Homography Estimation using Deep Learning for Registering All-22 Football Video Frames

HAMPUS FRISTEDT
Homography Estimation using Deep Learning for Registering All-22 Football Video Frames

HAMPUS FRISTEDT

Master in Computer Science
Date: June 19, 2017
Supervisor: Kevin Smith
Examiner: Tino Weinkauf
Swedish title: Homografiuppskattning med deep learning för registrering av bildrutor från video av amerikansk fotboll
School of Computer Science and Communication
Abstract

Homography estimation is a fundamental task in many computer vision applications, but many techniques for estimation rely on complicated feature extraction pipelines. We extend research in direct homography estimation (i.e. without explicit feature extraction) by implementing a convolutional network capable of estimating homographies. Previous work in deep learning based homography estimation calculates homographies between pairs of images, whereas our network takes single image input and registers it to a reference view where no image data is available. The application of the work is registering frames from American football video to a top-down view of the field. Our model manages to register frames in a test set with an average corner error equivalent to less than 2 yards.
Sammanfattning

Acknowledgements

I would like to thank Mikael Rousson and Signality for making this work possible. Mikael was immensely supportive and helped me greatly with his expertise. I would also like to thank the people at my school who helped me, most notably my supervisor Kevin Smith and examiner Tino Weinkauf.
# Contents

1 Introduction ............................................. 1
   1.1 Research question .................................. 3
      1.1.1 Objectives ...................................... 3
      1.1.2 Delimitations .................................... 4
      1.1.3 Related work .................................... 4

2 Background .............................................. 5
   2.1 Football video registration .......................... 5
   2.2 Homography .......................................... 7
   2.3 Convolutional neural networks ....................... 9
      2.3.1 Overview of convolutional neural networks .... 10
      2.3.2 Example networks ................................ 15
   2.4 Convolutional neural networks for homography estimation ............................................. 17

3 Methods ................................................... 19
   3.1 Dataset ............................................... 19
      3.1.1 Annotation ....................................... 20
      3.1.2 Generating ground truth homographies ......... 20
      3.1.3 Data augmentation ............................... 22
      3.1.4 Multiple ground truths .......................... 25
   3.2 Network architecture .................................. 27

4 Results .................................................. 30
   4.1 Data augmentation .................................... 30
      4.1.1 Augmentation by cropping ........................ 30
      4.1.2 Color augmentation ............................... 31
   4.2 Multiple ground truths ............................... 32
   4.3 Dropout ............................................... 33
   4.4 Batch normalization ................................... 34
Chapter 1

Introduction

Video registration is a fundamental problem in computer vision. It is the process of registering video frames into a canonical coordinate system defined by a reference image. Registering frames is necessary for any application that relies on the position of objects in a scene as the applications have no way to account for camera movements before frame registration. For example, an object might appear stationary if it moves with the same velocity as the camera observing it. Because of this reason, registration is a fundamental preprocessing step for most applications needing to analyze video taken with a non-fixed camera.

This work treats video registration in the context of American football (referred to as "football" hereafter). Specifically, we consider the problem of registering frames from All-22 football video to a top-down view of the football field (see figure 1.1).

The regular broadcast of football is filmed zoomed in and focused on the ball. This makes for entertaining footage, but is unsuitable when analyzing the game, as only the players currently close to the ball are visible. A better format is called All-22. All-22 is filmed primarily for coaches and the purpose of the format is to capture all active players (22 players; 11 on offense, 11 on defense). The camera is positioned high above the sidelines and looks down on the field. The camera undergoes pan, zoom and tilt transformations as the operator follows the ball. These transformations can all be represented by a homography. [13]
(a) A frame from All-22 video with annotated lines overlaid on top.

(b) The ground truth registration of the same frame to the canonical top-down view.

Figure 1.1: The red and blue lines represent the yard and hash lines respectively. The yellow dots are the intersection points between the lines. The hash lines run orthogonally through the hash marks.

A homography is a mathematical description of the relation between two views of the same plane. In our case, one of the views is the video frame and the other is the top-down reference view. The football field is the plane shared by both views. Homographies consist of translational and rotational transformations and can be calculated by manually selecting and extracting features that are common to both images, and calculating the transformation that maps the coordinates of the features in the target image to the coordinates of the same fea-
tures in the reference image. This process depends on the selected features and on how well these features can be extracted. Manually deciding the optimal selection of features to use is a very difficult task and the optimal selection varies greatly between applications.

