located with a given probability (an example can be seen in the photo 5.13).

The architecture consists of `DoubleConv` blocks, which are always made up of two spherical convolutions, each followed by `BatchNorm2d` and then the activation function `ReLU`. The model is at the beginning created by `DoubleConv(2, 64)`, where the parameters represent the number of in features and the number of out features. And then five `Down` blocks, which are used for downscaling and consist of `MaxPool` and `DoubleConv`. Out features in `Down` blocks are in this order: 128, 256, 512, 1024 and 512. Then there are `Up` blocks, which are used for upscaling and consist of `Upsample` method and `DoubleConv` block. Out features in `Up` blocks are in this order: 512, 256, 128, and 64. The last layer is `OutConv`, which consists of a single spherical convolution, and since we have one class, the out features for this layer are set to 1. A more detailed description of the architecture is in Figure 5.10. Used loss function is `Dice Loss` and used optimizer is `Adam` with a learning rate set to 0.0001. The optimizer is supplemented with a polynomial learning rate scheduler.

To save memory, the batch size is set to 4. The size of inputs is set to a width of 950 pixels and a height of 475 pixels (the maximum that can be set according to available GPU memory size). The presented architecture and parameters are, as in previous models, the best result of experimenting with the number of layers, loss functions, various learning rate values, etc.

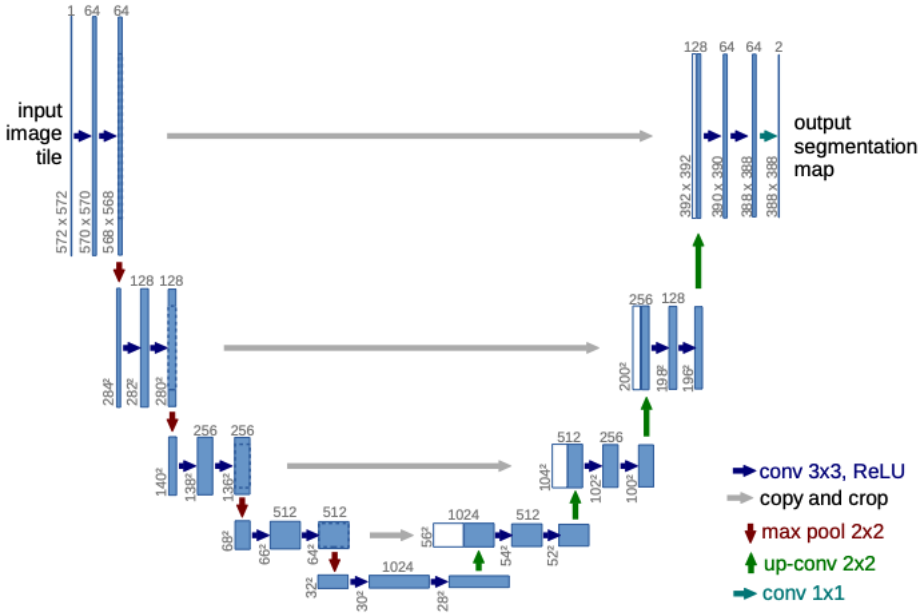

Figure 5.10: U-net architecture visualization for 32x32 input resolution. Blue boxes represent a multi-channel feature map. White boxes represent copied feature maps. Image taken from [25].

Because the idea of this model is to avoid fully connected layers (multi-layer perceptrons), it's necessary to solve the problem created by the fact that we have 2 inputs. Thus, without fully connected layers, the concatenation of these 2 inputs cannot be done after convolutions and therefore before fully connected layers, but concatentation needs to be done before convolutions. We have four options for concatenation. The first two options consist of placing the inputs, or in other words, the photos, either on top of each other or side by side, and then this goes as one input into the model. The problem with these two approaches is that they ruin the equirectangular format because if we will add the second input below it, first problem is that 2:1 resolution ratio will not match, but another problem is that what was on the equator in the original input will now be somewhere closer to the pole, and thus the convolution will process the photo incorrectly. Therefore, these two approaches cannot be used. Another approach is to place the cutout (without black edges) on the panorama, but there's a possibility that a lot of information will be lost and the network will not be able to predict the orientation correctly. The last and used approach is where, in our case, grayscale photos always have 1 color channel each. What can be done is that another channel is added to the panorama, which will consists of second input, and thus the input photo will have an image with two color channels as input, where the first is the panorama and the second is, for example, in our case, the cutout. For better understanding you can see it in Figure 5.11.

