CHETAN SRINIVASA KUMAR, B.Eng

URBAN VISUAL LOCALIZATION WITH MAP DATA

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Assoc-Prof. Dipl-Ing. Dr.techn. Friedrich Fraundorfer

Institute of Computer Graphics and Vision
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Horst Bischof

Graz, October 2020

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

This is a placeholder for the abstract. It summarizes the whole thesis to give a very short overview. Usually, this the abstract is written when the whole thesis text is finished.

Contents

List of Figures

1 Introduction

1.1 Motivation

An important task of computer vision is to predict accurately the pose of a camera (rotation + translation) in the real world (called the *pose*), given an image captured from it. This idea is christened *Visual Localization*

Visual localization is vital to applications that require robust navigation capabilities. An example that comes to mind would be the popular Autonomous Driving paradigm, wherein modalities such as GPS can be unreliable in dense urban areas. The camera, therefore, becomes a viable option to provide accurate estimates of the current position.

The standard way to tackle this problem would be to utilize *Structure from Motion(SfM)*. SfM utilizes corresponding points between two images and the geometry between two views to construct a 3D pointcloud of the scene. A query image's location is obtained by finding points of the 3D pointcloud that match with image points, and utilizing geometric techniques to fit a pose to these correspondences. SfM, however, does not scale very well to city-scale areas where obtaining suitable image data and a usable reconstruction are a major problem. It also does not behave well with later modifications to the structure of its representation.

A quicker but less robust way is to store a database of images of known locations in the required area, indexed by carefully chosen image-specific features. The images closest to the query image would be extracted using these features, and an approximate location could be interpolated. This approach is definitely more scalable, but is susceptible to false positives and failure if the sampling of images in an area are not dense enough.

Deep Learning's rise has affected Visual Localization as well. The seminal paper on PoseNet A. Kendall, Grimes, and R. Cipolla, 2015, proved that Localization is possible by training a Convolutional Neural Network (CNN) with image and pose data and then use it to predict the position of a new image in real-time. This approach definitely mitigates both the scale and accuracy problems, but the work by T. Sattler et al., 2019, proves that PoseNet variants are not as powerful as traditional SfM approaches. He proves that the PoseNet variants end up learning several "basis" poses, of which the prediction is a linear combination.

State-of-the-art methods have made strides in mitigating this problem.

We propose a method that utilizes a CNN to infer pose directly on the map raster of the area in question, given a 2D layout of buildings visible from a traveling car. We aim to address convincingly the problem of data acquisition, and scale of localization with this method.

1.2 Contribution

In this thesis, we estimate a coarse localization estimate of a car traversing through an urban setting.

We reformulate the visual localization problem as a 2d pose estimation directly on the map raster. Our Localization CNN (Brahmbhatt et al., 2017, MapNet) is trained on OSM map tiles of an area, where the pose of a tile is simply a 2d offset from some arbitrary reference tile. Overlap between the tiles is assumed.

The location is queried by feeding it a map-tile representation obtained from the pointcloud of the scene.

The method opens up a new data source for training visual localization networks - OpenStreetMap tile data, which covers most of the known urban world. We will aim to show that we can get reasonable pose estimates using just building/road pointclouds, using MapNet trained over large sections of a city.

We evaluate the method with the Oxford RobotCar sequence. We choose locations in the drive sequence which have distinct building shapes, and directly infer GPS locations from the MapNet trained on that section of Oxford.

1.3 Outline

We begin by describing the core concepts utilized in the method outlined in this thesis: the basic concepts of Structure from Motion(SfM), and a delineation of neural networks and convolutional neural networks (CNNs). We then describe the traditional solution to the localization problem, and proceed to the seminal localization CNN PoseNet. We later describe MapNet, and how we utilize it for our method. The process of getting from the PointCloud of the scene to a 2D Map-Tile representation ready for query is also described in this chapter. We then describe how we generate the dataset with which we train MapNet from OpenStreetMap.

Finally, we showcase our experiments and results and close with our conclusions.

2 Theoretical Background

2.1 The Basics of Structure from Motion

In this section, we review the fundamental principles of the SfM pipeline. The SfM method is the classical solution to the visual localization problem, and the diagram below outlines the basic steps of the SfM pipeline.

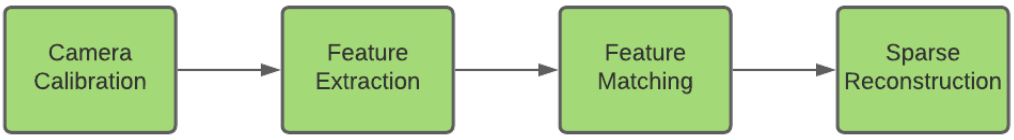

Figure 2.1: Simplified SfM Pipeline

We shall describe the basic ideas of each stage of the pipeline, as relevant to the localization problem.

2.1.1 Camera Calibration

Consider the case of projecting a point in 3D to a the image 2D plane, outlined in the figure below. The 2D point $\mathbf{x}$ can be described by the ray $\lambda[u\ v\ 1]^T$ in projective space.

The 3D point $\mathbf{x}$ can be projected onto the image plane by a projection matrix $\mathbf{P}$, thus:

$$\mathbf{x} = \mathbf{PX} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X}$$

Note the decomposition of the projection matrix $\mathbf{P}$ as $\mathbf{K}[\mathbf{R}|\mathbf{t})$, wherein $\mathbf{K}$ is called the *intrinsic matrix*. $[\mathbf{R}|\mathbf{t}]$ is a concatenation of the rotation and translation of the camera w.r.t some world coordinate system, called the *extrinsic matrix*.

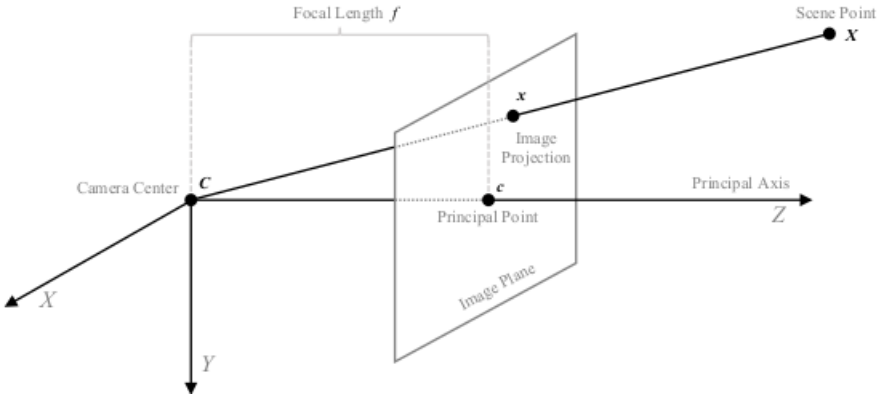

Figure 2.2: Pinhole Camera - taken from Schönberger, 2018

We model $\mathbf{K}$ as follows:

$$\mathbf{K} = \begin{bmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

f is the focal length, and c_x, c_y is the principal point. s is the shear angle between the axes, and a is the aspect ratio. Multiplying a homogenous coordinate in 3d-space will lead to a shift by the principal point, and scaling by f upon perspective division.

The aim of Camera Calibration is to recover the intrinsics and/or extrinsics. If we denote the 3D points as $\mathbf{X}$, and the corresponding 2D points as $\mathbf{x}$, we can formulate the following set of equations:

$$\mathbf{x} = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \mathbf{P} = \begin{bmatrix} \mathbf{P_1^T} \\ \mathbf{P_2^T} \\ \mathbf{P_3^T} \end{bmatrix}$$

where u, v are the coordinates of the 2D point, and $\mathbf{P_i^T}$ are transposes of the columns of $\mathbf{P}$.

$$\mathbf{x} = \mathbf{PX}$$

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{P_1^T} \\ \mathbf{P_2^T} \\ \mathbf{P_3^T} \end{bmatrix} \mathbf{X}$$

solve for λ, and we can write the above as:

$$\mathbf{P_3^T X} u = \mathbf{P_1^T X}$$
$$\mathbf{P_3^T X} v = \mathbf{P_2^T X}$$

Writing this system of equations as a postmultiplication of the $\mathbf{P_i^T}$'s gives us,

$$\mathbf{0} = \underbrace{\begin{bmatrix} \mathbf{X^T} & 0 & -\mathbf{X^T} u \\ 0 & \mathbf{X^T} & -\mathbf{X^T} v \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} \mathbf{P_1} \\ \mathbf{P_2} \\ \mathbf{P_3} \end{bmatrix}$$

If we solve the above equation for more than 6 points, we get more than 12 constraints (each equation contributes 2 constraints). The decomposition to intrinsic and extrinsic components can be acquired by an RQ decomposition, as the rotation $\mathbf{R}$ is an orthonormal matrix.