Convolutional neural networks have, with the advances in GPU computing, become the state of the art for many tasks in computer vision. They perform very well at tasks such as image classification and detection. Inspired by these recent successes in deep learning, we will investigate whether homographies can be estimated directly, i.e. without manual feature selection, using convolutional neural networks. For this purpose, we will establish a dataset of All-22 frames and their corresponding homographies. The dataset will be used to train and evaluate homography estimating convolutional neural networks.

1.1 Research question

How can convolutional neural networks be used to directly register frames of All-22 football video to a top-down view of the football field?

1.1.1 Objectives

The objective of the work is to directly map All-22 frames to a top-down view of the football field. We do this to map frame coordinates to real-world positions. However, the problem will be relaxed by ignoring the horizontal alignment of the frames. A successful mapping will align the yard lines and hash lines of the target image with the yard lines and hash lines of the reference view of the top-down field, but the exact yard line in the reference view does not matter. The practical consequence of this is that all frames will be registered to the center of the field. The registered frames will be used in a later stage of the video registration pipeline (not in this work), where the registered frame will be horizontally aligned.

Another objective of the work is to investigate whether direct image registration (i.e. without explicit feature extraction) is possible using current deep learning techniques. Despite the recent successes in deep learning based image processing, little research has been made about the viability of CNN-based image registration and homography estimation.
1.1.2 Delimitations

As mentioned above, all frames will be registered to the center of the field, with horizontal alignment being managed in a post-processing step outside of this work. This formulation considerably reduces the amount of training data required.

Another delimitation is that frames will be mapped one-by-one, with arbitrary order, instead of registering the frames in sequence. Technically, this makes the problem an instance of image registration, as opposed to video registration, but the problems are closely related.

1.1.3 Related work

At the time of writing, we know of only one paper investigating homography estimation using CNNs [5]. While [5] is very similar to our work, we have a single input and want to map to a reference view where the reference is defined without image data, whereas [5] estimated homographies between pairs of images. We explain [5] in detail in section 2.4.
Chapter 2

Background

This chapter contains the theoretical background used in the work. We also use this chapter to present related work and previous research related to the problem. The chapter is structured as follows: First, we present related research regarding football video registration. Then, we explain the general concept of homography. This is followed by a section containing an overview of convolutional neural networks, including some examples of modern networks. The final section of the chapter presents a paper on homography estimation using convolutional networks.

2.1 Football video registration

The problem of football video registration has previously been approached in a variety of ways. This section contains a summary and description of these approaches.

In [17], Intille wrote one of the earliest treatments of football video registration. His most important contribution was his approach to calculating the homography between the frames and the common coordinate system. His approach was to use the points of intersection between the yard lines and the lines defined by the hash marks ("hash lines"), as well as the points of intersection between the yard lines and sidelines. Intille [17] stated that his approach would yield about forty points for a typical frame. The problem with Intille [17]’s work was that even though the features used were simple when compared to other registration problems, they were still difficult to consistently detect. For example, a yard line that was detected in one frame might be
obstructed by a player in the next frame which would cause it to appear as two line segments. Another problem was that lines that were visible in one frame might be out of view in the next. Intille abandoned his approach for manual registration in a later work [18], although he did not explain why.

Hess and Fern [15] advanced research on the problem in two ways. First, they approached the problem using recent techniques in local feature-based registration, as opposed to previous work that relied on global distinctive features. Second, to overcome the problem of features being globally non-distinctive, they introduced the concept of local distinctiveness. For example, the logotype in the center of the field is globally distinctive. Other features, like the sets of hash marks that span the entire field, despite being globally non-distinctive, provide a lot of information of the location on the field. These are the features that Hess and Fern [15] refer to as locally distinctive.

Hess and Fern [15] suggested two roles of the locally distinctive features. The first was to track features between frames. Because video is sampled at a high rate (e.g. 30 frames per second), features often move in small steps between frames. Hess and Fern [15] would exploit this by searching for a currently detected feature’s locally distinctive match in the following frame relative to the position of the currently detected feature.

