COMPARISON OF DEEP LEARNING AND FEATURE MATCHING METHODS FOR HOMOGRAPHY ESTIMATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

David K. Niblick

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

December 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Avinash Kak, Chair

School of Electrical Engineering and Computer Engineering

Dr. Charles Bouman

School of Electrical Engineering and Computer Engineering

Dr. Mark Bell

School of Electrical Engineering and Computer Engineering

Approved by:

Dr. Dimitri Peroulis

Head of the School Graduate Program

ACKNOWLEDGMENTS

First and foremost, I want to thank God for blessing me with the opportunity to pursue my goals in life.

I want to thank Professor Avi Kak for bringing me into the Robotics Vision Lab and providing me with indispensable guidance at every step of the process. The insights you provided me made this possible.

To my fellow researchers in the Robotics Vision Lab, I appreciate the assistance and fellowship.

Finally, to my wife, I am eternally grateful for your love and support. Through combat deployments, living away from our family and friends, raising our three children, and now my research, you remain unwavering in your commitment to me. I can never show you how much you mean to me.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

ABBREVIATIONS viii

ABSTRACT ix

1 INTRODUCTION 1

2 LITERATURE REVIEW 4

2.1 Homography Estimation through Feature Matching 4

2.1.1 Direct Linear Transform 4

2.1.2 Key Point Detection and Description 5

2.1.3 Key Point Correspondences 5

2.2 Common Feature Matching Algorithms 6

2.2.1 Scale Invariant Feature Transform 6

2.2.2 Speeded Up Robust Features 11

2.2.3 Oriented FAST and Rotated BRIEF 13

2.3 Deep Learning Fundamentals 14

2.3.1 Fully Connected Layers 15

2.3.2 Convolutional Layers 16

2.3.3 Backpropogation 16

2.3.4 Training Enhancement Techniques 19

2.4 Homography Estimation through Deep Learning 20

2.4.1 Deep Image Homography Estimation 21

2.4.2 Homography Estimation from Image Pairs with Hierarchical Convolutional Networks 25

3 EXPERIMENT 27

Page

3.1 Implementation Details 27
3.2 Evaluation Procedures 28
3.3 Evaluation Metrics 30
4 RESULTS 32
4.1 Ideal Conditions 33
4.2 Noisy Conditions 34
4.3 Illumination Variance Conditions 39
4.4 Occluded Conditions 43
4.5 Additional Experiments 47
5 CONCLUSION 51
5.1 Discussion 51
5.2 Future Work 53
REFERENCES 55

LIST OF TABLES

Table Page

2.1 Digital Approximation of the Laplacian Operator 7

2.2 Discrete Approximation for $\partial^2/\partial x^2$ 11

2.3 Discrete Approximation for $\partial^2/\partial x^2$ 12

3.1 Experiment Summary 29

4.1 Median ACE and OR in Ideal Conditions 33

4.2 Median ACE and OR in Gaussian Noise 35

4.3 Median ACE and OR in Illumination Shift 40

4.4 Median ACE and OR in Occluded Conditions 43

4.5 Comparison of DH Trained in Grayscale to DH Trained in Color 47

4.6 Deep Image Homography Trained With Noise. 48

LIST OF FIGURES

Figure Page

2.1 Difference of Gaussian Pyramid 8
2.2 Layer in Difference of Gaussian Pyramid 8
2.3 SIFT Description Vector 10
2.4 Deep Image Homography Estimation Model 21
2.5 Hierarchical Homography Model 23
2.6 Hierarchical Homography Stack 24
4.1 Sorted ACE for Unaltered Dataset 34
4.2 Sorted ACE for Noise $\eta = 0.1$ 36
4.3 Sorted ACE for Noise $\eta = 0.3$ 37
4.4 Sorted ACE for Noise $\eta = 0.5$ 38
4.5 Sorted ACE for Illumination $\lambda = 1.2$ 40
4.6 Sorted ACE for Illumination $\lambda = 1.4$ 41
4.7 Sorted ACE for Illumination $\lambda = 1.6$ 42
4.8 Sorted ACE for Occlusion $\alpha = 0.2$ 44
4.9 Sorted ACE for Occlusion $\alpha = 0.4$ 45
4.10 Sorted ACE for Occlusion $\alpha = 0.6$ 46
4.11 DH Trained In Grayscale vs DH Trained In Color 48
4.12 Sorted ACE values for DH Trained to Noise in Unaltered Dataset 49
4.13 Sorted ACE values for DH Trained to Noise in Noise $\eta = 0.1$ Dataset . . . 49
4.14 Sorted ACE values for DH Trained to Noise in Noise $\eta = 0.3$ Dataset . . . 50
4.15 Sorted ACE values for DH Trained to Noise in Noise $\eta = 0.5$ Dataset . . . 50

ABBREVIATIONS

SIFT	Scale Invariant Feature Transform
ORB	Oriented FAST and Rotated BRIEF
FAST	Features from Accelerated Segment Test
BRIEF	Binary Robust Independent Elementary Features
SURF	Speeded Up Robust Features
DH	Deep Image Homography Estimation
HH	Homography Estimation from Image Pairs with Hierarchical Convolutional Networks
ACE	Average Corner Error in Euclidean Pixel Distance
LOG	Laplacian of Gaussian
DOG	Difference of Gaussian
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit

ABSTRACT

Niblick, David K. M.S., Purdue University, December 2019. Comparison of Deep Learning and Feature Matching Methods For Homography Estimation. Major Professor: Avinash C. Kak.

Planar homography estimation is foundational to many computer vision problems, such as Simultaneous Localization and Mapping (SLAM) and Augmented Reality (AR). However, conditions of high variance confound even the state-of-the-art algorithms. In this report, we analyze the performance of two recently published methods using Convolutional Neural Networks (CNNs) that are meant to replace the more traditional feature-matching based approaches to the estimation of homography. Our evaluation of the CNN based methods focuses particularly on measuring the performance under conditions of significant noise, illumination shift, and occlusion. We also measure the benefits of training CNNs to varying degrees of noise. Additionally, we compare the effect of using color images instead of grayscale images for inputs to CNNs. Finally, we compare the results against baseline feature-matching based homography estimation methods using SIFT, SURF, and ORB. We find that CNNs can be trained to be more robust against noise, but at a small cost to accuracy in the noiseless case. Additionally, CNNs perform significantly better in conditions of extreme variance than their feature-matching based counterparts. With regard to color inputs, we conclude that with no change in the CNN architecture to take advantage of the additional information in the color planes, the difference in performance using color inputs or grayscale inputs is negligible. About the CNNs trained with noise-corrupted inputs, we show that training a CNN to a specific magnitude of noise leads to a "Goldilocks Zone" with regard to the noise levels where that CNN performs best.

1. INTRODUCTION

A homography is a planar projective transformation represented by a 3×3 non-singular matrix in the homogeneous coordinate system. The matrix H maps homogeneous coordinates x to x'.

$$x' = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = Hx = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{1.1}$$

This transformation is foundational to many computer vision problems, including Simultaneous Localization and Mapping (SLAM), 3D reconstruction, and Augmented Reality (AR) [1] [2] [3]. Failing to accurately estimate a homography, especially in non-ideal conditions for automated applications, can cause significant error that propagates through the system. For example, in AR applications that require tracking the pose of a camera, even "minor" camera motions create estimation error that cause state-of-the-art systems to fail [3].

The baseline homography estimation pipeline depends on detecting features in a pair of images, determining the feature correspondences, solving for the homography, and then refining the homography with methods like Random Sample Consensus (RANSAC) [4]. Commonly used algorithms for feature detection and description include Scale Invariant Feature Transform (SIFT) [5], Speeded Up Robust Features (SURF) [6], and Oriented FAST and Rotated BRIEF (ORB) [7]. While these algorithms have been successfully used in various applications for over a decade, they all suffer significant error in non-ideal conditions. Noise, illumination variance, and occlusion are common occurrences in many automated computer visions applications that can cause significant error in traditional homography estimation pipelines. This homography estimation error potentially disrupts the entire application.

The recent successes of deep learning in computer vision applications have inspired research in using deep learning for homography estimation. Particularly in computer vision, Convolutional Neural Networks (CNNs) have recently achieved immense success at object detection, image synthesis [8], stereo matching [9], and more [10]. In these applications of CNNs, large amounts of data are used to "train" a neural network through backpropogation to accomplish pattern-related tasks that are otherwise highly complex for conventional algorithms. In this report, we implement two published research articles using neural networks as a solution to the homography estimation problem, both of which reported superior performance compared to the traditional pipeline.

The first of these two papers, "Deep Image Homography Estimation", by DeTone *et al.* was published in 2016 [11]. This was the first attempt at a drop-in replacement for the homography estimation pipeline with a deep learning solution. They altered the VGG-14 [12] architecture to calculate the four point parameterization of the homography matrix between an image and a warped image. They used the Common Objects in Context (COCO) dataset [12] for training and testing. A key contribution of their method is the process for generating training data, in which they warped an image with a randomly generated homography, and then used that homography as the ground truth. They reported results approximately 20% more accurate than ORB.

The following year, "Homography Estimation from Image Pairs with Hierarchical Convolutional Networks" by Nowruzi *et al.* [13] was published. They stacked multiple siamese convolutional networks end-to-end to achieve even higher accuracies. With this "boosting-like" effect, they reported a 67% error reduction to ORB.

In this report, we conduct a thorough comparison of DeTone's and Nowruzi's work against SIFT, SURF, and ORB on the Open 2017 dataset [14] [15]. Using Average Corner Error (ACE) as the primary metric, we analyze performance of these techniques under ideal, noisy, illumination varying, and occluded conditions. We then train a CNN in conditions of noise to measure performance compared against

the CNN trained without noise. Finally, we modify a CNN to use color inputs and measure the change in performance.

The results demonstrate that, although traditional methods retain a superior median ACE in ideal conditions, deep learning methods are consistently more robust in every other environment. When the CNNs are trained with noise-corrupted inputs, we show that training a CNN at a specific level of noise leads to a "Goldilocks Zone" with regard to the noise level where that CNN performs best. That is, the CNN-based homography estimator produces the best average performance when the noise level that was used during the training matches the noise level in the images on which the CNN is tested. For the case of color, we show that, with the same overall CNN architecture, the additional information in the color planes makes negligible contributions to the accuracies of the estimated homographies. The key takeaway is that one method should not be treated as "universally superior" to any other, but instead environmental conditions and engineering constraints need to be deliberately considered when choosing the appropriate technique for any application.

2. LITERATURE REVIEW

2.1 Homography Estimation through Feature Matching

Given a set of coordinates for four pairs of corresponding points between two images, the homography matrix can be derived with a direct linear transform. Traditionally, homography matrices are estimated by detecting features in a pair of images, determining the feature correspondences between the images, solving for the homography matrix, and then refining the results.

2.1.1 Direct Linear Transform

A homography, H, is a mapping of coordinates from x to x' in the homogeneous coordinate system, $x' = Hx$. By constraining the homogeneous coordinates x and x' to $x_3 = x'_3 = 1$, Equation 1.1 can be rewritten as

$$\mathbf{x}' \times \mathbf{Hx} = \begin{bmatrix} \mathbf{0}^T & -\mathbf{x}^T & x'_2\mathbf{x}^T \\ \mathbf{x}^T & \mathbf{0}^T & -x'_1\mathbf{x}^T \\ -x'_2\mathbf{x}^T & x'_1\mathbf{x}^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{bmatrix} = \mathbf{Ah} = \mathbf{0}^T \tag{2.1}$$

Not all three resulting equations are independent in this system. Therefore, using only the first two equations, the system simplifies to:

$$\begin{bmatrix} \mathbf{0}^T & -\mathbf{x}^T & x'_2\mathbf{x}^T \\ \mathbf{x}^T & \mathbf{0}^T & -x'_1\mathbf{x}^T \end{bmatrix} \begin{bmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{bmatrix} = \mathbf{Ah} = \mathbf{0}^T \tag{2.2}$$

Finally, given that the $\mathbf{H}$ is homogeneous, we can further constrain $h_{33} = 1$, and the resulting system can be expressed as:

$$\begin{bmatrix} 0 & 0 & 0 & -x_1 & -x_2 & -1 & x'_2x_1 & x'_2x_2 \\ x_1 & x_2 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1x_2 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} -x'_2 \\ x'_1 \end{bmatrix} = \mathbf{Ah} = \mathbf{b} \quad (2.3)$$

We now have two independent equations from a single correspondence. If we have $N \geq 4$ correspondences, we can stack the $2N$ equations as $\mathbf{A}_i\mathbf{h} = \mathbf{b}_i$, and create a system that allows us to solve for the eight terms in $\mathbf{h}$. The stacked $\mathbf{A}$ and $\mathbf{b}$ terms can be solved with Minimum Least Squares, $\mathbf{h} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$.