Figure 5.11: Channels concatenation visualization.

The above architecture represents the maximum we can use from currently available resources, where we are talking about available GPU memory with size of around 60 GB. Due to this limitation, it is not possible, for example, to add layers or increase the resolution of the image, which could help in more precise training.

Training results

The training lasted 94 epochs, where, as can be seen from the graph below, as we can see in Figure 5.12, overfitting occurred. The lowest achieved validation loss was 0.66. During training, the so-called `Dice score` was also measured, which can be understood as the similarity between the predicted image and the ground truth image. At the first epoch, this score was 0.19 and the best achieved was 0.33.

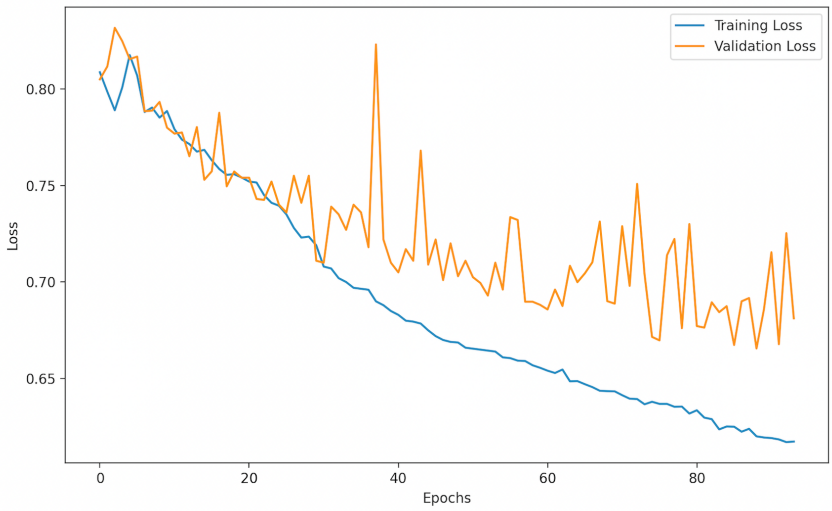

Figure 5.12: Test and validation loss graph.

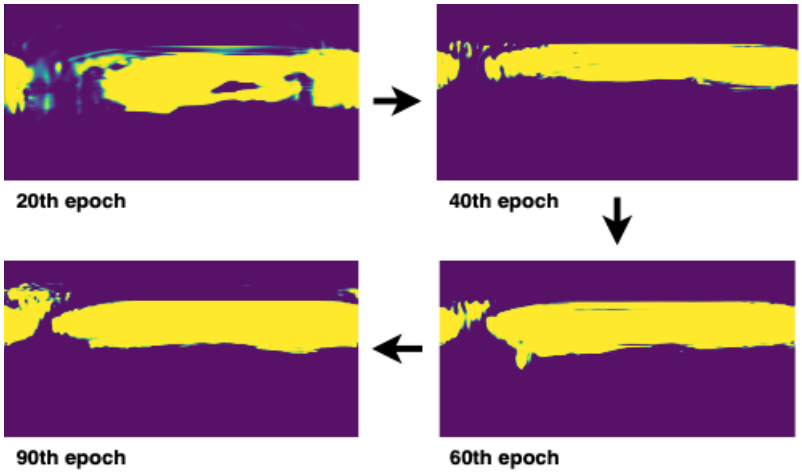

Figure 5.13: Probability maps during training.