The second role was to expand the set of matched model features by searching the region around a previously matched, globally distinctive feature for locally distinctive features. That is, once a set of features was found that were sufficient for a valid estimation of the registration transform, the same transform could be used to predict the future location of an unmatched feature. The neighboring region of the predicted location could then be searched for the feature, and if found, the set of feature correspondences could be extended.

Their results showed drastic improvements when adding locally distinctive features. The method of using only globally distinctive features averaged an error rate equivalent to 10 yards on the field, a significant amount. Adding their techniques for including locally distinctive features dropped the average error to around half a yard. However, the technique was not perfect. The error increased towards the end of the video, reaching a maximum equivalent to five yards. Hess and Fern [15] explained this as a possible effect of the fact that the camera often zooms in significantly towards the end of plays, to the point
Further improvements to the field came from Ghanem, Zhang, and Ahuja [11]. Their approach was to map entire images onto other images, without any explicit feature matching. The technique, named Robust Video Registration (RVR) by the authors, assumed nothing other than that outlier pixels (i.e. moving football players) are sparse. Ghanem, Zhang, and Ahuja [11] used the following formulation: Given a frame and a homography mapping that frame onto the following frame, the authors defined the error vector as the pixel difference between the mapped frame following transformation and the frame being mapped onto. This error vector was assumed to be sparse, and that the most correct homography maximized the sparseness of the error vector, i.e. minimized the $l_0$ “norm”. The problem was then relaxed by minimizing the $l_1$ norm instead of $l_0$, which reduced computational difficulty, and the problem was solved using a Lagrangian method. Ghanem, Zhang, and Ahuja [11] showed in their results that their RVR outperformed the standard SIFT [28] and RANSAC based method by an average of at least 10%. Also noteworthy is that the error of RVR was concentrated at the moving players, in contrast to the standard technique which showed registrations errors both at the players and the field lines. However, because Ghanem, Zhang, and Ahuja [11] registered frames onto preceding frames, as opposed to registering frames onto a canonical view of the football field, their technique required that the first frame be separately registered to the field. This is trivial to solve using manual registration, e.g. by point correspondences, but nevertheless, it is a problem that requires some attention. Another problem with their technique is that the solutions would drift from the initial homography as the errors accumulated over time.

### 2.2 Homography

A homography describes the relation between two images of the same plane. It can be used for image registration, image rectification and for calculating the movement of the camera that took the images. In the context of image registration, homographies are used to map the target image (the All-22 frame) onto the reference image (the top-down view).

Homographies are commonly represented by a 3x3 matrix with 8
parameters or degrees of freedom:

\[ H = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \]

\( h_9 \) is usually set to 1 to deal with the fact that the matrix has 9 elements, whereas the homography consists of only 8 parameters.

If \( p \) is a point in the target image, and \( p' \) is the corresponding point in the reference, then the homography matrix maps \( p \) onto \( p' \) like

\[ p' = Hp \quad (2.1) \]

where \( p = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \) and \( p' = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \) are 2D points with homogeneous coordinates.

Point correspondences between the reference image and the target image are often used to estimate \( H \). The algorithm for solving this problem is called the Direct Linear Transform (DLT). The following is a description of the DLT using the derivation provided by Dubrofsky [6].

By expanding equation 2.1 we get

\[
\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}
\]

\[
= \begin{pmatrix} h_1 x + h_2 y + h_3 \\ h_4 x + h_5 y + h_6 \\ h_7 x + h_8 y + h_9 \end{pmatrix}
\]

We can divide both the first and second row by the third row to get two equations

\[
-h_1 x - h_2 y - h_3 + u(h_7 x + h_8 y + h_9) = 0 \quad (2.2)
\]

\[
-h_4 x - h_5 y - h_6 + v(h_7 x + h_8 y + h_9) = 0 \quad (2.3)
\]

These equations can be written in matrix form

\[ A_i h = 0 \quad (2.4) \]
where

\[
A_i = \begin{pmatrix}
-x & -y & -1 & 0 & 0 & 0 & ux & uy & u \\
0 & 0 & 0 & -x & -y & -1 & vx & vy & v
\end{pmatrix}
\]

and \(h_9\) is set to 1 so

\[
h = (h_1 \ h_2 \ h_3 \ h_4 \ h_5 \ h_6 \ h_7 \ h_8 \ 1)^T
\]