Distortions due to actual lens shapes can be modeled as a simple radial distance-based displacement of the 2d points, as follows:

$$\bar{\mathbf{x}} = [u\ v]^T (1 + k_1 r^2 + k_2 r^2 + ...)$$

For accurate SfM, it is crucial to get a good estimation of the camera projection matrix.

2.1.2 Feature Extraction

Features are descriptions of special locations of the image that are invariant under transformations of the image, spatial or otherwise.

Features can either be learned, or handcrafted. We outline the fundamentals of handcrafted features, as only these are potentially relevant in our pipeline to construct pointclouds.

Keypoint Extraction

We first have to identify interesting points in the image before we can invariantly describe them. One standard keypoint detector is the Harris detector, upon which most other handcrafted detectors are based. We outline below the central ideas of the Harris detector.

Without loss of generality, assume that a grayscale image I is used. For a small shift of $[\Delta x\ \Delta y]^T$ of some pixel located at $[x\ y]^T$ in a window W of the image I:

$$f(\Delta x, \Delta y) = \sum_{(x_k, y_k) \in W} (I(x_k, y_k) - I(x_k + \Delta x, y_k + \Delta y))^2$$

is the squared error sum obtained by shifting the window by $[\Delta x\ \Delta y]^T$.

We can approximate $I(x + \Delta x, y + \Delta y)$ by a Taylor expansion and keep only the first order terms. This leads us to the following expression for f:

$$f(\Delta x, \Delta y) = \sum_{(x,y) \in W} (I_x(x,y)\Delta x + I_y(x,y)\Delta y)^2$$

Expanding the squares, and rewriting as a quadratic matrix multiplication leads us to:

$$f(\Delta x, \Delta y) \approx [\Delta x \ \Delta y] \underbrace{\sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}}_{\mathrm{M}} [\Delta x \ \Delta y]^T$$

Here, M is called the *structure tensor*. We use the structure tensor to calculate the *Harris Response* as follows:

$$R = det(M) \ - \ k.trace(M)$$

We choose as keypoint only those locations $[x \ y]$ for which R is high. At these locations, both eigenvalues are high. This means that the error increases uniformly in all directions, and therefore the point is most likely a corner.

Descriptor extraction

There are several handcrafted feature descriptors which describe points of interest robustly, but SIFT (Shift Invariant Feature Transform) is the foundational one and the most widely used. The SIFT detector+descriptor works basically in the following stages:

SIFT Detector:

- Create scale pyramid of the image using a Gaussian kernel.
- Create a DoG (Difference of Gaussians) pyramid by taking the difference between successive scales.
- Detect extremal points by comparing to 8 neighbours in current, above and below scales.
- Filter out locations from previous step further with a Harris-like procedure described in the previous section.
- Assign as orientation of the point the dominant orientation of its gradient.

SIFT Descriptor: We know the scale, location and orientation of each keypoint.

- Around each keypoint, consider a 16x16 window. Subdivide this into 16 4x4 blocks.
- For each 4x4 block, create an 8-bin orientation histogram.
- Concatenate the 16 8-bin histograms to get a 128-dimensional vector. This is our final SIFT descriptor.

2.1.3 Feature Matching

Given descriptors for interest points for a pair of images (obtained perhaps with procedures like the ones outlined above), we seek to obtain corresponding points. This procedure is often done in a brute force manner for small images (each descriptor is compared to every other descriptor). However, this method is not feasible for images with a large number of features.

The standard solution to the above problem is to utilize tree methods, such as K-d trees, to speed up the search for a matching descriptor candidate. A popular method is the FLANN (Fast Approximate Nearest Neighbour search algorithm).

A pair of descriptors are compared using the L1 distance (for descriptors such as SIFT, SURF, etc) or the Hamming distance (for binary descriptors such as BRIEF).

The output of this stage in the pipeline is the set of corresponding points of the image pair.

2.1.4 Sparse Reconstruction

Triangulation

Given a set of corresponding points in a two-view setting, we can check for the location of the 3D point that projects on to these correspondences by intersecting rays from these correspondences. This idea is the basis of *triangulating* for the location of the 3d point.

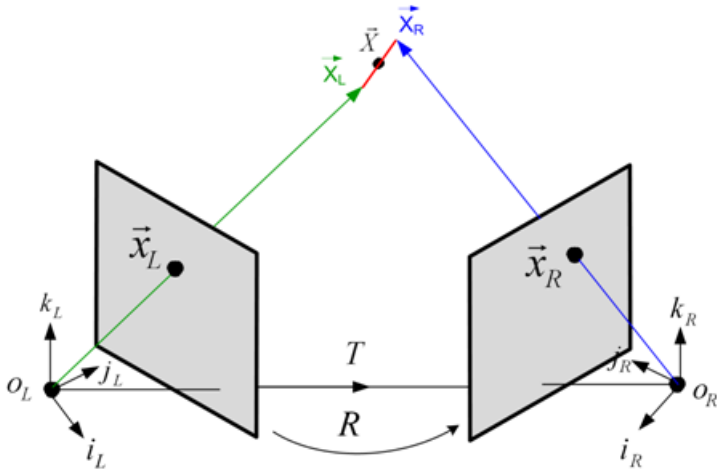

Figure 2.3: Triangulation in SfM, taken from Schönberger, 2018

The 3D point $\mathbf{X}$ projects on to the left image as $\mathbf{x_L} = \mathbf{P_L X}$ and onto the right image as $\mathbf{x_R} = \mathbf{P_R X}$, where $\mathbf{P_L}$ and $\mathbf{P_R}$ are the poses of the left and right images respectively. Since the camera poses and the 2D points are known, we can solve for the 3D point X in the above equations.

Refinement of reconstruction

The 3D points so constructed could have inaccuracies due to measurement errors and drift. A procedure called bundle adjustment is proposed to refine the triangulated points. Its objective function is:

$$\min \mathbf{a_j}, \mathbf{b_i} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij} d(\mathbf{Q}(\mathbf{a_j}, \mathbf{b_i}), \mathbf{x_{ij}})^2$$

Camera i is parameterized by $\mathbf{a_i}$, and $\mathbf{b_j}$ is the j^{th} 3D point.$\mathbf{Q}(\mathbf{a_j}, \mathbf{b_i})$ is the predicted position of point i on image j, and $\mathbf{d}$ is the euclidean distance.

We can see that Bundle Adjustment minimizes overall reprojection error of each 3D point, and is optimized by the Levenberg-Marquardt algorithm.

2.2 The Basics of Convolutional Neural Networks (CNNs)

2.2.1 The Fundamentals of Deep Neural Networks

Neural networks are essentially a composition of alternating linear and nonlinear functions, nested to an arbitrary depth. These nonlinearities are usually fixed, and are called *activation functions* - eg. Logistic Sigmoid, tanh, Rectified Linear Unit (ReLU).

Logistic Sigmoid	$g(x) = \frac{1}{1+e^{-x}}$
tanh	$g(x) = \frac{e^{2x}-1}{e^{2x}+1}$
ReLU	$g(x) = max(0, x)$

The nesting level is called *layer* in neural network parlance.