Figure 5.14: Visualization of prediction and ground truth error.

It was possible to estimate the pitch angle with an accuracy of 39% if we tolerate an error of 10 degrees, 53% if we tolerate an error of 15 degrees and 61% if we tolerate an error of 20 degrees. The value was obtained as an average between the Y position of the lowest and highest pixel. However, it was necessary to remove lower probabilities from the estimated probability map with a threshold of 0.999999 before finding the average.

What we see at first glance from the probability maps in Figures 5.13 and 5.14, is that we cannot extract the yaw angle in a reasonable way, similar to the pitch angle. Therefore, I decided to create a model with the same architecture that will estimate only the yaw angle, but the input will be a cutout set at the correct position of the pitch angle. After 120 epochs, as we can see in Figure 5.15, overfitting had occurred, we achieved a validation loss of 0.41 and a dice score of 0.58.

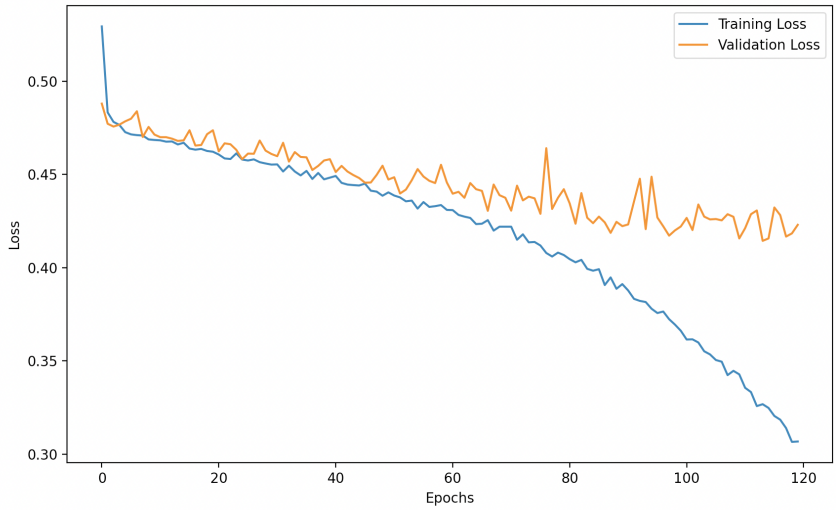

Figure 5.15: Test and validation loss graph.

Figure 5.16: Visualization of prediction and ground truth error.

As we can see in this submodel (compared between Figures 5.16 and 5.14, where the same cutouts were used), no reasonable results were achieved that we could get from probability maps, in contrast with model that estimates the pitch angle, but visually, it seems to have better results than with the first model, But I tried to measure accuracy, with similar method like in case of pitch angle. I made an average of leftmost and rightmost pixel. If we tolerate and an error of 10 degrees accuracy is 6%, if we tolerate an error of 15 degrees accuracy is 10% and if we tolerate and error of 20 degrees accuracy is 16%.

There could be several reasons for the not fully successful training for estimating pitch and yaw angles. An important issue, in my opinion, is that it would be better if we will have the photo in a higher resolution, which showed as an improvement in accuracy for all models, but this was not possible due to the capacity of the GPU memory. For the same reason, it was not possible to make the network deeper. As I have mentioned, pretrained models are typically used for many tasks, which, in our case, is not possible and this could also be a key factor in training and accuracy.

5.3.4 Models conclusion

In each model, I encountered the limits of current hardware or the current situation in the world of spherical convolutions implementations. The only model that led to at least partial results was the last model, where we addressed the problem using probability prediction. After explaining the issues with each type of model with different ideas, based on results I consider the direction of probability prediction to be the most sensible. One of the benefits of spherical convolutions, namely rotation equivariance, which the `SphereNet` implementation [12] does not ensure, may not be so problematic in certain situations. In one of the use cases for our task, which is to estimate camera orientation in photos taken by people, typically, people automatically tries to keep the photo aligned in the roll angle and not rotated. In other situations, it can be necessary for the convolution to also ensure this property. Model for estimating pitch can be found on this link [13] and model for estimating yaw can be found on this link [14].