Because each point correspondence \((p, p')\) gives us 2 equations, we only need 4 such correspondences to solve for all 8 parameters of the homography. That is, given 4 point correspondences with their matrices \(A_1, A_2, A_3, A_4\) we can stack the matrices \(A_i\) to get a single 8x9 matrix \(A\). The solution to the homography can then be found by solving

\[
Ah = 0
\] (2.5)

However, a more robust solution can be calculated numerically if more correspondences are available, e.g. by using least squares [17] or RANSAC [9, 15].

An alternative homography representation was investigated by Baker, Datta, and Kanade [2]. Instead of using the usual 3x3 matrix, 4 fixed points in one image can be combined with 4 variable points in the other to parameterize the homography. That is, because the matrix \(H\) can be calculated using the DLT, we only need the fixed points \(p_i\) and variable points \(p'_i\) to parameterize the homography. As \(p_i\) are fixed, we only need to store the 4 points \(p'_i\) to parameterize the homography.

Baker, Datta, and Kanade [2] performed a number of experiments and found that the 4-pt parametrization performed better than the 3x3 parametrization. However, they could not make any conclusions on whether the 4pt parametrization is the best parametrization or how the 4 fixed points are to be chosen. An additional benefit of the 4-pt parametrization is that it provides a geometrically intuitive representation of the homography.

### 2.3 Convolutional neural networks

This section is split into two subsections. The first subsection contains an overview of convolutional neural networks and the techniques used when working with them. The second subsection presents some examples of modern convolutional networks.
2.3.1 Overview of convolutional neural networks

Convolutional neural networks (CNNs) are the state of the art for many applications in computer vision. They have recently been used to great effect for solving tasks such as semantic segmentation [27] and image classification [23]. The following is a general description of CNNs.

LeCun et al. [24] conducted an early work on CNNs that applied CNNs and gradient-based learning to handwriting recognition. In addition to presenting a CNN for recognizing handwriting recognition (LeNet-5), they described the problems with using fully connected networks (also known as multilayer perceptrons, abbreviated MLP) for image classification. One of these problems is that MLPs have weights for every input. This is manageable for small images (e.g. a 32x32 RGB image means 32x32x3=3072 weights in the first layer), but it quickly becomes a problem as image resolution increases, both in terms of complexity and overfitting. Also, MLPs do not account for spatial structure in that input pixels that are far apart are treated the same as pixels that are close to each other. This makes MLPs unsuitable for detecting the local features that are required when processing image data.

LeCun et al. [24] described three architectural ideas that make CNNs suitable for image processing: local receptive fields, shared weights, and sub-sampling. Local receptive fields are conceptually similar to neurons in a cat’s visual system, described by Hubel and Wiesel [16]. They let neurons detect local features such as edges, points and curves. Following layers can then use these features to describe higher-order features. The idea is to use collections of several small filters in convolutional layers. Filters process small neighborhoods of the input and output the dot product of the input and the values (weights) of the filter. The output of a filter is called a feature map and describes the feature detected by the filter.

If detected features are useful in one part of the input, they are probably useful in another part of the input. This knowledge is applied to CNNs as the weights of a filter do not change when the input region changes. This is called weight sharing.

Non-linearity is provided to the network by non-linear activation functions. The activation functions follow the other layers in the network (convolutional or fully-connected) and the choice of function can impact the performance of the network. Sigmoid, e.g. \( f(x) = \tanh(x) \)
or \( f(x) = (1 + e^{-x})^{-1} \), were long used as the standard way of modeling a neuron’s output, but the success of networks such as AlexNet \([23]\) has made the simpler Rectified Linear Unit (ReLU, \( f(x) = \max(0, x) \)) popular. Besides being simpler, ReLUs have the benefit of increasing the learning rate of the network as shown by Krizhevsky, Sutskever, and Hinton \([23]\). Sigmoids can also be saturated, which can slow down the learning of the network. ReLUs avoid this problem by scaling linearly for values \( > 0 \).