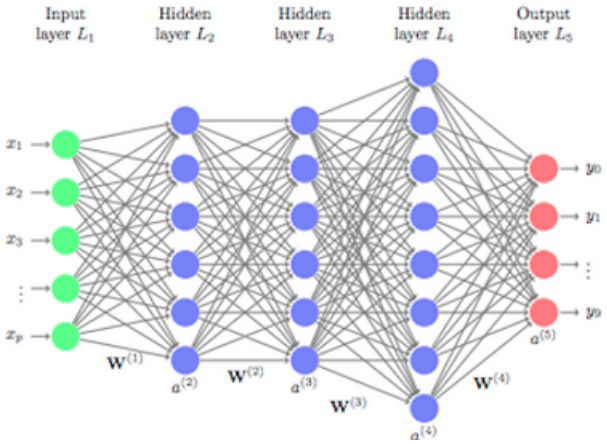

Figure 2.4: Typical neural network architecture, taken from the DeepAI website

The output at layer $l + 1$ can be expressed as a function of the previous layer's output as follows:

$$a^{(l+1)} = g(\underbrace{W^{(l+1)}a^{(l)} + b^{(l+1)}}_{z^{(l)}})$$

This deeply composed function has enough degrees of freedom to theoretically fit any target function. The fitting procedure is performed by varying the weight matrix of all the layers ($W_{(l)}$). Gradient descent is the algorithm of choice to arrive at the optimal weights.

Gradient descent relies on varying the weights in the direction of the gradient of the loss function w.r.t the weights - this is the direction of steepest descent. However, to apply this procedure, we must first obtain the gradient of the loss function w.r.t the weights. This is obtained by a procedure christened *backpropagation*.

1. **Input** x**:** Set the corresponding activation a^1 for the input layer.
2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. **Output error** δ^L**:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

Figure 2.5: The Backpropagation Algorithm, taken from http://neuralnetworksanddeeplearning.com

Backpropagation is actually a sequential application of the *chain rule*. If our error function is L, then backpropagating to find the weights of the first layer W_0 would function as follows:

$$\frac{\partial E}{\partial W^{(0)}} = \underbrace{\underbrace{\frac{\partial E}{\partial y^{(l)}}\frac{\partial y^{(l)}}{\partial z^{(l)}}}_{\delta^{(l)}}\frac{\partial z^{(l)}}{\partial y^{(l-1)}}\frac{\partial y^{(l-1)}}{\partial z^{(l-1)}}\cdots}_{\delta^{(l-1)}}\frac{\partial y^{(1)}}{\partial W_{(0)}}$$

The above architecture can technically learn to fit any function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$.

2.2.2 Convolutional Neural Networks

The standard feedforward neural network works only to fit to vector valued functions. Input of any spatial structure would have to first be unrolled into a 1D vector before it being used in this network. However, if we wanted to preserve spatial relationships during the learning procedure (eg. with images), we would need to formulate a new architecture.

Convolutional Neural Networks are the defacto method to address fitting functions to 2D data where spatial relationships are crucial. Thus, images are the most obvious and abundant sources of data for the CNN. CNNs have been applied successfully to several computer vision tasks such as segmentation, classification, object recognition, etc. and near or even superhuman performance has been achieved in these tasks.

We begin by defining the application of the square shaped convolution operator H on image I of dimensions M X N as follows.

$$Conv(y,x) = \sum_{m=0}^{M} \sum_{n=0}^{N} H(m,n)I(y-m, x-n)$$

We can see that the convolution operation is a sum of the image intensities within the window, weighted by the values of the convolution kernel. For accesses outside the image area, padding (by reflection, zeros, etc.) is performed.

CNNs are composed of three different layers:

- The *Convolution Layer*, where the output is a set of images (called feature maps), that are the results of convolution of the input layer by a set of kernels of a certain dimension.

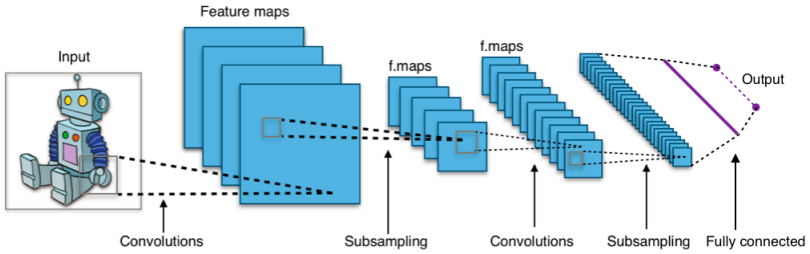

Figure 2.6: The Typical CNN, taken from Wikipedia

- The *Pooling/Subsampling Layer*, where a cluster of points around a each point is considered. The average or maximum of this cluster is taken as the output, thereby reducing the dimensionality of the feature maps.

- The *Fully Connected Layer* flattens the 2D feature maps into 1D vector outputs, which can then be used for conventional tasks such as classification or regression.

2.2.3 Semantic Segmentation

Semantic Segmentation is the task of predicting a class label for each pixel of a given image. The seminal work is the Fully Convolutional Network (FCN), which uses the features of a classification network (experiments were performed on GoogLeNet, VGG and the AlexNet) to predict a pixel-wise semantic mask for the image.

It is worth briefly looking into the details of GoogLeNet, VGG and AlexNet which are the basic classificaion networks and then proceed on to the usage of their features to predict the semantic mask via the CNN.

AlexNet

AlexNet is of historical significance to CNN architectures, as it was the first network to win the ImageNet classification challenege and was the

first to utilize the GPU to speedup its computations. It also introduced the architectural parlance that is now today's terminology of CNNs.

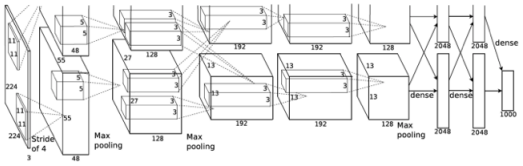

Figure 2.7: The Architecture of AlexNet, taken from Krizhevsky, Sutskever, and Hinton, 2012

AlexNet is composed of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer. Usually, the ReLU activation is used as the preferred nonlinearity. It addresses the problem of overfitting to data by using the principles of Dropout and data augmentation.

VGG

VGG is an AlexNet derivative, but made with the following improvements.

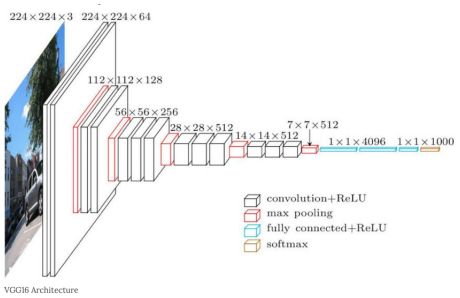

Figure 2.8: The Architecture of VGG, atken from Simonyan and Andrew Zisserman, 2015

VGG, has several differences that separates it from similar models. Unlike AlexNet that uses large receptive fields, VGG uses small receptive fields (3x3 kernel with stride 1). The addition of 3 ReLU units makes the decision function more discriminative. There are also fewer parameters. VGG uses

1x1 convolutions to make the decision function behave in a more non-linear manner without changing the receptive fields. The smaller convolution filters allows VGG to have a larger number of weight layers, which leads to improved performance. This, however, is a feature shared by GoogLeNet, which is delineated in the next section.

GoogLeNet

The main idea behind the GoogleNet architecture was the use of the *Inception* module, which created features of different receptive fields and aggregated them. Note the addition of the 1x1 convolutions from the previous layer before convolution with the set of kernels, just to reduce dimensionality of the incoming feature volume.

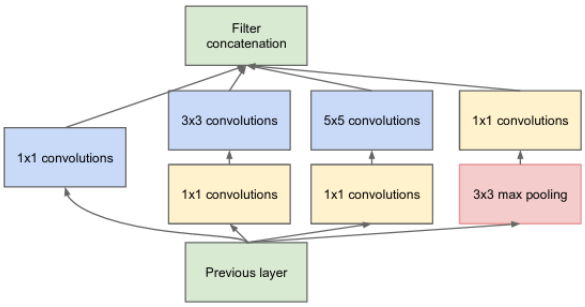

Figure 2.9: Inception Module, taken from Szegedy et al., 2015

The GoogleNet has a depth of 22 layers, with 27 pooling layers. It consists of 9 inception modules stacked linearly in total, where the ends of the inception module is connected to a global average pooling layer. The intuition is that we learn features of varying receptive fields, each of which "zoom-out" with a stacking of an inception module. The below figure depicts pictorially the architecture of the GoogLeNet:

Fully Convolutional Networks for Semantic Segmentation

Fully Convolutional Neural Networks (FCNs) by Long, Shelhamer, and Darrell, 2015 in the use of CNNs for Semantic Segmentation.