2.1.2 Key Point Detection and Description

In order to create correspondences between two images, an algorithm needs to detect key points and describe them in a way that reliably distinguishes them from other points despite variations in illumination, rotation, and scale. Ideally, these key points are also distinguishable in noisy conditions. SIFT, SURF, and ORB are widely used methods for both detection and description of these key points. When the detection and description algorithms work perfectly, the key point locations and description vectors remain consistent despite any transformation.

2.1.3 Key Point Correspondences

After key points are detected and described, they need to be matched between the original and transformed images. To do this, the description vectors must be compared. Common ways to determine correspondences include finding minimal

least squares between the description vectors, or finding the maximal normalized cross correlation.

Once correspondences are determined, they need to be refined as much as possible. Random Sample Consensus (RANSAC) [4] is commonly used to refine the correspondence list. RANSAC is an outlier rejection algorithm that, through random trials, seeks to maximize an "inlier" set of data points and reject any "outlier" data points. It randomly selects two data points (such as estimated homographies), extrapolates a line, then builds an inlier set based on the data points that lie within a chosen threshold, δ, of the line. After executing N random trials, the largest inlier set is selected, and the final result is calculated using all of the points in that set.

While a homography can be calculated with a minimum of four correspondences, it is recommended to use as many *quality* correspondences as possible. Calculating a homography with less than eight correspondences results in outputs that are too noisy for any practical use.

2.2 Common Feature Matching Algorithms

2.2.1 Scale Invariant Feature Transform

The Scale Invariant Feature Transform (SIFT) algorithm was developed by Lowe *et al.* and published in 2004 [5]. SIFT uses a Laplacian of Gaussian Pyramid, approximated as a Difference of Gaussian Pyramid, to find key points by comparing local extrema at different scales. It then creates the description vector by determining the dominant local orientation and combining the histograms of relative gradient measurements from smaller local neighborhoods into a vector.

The purpose of a key point is to be easily distinguishable from other objects in an image. One approach to accomplish this is to detect a point on an image where the local change of contrast is high. These local extrema can be identified with the Laplacian Operator, $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$. The kernel of the Laplacian Operator is generally convolved with an image, highlighting the sections of high contrast change. A simple

Table 2.1.: Digital Approximation of the Laplacian Operator. The 3×3 kernel of the Laplacian Operator measures for large changes in contrast, and can be used to detect edges and key points.

0	-1	0
-1	4	-1
0	-1	0

3×3 Laplacian kernel is shown on Table 2.1. Before using the operator, it is helpful to smooth with the Gaussian kernel in order to avoid noisy results.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(-x^2+y^2)/(2\sigma^2)} \tag{2.4}$$

The σ is a scaling value for the smoothing effect. When combining the Laplacian operator to the Gaussian filter, the result is the Laplacian of Gaussian, which can be simplified to:

$$LoG = \frac{\partial}{\partial\sigma} G(x, y, \sigma) = \sigma\nabla^2 G(x, y, \sigma) \tag{2.5}$$

In application, this can be even further approximated to a Difference of Gaussian Pyramid (DoG), where G is the Gaussian smoothing filter, I is an image, and D is a layer in the DoG, and $*$ is a convolution:

$$D(x, y, \sigma_{12}) = (G(x, y, \sigma_2) - G(x, y, \sigma_1)) * I(x, y) \tag{2.6}$$

Essentially, by slightly adjusting the σ value while smoothing an image, and then finding the difference, locations where the gradient is high in the x, y, and σ dimensions are detected. These are potential key points.

Adjusting σ by a large amount, such as 2σ, changes the "octave" at which the image is smoothed, and therefore the scale at which extrema are detected. Instead of increasing the kernel size at each octave (and therefore increasing the computational cost), the resolution can be decreased by half. Ignoring every other column/row while

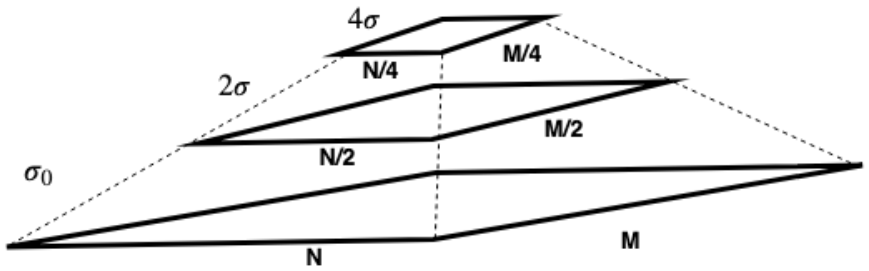

Fig. 2.1.: Difference of Gaussian Pyramid

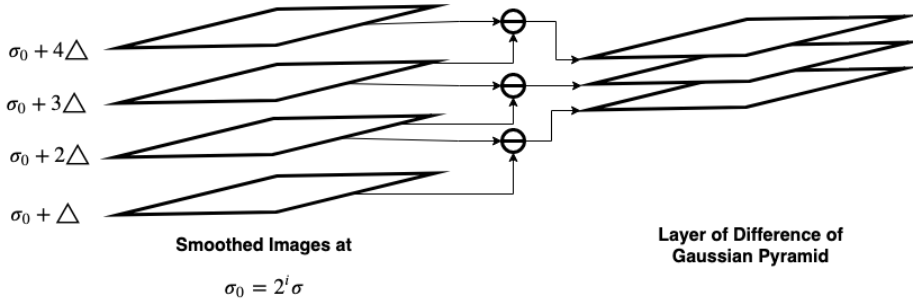

Fig. 2.2.: Layer in Difference of Gaussian Pyramid

moving up an octave achieves the same spatial-scale results per Nyquist's theorem, but at much less computational cost.

SIFT makes a pyramid of images smoothed at different scale octaves of $\sigma_i = 2^i\sigma_0$, as seen in Figure 2.1. At each octave, five smoothed images are created at slightly different $\sigma_i + j\triangle$ scales. From those five smoothed images, three DoG images are calculated, creating a $N \times M \times 3$ volume, as seen in Fig. 2.2.

In the middle image, local extrema are detected by comparing every point with eight adjacent pixels at the same scale, nine adjacent pixels at the scale slightly above and slightly below. If any point is greater or less than all 26 pixels, it is selected as a potential key point at that scale.

Key points found at various scales need to be localized to the original resolution. This is accomplished by using a Taylor Series expansion of $D(x, y, \sigma)$ in vicinity of $\mathbf{x}_0 = (x_0, y_0, \sigma_0)^T$, which can be expressed as:

$$D(\mathbf{x}) = D(\mathbf{x}_0) + \mathbf{J}^T(\mathbf{x}_0)\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{Hess}(\mathbf{x}_0)\mathbf{x} \tag{2.7}$$

where $\mathbf{x}_0$ is an incremental change from $\mathbf{x}$, $\mathbf{J}$ is the gradient vector, and $\mathbf{Hess}$ is the Hessian. For a value to be at the true extrema, $\frac{\partial D(\mathbf{x})}{\partial \mathbf{x}} = 0$ must be true. Applying the derivative to Equation 2.7 results in $0 = \mathbf{J}(\mathbf{x}_0) + \mathbf{Hess}(\mathbf{x}_0)\mathbf{x}$. Solving for $\mathbf{x}$:

$$\mathbf{x} = -\mathbf{Hess}^{-1}(\mathbf{x}_0)\mathbf{J}(\mathbf{x}_0) \tag{2.8}$$

Equation 2.8 allows for pixel localization down to sub-pixel accuracy at original resolution.

With key points detected at multiple scales, and localized to sub-pixel accuracy at the original resolution, they must be filtered. Any key points with a weak response and those that resulted from edges must be removed. Without filtering, the computational load is increased and the results are diluted with key points that aren't distinct enough for a viable description vector. Weak key points can be easily filtered by setting a threshold to the $|D(\mathbf{x})|$ value. Per Lowe *et al.* [5], removing extrema of $|D(\mathbf{x})| < 0.03$ gives good results. To remove edges, a Harris Corner Detector is used, calculating the trace and determinant of the 2×2 Hessian matrix. Lowe *et al.* [5] empirically found that using a ratio of 12.1 for the threshold of the square of trace over the determinant provided good results.

$$\frac{Tr(\mathbf{Hess})^2}{Det(\mathbf{Hess})} < 12.1 \tag{2.9}$$

After key points are detected, localized, and filtered, a descriptor vector that is both unique and robust against variance must be calculated. This is achieved by calculating a dominant gradient in the neighborhood surrounding the key point, and then accumulating gradient magnitudes and orientations of cells into a histogram. A dominant gradient is calculated by constructing a 36-bin histogram, each bin corresponding to a gradient direction θ. The bins are filled by the associated magnitudes, m.

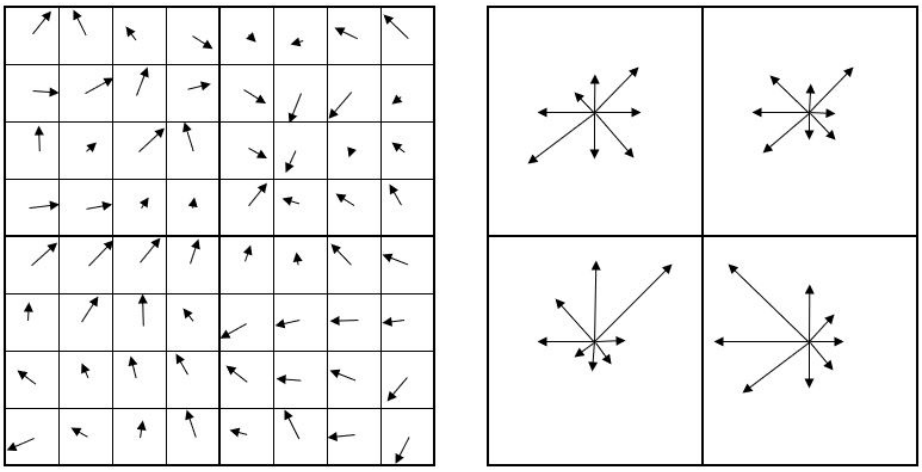

Fig. 2.3.: SIFT Description Vector. Displayed (left) is an 8×8 local key point region with gradient magnitudes and orientations. After rotating the orientations per the dominant gradient and weighting the magnitudes by a Gaussian window, they are aggregated into eight-bin histograms (right). Concatenating these histograms forms the description vector. This figure shows an 8×8 region broken into four cells, but SIFT uses a 16×16 region broken into 16 cells.

$$m(x,y) = \sqrt{(I(x+1,y) - I(x-1,y))2 + (I(x,y+1) - I(x,y-1))^2} \tag{2.10}$$

$$\theta(x,y) = atan2\left(\frac{I(x,y+1) - I(x,y-1)}{I(x+1,y) - I(x-1,y)}\right) \tag{2.11}$$

The peak of the histogram determines the dominant direction for the key point, increasing the robustness to rotational variance.

After determining the local dominant direction, the description vector is calculated from a 16×16 neighborhood around the key point, which is further broken into 16 4×4 cells (Fig. 2.3). The magnitudes of the gradients in the neighborhood are weighted by a Gaussian kernel whose σ is half the width of the neighborhood, decreasing the impact of pixels farther away from the key point. For each 4×4 cell, an eight-bin histogram is calculated from the weighted gradient magnitude, relative to the

key point dominant vector. Stringing together these eight-bin histograms (16 in all) results in a 128-element descriptor. This descriptor is normalized to unity, increasing robustness to illumination variance.

2.2.2 Speeded Up Robust Features

In an attempt to speed up the feature matching process, Bay *et al.* developed Speeded Up Robust Features (SURF) in 2006 [6]. SURF uses integral images to efficiently calculate local extrema based on the determinant of the Hessian over a Gaussian-smoothed image. Haar wavelets are used to calculate the local dominant direction, and the description vector is created from summations of the first order derivative.

Instead of using the Difference of Gaussian to detect extrema, SURF uses the Hessian matrix of the Gaussian kernel. The determinant of the Hessian measures how the curvature in the "image surface" varies in every direction. If the determinant is high, then a local extrema is present, and therefore a key point. The second order derivative of the Gaussian kernel is $h(x) = \frac{-1}{\sqrt{2\pi\sigma^3}}[1 - \frac{x^2}{\sigma^2}]e^{-x^2/2\sigma^2}$. For computational efficiency, kernel for $\partial^2/\partial x^2$ can be approximated in discrete form as seen in Table 2.2. The kernel for $\partial^2/\partial y^2$ would be the transpose of $\partial^2/\partial x^2$. The kernel for $\partial^2/\partial x \partial y$ can be approximated in discrete form as seen in Table 2.3.