[13] https://nextcloud.fit.vutbr.cz/s/tozKxRMkC57Zaex
[14] https://nextcloud.fit.vutbr.cz/s/CcoxLkYzSg3Xje9

Chapter 6

Testing and experiments

Some results were presented in previous section 5.3. In this chapter, we will test the accuracy and behavior of the `SphereNet probability prediction` model for pitch angle and for yaw angle we are testing with submodel trained for estimating yaw angle, because this models brought us the best results. First, we will try to test the trained model on the `PAC` dataset, which, contains rotation around the Z-axis (roll angle). Then, we will try to test the accuracy on the `GeoPose` dataset, where at first we test the accuracy on models without fine-tuning, and finally, we will fine-tune the model and compare the results.

Each test consists of estimating the pitch and yaw angles separately, where for each angle we test the accuracy with a tolerance of error of 10, 15, and 20 degrees.

6.1 Rotated PAC dataset

The input for this experiment was a dataset consisting of panoramas and cutouts, where the cutouts were randomly rotated around the Z-axis (roll angle). The results for this dataset can be seen in the table 6.1.

	10° error	**15° error**	**20° error**
Pitch	19%	26%	36%
Yaw	8%	10%	10%

Table 6.1: Accuracy for rotated PAC dataset

6.2 GeoPose3K

In this test, we will calculate accuracy on the GeoPose3K dataset 5.1.3, where the first test will be on models without fine-tuning, and the second test will be on models where we perform fine-tuning on the training part of the GeoPose3K dataset.

As can be seen in Figures 6.1 and 6.2 compared to 5.16, it was necessary to use a more sophisticated algorithm to get the angle from the probability map. Since we can see that in this case, the set threshold does not eliminate small unwanted segments, I decided to implement functionality that always selects the largest segment, from which the average between the leftmost and rightmost pixel is calculated.

6.2.1 Without fine-tuning

The results for GeoPose dataset without fine-tuning can be seen in the table 6.2.

	10° error	**15° error**	**20° error**
Pitch	57%	80%	91%
Yaw	14%	20%	26%

Table 6.2: Accuracy for GeoPose3K dataset without fine-tuning

The reason for the high accuracy for the pitch angle is that, according to [1], often it is enough in the transformation from cylindrical to equirectangular projection to simply stretch the photo to a 2:1 ratio, so there will be black edges. Thus, it is easier for the model to achieve accuracy.

6.2.2 Fine-tuning

The results for GeoPose dataset with fine-tuning can be seen in the table 6.3.

	10° error	**15° error**	**20° error**
Pitch	76%	92%	96%
Yaw	16%	28%	42%

Table 6.3: Accuracy for GeoPose3K dataset with fine-tuning

Similarly, in this experiment, the reason for the high accuracy of pitch angle is the same as in the previous experiment.

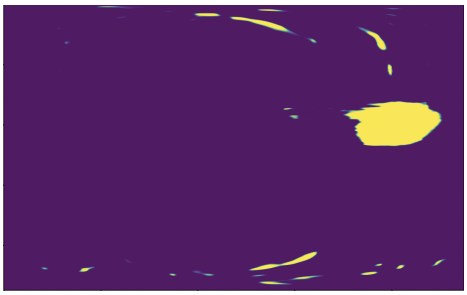

Figure 6.1: Example of probability map for GeoPose dataset without fine-tuning.

Figure 6.2: Example of probability map for GeoPose dataset with fine-tuning.

In Figure 6.3 below, we can see one of the predictions compared with the ground truth image from the GeoPose dataset.