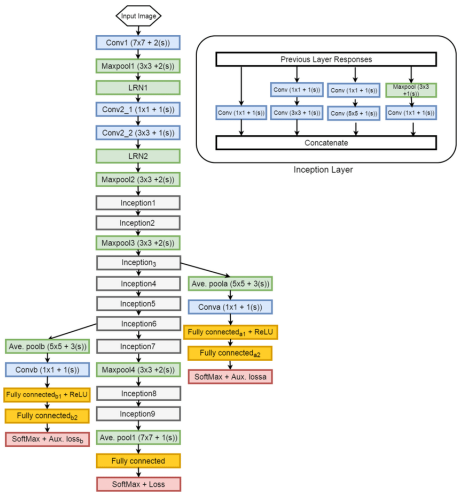

Figure 2.10: GoogLeNet Architecture, taken from Karri, Chakraborty, and Chatterjee, 2017

FCNs start with a backbone classification network (such as AlexNet, VGG or GoogLeNet which we have described above), but with several adaptations which we will describe now.

In classification, conventionally, an input image is downsized and goes through the convolution layers and fully connected (FC) layers, and output one predicted label for the input image, as follows:

Conventional classification networks downsize the input image before sending them through the convolution and fully connected (FC) layers before predicting one label for the image, as shown in the below:

Figure 2.11: Conventional classification networks, taken from Long, Shelhamer, and Darrell, 2015

If we turn the FC layers into 1x1 convolutional layers and the image is not downsized, the output will not be a single label. The output is blown back to the input image resolution by upsampling (bilinear interpolation).

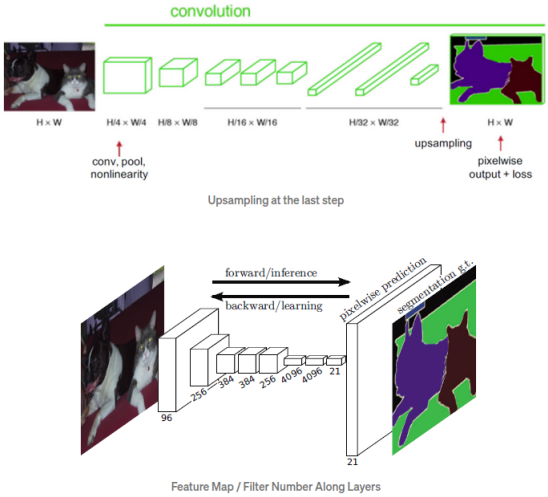

Figure 2.12: Fully Convolutional Layer, taken from Long, Shelhamer, and Darrell, 2015

Note that we can lose a lot of fine features while going through the pooling operations of the network, so the earlier features containing finer information are added onto the later stages as shown below:

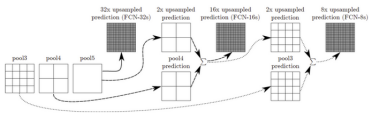

Figure 2.13: FCN structure, taken from Long, Shelhamer, and Darrell, 2015

Depending on the stage at which we upsample and fuse features, we get FCN-32, FCN-16 and FCN-8's. Obviously, FCN-8's produce the best performance as it gets closest to the input resolution and incorporates fine and coarse feature fusions at several scales.

FCNs were built on by several authors incorporating more refinement stages (Badrinarayanan, Alex Kendall, and Roberto Cipolla, 2016), efficient non-linear upsampling schemes (Liu, Rabinovich, and Berg, 2015), adding global context () Zhao et al., 2017) and pyramid pooling for context aggregation. Yu and Koltun, 2016 proposed a context module the uses dilated convolutions to enlarge the receptive field. Chen et al., 2017 proposed using multiple

parallel dilated convolutions with different sampling rates for multi-scale learning in addition to using CRFs for post-processing. A. Valada et al., 2017 introduced multi-scale residual blocks with dilated convolutions as parallel convolutions to enable faster inference without compromising the performance.

3 Related Work

3.1 Localization against a 3D Pointcloud

Traditionally, the task of Visual Localization is performed using the components of SfM. Given a query image whose pose we are required to find in some 3D reconstruction, the basic sequence of steps followed are:

- *Extract descriptors* for the query image.
- *Match* 2D query image keypoints to 3D Reconstruction points.
- *Refine pose* of the query image using the 2D-3D matches of the previous step, and the PnP algorithm.

Descriptor extraction has been detailed in the previous section. We will describe briefly the proceeding two steps.

3.1.1 2D-3D Matching

Once we have the features of the query image, we can associate them with corresponding image features in the reconstruction. Each image feature has also an association with some 3D point in the reconstruction, and by transitivity we can assert an association between the query keypoints and a 3D point of the reconstruction.

A tree structure like the k-d tree is used to hash the reconstruction's features of the reconstruction for rapid matching of the query image features against large reconstructions.

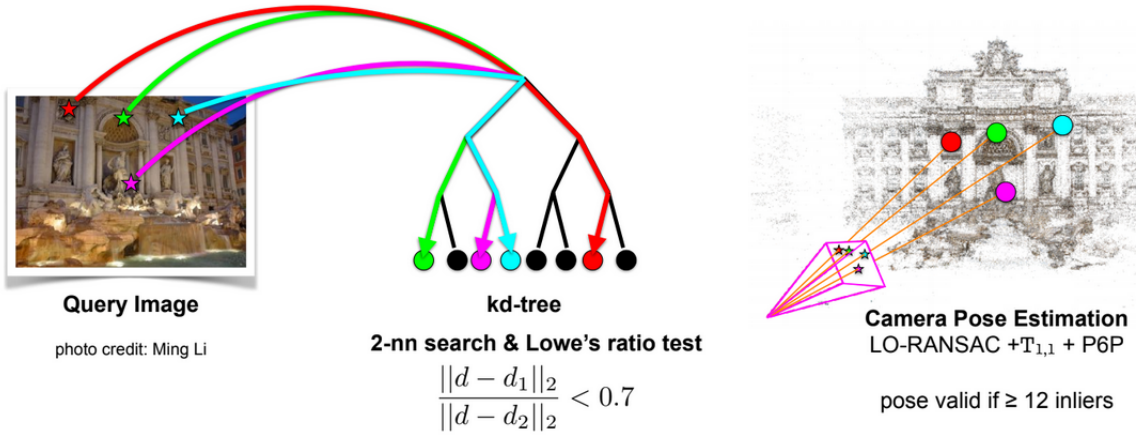

Figure 3.1: Feature Based Localization, taken from Torsten Sattler's ECCV'18 slides

3.1.2 Pose Refinement with RANSAC + Perspective-n-Pose

Now that we know which 3D points are visible from the query image, we can try to figure out an affine transform of the 3D points that yields a minimum reprojection error when projected on the query image.

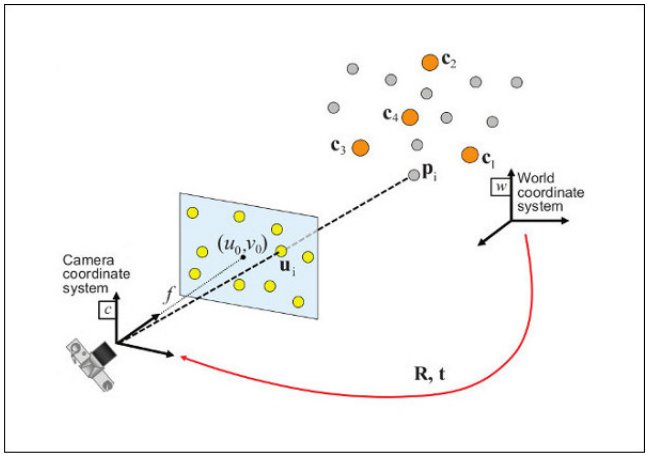

Figure 3.2: Perspective-n-Pose Algorithm, taken from OpenCV Tutorials

The reprojection error objective to be minimized is:

$$argmin_{R,t} \sum_{i=1}^{N} \|\hat{x}_i - x_i\|^2$$

The projection from 3D point to 2D point i is:

$$\mathbf{x_i} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X_i}$$

The minimization is usually performed with the Levenberg-Marquardt algorithm.