Table 2.2.: Discrete Approximation for $\partial^2/\partial x^2$. The second order derivative of a Gaussian kernel can be approximated in the discrete form, reducing the computational complexity at insignificant loss to fidelity.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
1	1	1	-2	-2	-2	1	1	1
1	1	1	-2	-2	-2	1	1	1
1	1	1	-2	-2	-2	1	1	1
1	1	1	-2	-2	-2	1	1	1
1	1	1	-2	-2	-2	1	1	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Table 2.3.: Discrete Approximation for $\partial^2/\partial x \partial y$. The second order derivative of a Gaussian kernel can be approximated in the discrete form, reducing the computational complexity at insignificant loss to fidelity.

0	0	0	0	0	0	0	0	0
0	-1	-1	-1	0	1	1	1	0
0	-1	-1	-1	0	1	1	1	0
0	-1	-1	-1	0	1	1	1	0
0	0	0	0	0	0	0	0	0
0	1	1	1	0	-1	-1	-1	0
0	1	1	1	0	-1	-1	-1	0
0	1	1	1	0	-1	-1	-1	0
0	0	0	0	0	0	0	0	0

In order to further decrease the computational cost, the kernels are applied over the *integral* of the image. The integral image is formed by summing the pixel values along each step across a row or down a column.

$$II(x,y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} f(i,j) \tag{2.12}$$

With the integral image $II(x,y)$, the calculation of the sum of pixels in any rectangular area in the image can be accomplished with four memory look-ups and three additions. The Hessian operator can be approximated by convolving the kernels, resulting in a series of additions and subtractions on sums of rectangular areas. Since a kernel of any size can be calculated in the same number of computations, it is not necessary for SURF to downsample an image and then localize the key point back to the original resolution. Scale invariance is achieved by simply using larger kernels to identify extrema.

Much like SIFT, SURF calculates the "local dominant direction" and determines a descriptor vector with respect to it. At scale σ, the local dominant vector is determined by calculating the first order derivatives across a 6σ area using Haar wavelets. A sample Haar Wavelet to calculate d_x is $\boxed{1 \mid -1}$, and d_y is $\begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline \end{array}$, sized appropriately to σ scale while maintaining symmetry. The d_x and d_y values are weighted by

distance. The dominant direction is determined by "scanning" the weighted scatter plot with a 60° arc, finding the direction with the largest weighted d_x and d_y values. To derive the vector, a $20\sigma \times 20\sigma$ neighborhood is oriented along the dominant local vector, and then broken into 16 cells of $5\sigma \times 5\sigma$. Within each cell, four-element vectors are calculated by $(\sum d'_x, \sum d'_y, \sum |d'_x|, \sum |d'_y|)$, which are the outputs of the Haar kernels. (Again, using the Haar kernels on the integral image makes this a relatively quick operation at any scale.) Those 16 vectors are concatenated together, forming a rotation-invariant 64-element descriptor vector. Like SIFT, the descriptor vector is then normalized to unity to increase robustness against illumination variance.

2.2.3 Oriented FAST and Rotated BRIEF

As an even faster (and open source) alternative to SIFT or SURF, Rublee *et al.* developed ORB, which was published in 2011. ORB combines FAST (Features from Accelerated Segment Test) for key point detection and BRIEF (Binary Robust Independent Elementary Features) for a binary description vector.

FAST detects key points with one parameter. It measures the intensity of a center pixel relative to a circle of pixels around it. (ORB uses FAST with a circle of radius nine.) After thresholding the results to capture a target N number of key points, it orders the key points using a Harris Corner Detector. Without the Harris Corner Detector, multiple "key points" along a straight edge would be included in the list, even though they are not very distinguishable. Finally, since there is no intrinsic scale-invariance to FAST, the process is repeated at multiple downsampled resolutions.

In order to orient FAST on the local dominant gradient for rotational invariance, ORB measures the intensity centroid. Assuming the key point's intensity is slightly offset from the center, it finds this centroid C by measuring the moments m defined by:

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \tag{2.13}$$

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{2.14}$$

The orientation of the vector from the origin of the key point to centroid is defined as:

$$\theta = atan2(m_{01}, m_{10}) \tag{2.15}$$

ORB determines a descriptor vector by using a variant of BRIEF. BRIEF creates a binary vector of 256 elements based on pixel intensity tests on a smoothed image. A common binary test, for example, could be a Gaussian-weighted distribution around the center of the patch. However, Rublee *et al.* used machine learning with a greedy search to determine the highest performing set of uncorrelated binary tests, called rBRIEF. This variation of BRIEF is used in ORB.

2.3 Deep Learning Fundamentals

Although the concepts of deep learning with neural networks have existed for decades, recent advancements enabled these techniques to match or exceed the state-of-the-art in many problems. Deep learning involves a forward pass of input, through a neural network, to an output, calculation of loss between the output and a known ground truth, backpropogation of the loss throughout the network, and gradient descent to determine the step size and direction with which to adjust the parameters of the network. In order to achieve this training cycle, *labeled* data are necessary. The cycle of forward pass, loss calculation, backpropogation, and gradient descent is repeated numerous times, with each iteration forcing the parameters of the network to "step" closer and closer to a viable solution. Additionally, many neural networks employ regularization techniques to improve the efficiency of training.

2.3.1 Fully Connected Layers

Fully connected layers attempt to model a collection of neurons, somewhat resembling the function of the human mind. Each neuron takes a collection of input values, sums them, adds a bias term, and then applies some non-linear function to the result.

$$out_m = f_{act}\left(\sum_n \left[w_{m,n} * input_n\right] + b_m\right) \tag{2.16}$$

Here, out_m is the output of neuron m, $input_n$ is the output of some other neuron n, $w_{m,n}$ is a "weighting" factor applied to the input of neuron n as it enters neuron m, and b_m is a biasing term added to the result of all the weighted inputs. The activation function, f_{act}, is a non-linear function that shapes the output of a neuron before it passed to other neurons. Non-linearity is a necessary attribute for activation functions, as it adds the complexity required for solving otherwise intractable problems. A commonly used activation function is the Rectified Linear Unit (ReLU), defined as:

$$f(x)_{relu} = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \tag{2.17}$$

The ReLU function is simple to implement, fast to calculate, and avoids diminished gradients as activations are chained together, all while achieving the required complexity of non-linearity.

If these neurons are organized into layers, where the outputs of layer $i-1$ act as the inputs to layer i, the process can be simplified to a series of matrix multiplications, as seen below:

$$\mathbf{Out}_i = f_{act}\left(\mathbf{W}_i * \mathbf{Out}_{i-1} + \mathbf{B}_i\right) \tag{2.18}$$

The output of a fully connected layer i is $\mathbf{Out}_i$, $\mathbf{W}_i$ is the matrix of weights connecting the previous layer to the current layer, and $\mathbf{B}_i$ is the bias values applied after summation of weighted inputs.

2.3.2 Convolutional Layers

Convolutional layers replace the neuron from a fully connected layer with a convolutional kernel containing learned weights. The motivation for a convolutional layer is to avoid the massive number of weights required to process 2D images. For example, even a relatively small image of 128×128 requires the first layer to contain 16,384 neurons. Large networks take an extremely long time to train, require massive amounts of memory, and can suffer from "overfitting". (Overfitting occurs when a network succeeds only on data it has already trained on, but fails at data it hasn't seen. In other words, the network fails to generalize.) Additionally, convolutional layers retain spatial information instead of treating every input pixel as independent of other pixels.

A single kernel of a 2D convolutional layer contains $M \times N \times In\_Channels + 1$ weights. (The addition accounts for the bias term.) The input is convolved channel by channel, with a bias added to the results at every step, creating a 2D output "feature map". Depending on the implementation, padding can be added to the input in order to handle border effects. The elements of the feature map are passed through an activation function (such as ReLU) to achieve the complexity of non-linearity. A convolutional layer generally includes multiple kernels, each of which corresponds to a feature map in the output. The output of the layer is comprised of the stacked feature maps of the kernels, which in turn is passed to the next layer.

2.3.3 Backpropogation

After the input passes through all layers and the forward pass is complete, the loss between the output and a ground truth needs to be calculated. A common method

for calculating loss is to use the Mean Squared Error (MSE) metric. This is the mean of squared differences between elements in the output and ground truth vectors.

$$L_{MSE} = \frac{1}{N}\sum_{i}^{N}(x_i - y_i)^2 \tag{2.19}$$

The calculated loss is propagated backward through the network using the chain rule of derivatives. The backpropogation of loss determines the quantitative impact each parameter had on the output per input, and therefore how much each parameter should be adjusted to force the output closer to the ground truth. This parameter-output sensitivity, aggregated into a gradient, drives the training process.

The gradient can be calculated through the following process. The symbol $w^k_{m,n}$ represents the parameter for the weight between node m in layer $k-1$ and node n in layer l. (Bias is included as $w^k_{0,n}$.) The activation is the weighted sum of inputs before nonlinearity. This is represented for node n in layer k as a^k_n. The output of node n in layer k, after the nonlinear function is applied, is represented as o^k_n. Therefore in the forward pass,

$$o^k_n = f_{act}\left(a^k_n\right) \tag{2.20}$$

$$a^k_n = \sum_{i\in k-1}\left[w^k_{i,n} f_{act}\left(a^{k-1}_i\right)\right] = \sum_{i\in k-1} w^k_{i,n} o^{k-1}_i \tag{2.21}$$

After calculating loss, the chain rule is applied to find the sensitivity of loss L to a specific parameter, $w_{m,n}$.

$$\frac{\partial L}{\partial w^k_{m,n}} = \frac{\partial L}{\partial a^k_n}\frac{\partial a^k_n}{\partial w^k_{m,n}} \tag{2.22}$$

The first term of Equation 2.22 can be defined as,

$$\frac{\partial L}{\partial a^k_n} = \delta^k_n = \sum_{i\in k+1}\left[\frac{\partial L}{\partial a^{k+1}_i}\frac{\partial a^{k+1}_i}{\partial a^k_n}\right] = \sum_{i\in k+1}\left[\delta^{k+1}_i\frac{\partial a^{k+1}_i}{\partial a^k_n}\right] \tag{2.23}$$

Using Equation 2.21, the first order derivative between layers for node activation is,

$$\frac{\partial a_n^k}{\partial a_m^{k-1}} = \frac{\partial}{\partial a_m^{k-1}} \sum_{i \in k-1} \left[w_{i,n}^k f_{act} \left(a_i^{k-1} \right) \right] = w_{m,n}^k f'_{act}(a_m^{k-1}) \tag{2.24}$$

Substituting Equation 2.24 into Equation 2.23 simplifies to,

$$\frac{\partial L}{\partial a_n^k} = \delta_n^k = \sum_{i \in k+1} \left[\delta_i^{k+1} w_{n,i}^{k+1} f'_{act} \left(a_n^k \right) \right] = f'_{act} \left(a_n^k \right) \sum_{i \in k+1} \left[\delta_i^{k+1} w_{n,i}^{k+1} \right] \tag{2.25}$$

The second term of Equation 2.22 can be redefined by substituting Equation 2.21,

$$\frac{\partial a_n^k}{\partial w_{m,n}^k} = \frac{\partial}{\partial w_{m,n}^k} \left(\sum_{i \in k-1} w_{i,n}^k o_i^{k-1} \right) = o_m^{k-1} \tag{2.26}$$

Combining Equations 2.25 and 2.26 simplifies the solution in Equation 2.26, from which the parameter gradient can be derived.

$$\frac{\partial L}{\partial w_{m,n}^k} = \delta_n^k o_m^{k-1} = o_m^{k-1} f'_{act} \left(a_n^k \right) \sum_{i \in k+1} \left[\delta_i^{k+1} w_{n,i}^{k+1} \right] \tag{2.27}$$

With the gradients calculated, a gradient descent algorithm is used to update the parameters in the most efficient way possible. One of the most widely-used algorithms, known as Adam, was developed by Diederik *et al.*, and was published in 2014 [16]. This gradient-based stochastic optimization algorithm estimates the expected mean and variance of the gradients to adjust the update rates for each parameter. To calculate the moments, it uses exponentially moving averages as follows:

$$m_i = \beta_1 m_{i-1} + (1 - \beta_1) g_i \tag{2.28}$$

$$v_i = \beta_2 v_{i-1} + (1 - \beta_2) g_i^2 \tag{2.29}$$

The moving average for the first moment (or expectation) is m, v is the moving average for the second moment (or variance), g is the gradient, and β_1 and β_2 are constants that determine the influence of previous iterations on the current iteration. The values are initialized to zero. In order to get the moving averages closer to the "true" moments, they are each adjusted with a bias correction:

$$\widehat{m}_i = \frac{m_i}{1 - \beta_1^i} \tag{2.30}$$

$$\widehat{v}_i = \frac{v_i}{1 - \beta_2^i} \tag{2.31}$$

Finally, the actual parameters are updated as follows:

$$w_i = w_{i-1} - \eta \frac{\widehat{m}_i}{\sqrt{\widehat{v}_i} + \epsilon} \tag{2.32}$$

The global learning rate is represented as η, and ϵ is a small constant to avoid division by zero.