Sub-sampling is handled by pooling layers. Pooling layers downsample the input non-linearly. The most common type of pooling layer is max pooling, but earlier works such as \([24]\) used mean pooling. Pooling layers split the input into sub-regions, and output a value for each sub-region. The value is determined by the pooling function, e.g. the mean of the sub-region for mean pooling. Sub-sampling is done to reduce the size of the feature maps, which reduces the number of parameters and the importance of a feature’s exact position in the input, i.e. to increase translation invariance.

Fully-connected layers are often used as the last stage of CNN. For example, the VGG networks created by \([33]\), have three fully-connected layers before the output layer. A fully-connected layer has connections for all activations in the previous layer. In the context of CNNs, fully-connected layers are used so that the network can learn a function of the previously learned visual features.

![Figure 2.1: Figure from LeCun et al. \([24]\) depicting LeNet-5: an early CNN for handwriting recognition.](image)

To train the network, a loss function is required. A loss function takes a ground truth and a prediction, and outputs a value, a loss, describing the size of the prediction error. During training, the optimizer will attempt to minimize the loss by tuning the parameters of the network. The choice of loss function varies depending on the application.
Classification networks commonly use the Softmax loss, while regression networks often use the Euclidian loss.

Most deep learning libraries contain several optimization algorithms, or optimizers, that are used for training the network. The majority of the algorithms are based on the concept of gradient descent. Gradient descent is based on taking steps in the negative direction of the gradient at the current point, as that direction describes the steepest descent towards the local minimum. The size of the steps is called the learning rate and determines how fast to approach the minimum. If the learning rate is too large, the optimizer risks stepping over the minimum, and in the worst case, the optimizer gets stuck continuously skipping across the minimum. Some optimizers, such as Adagrad [7], RMSProp [36] and Adam [21] attempt to minimize the risk of this happening by having adaptive learning rates.

When using gradient descent, we update the parameters of the model after the entire training set has been used. This method provides the most accurate estimation of the gradient, but because the parameters are updated so rarely the model can become slow to train. The solution to this is to split the dataset into smaller mini-batches, feed these to the network, and update the weights after each mini-batch. This approximation of gradient descent is called stochastic gradient descent (SGD). The downside with using SGD and mini-batches is that it reduces the accuracy of the gradient estimation, but as Bengio [3] pointed out, even the true gradient direction is only the steepest descent locally so it is often better in practice to use smaller mini-batches and update weights often, rather than focus on accurate gradient estimations with fewer weight updates. A sweep over the entire dataset is often referred to as an epoch.

A common problem in machine learning is overfitting. A system that has overfitted has adapted to noise in the training data, causing the system to perform poorly on validation and test data. Overfitting can be caused by having a training dataset that is too small or an overly complex model. Methods that combat overfitting are commonly referred to as regularization methods. One of the simplest regularization methods when training neural networks is called dropout [34]. Dropout refers to the technique of temporarily removing network units during training. The choice of which units to drop is decided at random, where each unit has a probability $p$ of being retained, with $p = 0.5$ being a commonly used value for hidden units. This amounts to a col-
lection of "thinned" networks being trained, where the networks share parameters and each network is rarely trained. Despite the simplicity of the technique, dropout has proven very useful. Srivastava et al. [34] compared networks with and without dropout on multiple datasets, including the popular MNIST, CIFAR-10 and ImageNet datasets. They found that dropout improved the generalization performance on all datasets when compared with the networks that did not use dropout. A drawback with dropout is that it increases training time. Srivastava et al. [34] noted that a dropout network typically takes 2-3 times longer to train than its standard counterpart.

Typically, dropout has only been applied to the fully-connected layers in CNNs. However, some recent studies have suggested that dropout can be successfully used even on convolutional layers. For example, Gal and Ghahramani [10] described a Bayesian CNN with Monte Carlo dropout that improved the state-of-the-art results for CIFAR-10 [22]. Srivastava et al. [34] even managed to use "standard" dropout (as opposed to Monte Carlo dropout) on convolutional layers successfully, improving a network classifying the Street View House Numbers (SVHN) dataset [30]. However, the results required some manual parameter tweaking. In convolutional layers close to the input layer, $p$ was set high ($p = 0.9$), to prevent the network from discarding necessary features too early, while later layers used lower values for $p$.