To obtain a robust estimate of our pose, we sample sets of points and choose that pose which has maximum overall inliers (minimum overall reprojection error). This is the basic idea of the RANSAC procedure.

3.2 Localization with Global Image Descriptors

The basic idea of image based localization is to localize a query image by using strategies to find images (of known pose) in a database that are closest to it, and then compute a pose with respect to these set of "close" images. The method of comparison employed between a pair of images here is the distance between their global image descriptors.While localizing against a huge pointcloud, the global descriptors can be used as a preliminary step in identifying a set of candidate images to form 2D-3D associations.

This reduces the search space considerably for matching query image features. The paper by Torsten Sattler, Leibe, and Kobbelt, 2012 is typical of these class of techniques.

3.2.1 Visual Vocabulary Trees

Nister and Stewenius, 2006 is the seminal publication that exhibited early the possible use of Vocabulary trees to search visual cues. Intuitively speaking,

visual words are a histogram over frequently occuring image content.This content is usually characterized by descriptors such as SIFT, etc. SIFT is the standard handcrafted descriptor for visual words, as they are highly distinctive. The general procedure to build tese visual words is detailed below:

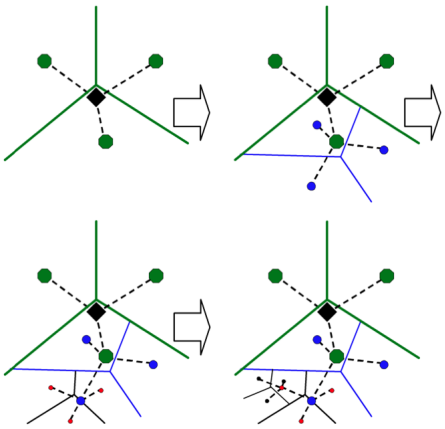

Figure 3.3: Construction of the visual words with hierarchical k-means tree, taken from Nister and Stewenius, 2006

- For a large, diverse database of images, aggregate all the SIFT descriptors from each image. This will be our training data.
- Run an initial k-means clustering on the training data and get the k cluster centers.
- The training dataset is then partitioned into k sets, where each set contains a cluster center and a set of descriptor vectors closest to it.
- Apply the previous two steps to each of the k sets, a predetermined number of steps. We now have a hierarchical k-means tree.

In the online phase, when we need to compute a visual word descriptor for an image, all we need to do is take every (SIFT) descriptor of the image and compute which of the k clusters of our vocabulary tree is closest to the word.

This "closeness" is computed by propagating the descriptor vector down the tree for each level-1 node, yielding a score for each node by computing inner products with each descriptor in the tree.

After doing this scoring for all the descriptors in the image and computing a histogram, we have our visual words description for the image.

3.2.2 VLAD

The VLAD descriptor, by R. Arandjelovic and A. Zisserman, 2013, serves as a global image descriptor. It also uses a vocabulary tree, and is computed as follows:

- Compute a visual vocabulary tree as described previously, for some database of images.
- Extract regions with an affine invariant detector.
- Describe regions with the 128 dimensional SIFT descriptor.
- Assign each descriptor to one of the k clusters of the vocabulary tree.
- Compute and accumulate residuals (difference between cluster center and assigned descriptors) for each of the k cluster centers. Formally, this can be defined as:

$$V(i,j) = \sum_{i=0}^{N} a_k((x)_i)(x_i(j) - c_k(j))$$

  where $x_i(j)$ and $c_k(j)$ are the i-th descriptor and k-th cluster center respectively. $a_k((x)_i)$ denotes the membership of the descriptor in cluster center k.

- Concatenate the final residuals from the k cluster centers into a $kx128$ dimensional dimensional descriptor, referred to as unnormalized VLAD.

The VLAD descriptor can then be normalized with an L2 function, a signed squared root, or the intra-normalization (wherein the VLAD is normalized within each cluster before concatenation, before L2 normalization).

3.2.3 NetVLAD

NetVLAD, by Relja Arandjelovic et al., 2015, seeks to incorporate the VLAD idea into a CNN, which can potentially learn much better features than the handcrafted SIFT descriptor. However, the obstacle to including VLAD as a layer would be the hard assignment of a descriptor to one of the clusters , i.e. $a_k(\mathbf{x})$. This is not differentiable. To get around this problem, we change the hard assignment to a softer assignment as follows:

$$a_k(\mathbf{x_i}) = \frac{e^{-\alpha\|\mathbf{x_i}-\mathbf{c_k}\|}}{\sum_{k'} e^{-\alpha\|\mathbf{x_i}-\mathbf{c'_k}\|}}$$

which assigns descriptor $\mathbf{x_i}$ to to cluster $\mathbf{c_k}$ depending on proximity. Expanding squares and cancelling terms, we can rewrite the assignment function as:

$$a_k(\mathbf{x_i}) = \frac{e^{\mathbf{w_k^T x_k}+b_k}}{\sum_{k'} e^{\mathbf{w_k'^T x'_k}+b'_k}}$$

where $\mathbf{w_k} = w\alpha\mathbf{c_k}$ and $b_k = -\alpha\|\mathbf{c_k}\|^2$. Implementation-wise, this soft-assignment function can be broken up as a convolution for the linear part, and then a softmax for the normalization. Overall, the implementation can be summed up pictorially as:

3.3 Localization by Regressing Pose with CNNs

CNNs trained on image-pose pairs of a scene have enabled the of poses directly from a query image of that scene. The seminal work on this type of localization is PoseNet.

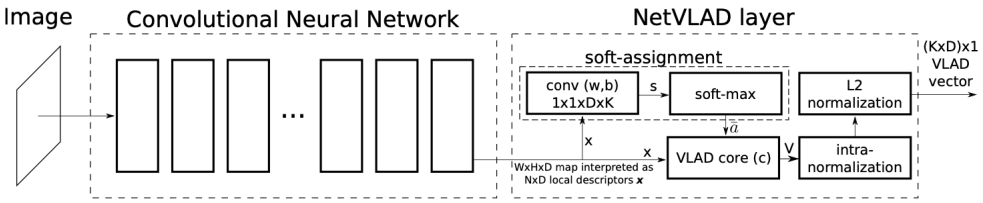

Figure 3.4: The NetVLAD network architecture, taken from Relja Arandjelovic et al., 2015

3.3.1 PoseNet

PoseNet takes as input a 224x224 RGB image, and regresses the 6-DoF pose of that image relative to the scene it was trained on. The output of the PoseNet is a 6 vector $[\mathbf{x}, \mathbf{q}]$, where $\mathbf{x}$ is the translation and $\mathbf{q}$ is a unit quaternion parameterizing rotation.

PoseNet Loss

The loss function for the CNN of PoseNet is defined as follows:

$$loss(I) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2 + \beta \|\mathbf{q} - \frac{\mathbf{q}}{\|\mathbf{q}\|}\|_2$$

Note that the β parameter balances the relative importance of translation and rotation loss.

PoseNet Architecture

PoseNet, by A. Kendall, Grimes, and R. Cipolla, 2015 utilizes as backbone the GoogleNet Architecture. GoogleNet is a 22 layer deep CNN which was state of the art at the time for classification.

However, the following modifications were made by the authors of PoseNet to convert the GoogLeNet backbone into a pose regressor network.

- The 3 softmax classifiers were replaced with an affine regressor (fully connected layers) which output a 7-dimension vector representing pose.
- Another fully connected layer of size 2048 was added before the final layer, so it could be used as a feature vector for localization.
- Normalize quaternion orientation vector to unit length at test time.

PoseNet was trained on image, pose pairs obtained from SfM and was able to get competitive results on standard datasets such as Cambridge and 7Scenes. It spawned several variants for visual localization that used its base architecture as a backbone, and we shall explain how we use one such variant (Mapnet) in our approach to solve the localization problem, as detailed in the next chapter.

3.3.2 VLocNet

An improvement upon PoseNet was VLocNet by Abhinav Valada, Radwan, and Burgard, 2018. In order to estimate the global pose of a pair of query frames accurately with respect to some scene, cues from Visual Odometry (relative motion between frames) are used jointly with the global pose of the frames. The idea is that the relative motion estimates from VO would help constrict the search space considerably.