2.3.4 Training Enhancement Techniques

Most neural networks employ additional techniques to avoid overfitting and increase training efficiency. Overfitting can be avoided by adding noise to the process and minimizing the required number of parameters.

Batch normalization, a concept originally published for deep learning in 2015 [17], applies the same strategy of normalizing an input to a machine learning process, but between the hidden layers of a neural network. By subtracting the batch mean and standard deviation of the output of one hidden layer prior to next layer, the training process avoids internal covariate shift, and therefore becomes more stable and efficient. Without using batch normalization, it is possible for inputs of orders of magnitude larger than the rest to dominate the learning process. To make batch normalization more flexible, instead of moving the distribution to zero mean and unit

variance, it introduces two new learn-able parameters that allow the training process to determine the best way to shape any input distribution to particular layer. Using a "learned" mean and variance keeps any particular weight or layer from dominating the process while still being flexible enough to allow the network adapt. Ultimately, it allows for use of larger learning rates, decreasing the time and data necessary for a network to learn desirable features.

Pooling is a process of combining outputs of a convolutional layer and selecting one to represent the entire sector of a feature map. For example, in a 2×2 max pooling, the output $M \times N$ feature map is divided into blocks of 2×2, and only the largest value of each block is passed onto the next layer. This is a way to efficiently downsample the feature maps such that information loss is minimized. Not only does pooling minimize the number of parameters in a model, but it has also been shown to help avoid overfitting.

Finally, dropout is a regularization technique that removes a random portion of a layer's neurons during training. The idea was originally published in 2014 [18] as a technique to enable using larger networks without overfitting. This forces the network to not "depend" on certain neurons and ignore others, while also adding noise to the process. Although this might actually slow down the training process slightly, it greatly decreases the chance of overfitting the training data. When the network is used for evaluation, dropout is removed, leaving all neurons activated for inference.

2.4 Homography Estimation through Deep Learning

In recent years, deep learning has proven effective at analyzing complicated patterns which often confound "hand-crafted" algorithms. Particularly in computer vision, Convolutional Neural Networks (CNNs) recently achieved immense success at tasks such as object detection, image synthesis, stereo matching [8] [10] [9]. For this report, we analyze two deep learning methods as a drop-in replacement of the entire homography estimation pipeline.

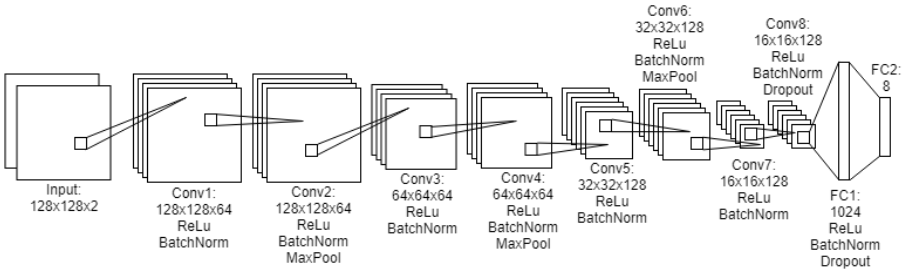

Fig. 2.4.: Deep Image Homography Estimation Model. The network architecture consists of eight convolutional layers in sequence, followed by two fully connected layers. Each layer uses ReLu activation (except for the last fully connected layer) and batch normalization. Max pooling is applied to layers two, four, and six. Dropout is applied after the last convolutional layer and first fully connected layer.

2.4.1 Deep Image Homography Estimation

"Deep Image Homography Estimation", developed by DeTone *et al.*, and published in 2016 [11], is the first deep learning solution to homography estimation that replaces the entire pipeline. It uses the popular VGG-14 Net [8] to process two patches from similar images (or after a warping is applied to a single image). The two input patches are converted to grayscale, and then are stacked onto each other. The architecture uses eight convolutional layers, followed by two fully connected layers (Figure 2.4). The convolutional layers use a 3×3 kernel. ReLu activation and batch normalization follows each layer. Max Pooling occurs prior to the third, fifth, and seventh convolutional layers. Dropout with a factor of 0.5 is applied before each fully connected layer.

The output is a parameterized form of the estimate homography matrix $\mathbf{H}$. Specifically, the output is the Four Point Parameterization, $\mathbf{H}_{4pt}$. (To avoid confusion throught the rest of the paper, $\mathbf{H}_{mat}$ refers to the original homography matrix, while $\mathbf{H}_{4pt}$ refers to the Four Point Parameterization.) Instead of directly estimating the homography matrix, it outputs the difference between the original corner locations and warped corner locations. If $\mathbf{x}' = \mathbf{H}_{mat}\mathbf{x}$, and $\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix}$,

$$\mathbf{H}_{4pt} = \begin{bmatrix} u_1 - u'_1 & v_1 - v'_1 \\ u_2 - u'_2 & v_2 - v'_2 \\ u_3 - u'_3 & v_3 - v'_3 \\ u_4 - u'_4 & v_4 - v'_4 \end{bmatrix} \tag{2.33}$$

By adding $\mathbf{H}_{4pt}$ to the original corner locations, we derive four correspondences and therefore can use Equation 2.3 to map between $\mathbf{H}_{4pt}$ and $\mathbf{H}_{mat}$. DeTone et al. describes the reasoning for using $\mathbf{H}_{4pt}$ as follows:

This raises the interesting question: Why not use the 8 numbers that uniquely characterizes a 3×3 homography between a pair of images for the output of a neural network,[1] as opposed to the eight numbers for the differences in the corresponding-pixel coordinates as shown above? The main problem with using the homography elements directly in a CNN-based learning framework is that the homography matrix encodes both translations and rotations in ways that depend on the nature of homography. For example, for an affine homography, all the rotation is encoded in the upper 2×2 submatrix of the homography and all the translations in the first two elements of the rightmost column. This makes for non-uniform semantics for the elements of the output vector of a neural network if the vector is to represent the elements of a homography matrix. On the other hand, when a 3×3 homography is represented by a matrix like the one shown in Equation 2.33, all of its elements carry the same meaning. That makes it more likely that a neural network would be able to carry out the optimization needed for estimating those elements.

At the time of publication, there were no labeled homography datasets available large enough for training. However, DeTone *et al.* devised a self-supervised method to train with any natural image dataset. A patch is randomly selected from the image, no closer than ρ pixels from any image corner. A random perturbation (of no

[1]Since a 3×3 homography, $\mathbf{H}_{mat}$, is a linear mapping in projective coordinates, the homography is invariant to multiplication by a scalar. Another way of saying the same thing is that all the information in a homography matrix resides in the ratios of each of the nine elements with respect to a chosen element, which is typically $\mathbf{H}_{mat}[2, 2]$

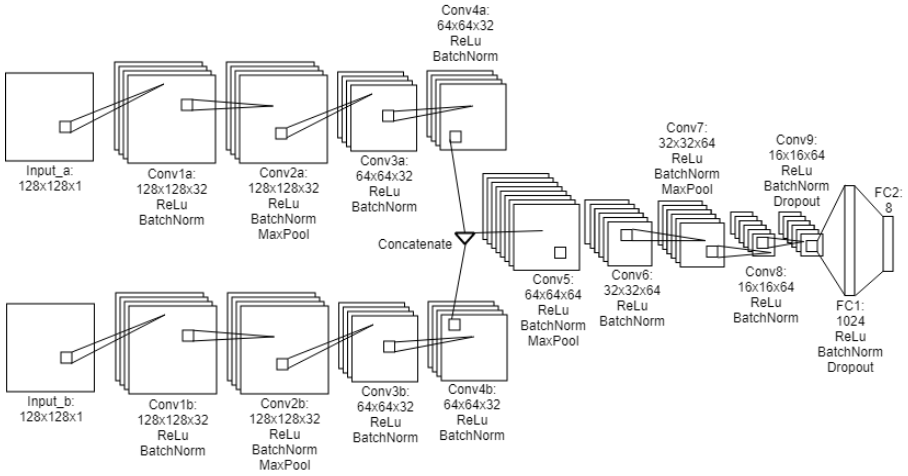

Fig. 2.5.: Hierarchical Homography Model. This siamese network uses eight convolutional layers followed by two fully connected layers. The first four layers run parallel and share weights, and then are concatenated before layer five. All layers use ReLu activation (except for the last layer) and batch normalization. Max pooling is applied after the second, fifth, and seventh layers. Dropout is applied after the last convolutional layer and first fully connected layer.

more than ρ pixels) is applied to the corners of the patch, creating warped corners. The correspondences between the original corners and warped corners are used in Equation 2.3 to calculate a homography, $\mathbf{H}_{mat}$. The homography is applied to the image, and a patch (from same coordinates as original patch) is extracted from the warped image as the warped patch. The original and warped patches are stacked and input into the neural network, with the corner perturbation values used as the ground truth, $\mathbf{H}_{4pt}^{target}$. This "label on the fly" method is both simple and incredibly flexible. The ability to use any image dataset for training and testing was critical for a successful deep learning solution to the homography estimation problem.

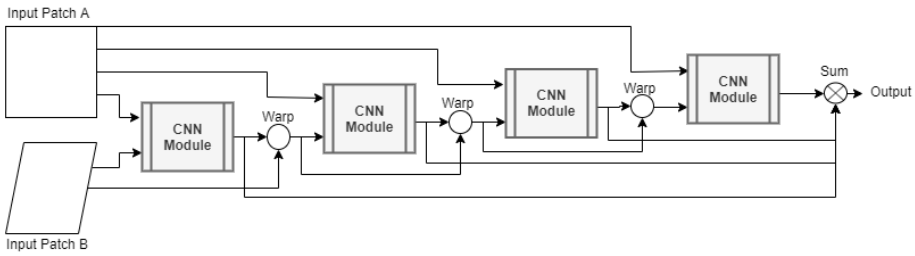

Fig. 2.6.: Hierarchical Homography Stack. In order to achieve a better accuracy, multiple networks are used in series. The error bounds for each network is less, allowing the following network to train to a more accurate standard. The warped patch (Input Patch B) is warped by the difference between the previous target homography and estimated output. At the very end, all estimated homographies are summed together to calculate the original target homography.

2.4.2 Homography Estimation from Image Pairs with Hierarchical Convolutional Networks

Published a year later, "Homography Estimation from Image Pairs with Hierarchical Convolutional Networks" by Nowruzi *et al.* [13] attempts to improve on accuracy by using multiple siamese CNNs stacked end-to-end. Much like "Deep Image Homography Estimation" [11], Nowruzi's work takes an input of two image patches and outputs the estimated corner perturbation values $\mathbf{H}_{4pt}$, which can be mapped to a homography matrix $\mathbf{H}_{mat}$.

The architecture also uses eight convolutional layers followed by two fully connected layers (Fig. 2.5). However, the first four convolutional layers are two parallel branches with shared weights. The first branch takes the original patch as input, and the second branch takes the warped patch, as opposed to stacking the patches and inputting into the same layer. The branches merge after the fourth layer, concatenating along the feature dimension. Each layer is followed by ReLu activation and batch normalization. Max Pooling is applied after the second, fifth, and seventh convolutional layers. Dropout of factor 0.5 is applied prior to the first fully connected layer.

This architecture is repeated as separate modules stacked end-to-end (Fig. 2.6). The original patch and warped patch are fed into the first module, which estimates $\mathbf{H}_{4pt}$. That estimated homography is subtracted from the target to create a new target for the next module.

$$\mathbf{H}_i^{target} = \mathbf{H}_{i-1}^{est} - \mathbf{H}_i^{target} \tag{2.34}$$

The new target is used to create a new warped patch. The input data to the following module is the original patch and new warped patch, with the new target as ground truth. ($\mathbf{I}$ represents the input image patch.)

$$(\mathbf{I}_i^{original}, \mathbf{I}_i^{warped}) = (\mathbf{I}_{i-1}^{original}, \mathbf{I}_{i-1}^{original} * \mathbf{H}_i^{target}) \tag{2.35}$$

This process is repeated iteratively, with a "boosting" like effect that drives error down. In other words, the magnitude of error residuals shrink between each iteration. By training each module specific to those error residual ranges, the overall accuracy is increased.

3. EXPERIMENT

We implemented and analyzed the performance of the two deep learning solutions, as well as the baseline methods SIFT, SURF, and ORB. We execute the homography estimation tests on the Open Images 2019 Test Set [14] [15]. Using this dataset, we conduct four experiments by simulating conditions of variance, including ideal (i.e. unaltered input images), noise, illumination shifts, and random occlusions. Performance was measured with the Average Corner Error (ACE), as well as the Outlier Ratio (OR). Finally, we report the results in tables comparing ACE and OR, as well as figures illustrating the distribution of sorted ACE values for each method per experiment.