A common preprocessing step in deep learning is to normalize the input data. Normalization refers to the process of transforming the data into a format that is centered at a preferred value, scaled to a preferred range. In the case of normally distributed inputs, this is done by subtracting the mean and dividing by the standard deviation, making the data zero-centered and normalized

$$\tilde{x} = \frac{x - \mu_x}{\sigma_x}$$

This can be beneficial for several reasons. For example, when considering CNNs and image input, images might be provided with varying contrast and exposure, in which case subtracting the mean would equate the exposure, and dividing the standard deviation would equate the contrast. Given enough data, neural networks can learn to ignore contrast and exposure, but normalization can reduce the difficulty of training [32].
Ioffe and Szegedy [19] even found that normalization was beneficial when added to each layer’s input. The problem they wanted to solve, which they referred to as *internal covariate shift*, is that the input distributions between layers changes as parameters were updated in previous layers, making saturating non-linearities (e.g. Sigmoids) difficult to train and requiring lower training rates. They addressed this by introducing *batch normalization*.

Batch normalization is the process of performing normalization for each mini-batch, and introducing two learnable parameters $\gamma$ and $\beta$ so that

$$ y_i = \gamma \left( \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \beta $$

where $y_i$ is the "batch normalized" value of input $x_i$, and $\mu_B, \sigma_B$ are the mean and standard deviation over the batch. $\epsilon$ is a small constant added for numerical stability. The effect of the formulation is that it enables the network to learn to which degree normalization should be applied. In the case where the raw input would yield a lower loss, the network can set $\gamma$ to $\sqrt{\sigma_B^2 + \epsilon}$ and $\beta$ to $\mu_B$, resulting in $y_i = x_i$.

The results of Ioffe and Szegedy [19] were impressive. They managed to improve upon the previous state-of-the-art in ImageNet [4] classification by attaching batch normalization to an ensemble of Inception-type networks. [35] An additional benefit of batch normalization pointed out by Ioffe and Szegedy [19] is that it regularizes the model, which in their experiments meant that dropout could be either reduced or removed entirely.

Many state-of-the-art benchmarks are set using ensembles of networks. Ensembles are typically built of multiple networks where the networks vary in some way, e.g. by different initializations. The results of the ensembles are constructed from the average of the network outputs. Lee et al. [25] investigated the optimal strategy of creating ensembles by comparing different ensemble strategies and by proposing new methods.

Typically, ensemble members are trained independently, but Lee et al. [25] introduced a unified, ensemble-aware loss that enabled ensembles to be trained collectively. The loss was based upon Multiple Choice Learning [12]. We will now summarize the description of MCL from Lee et al. [25].
Given a dataset \( D = \{(x_1, y_1), \ldots, (x_N, y_N)\} \), where \( x_i \in X \) represents the feature vectors and \( y_i \in Y \) the ground truth vectors, as well as set of predictors \( \{\theta_1, \ldots, \theta_M\} \) such that \( \theta_m : X \to Y \). With MCL, the goal is to minimize the oracle loss, \( L(x, y) = \min_{m \in [1, M]} \ell(\theta_m(x), y) \), of the predictors for samples \( x \in X \) and \( y \in Y \). Lee et al. [25] used this by feeding every sample in the dataset to every member of the ensemble, but the gradient descent update was performed using only the subset of samples on which each member achieved the lowest loss. See figure 2.2 for details.

Figure 2.2: Algorithm from the appendix of Lee et al. [25] describing their approach integrating MCL with Stochastic Gradient Descent (SGD).

This resulted in ensemble members diversifying and specializing at classifying different labels. For example, Lee et al. [25] found that one member specialized at cats and another member at cars when they trained an ensemble on CIFAR-10 [22]. The individual networks performed poorly (accuracy of 19-27%), but collectively the ensemble performed well (oracle accuracy of 93%).

### 2.3.2 Example networks

The VGG networks were developed by Simonyan and Zisserman [33]. The aim of the networks was to investigate how CNN depth effects accuracy and the networks performed very well in both localization and classification tasks. The VGG networks used multiple 3x3 filters (the smallest size able to represent up/down, left/right and center) stacked
depthwise as opposed to single large filters. For example, three layers with 3x3 filters (without pooling between layers) effectively function as a single 7x7 filter, with the advantage of having fewer weights. Also, because each layer can be followed by a ReLU, the stack of 3x3 filters have the additional advantage of being able to express more complex non-linear functions. Another characteristic trait of the VGG networks is that the number of filters was doubled following each pooling layer, until the number of filters reached 512. This has the effect of keeping the time complexity consistent between layers, as every time the feature map is halved, the number of filters is doubled.