Given a pair of frames (I_t, I_{t-1}), the network aims to regress the absolute pose $\mathbf{p_t} = (\mathbf{x_t}, \mathbf{q_t})$ and the relative pose $\mathbf{p_{t,t-1}} = (\mathbf{x_{t,t-1}}, \mathbf{q_{t,t-1}})$ where $\mathbf{x}$ is the translation 3-vector, and $\mathbf{q}$ is the rotation 4-vector. The semantic branch predicts M_t, the pixelwise semantic mask of C classes for image I_t

Architecture

The backbone network for the VLocNet shown above is a modified ResNet-50 network (He et al., 2015).

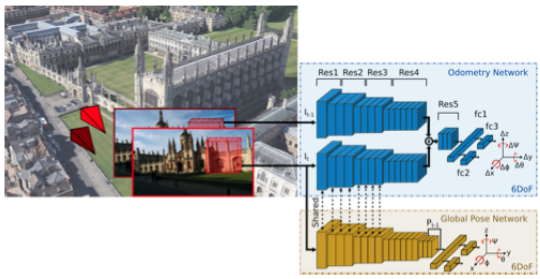

Figure 3.5: VLocNet architecture, taken from Abhinav Valada, Radwan, and Burgard, 2018

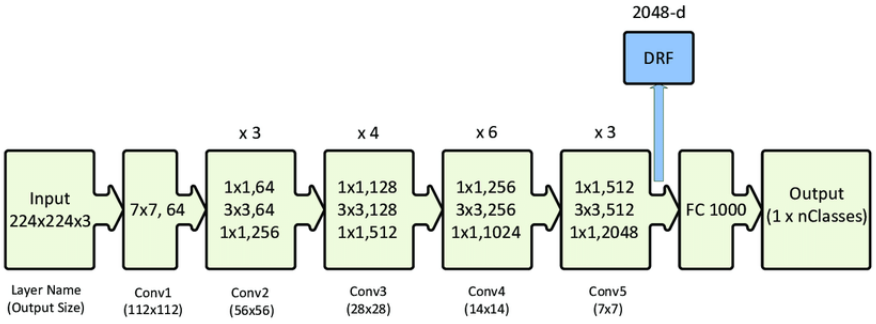

Figure 3.6: ResNet50 architecture, taken from Mahmood et al., 2020

However, the ResNet50 is modified as follows:

- Replace conventional ReLUs with ELU activation functions, which have been found to be more robust to noise and are faster to converge.

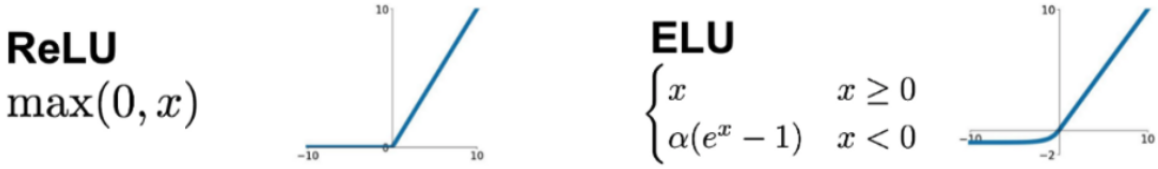

Figure 3.7: The ReLU and the ELU activation functions, taken from https://medium.com/hyunjulie/activation-functions-a-short-summary-8450c1b1d426

- After the fifth layer, add a global average pooling layer. Add three inner product layers (*fc1*, *fc2*, *fc3*) after, of dimensions 1024, 3, and 4. *fc2* and *fc3* regress the translation and rotation respectively.

Instead of directly regressing the pose from the network, an following extra step is introduced: the fusion of the previous timestep's final downsampling layer output with that of the current timestep's final downsampling layer output. This happens for pairs of frames in a sequence, and forces the network to learn motion cues in the sequence.

This scheme in combination with the *Geometric consistency loss* described in the following section enables the network to learn motion-specific cues in the temporal dimension.

Learning Pose Regression

The usual euclidean loss between predicted and ground truth pose is augmented here with an additional term to constrain the error between the relative motion predicted by the odometry stream of the network, and that available from the ground truth. If we denote our neural network as f, and its parameters as θ, we can build up to the final loss function as follows:

The translational and rotational losses between two consecutive frames are defined as:

$$\mathcal{L}_{xRel}(f(\theta|I_t)) = \|\mathbf{x}_{t,t-1} - (\hat{\mathbf{x}}_t - \hat{\mathbf{x}}_{t-1})\|$$
$$\mathcal{L}_{qRel}(f(\theta|I_t)) = \|\mathbf{q}_{t,t-1} - (\hat{\mathbf{q}}_t - \hat{\mathbf{q}}_{t-1})\|$$

These two losses are each weighted exponentially with a learnable exponent, and added together to form the relative loss.

$$\mathcal{L}_{Rel}(f(\theta|I_t)) = \mathcal{L}_{xRel}(f(\theta|I_t))exp(-\hat{s}_{x_{Rel}}) + \hat{s}_{x_{Rel}} + \mathcal{L}_{qRel}(f(\theta|I_t))exp(-\hat{s}_{q_{Rel}}) + \hat{s}_{q_{Rel}}$$

The absolute loss between pose prediction and ground truth has a similar structure:

$$\mathcal{L}_x(f(\theta|I_t)) = \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|$$
$$\mathcal{L}_q(f(\theta|I_t)) = \|\mathbf{q}_t - \hat{\mathbf{q}}_t\|$$

The individual euclidean translation and rotation losses are put together in a cumulative absolute pose loss terms as follows:

$$\mathcal{L}_{Euc}(f(\theta|I_t)) = \mathcal{L}_x(f(\theta|I_t))exp(-\hat{s}_x) + \hat{s}_x + \mathcal{L}_q(f(\theta|I_t))exp(-\hat{s}_q) + \hat{s}_q$$

We put both relative and absolute losses together to get the following loss term:

$$\mathcal{L}_{Loc}(f(\theta|I_t)) = \mathcal{L}_{Rel}(f(\theta|I_t)) + \mathcal{L}_{Euc}(f(\theta|I_t))$$

Learning Visual Odometry

Another branch of the overall network learns to predict visual odometry $\mathbf{p}_{t,t-1} = (\mathbf{x}_{t,t-1}, \mathbf{q}_{t,t-1})$, given a pair of frames in a sequence (I_t, I_{t-1}). This is done by employing a dual stream architecture, wherein each branch is identical to one another, and is based on the ResNet-50 architecture.

The feature maps before the last downsampling stage of both stages are concatenated, convolved through the last residual block, followed by an inner product layer and two pose regressors for estimating pose of both

frames. This loss between predicted and ground truth motion is expressed by the following loss function:

$$\mathcal{L}_{vo}(f(\theta|(I_t, I_{t-1}))) = \mathcal{L}_x(f(\theta|(I_t, I_{t-1})))exp(-\hat{s}_{x_{vo}}) + \hat{s}_{x_{vo}} + \mathcal{L}_q(f(\theta|(I_t, I_{t-1})))exp(-\hat{s}_{q_{vo}}) +$$

The parameters between the odometry network here, and the global pose regression network are shared. This sharing enables an inductive transfer of information between both networks.

Learning Semantics

The authors propose two variants of a semantic learning branch: a single task architecture that predicts a pixel-wise segmentation mask for a monocular image, and a multitask architecture that incorporates self-supervised warping and adaptive fusion layers in the segmentation process.

For the *single-task architecture*, an encoder-decoder model is used. The encoder is a ResNet-50 architecture, which learns highly discriminative semantic features 16 times downsampled at the output layer. The decoder consists of two deconvolution layers, a skip convolution from the encoder for fusing high-resolution semantic maps and upsampling to the input feature resolution. At the output of the network, we get classification scores per pixel. We get the probability of assigning a class to a pixel given an image, by applying softmax to the per pixel classfication scores. The loss function is therefore defined as the maximum log-sum of all points in this distribution.