3.1 Implementation Details

To implement SIFT, SURF, and ORB, we used OpenCV 3's [19] default implementation. After determining correspondences, we solve the homography and refine it with RANSAC. We implemented and trained the neural networks using the PyTorch library [20]. The dataset for training and validation was the COCO 2017 Unlabeled dataset [12], split by 100k images for training and 23k images for validation. By using a different dataset for training/validation and for testing, we ensured that the neural networks generalized appropriately. Labels were created "on the fly" per the method described below during training and validation. We used the Adam optimizer [16] for both networks, training for approximately 30 epochs at a learning rate of 0.005. (One epoch here is defined as a single pass over the entire training dataset.) The learning rate was halved every five epochs. PyTorch's `MSELoss` function was used as the loss function.

All tests were conducted on the Open Images 2019 Test Set (approximately 100k images) [14] [15]. The process developed by DeTone *et al.* and described in Section 2.4.1 was used to develop ground truth homographies from the data. For each image, four corners were selected, and then a random perturbation was applied to each corner. The homography that maps the original corners to the perturbed corners was derived. The image was transformed by the homography to create a warped image. A patch from the original image and the warped image were used as the data, and the homography (both as $\mathbf{H}_{mat}$ and $\mathbf{H}_{4pt}$) was the ground truth. The inputs were normalized to the dataset mean and variance, and values were scaled to a $(-1, 1)$ range.

3.2 Evaluation Procedures

We conducted four experiments over the entire dataset. Each experiment simulates a different form of variance experienced in natural conditions. The four experiments are Ideal Conditions (where the dataset is unaltered), Gaussian Noise, Illumination Shift, and Occlusion. For the latter three experiments, the process is repeated three times with different variance values in order to better illustrate the sensitivity of each method as the variance increases. Table 3.1 breaks down each experiment with the variance values.

To simulate noise, we added normal Gaussian noise scaled by a factor η (such as 0.4) relative to pixel range, and clipped the results to normalized pixel range.

$$X_{noisy} = min(max(X + \eta\mathcal{N}(0, 1), -1), 1) \tag{3.1}$$

To simulate illumination, we multiplied the pixel values in the warped patch by a factor λ (such as 1.4) and clip the results to normalized pixel range.

Table 3.1.: Experiment Summary. Four experiments are conducted, each one simulating a type of variance. In each experiment (except for Ideal which leaves the dataset unaltered), three different magnitudes of variance are used to better illustrate the sensitivity of each method.

Experiment	Magnitude of Variance		
Ideal Conditions	*No Variance*		
Gaussian Noise	$\eta = 0.1$	$\eta = 0.3$	$\eta = 0.5$
Illumination Shift	$\lambda = 1.2$	$\lambda = 1.4$	$\lambda = 1.6$
Occlusion	$\alpha = 0.2$	$\alpha = 0.4$	$\alpha = 0.6$

$$X_{illum} = min(max(\lambda X, -1), 1) \tag{3.2}$$

To simulate occlusions, We replaced an $n \times n$ box of random location with a random color in the warped image. The size of $n \times n$ is determined by a factor α (such as 0.6) relative to the total patch space. In other words, if $\alpha = 0.6$, then 60% of the pixels in the warped patch are replaced by a box of random color.

After comparing CNNs to the baseline feature matching methods, we analyzed the affect of training on noisy data will have on performance. We used the trained Deep Image Homography Net, train it for an additional 30 epochs in $\eta = 0.1$ noisy data, and then measure performance. We then train it for another 30 epochs on $\eta = 0.3$, tested again, and then repeat with $\eta = 0.5$.

Finally, to measure the effect of using color images instead of grayscale images for input, we modified the Deep Image Homography Net to accept six channels for input instead of two. We then train the network from scratch on color images, and measure performance against the grayscale counterpart.

3.3 Evaluation Metrics

The primary metric for testing accuracy was Average Corner Error. This is the euclidean distance between corner locations after applying the target and output homographies. The value was averaged over the four corners.

$$ACE = \frac{1}{4}\sum_{i=1}^{4} ||\mathbf{H}_{mat}^{target}\mathbf{x}_i - \mathbf{H}_{mat}^{output}\mathbf{x}_i|| \tag{3.3}$$

Here, $\mathbf{x}_i$ refers to one of the four corners of the original patch, $\mathbf{H}_{mat}^{target}$ is the ground truth homography in the original matrix parameterization, and $\mathbf{H}_{mat}^{output}$ is the estimated homography from the technique being evaluated.

We present median ACE as opposed to mean ACE due to the unbounded error at the positive extreme. The ACEs of the highest error greatly skew the mean. In most applications, a homography that maps points over 50 pixels away from the ground truth is not much more useful than one that maps points over 500 pixels away - both will cause the system to fail. However, the latter case has a much greater effect on the mean. This skewing effect heavily favors deep learning methods, and fails to illustrate which method is most accurate for the majority of the data.

However, the median alone fails to convey enough information about how these methods compare. Previous publications on homography estimation with deep learning did not report on how error is distributed over the dataset for each method [11] [13]. Only the mean was used to represent accuracy. Therefore, in this report we also include a novel metric described as the Outlier Ratio. This is the ratio of extremely high outlier ACE values compared to the rest of the dataset. For the sake of consistency, **we define a value to be an outlier if it exceeds an ACE of 50 pixels**, or is undefined because the feature-matching method failed to produce enough correspondences for a solution. We choose 50 for two reasons. First, if a homography maps points that are greater than 50 pixels away from ground truth in an image patch of size 128×128, that estimated homography has little practical use.

Many systems will fail to achieve desirable results if it depends on a homography with such large error. Second, as illustrated by the figures in the "Results" section of this report, every method that reaches a sorted ACE of 50 experiences extreme error growth beyond that point. The set of values with an ACE greater than 50 contains such extremely large values when compared to the rest of the results that it is prudent to classify such a set as "outliers". This OR metric is important to gaining greater insight into the performance of each method. It can be heuristically thought of as the rate at which a certain method will produce a practically unusable homography in given conditions.

4. RESULTS

In this section, we provide results for our comparative study of homography estimation methods. We used the Open Images 2019 Test Set (approximately 100k images) [14] [15] for all testing. The method for extracting labels can be found in Section 3.1. We executed four experiments, each one corresponding to a type of variance. In Ideal Conditions, the dataset is left unaltered as a baseline. Gaussian Noise experiments with varying degrees of adding values from a normal Gaussian distribution to the pixels in both input patches. Illumination Shift adds a fixed value to each pixel in the warped input patch. Occlusion removes a portion of pixels from the warped input patch to simulate a shadow or object blocking that viewpoint. For the Gaussian Noise, Illumination Shift, and Occlusion experiments, three different magnitudes of variance are used to illustrate the sensitivity of each method to the variance. The experiments are summarized in Table 3.1.

Two metrics are used to measure performance across these experiments. The first is Average Corner Error (ACE), which is the average euclidean distance of pixels from the estimated homography mapped point to the ground truth homography mapped point. The second is Outlier Ratio (OR), which is the ratio of outputs with ACE above 50 to total outputs. Thorough explanations for both metrics can be found in Section 3.3. Finally, we present figures that compare the sorted ACE values as line charts for each method in each experiment. This best illustrates the distribution of error over the dataset, and therefore conveys a qualitative sense of consistency. The lines depict sorted ACE values, from least to greatest, for each input patch pair. The lines of each method are combined into one figure (per experiment) to more easily compare the methods.

For brevity, DH refers to "Deep Image Homography Estimation" by DeTone, *et al.*, and HH refers to "Homography Estimation from Image Pairs with Hierarchical

Convolutional Networks" by Nowruzi *et al.*. In the tables below, the best ACE is in bold. The methods above the dotted line are deep learning, and the methods below are feature matching.

4.1 Ideal Conditions

In ideal conditions, where the dataset is unaltered, feature matching methods are generally more accurate, while deep learning methods are more consistent. SIFT has the lowest median ACE at 0.89. DH has the lowest OR at 0, meaning every output had an ACE below 50 pixels. HH had the second lowest median ACE at 2.25 with an OR near 0. Put a different way, SIFT was more accurate than any other method for approximately 90% of the dataset, while HH was most accurate for approximately 10%. However, in that 10%, SIFT experienced tremendously high error.

Between the two deep learning methods, HH outperformed DH for nearly all of the 100k inputs.

For the feature matching methods, SIFT had lowest error for the entire dataset, and ORB had the highest.

Table 4.1.: SIFT has the lowest median ACE, followed by HH. DH had a perfect OR, meaning every ACE was below 50 pixels. SIFT has the best OR of the feature matching methods. ORB was the worst-performing method, with a high median ACE and OR.

Method	Median ACE	OR
DH	3.97	0
HH	2.25	≤ 0.01
SIFT	**0.89**	0.08
SURF	3.15	0.09
ORB	7.82	0.12

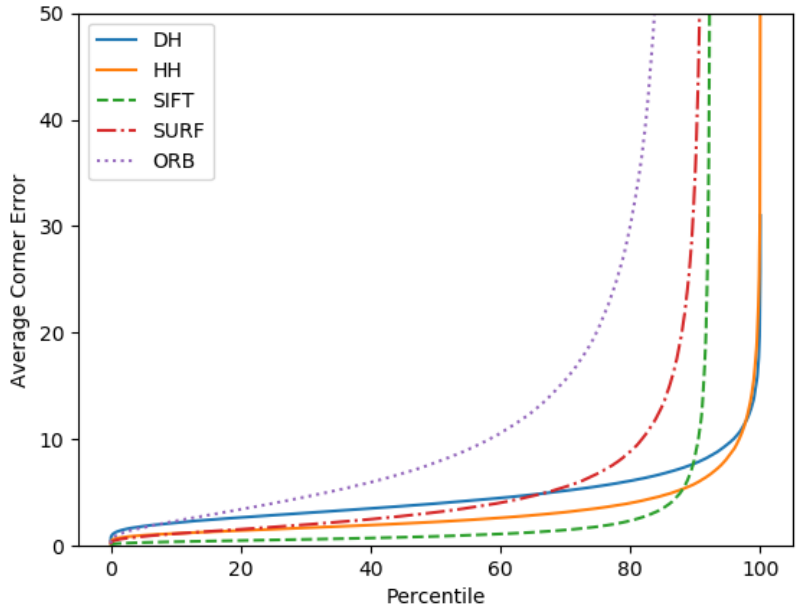

Fig. 4.1.: Sorted ACE for Unaltered Dataset. SIFT gives superior performance for 90% of the data. It has sub-pixel ACE for approximately half the dataset. HH remains consistently the second best, and is most accurate for approximately 8% of the data. In that 8% of the dataset, SIFT has incredibly high error making it unusable in most practical applications.

4.2 Noisy Conditions

Feature matching methods are much more sensitive to noise than deep learning methods. Although SIFT and SURF maintain a lower median ACE than deep learning methods at $\eta = 0.1$, by $\eta = 0.3$ both SIFT and SURF fail to determine enough correspondences to determine a solution for over half the dataset. The deep learning methods, though affected by noise, remain much more robust. Even at the highest value of noise, the highest OR of either deep learning method is 0.03, compared to the highest OR of feature matching methods at 0.94. This means that, at the highest level of noise, SIFT is practically unusable for 94% of the dataset, whereas both deep learning methods provide "reasonable" results for at least 97% of the dataset. Even

ORB, the most robust feature matching method, fails to achieve the same robustness as either deep learning method.

Both deep learning methods degrade in noise, although HH experiences less degradation than DH. From a qualitative perspective, the effect is roughly the same on both, just at different scales.

Noise has significant impact to feature matching methods. While all three methods are still viable at a "low" level of noise, SIFT and SURF degrade very quickly at "moderate" and "high" noise levels. ORB, however, degrades at a much lower rate than the other feature matching methods. This is consistent with the findings of Rublee *et al.*, where ORB was more robust against noise than SIFT or SURF [7].

Table 4.2.: Median ACE and OR in Noise. Depicted are the ACE and OR for each method in noise scaled to $\eta = 0.1$, $\eta = 0.3$, and $\eta = 0.5$. Above the dotted line are deep learning methods, and below it are feature matching methods. Although SIFT has the best median ACE in "low" noise conditions, it is highly sensitive when the level of noise increases. HH has the best OR for every noise level, as well as the best median ACE for "moderate" and "high" noise levels.