[1] https://paulbourke.net/panorama/pano2sphere/

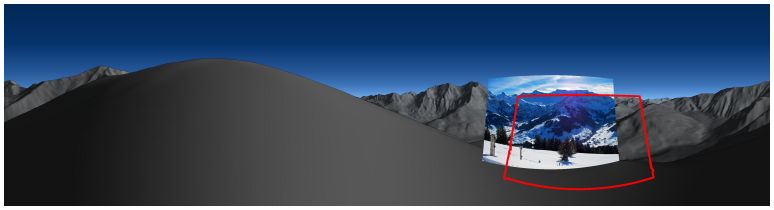

Figure 6.3: GeoPose3K prediction and ground truth visualization.

6.3 Comparison

Other methods mentioned in section 2.3.3 achieve better results, where we are talking about accuracy to the degree of units. However, I decided to compare fine-tuned model on GeoPose dataset with the results of the work [6], which is similar to ours in several ways. For example, semantic segmentation is used, which is an inspiration for the probability prediction method. Similarly, the mentioned work also estimates the orientation of an input photo with a model in a mountainous environment, similarly to our GeoPose dataset. The authors state that the matching is accurate to under 0.2 degrees, which, as can be seen in Table 6.3, is a difference compared to our accuracy. In our model, the average error in degrees for the pitch angle is 6.45 degrees, and for the yaw angle is 66.44 degrees. In the work [8], the authors achieve the best result with an error of 1.92 degrees on the mountainous dataset.

6.4 Memory and time requirements

The training was conducted on the LUMI supercomputer [2], where the most crucial parameter for us is the size of the GPU memory. For this hardware, it is 64GB.

To train the SphereNet probability prediction model, 55GB of GPU memory is needed. I would like to remind that, we use an image resolution of 950x475 pixels and a `batchsize` of 4. Because of generated error during testing, occured when testing a larger image resolution from which the exact memory usage is not clear, I decided to test the usage at a resolution of 475x237 pixels, which is half of the original resolution. At this setting, 21GB of GPU memory is used. If we will use a `batchsize` set to 2, then we would need 41GB of GPU memory. From this, we can get an idea of how memory usage changes with the modification of mentioned parameters.

The time for one epoch of the model using the PACNoRollProb dataset 5.1.2 was 1 hour.

[2] https://www.lumi-supercomputer.eu

Chapter 7

Conclusion

In this work, spherical convolutions were introduced and clarified, along with their motivation and applications, where our application was camera orientation estimation using the mentioned type of convolutions.

Several models were created for resolving this task, where each has a different idea behind it and brings various results. The best results were shown by the direction of probability prediction, where for each angle, a separate model is created where initially, the pitch angle is estimated and then the yaw angle in case we are estimating only these 2 angles.

In addition to the created models, several datasets were developed, which are a significant contribution, as there are not many datasets that contain panoramas in equirectangular format and even fewer datasets that contain panoramas in stereographic format. Part of the work also includes programs that generate these datasets, and thus they can be expanded.

We encountered the limits of current hardware, specifically the size of GPU memory. Therefore, the continuation of this work could be optimization in several directions, such as the use of 16-bit floating-point numbers instead of 32-bit ones. The results of this probability prediction method are only partial, but a direction has been found for solving the same or similar problems. Also, after using the created model, other algorithms in combination with this could be used for estimating camera orientation. An option is also the use of Vision transformers, which can work better with large resolutions, because increasing the resolution has proven to be profitable. We could also look at the problem through classification, where, for example, we could first estimate the angle interval and add this as information to other models. Extending this work in a slightly different direction would be the estimation of the FOV of a photo in the panorama, because if we don't have this information, we do not know how to place the photo correctly in the equirectangular format.

Bibliography

[1] *Cylindrical Projection* [online]. [cit. 2023-10-06]. Available at: https://wiki.panotools.org/Cylindrical_Projection.

[2] *Equidistant Cylindrical Projection* [online]. [cit. 2024-04-22]. Available at: https://surferhelp.goldensoftware.com/projections/Equidistant_Cylindrical_Projection.htm.