If we have a set of training images, $\mathcal{T} = (I_n, M_n), n = 1, ...N$, where $I_n = u_r|r = 1, ...\rho$ is the set of training images consisting of ρ pixels, and $M_n = m_r^n|r = 1, ...\rho$ is the set of corresponding ground truth masks with a semantic label per pixel, $m_n^r = 1, ...C$.

Using the per-pixel classification scores s_j we can model the probability of a pixel being assigned a semantic class with the softmax function, as follows:

$$p_j(u_r, \theta | I_n) = \frac{exp(s_j(u_r, \theta)}{\sum_k^C exp(s_k(u_r, \theta)}$$

The optimal network parameters θ is estimated by minimizing the following loss function:

$$\mathcal{L}_\int]\}(\mathcal{T}, \theta) = \sum_{n=1}^{N} \sum_{r=1}^{\rho} \sum_{j=1}^{C} \delta_{m_{n,j}^r} p_j(u_r, \theta | I_n)$$

for $(I_n, M_n) \in \mathcal{T}$.

The *multitask architecture* utilizes a self-supervised warping method, wherein the pose of the previous timestep's image as predicted by the odometry stream is utilized. A depth image of the previous timestep, D_t, is predicted using a separate network Mayer et al., 2016, and the previous image's feature maps (outputs from layers *Res4f* and *Res5c* of the previous timesteps - this is marked in Figurerw3.7) are now warped onto the current image's features.

Formally, if the projection function is π, we can warp on the previous timestep's pixels u_r onto the current timestep $\hat{u_r}$ with relative pose $p_{1,t-1}$

$$\hat{u_r} = \pi(T(p_{1,t-1})\pi^{-1}(u_r, D_t(u_r)))$$

This operation allows robustness to camera angle deviations, object scale and frame distortions. It is also a feature augmenter, and thereby enforces consistent learning of consistent semantics.

Summary

The final loss function is an accumulation of the semantic, odometry and absolute pose losses as defined above. The reasons behind training absolute pose, odometry and semantics can be articulated with two points: to enable the absolute pose regression network to encode geometric an semantic

information while training, and to enable inductive transfer between domain specific information.

This is the first network wherein the results of Localization is consistently equal to or better than the SfM/Feature desriptor approach. For detailed analysis of the results please refer the paper.

3.4 Cross-View Localization

Cross-View Localization is the task of localizing a query image against a database of images captured from a different view than the typical query image. The most usual example is localizing a ground-view image against a database of aerial/satellite imagery.

The seminal work in this area submitted by Workman and Jacobs, 2015, where they fine-tuned AlexNet on ImageNet and Places datasets. They proved the efficacy of CNNs as a feature extractor for cross-view image association by creating a dataset of the Charleston San Francisco aresa as follows: aerial imagery for a predetermined 40 sq.km region was downloaded from Bing maps, and was associated by GPS tagging to ground level images obtained via various open-source image databases such as flickr and Google StreetView. The pre-trained AlexNet was fed these images, and the final layer output is taken as the corresponding feature vector.

Their experiments were able to prove the high discriminative nature of CNNs for this purpose. The following graphs from their paper shows the performance of CNN features against the state of the art method by Lin.et.al (Lin, Belongie, and Hays, 2013) at the time (which sought to learn the relationship between cross-view image pairs with a Support Vector machine based method).

Other methods that utilize CNNs for Cross View localization were then spawned. These myriad methods each utilize a different backbone CNN architecture, and incorporated more information about the image to make the CNN output more distinctive features.

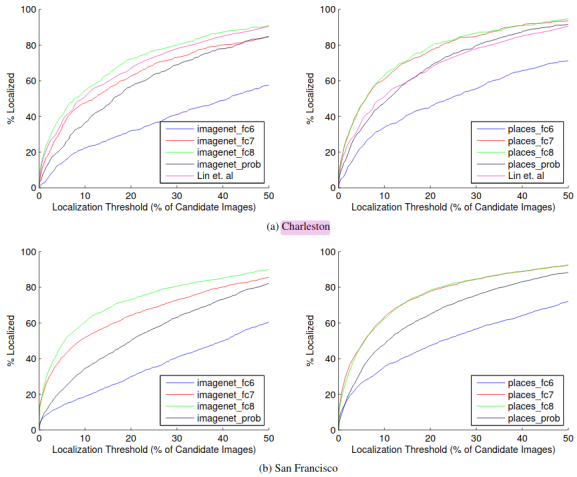

Figure 3.8: AlexNet features (Jacobs et.al) vs SVM learned (Lin et.al) Cross-View Localization, taken from Workman and Jacobs, 2015

Workman et al. submitted that fine- tuning the aerial branch by minimizing the distance between aerial and ground images results in improved localization performance. Vo and Hays (Vo and Hays 2016) conducted thorough evaluations on the network suited best for cross-view localization: i.e. binary classification (image retrieval), Triplet or Siamese CNNs. Hu et.al stacked a NetVLAD layer upon a VGG Network to endow VGG features the viewpoint invariance that NetVLAD possesses.Liu and Li (2019) added per-pixel orientation information into their CNN so that the features learned are sensitive to pixel orientation as another discriminative feature.

We outline here a method that is state of the art in the Cross View Localization domain, and builds upon the idea of using a CNN to learn features.

3.4.1 Optimal Feature Transport for Cross-View Image Geo-Localization

This paper by Shi et al., 2019. deals with cross-view geo localization, the task of finding the location of a given ground-view image in a large satellite map. As we have detailed before, this paper too is based on the premise of using

DCNNs to extract features that can be used for comparing a ground-view image against an aerial image database. There are however two insights that are incorporated by the authors:

- Spatial layout of the features is to be taken into account - i.e. their relative position with respect to other features in the image plane.
- Take into consideration that the ground and aerial images are, for the lack of a better word, of different "domains".

To tackle both of these problems, the direct approach of learning a transport matrix to transfer the ground-domain features to the aerial domain features is utilized.

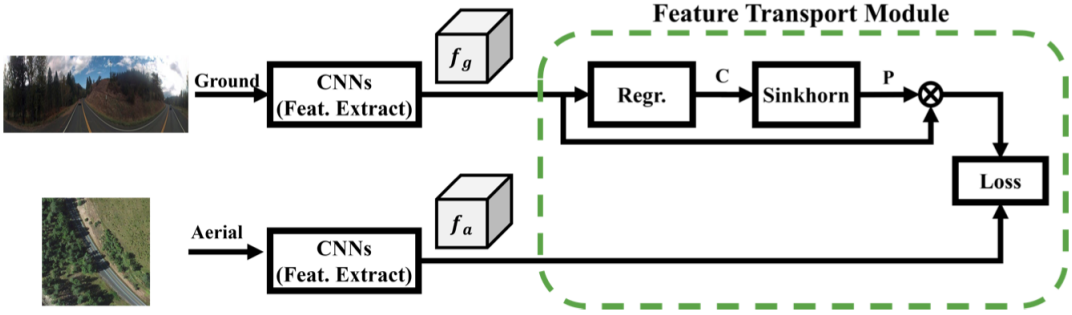

Figure 3.9: The Cross View Feature Transport Module, taken from Shi et al., 2019

Two CNN branches (VGG, Simonyan and Andrew Zisserman, 2015.) are used to learn the aerial and ground features, but the improvisation is in the inclusion of the new feature transport module. Previous approaches are similar upto the usage of two CNN branches to extract features for the aerial and ground images. However they use a feature aggregation such as pooling is used before minimizing the loss between features, which discards spatial layout information.

The feature transport module is actually a set of cost matrices that transform ground to aerial features. This does away with the lossy aggregation operation, and explicitly models the domain change.

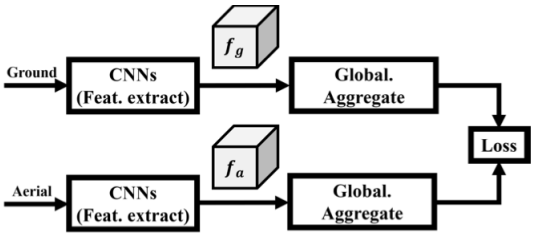

Figure 3.10: Older Cross-View localization architectures, taken from Shi et al., 2019

Feature Transport

$\mathbf{f}^i(g) \in R^{h \times w}$ and $\mathbf{f}^i(a) \in R^{h \times w}$ indicate the ground and aerial feature maps, wherein i is the feature channel number an h and w indicate the width and height of the features.