Noise Factor η	0.1	0.3	0.5
Method	Median ACE / OR		
DH	13.60 / ≤ 0.01	21.73 / ≤ 0.01	28.49 / 0.03
HH	6.79 / ≤ 0.01	**13.38** / ≤ 0.01	**17.30** / ≤ 0.01
SIFT	**2.33** / 0.18	NAN / 0.66	NAN / 0.94
SURF	5.42 / 0.17	NAN / 0.62	NAN / 0.93
ORB	12.20 / 0.16	26.50 / 0.34	53.12 / 0.52

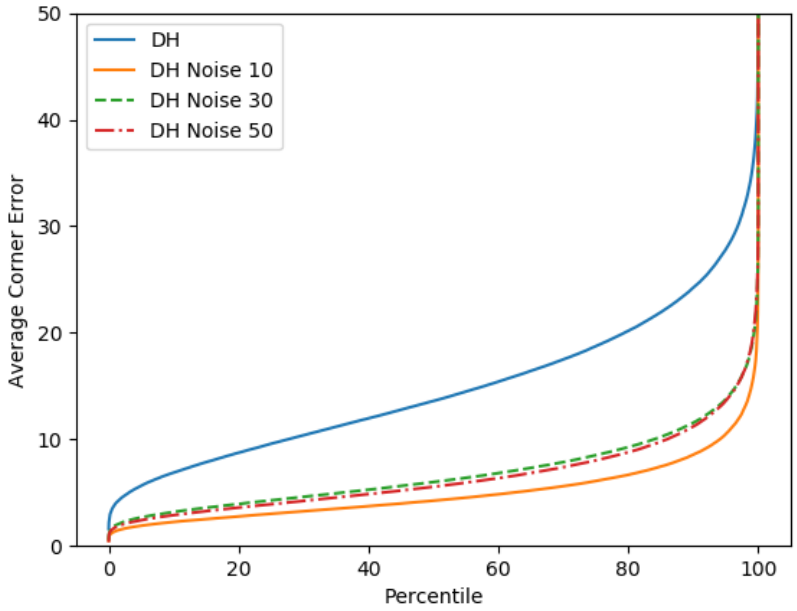

Fig. 4.2.: Sorted ACE for Noise $\eta = 0.1$. This figure depicts performance of each method using sorted ACE values, by least to greatest. SIFT has the lowest ACE for approximately 80% of the dataset. For the other 20%, HH has a lower ACE, and only exceeds an ACE of 50 for less than 0.01% of the dataset.

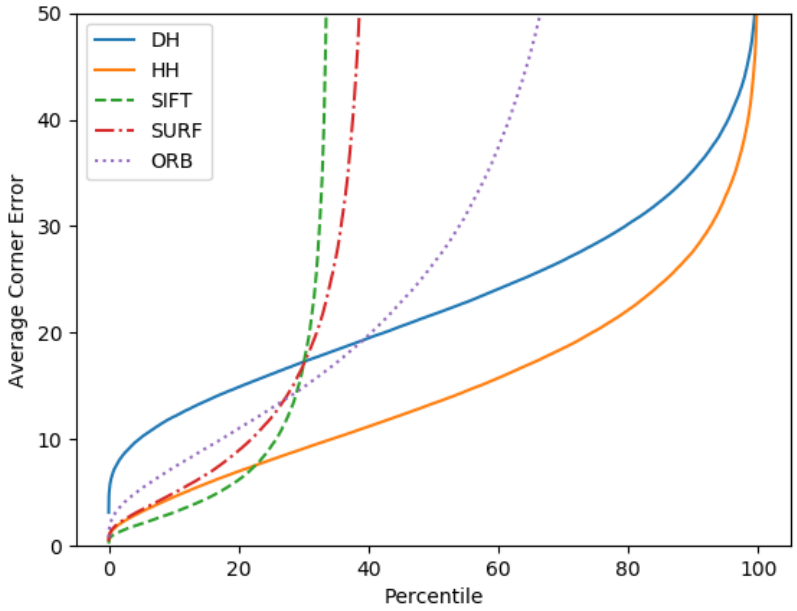

Fig. 4.3.: Sorted ACE for Noise $\eta = 0.3$. This figure depicts performance of each method using sorted ACE values, by least to greatest. At a "moderate" level of noise, SIFT and SURF give viable results for less than 40% of the dataset. ORB proves to be more robust to noise, but not as robust as the deep learning methods, both of which remain below an ACE of 50 for over 99% of the dataset.

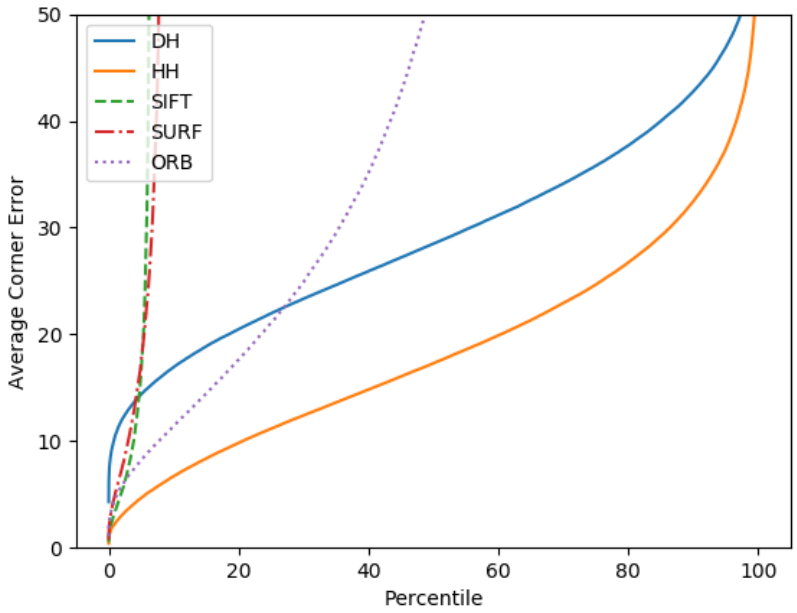

Fig. 4.4.: Sorted ACE for Noise $\eta = 0.5$. This figure depicts performance of each method using sorted ACE values, by least to greatest. The same trends exhibited in Figure 4.3 are illustrated here, but to a greater extent. SIFT and SURF degrade in accuracy almost immediately, while the deep learning methods maintain a reasonable accuracy for the majority of the dataset. ORB outperforms DH for approximately 20% of the dataset, but it too degrades quickly after that point.

4.3 Illumination Variance Conditions

Feature matching has a better median ACE at "low" and "moderate" levels of illumination shift, but at "high" illumination shift, deep learning achieves a lower median ACE. The effects somewhat resemble that of Gaussian noise, but not to the same magnitude. While deep learning is still robust, especially at higher variance intensity, that robustness is not as pronounced as it is with Gaussian noise.

Between the two deep learning methods, DH is impacted much more significantly by illumination shift. Even a "low" illumination shift causes a rise in median ACE of almost 20 when compared to ideal conditions. As the illumination shift intensity increases, DH's median ACE only rises a moderate amount. HH is very robust, only increasing slightly in median ACE even at the highest illumination shift. The discrepancy between the two deep learning methods could be explained by the fact that HH is based on a siamese network, where two input patches are transformed by four convolutional layers before the features are combined. By contrast, DH stacks the input patches at the very beginning. The discrepancy in input patches, therefore, has a larger impact, as there is no opportunity for the network to isolate invariant features within each patch, independent of the other.

Of the feature matching methods, SIFT remains consistently the best in both median ACE and OR. SURF is most sensitive to illumination shift, experiencing the greatest degradation to both median ACE and OR as illumination shift intensity increases.

Table 4.3.: Median ACE and OR in Illumination Shift. Depicted are the ACE and OR for each method in illumination shift scaled to $\lambda = 1.2$, $\lambda = 1.4$, and $\lambda = 1.6$. SIFT has a lower median ACE at $\lambda = 1.2$ and $\lambda = 1.4$, but HH proves to be more robust with a lower median ACE at $\lambda = 1.6$. Additionally, HH has the lowest OR at every illumination shift intensity.

Illumination Factor λ	1.2	1.4	1.6
Method	Median ACE / OR		
DH	23.08 / ≤ 0.01	26.45 / ≤ 0.01	28.38 / ≤ 0.01
HH	2.35 / ≤ 0.01	2.86 / ≤ 0.01	**5.50** / ≤ 0.01
SIFT	**0.95** / 0.10	**1.35** / 0.17	10.70 / 0.47
SURF	3.41 / 0.12	5.65 / 0.26	NAN / 0.67
ORB	8.48 / 0.18	11.92 / 0.26	48.26 / 0.50

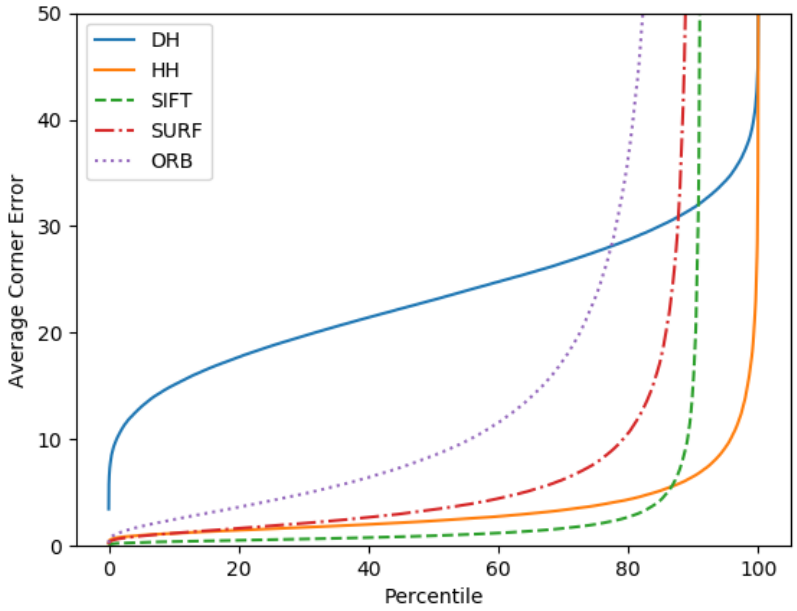

Fig. 4.5.: Sorted ACE for Illumination $\lambda = 1.2$. This figure depicts performance of each method using sorted ACE values, by least to greatest. SIFT has the lowest ACE for 90% of the dataset, after which HH has the best ACE. DH is the most sensitive to illumination shift, even at the "low" intensity of $\lambda = 1.2$, and therefore has the worst ACE for approximately 75% of the dataset.

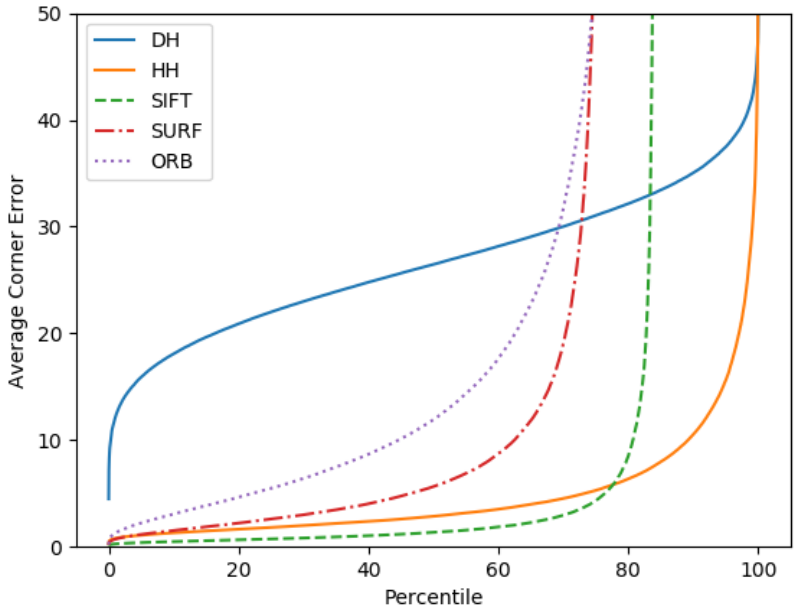

Fig. 4.6.: Sorted ACE for Illumination $\lambda = 1.4$. This figure depicts performance of each method using sorted ACE values, by least to greatest. SIFT has a lowest ACE for 83% of the dataset, with HH having the lowest for the remaining 17%

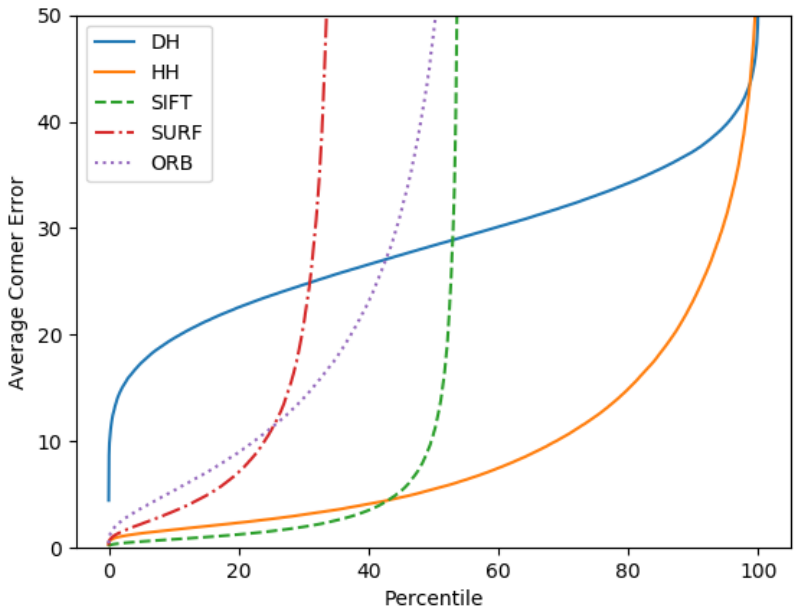

Fig. 4.7.: Sorted ACE for Illumination $\lambda = 1.6$. This figure depicts performance of each method using sorted ACE values, by least to greatest. HH has the lowest ACE for 56% of the dataset, as well as an OR below 0.01%. Although DH has the worst ACE for approximately 30% of the dataset, it is still better than any feature matching method for over half of the dataset.