[3] *Omnidirectional (360-degree) camera* [online]. [cit. 2023-09-28]. Available at: https://en.wikipedia.org/wiki/Omnidirectional_(360-degree)_camera.

[4] *The Three Main Families of Map Projections* [online]. [cit. 2023-10-06]. Available at: https://www.mathworks.com/help/map/the-three-main-families-of-map-projections.html.

[5] Baatz, G., Saurer, O., Köser, K. and Pollefeys, M. Leveraging Topographic Maps for Image to Terrain Alignment. In: *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*. 2012, p. 487–492. DOI: 10.1109/3DIMPVT.2012.33.

[6] Baboud, L., Čadík, M., Eisemann, E. and Seidel, H.-P. Automatic photo-to-terrain alignment for the annotation of mountain pictures. In: *CVPR 2011*. 2011, p. 41–48. DOI: 10.1109/CVPR.2011.5995727.

[7] baeldung. *Translation Invariance and Equivariance in Computer Vision* [online]. [cit. 2024-04-22]. Available at: https://www.baeldung.com/cs/translation-invariance-equivariance.

[8] Brejcha, J. and Čadík, M. Camera Orientation Estimation in Natural Scenes Using Semantic Cues. In: *2018 International Conference on 3D Vision (3DV)*. 2018, p. 208–217. DOI: 10.1109/3DV.2018.00033.

[9] Camposeco, F., Cohen, A., Pollefeys, M. and Sattler, T. Hybrid Camera Pose Estimation. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 2018, p. 136–144. ISBN 9781538664209.

[10] Chu, H., Gallagher, A. and Chen, T. GPS Refinement and Camera Orientation Estimation from a Single Image and a 2D Map. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, p. 171–178. DOI: 10.1109/CVPRW.2014.31.

[11] Cohen, T. S., Geiger, M., Köhler, J. and Welling, M. Spherical CNNs. In: *International Conference on Learning Representations*. 2018.

[12] Coors, B., Condurache, A. P. and Geiger, A. SphereNet: Learning Spherical Representations for Detection and Classification in Omnidirectional Images. In: *Computer Vision – ECCV 2018*. Cham: [b.n.], p. 525–541. ISBN 3030012395.

[13] Esteves, C., Allen Blanchette, C., Makadia, A. and Daniilidis, K. Learning SO(3) Equivariant Representations with Spherical CNNs. *International Journal of Computer Vision*. 2017, vol. 128, p. 588 – 600. DOI: 10.1007/s11263-019-01220-1.

[14] Francesco. *Does Dragonfly provide the orientation (yaw, pitch and roll)?* [online]. [cit. 2023-01-21]. Available at: https://dragonflycv.com/support/knowledge-base/does-dragonfly-provide-the-orientation-yaw-pitch-and-roll/.

[15] Francesco. *What is a neural network?* [online]. [cit. 2024-03-14]. Available at: https://www.ibm.com/topics/neural-networks.

[16] GISGeography. *Azimuthal Projection: Orthographic, Stereographic and Gnomonic* [online]. [cit. 2024-04-22]. Available at: https://gisgeography.com/azimuthal-projection-orthographic-stereographic-gnomonic/.

[17] hettiarachchi, H. *Unveiling Vision Transformers: Revolutionizing Computer Vision Beyond Convolution* [online]. 2023 [cit. 2024-03-14]. Available at: https://medium.com/@hansahettiarachchi/unveiling-vision-transformers-revolutionizing-computer-vision-beyond-convolution-c410110ef061.

[18] Jang, J., Jo, Y., Shin, M. and Paik, J. Camera Orientation Estimation Using Motion-Based Vanishing Point Detection for Advanced Driver-Assistance Systems. *IEEE Transactions on Intelligent Transportation Systems*. 2021, vol. 22, no. 10, p. 6286–6296. DOI: 10.1109/TITS.2020.2990983.