Ideally, both the ground and aerial images must be used in the computation of the cost matrix $\mathbf{C}$. For purposes of efficiency, we postpone the inclusion of aerial images to the loss function and instead use a regression block to compute $\mathbf{C}$.

Our transport matrix $\mathbf{P}$, has to satisfy the following strictly convex loss condition:

$$\mathbf{P}^* = argmin_P \ < \mathbf{P}, \mathbf{C} >_F - \lambda h(\mathbf{P})$$

where h is the entropy regularization, and the norm is the Frobenius norm. To minimize this particular functional objective, the Sinkhorn algorithm is utilized. The Sinkhorn algorithm first applies an exponential kernel on $\mathbf{C}$ and then iteratively converts $\mathbf{C}$ to a doubly stochastic matrix (rows and columns add up to one). If row and column normalizations are denoted as:

$$\mathbf{C}' = exp(-\lambda C)$$

$$\mathcal{N}_{i,j}^r = \frac{c'_{i,j}}{\sum_{k=1}^{N} c'_{k,j}}$$

$$\mathcal{N}_{i,j}^{c} = \frac{c'_{i,j}}{\sum_{k=1}^{N} c'_{i,k}}$$

For the m-th iteration, the Sinkhorn operation is given as

$$\mathbf{S}_m(\mathbf{C}') = \mathcal{N}^r(\mathcal{N}^c \mathcal{S}_{m-1}(\mathbf{C}'))$$

So when the iterations converge, $\mathbf{P}^* = \mathbf{S}_m(\mathbf{C}')$.

It is to be noted that the Sinkhorn operation is differentiable. Thus, when the ground feature maps are transported using the $\mathbf{P}^*$ and the loss is backpropagated, the domain transfer is also learned.

$$\mathcal{L}_{triplet} = log(1 + e^{\gamma(d_{pos} - d_{neg})})$$

where d_{pos} and d_{neg} denote the l2 distance of all the positive and negative aerial features from the anchor ground feature. The network outperforms the state of the art at the time of its writing (CVM-Net, Hu et al., 2018), as outlined in the graph below:

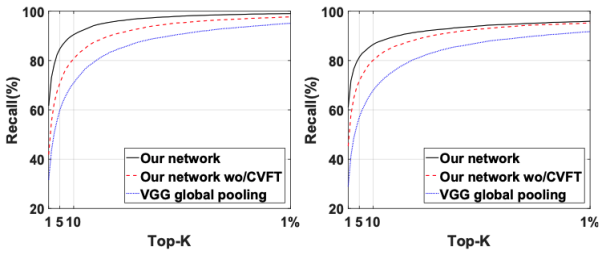

Figure 3.11: Performance of CVFT against CVM-Net on the CVUSA and CVACT datasets respectively, taken from Shi et al., 2019

Appendix

Bibliography

Arandjelovic, R. and A. Zisserman (2013). "All About VLAD." In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1578–1585. DOI: 10.1109/CVPR.2013.207 (cit. on p. 25).

Arandjelovic, Relja et al. (2015). "NetVLAD: CNN architecture for weakly supervised place recognition." In: *CoRR* abs/1511.07247. arXiv: 1511.07247. URL: http://arxiv.org/abs/1511.07247 (cit. on pp. 26, 27).

Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla (2016). *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. arXiv: 1511.00561 [cs.CV] (cit. on p. 19).

Brahmbhatt, Samarth et al. (2017). "MapNet: Geometry-Aware Learning of Maps for Camera Localization." In: vol. abs/1712.03342 (cit. on p. 2).

Chen, Liang-Chieh et al. (2017). *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. arXiv: 1606.00915 [cs.CV] (cit. on p. 19).

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition." In: *CoRR* abs/1512.03385. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (cit. on p. 28).

Hu, Sixing et al. (June 2018). "CVM-Net: Cross-View Matching Network for Image-Based Ground-to-Aerial Geo-Localization." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 38).

Karri, Sri Phani, Debjani Chakraborty, and Jyotirmoy Chatterjee (Feb. 2017). "Transfer learning based classification of optical coherence tomography images with diabetic macular edema and dry age-related macular degeneration." In: *Biomedical Optics Express* 8, p. 579. DOI: 10.1364/BOE.8.000579 (cit. on p. 18).

Kendall, A., M. Grimes, and R. Cipolla (Dec. 2015). "PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization." In: *2015*

IEEE International Conference on Computer Vision (ICCV), pp. 2938–2946. DOI: 10.1109/ICCV.2015.336 (cit. on pp. 2, 27).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., pp. 1097–1105. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. on p. 16).

Lin, T., S. Belongie, and J. Hays (2013). "Cross-View Image Geolocalization." In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 891–898. DOI: 10.1109/CVPR.2013.120 (cit. on p. 34).

Liu, Wei, Andrew Rabinovich, and Alexander C. Berg (2015). *ParseNet: Looking Wider to See Better*. arXiv: 1506.04579 [cs.CV] (cit. on p. 19).

Long, Jonathan, Evan Shelhamer, and Trevor Darrell (2015). *Fully Convolutional Networks for Semantic Segmentation*. arXiv: 1411.4038 [cs.CV] (cit. on pp. 17–19).

Mahmood, Ammar et al. (Jan. 2020). "Automatic Hierarchical Classification of Kelps Using Deep Residual Features." In: *Sensors* 20.2, p. 447. ISSN: 1424-8220. DOI: 10.3390/s20020447. URL: http://dx.doi.org/10.3390/s20020447 (cit. on p. 29).

Mayer, Nikolaus et al. (June 2016). "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 33).

Nister, D. and H. Stewenius (2006). "Scalable Recognition with a Vocabulary Tree." In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2, pp. 2161–2168. DOI: 10.1109/CVPR.2006.264 (cit. on pp. 23, 24).

Sattler, T. et al. (2019). "Understanding the Limitations of CNN-Based Absolute Camera Pose Regression." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3297–3307 (cit. on p. 2).

Sattler, Torsten, Bastian Leibe, and Leif Kobbelt (2012). "Towards Fast Image-Based Localization on a City-Scale." In: *Outdoor and Large-Scale Real-World Scene Analysis*. Ed. by Frank Dellaert et al. Berlin, Heidelberg: Springer Berlin Heidelberg (cit. on p. 23).

Schönberger, Johannes Lutz (2018). "Robust Methods for Accurate and Efficient 3D Modeling from Unstructured Imagery." PhD thesis. ETH Zürich (cit. on pp. 6, 11).

Shi, Yujiao et al. (2019). "Optimal Feature Transport for Cross-View Image Geo-Localization." In: *CoRR* abs/1907.05021. arXiv: 1907.05021. URL: http://arxiv.org/abs/1907.05021 (cit. on pp. 35–38).

Simonyan, K. and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *CoRR* abs/1409.1556 (cit. on pp. 16, 36).

Szegedy, Christian et al. (2015). "Going Deeper with Convolutions." In: *Computer Vision and Pattern Recognition (CVPR)*. URL: http://arxiv.org/abs/1409.4842 (cit. on p. 17).

Valada, A. et al. (2017). "AdapNet: Adaptive semantic segmentation in adverse environmental conditions." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4644–4651. DOI: 10.1109/ICRA.2017.7989540 (cit. on p. 20).

Valada, Abhinav, Noha Radwan, and Wolfram Burgard (2018). "Deep Auxiliary Learning for Visual Localization and Odometry." In: *CoRR* abs/1803.03642. arXiv: 1803.03642. URL: http://arxiv.org/abs/1803.03642 (cit. on pp. 28, 29).

Workman, S. and N. Jacobs (2015). "On the location dependence of convolutional neural network features." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 70–78. DOI: 10.1109/CVPRW.2015.7301385 (cit. on pp. 34, 35).

Yu, Fisher and Vladlen Koltun (2016). *Multi-Scale Context Aggregation by Dilated Convolutions*. arXiv: 1511.07122 [cs.CV] (cit. on p. 19).

Zhao, Hengshuang et al. (2017). *Pyramid Scene Parsing Network*. arXiv: 1612.01105 [cs.CV] (cit. on p. 19).