4.4 Occluded Conditions

Relative to noise and illumination shift, feature matching is not as sensitive to occlusions. SIFT retains the lowest median ACE for all three levels of occlusion. Deep learning methods remain more consistent and have the lowest OR. This suggests that, as long as there is enough opportunity for distinct key points to be identified and matched, feature matching will experience little degradation.

HH has the lowest ACE values of the deep learning methods, and is less sensitive to increase in occlusion level.

SIFT performs better in ACE values and OR than any other feature matching method. ORB is the only method that severely degrades with increasing occlusion. This suggests that ORB does not detect as many key points, and therefore does not create as many correspondences, leading to more numerical instability in the solution.

Table 4.4.: Median ACE and OR in Occluded Conditions. Depicted are the ACE and OR for each method in occlusion scaled to $\alpha = 0.2$, $\alpha = 0.4$, and $\alpha = 0.6$. SIFT has the best median ACE for every amount of occlusion, and experiences only minor degradation of OR at $\alpha = 0.6$. Neither deep learning method exceeds an ACE of 50 for more than 0.01% of the dataset.

Occlusion α	0.2	0.4	0.6
Method	Median ACE / OR		
DH	4.74 / ≤ 0.01	6.99 / ≤ 0.01	11.23 / ≤ 0.01
HH	2.27 / ≤ 0.01	2.39 / ≤ 0.01	3.01 / ≤ 0.01
SIFT	**0.91** / 0.10	**0.99** / 0.13	**1.29** / 0.21
SURF	3.52 / 0.12	4.85 / 0.20	14.94 / 0.41
ORB	9.75 / 0.21	22.04 / 0.39	96.03 / 0.74

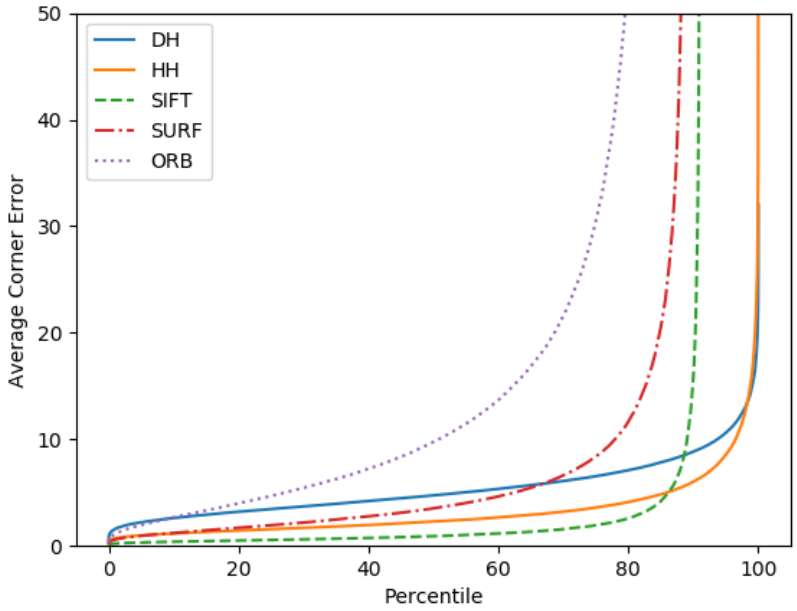

Fig. 4.8.: Sorted ACE for Occlusion $\alpha = 0.2$. This figure depicts performance of each method using sorted ACE values, by least to greatest. When 20% of the pixels are removed from the warped patch, the effect is minimal. The same trends found in the ideal conditions (Figure 4.1) are present in conditions of minor occlusion as well.

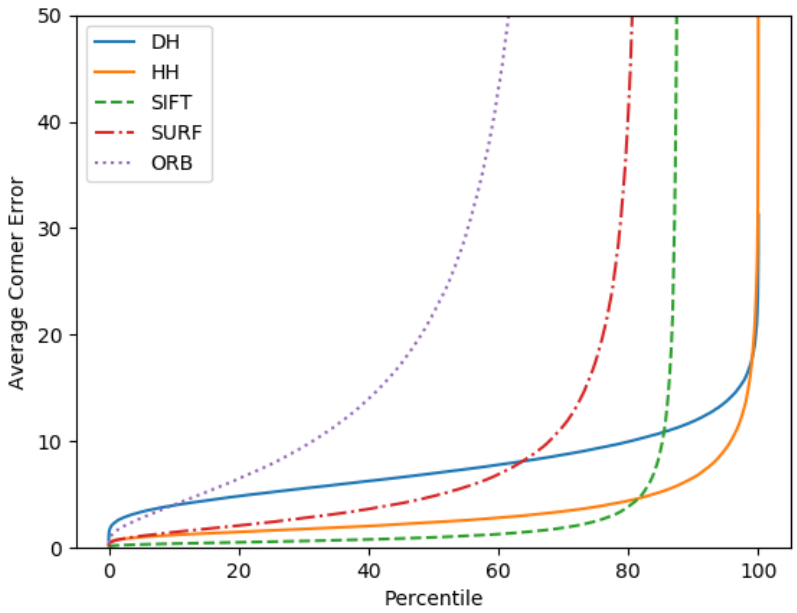

Fig. 4.9.: Sorted ACE for Occlusion $\alpha = 0.4$. This figure depicts performance of each method using sorted ACE values, by least to greatest. ORB is the only method significantly impacted by occlusions affecting 40% of the pixels in the warped patch. ORB exceeds an ACE of 50 for nearly half the dataset as a result, while the other methods remain near their levels at ideal conditions.

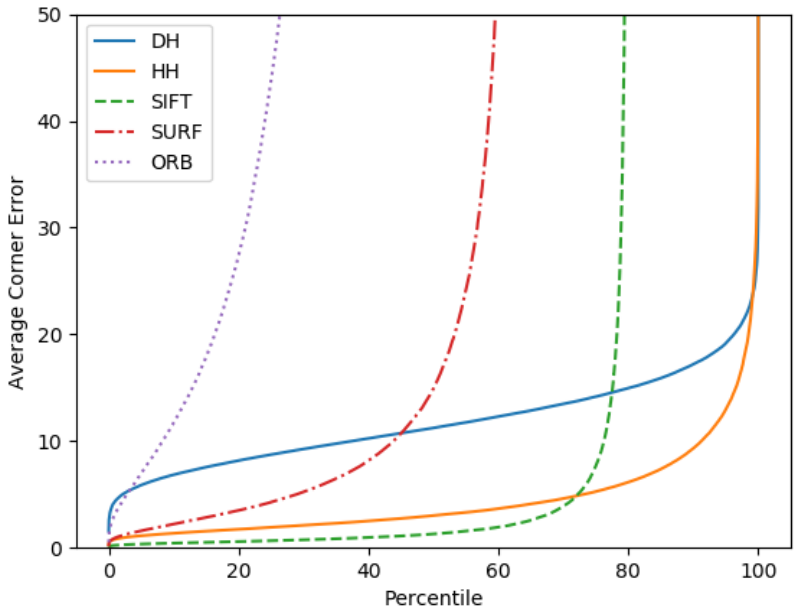

Fig. 4.10.: Sorted ACE for Occlusion $\alpha = 0.6$. This figure depicts performance of each method using sorted ACE values, by least to greatest. With over half of the warped patch removed due to occlusion, SIFT has the lowest ACE for nearly 80% of the dataset. The deep learning methods also experience relatively little change compared to ideal conditions. The impact on ORB is the most significant, as ORB is now viable for less than 20% of the dataset.

4.5 Additional Experiments

As shown in Table 4.5, training the DH CNN with color inputs instead of grayscale inputs resulted in a slightly larger median error for the ACE metric.

Table 4.5.: Comparison of DH Trained in Grayscale to DH Trained in Color.

Method	Median ACE	OR
DH Grayscale	3.97	0
DH Color	4.74	0

The performance plots are shown in Fig. 4.11. The graph for the grayscale case is consistently below the graph for color, implying that, with the same CNN architecture, the grayscale images result in slightly more accurate homographies than the color images.

Table 4.6 presents the results obtained when the DH is trained with different amounts of noise added to the training dataset. The CNN based homography estimator achieved significantly superior performance when trained to the specific level of noise. However, this performance enhancement came with a small degradation in the performance for the noiseless case (Ideal). Additionally, training at a lower level of noise only slightly improved the performance at higher levels of noise.

Shown in Fig. 4.15 are the percentile plots for the case when the three different DH networks, each trained with one of the three noisy input datasets as listed in Table 4.6), are tested on the same input-plus-noise data set corresponding to $\eta = 0.5$. Since the red-dotted plot for the case "DH Noise 50" is the slowest rising graph in the figure, this again shows that one gets the best results when the level of noise at which the DH is trained matches the level of noise in the input images.

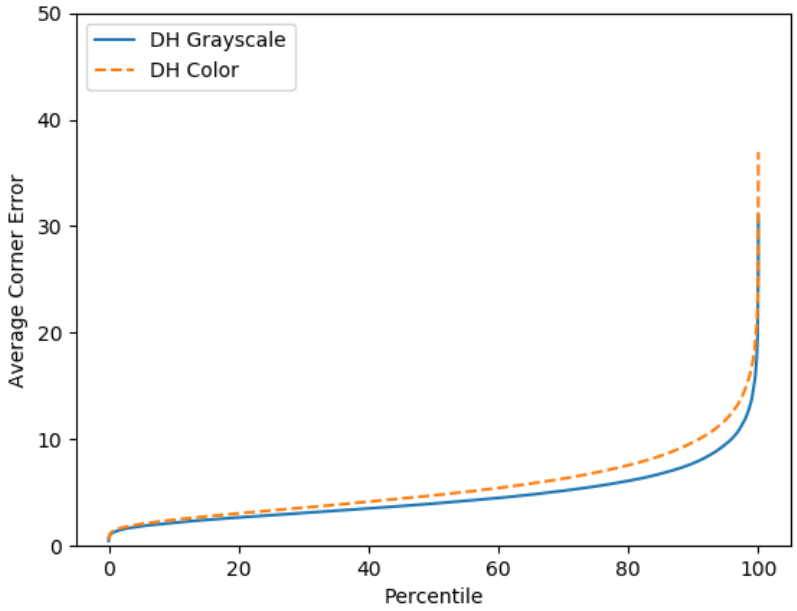

Fig. 4.11.: DH Trained In Grayscale vs DH Trained In Color

Table 4.6.: Deep Image Homography Trained With Noise. Depicted are the ACE and OR for Deep Image Homography CNNs trained to specific levels of noise, then compared in noisy conditions.