[19] Lee, J.-K. and Yoon, K.-J. Real-time joint estimation of camera orientation and vanishing points. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, p. 1866–1874. DOI: 10.1109/CVPR.2015.7298796.

[20] Liu, X., Deng, Z. and Yang, Y. Recent progress in semantic image segmentation. *Artificial Intelligence Review*. Springer Science and Business Media LLC. june 2018, vol. 52, no. 2, p. 1089–1106. DOI: 10.1007/s10462-018-9641-3. ISSN 1573-7462.

[21] Melekhov, I., Ylioinas, J., Kannala, J. and Rahtu, E. Relative Camera Pose Estimation Using Convolutional Neural Networks. In:. November 2017, p. 675–687. DOI: 10.1007/978-3-319-70353-4_57. ISBN 978-3-319-70352-7.

[22] Naval, P. Camera Pose Estimation by Alignment from a Single Mountain Image. *International Symposium on Intelligent Robotic Systems*. 1998, p. 157–163.

[23] Naval, P., Mukunoki, M., Minoh, M. and Ikeda, K. Estimating Camera Position and Orientation from Geographical Map and Mountain Image. In: *38th Research Meeting of the Pattern Sensing Group, Society of Instrument and Control Engineers*. April 1997, p. 9–16.

[24] Porzi, L., Rota Bulò, S., Lanz, O., Valigi, P. and Ricci, E. An automatic image-to-DEM alignment approach for annotating mountains pictures on a smartphone. *Machine Vision and Applications*. february 2017, vol. 28. DOI: 10.1007/s00138-016-0808-0.

[25] Ronneberger, O., Fischer, P. and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In:. October 2015, vol. 9351, p. 234–241. DOI: 10.1007/978-3-319-24574-4_28. ISBN 978-3-319-24573-7.

[26] Shavit, Y. and Ferens, R. Introduction to camera pose estimation with deep learning. *ArXiv preprint arXiv:1907.05272*. 2019.

[27] Su, Y.-C. and Grauman, K. Learning spherical convolution for fast features from 360 imagery. In: *Advances in Neural Information Processing Systems*. 2017, vol. 30.

[28] Uniqtech. *Multilayer Perceptron (MLP) vs Convolutional Neural Network in Deep Learning* [online]. 2018 [cit. 2024-03-14]. Available at: https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1.

[29] Xu, M., Wang, Y., Xu, B., Zhang, J., Ren, J. et al. A critical analysis of image-based camera pose estimation techniques. *Neurocomputing*. 2024, vol. 570, p. 127125. DOI: 10.1016/j.neucom.2023.127125. ISSN 0925-2312.

[30] Zhang, C., Budvytis, I., Liwicki, S. and Cipolla, R. Rotation Equivariant Orientation Estimation for Omnidirectional Localization. 2020, p. 334–350. DOI: 10.1007/978-3-030-69538-5_21.

Appendix A

Contents of the included storage media

```
.
├── README.md                    - Documentation and manual
├── requirements.txt             - Required python dependencies
├── poster.pdf                   - Poster
├── video.mp4                    - Video
├── latex                        - Latex files
├── install                      - Script for installation
├── run                          - Script for executing programs
└── src                          - Scripts and programs
    ├── COESCNN                  - Models
    │   ├── S2CNNModel           - S2CNN model
    │   ├── SphereNetModel       - SphereNet model
    │   └── SphereNetProbModel   - SphereNet probability prediction model
    ├── datasetGenerator         - SPPAI dataset generator
    ├── preprocessing            - Preprocessing
    │   ├── PAC                  - PAC dataset generator
    │   ├── Probability          - Probability datasets generator
    │   ├── equir2stereo         - Script for transforming equirectangular
    │   │                          to stereographic projection
    │   ├── geoPose              - Equirectangular and stereographic
    │   │                          GeoPose dataset generator
    │   ├── lib                  - Library for creating cutouts
    │   └── utils                - Utilities for dataset generators
    └── testing                  - Scripts for models testing and loading
```