Noise η	0	0.1	0.3	0.5
Method	Median ACE			
DH	**3.97**	13.60	21.73	28.49
DH Noise 10	10.08	**4.24**	20.93	26.40
DH Noise 30	6.25	5.99	**4.99**	20.17
DH Noise 50	5.66	5.54	5.62	**5.70**

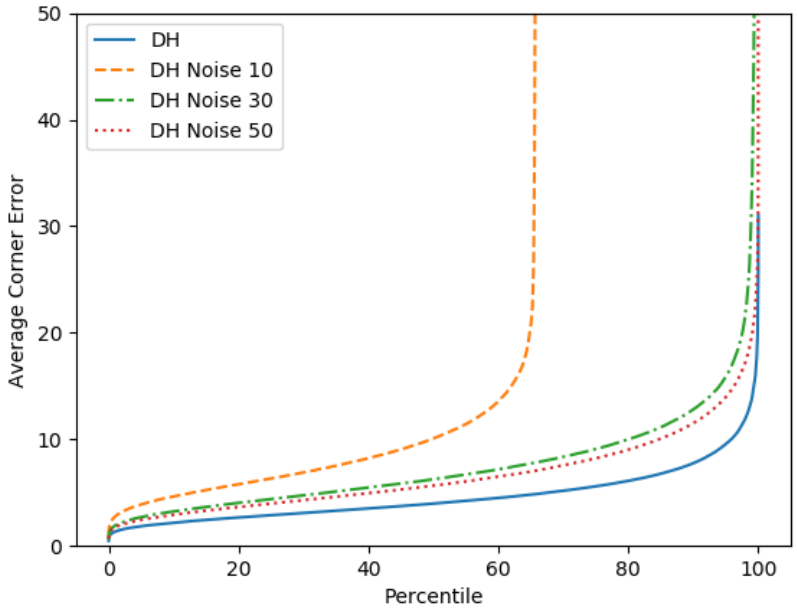

Fig. 4.12.: Sorted ACE values for DH Trained to Noise in Unaltered Dataset

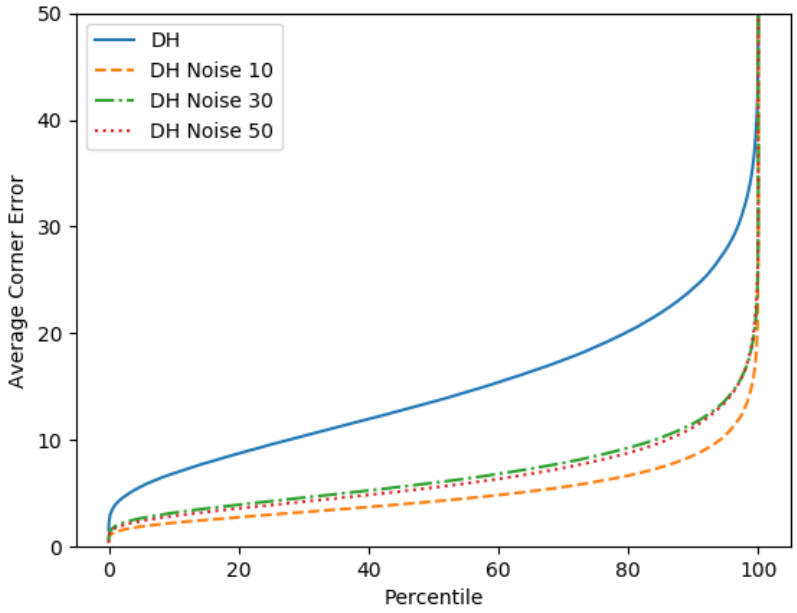

Fig. 4.13.: Sorted ACE values for DH Trained to Noise in Noise $\eta = 0.1$ Dataset

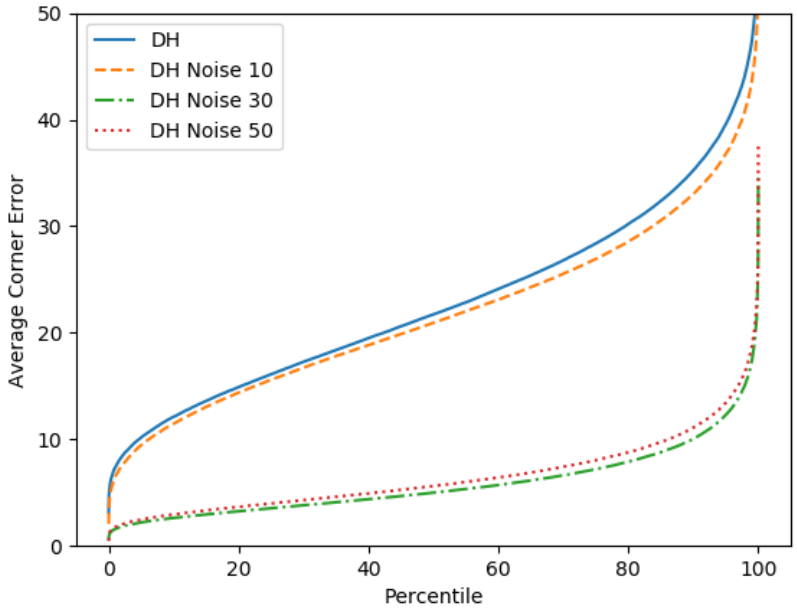

Fig. 4.14.: Sorted ACE values for DH Trained to Noise in Noise $\eta = 0.3$ Dataset

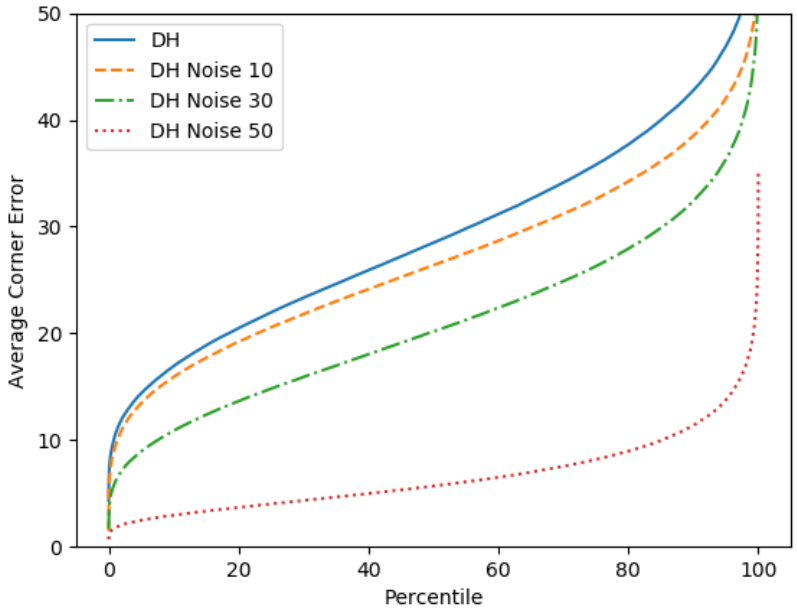

Fig. 4.15.: Sorted ACE values for DH Trained to Noise in Noise $\eta = 0.5$ Dataset

5. CONCLUSION

5.1 Discussion

In this paper, we analyzed the planar homography estimation performance of two deep learning solutions, "Deep Image Homography Estimation" by DeTone *et al.* and "Homography Estimation from Image Pairs with Hierarchical Convolutional Networks" by Nowruzi *et al.*, and three feature-matching solutions, including SIFT, SURF, and ORB. We conducted four experiments, each with a different form of variance applied to the dataset, to compare robustness of each method. The experiments were in ideal conditions, Gaussian noise, illumination shift, and random occlusions. Performance was measured with two metrics. The first was Average Corner Error (ACE), which is a measurement of accuracy based on average euclidean distance between corner points transformed by ground truth homography and corner points transformed by estimated homography. The second was Outlier Ratio, which is the size of the set of outputs with incredibly high error over the total size of the set of outputs.

The state-of-the-art feature matching is still the most accurate (per median ACE) in ideal conditions and certain conditions of minor variance. In particular, SIFT is the only method capable of achieving sub-pixel accuracy for over half the dataset, despite being the oldest method compared in this report. However, in conditions of moderate to heavy variance, particularly with Gaussian noise, deep learning methods achieve a lower median ACE. Feature matching methods consistently suffered higher error as a function of increasing variance, while the same variance had a much smaller impact on the deep learning methods. It should be noted that neither deep learning method was trained in conditions of variance. The robustness was learned even from training in ideal conditions.

No matter what conditions we applied, deep learning produced more consistent results than feature matching. Even in ideal conditions, SIFT produced outputs of tremendous error on 8% of the dataset. The other feature matching methods were even worse. The deep learning methods, however, produced such outputs for less than one percent of the dataset in every experiment. Many applications, such as real-time tracking for AR, might benefit in consistency over absolute error. For example, if the application feeds the output from one frame as input for the next, and it needs those outputs to be "close enough" to the ground truth, HH would be preferable to SIFT according to the results above.

Our experiments have also demonstrated that the robustness to noise for CNN-based homography estimators can be further improved by training with noisy data. Our experiments showed that the performance improved significantly at the specific level of noise with which the CNN trained. However, that improved performance comes at a slight degradation to accuracy in ideal conditions. There was only little improvement to performance in higher magnitudes of noise. Essentially, by training to a specific magnitude of noise, a "Goldilocks Zone" is created where that CNN performs best. More experiments need to be conducted to determine the ideal process for training in noise that optimizes performance across all potential magnitudes of noise.

Explicitly including color channels at the input for training CNN based homography estimators did not produce significantly better results for CNN based methods. Despite the additional information provided by the different color channels, the CNN-based homography estimation actually performed slightly worse. However, it is possible that a more elaborate architecture would yield higher accuracies for the homographies estimated from color images. Additionally, given the redundancy between the RGB channels of a color image, such inputs might increase robustness against noise. More research is necessary to find the optimal architecture and training methods that best leverages the additional information available in color images.

Performance aside, there are many other factors that further differentiate the various methods. SIFT and SURF, for example, are both patented, while ORB and deep learning methods are not. DH requires approximately 400MB of space to save the model parameters, and HH requires approximately 800MB of space for all four modules. While there are plenty of leaner architectures that could possibly be used without significant impact to performance, deep learning methods will require more space than a hand-crafted algorithm.

Ultimately, one method should not be treated as "universally superior" to any other, but instead environmental conditions and engineering constraints need to be deliberately considered when choosing the appropriate technique for any application. Deep learning is a viable solution to the homography estimation problem, and comes with many advantages. While not yet strictly as accurate as the best "hand-crafted" algorithms, they have a better break point and perform better in conditions of variance (noise, illumination shifts, and occlusions). Engineers need to consider acceptable error bounds, importance of error over consistency, variance conditions, etc. Although deep learning methods in homography estimation are not strictly superior than traditional feature matching, they are a perfectly valid tool for many circumstances.

5.2 Future Work

More research needs to be done to quantify the advantages and disadvantages of training a CNN in noise to enhance the robustness against that noise. An optimal training process should be developed that maximizes accuracy across the full spectrum of expected noise.

Using color inputs with CNNs carry potential for more accuracy and robustness to noise, given the increased amount of information and redundancy across the RGB channels. Research should attempt to leverage that information in the best way

possible. A larger number of parameters, particularly in the layers closest to input, could increase lead to better accuracies.

Future research should attempt to combine the robustness of deep learning implementations with the accuracy of feature matching. This could be accomplished in a few ways. One potential method could use a decision-based algorithm that determines the best method for estimation based on the conditions of input data. Properties of the input data can be used to determine levels of noise, illumination shift, etc. Such information could be used to determine the best homography estimation method to use for improved overall results.

Another method could use a homography determined by deep learning to "guide" a feature matching solution by narrowing possible locations for key point matches. In other words, a neural network could act as a rectification step. This would bound the error of overall homography estimation, ensuring that the worst case results in what a neural network could estimate, which is still generally reasonable, but also achieving the sub-pixel accuracy of feature matching in the best case.

Finally, continued research should be done to optimize neural network architecture to the homography estimation problem. CNN-based solutions to homography estimation are still relatively new, with much potential for improvement. Insights discovered with homography estimation through CNNs could apply to many other regression problems with image-based inputs.

REFERENCES

REFERENCES

[1] E. Dubrofsky, "Homography estimation," 2009.

[2] E. Vincent and R. Laganière, "Detecting planar homographies in an image pair," *ISPA 2001. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis. In conjunction with 23rd International Conference on Information Technology Interfaces (IEEE Cat.*, pp. 182–187, 2001.

[3] H. Liu, G. Zhang, and H. Bao, "Robust keyframe-based monocular slam for augmented reality," in *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, Sep. 2016, pp. 1–10.

[4] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. [Online]. Available: http://doi.acm.org/10.1145/358669.358692

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[6] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[7] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf." in *ICCV*, vol. 11, no. 1. Citeseer, 2011, p. 2.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[9] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg, "Matchnet: Unifying feature and metric learning for patch-based matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3279–3286.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[11] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Deep image homography estimation," *CoRR*, vol. abs/1606.03798, 2016. [Online]. Available: http://arxiv.org/abs/1606.03798

[12] T.-Y. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014.

[13] F. E. Nowruzi, R. Laganiere, and N. Japkowicz, "Homography estimation from image pairs with hierarchical convolutional networks," in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, Oct 2017, pp. 904–911.

[14] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, and V. Ferrari, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *arXiv:1811.00982*, 2018.

[15] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, S. Kamali, M. Malloci, J. Pont-Tuset, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy, "Openimages: A public dataset for large-scale multi-label and multi-class image classification." *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, pp. 448–456. [Online]. Available: http://jmlr.org/proceedings/papers/v37/ioffe15.pdf

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[19] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

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