The results of [33] showed that network depth is beneficial for classification accuracy. However, other researchers found that increasing depth can have surprising consequences. In their paper describing their Residual Networks (ResNets), He et al. [14] explained that adding layers of a sufficiently deep network often increased the training error. They called this the degradation problem. This was surprising because networks should be able to learn identity mappings (that is, learn the ability to make redundant layers output their input without modification; \( f(x) = x \)), but the experiments of He et al. [14] showed that the identity mappings were difficult to learn in practice. Their solution was to introduce deep residual learning. Instead of having stacks of layers that directly estimate the desired mapping \( H(x) \), they reformulated the problem using the residual mapping \( F(x) = H(x) - x \) and let the stack of layers learn the mapping \( H(x) = F(x) + x \). Their hypothesis was that the residual mapping would be easier to learn than the direct mapping. The reasoning being that, if an identity mapping was desired, it would be a lot easier to learn a zero residual than it would be to learn an identity mapping through several non-linearities.

![Figure 2.3](image)

Figure 2.3: Figure from He et al. [14] depicting their residual learning blocks.
In practice, this was done by introducing shortcut connections between layers. The shortcut connections provided the identity mapping $x$ and result of the network layers provided $F(x)$. The two parts were combined by addition, which meant that if the identity mapping was desired, the network simply had to zero all the weights in the layers.

The residual learning formulation proved successful. He et al. [14] showed that two 18 layer networks, one with shortcut connections (ResNet) and one without, were able to perform equally well at ImageNet [4] validation, but the ResNet was able to converge faster. The differences between the ResNets and their counterparts increased as the networks grew in depth. At 34 layers, the ResNet showed a top-1 error that was 3.5% lower than its plain counterpart on ImageNet validation. The ResNets scaled well even to extreme depths. He et al. [14] used an ensemble including two 152-layer ResNets to win 1st place in the Large Scale Visual Recognition Challenge 2015 (ILSVRC2015).

### 2.4 Convolutional neural networks for homography estimation

DeTone, Malisiewicz, and Rabinovich [5] investigated how homographies can be directly estimated using CNNs. They proposed and evaluated two VGG16-like [33] networks. Both networks were identical until the final layers, where the first variation was a simple regression network that output 8 real values describing the homography and the second was a classification network that used 21 quantization bins for each of the 8 output dimensions, resulting in 168 output neurons.

DeTone, Malisiewicz, and Rabinovich [5] used the offset between the corner positions of the target and reference images to parametrize the homographies, yielding a representation based on the 4-pt homography parametrization that Baker, Datta, and Kanade [2] investigated. Their training data was generated by cropping 128x128 greyscale patches from the MS-COCO dataset [26]. For each patch $A$ cropped at position $p$, they randomly perturbed the corners to generate a transformed copy. Then, the homography $H^{AB}$ between patch $A$ and its transformed copy was computed. The homography $(H^{AB})^{-1}$ was applied to the entire image, and a new patch $B$ was cropped from the transformed image at position $p$. The patches $A$ and $B$ were then stacked channel-wise and fed into the network, with homography $H^{AB}$ as the
target vector.
While the classification network was more complex than the regression network, it was able to produce a confidence for each of the corners. Their results showed that the regression network performed significantly better when the corner confidences are not required.

![Figure 2.4: Figure from DeTone, Malisiewicz, and Rabinovich [5] depicting their homography estimating networks.](image)

Their results also showed that the regression network slightly outperformed a “traditional” homography estimator. The traditional estimator was based on ORB [31, 29] (a feature descriptor designed to replace SIFT) and RANSAC [9]. The authors also pointed out that it is simpler to train a network for application-specific homography estimation than it is to optimize an ORB + RANSAC based estimator for a specific application, which probably is the largest benefit of using the CNN-based approach.
Chapter 3

Methods

In this chapter we describe the methods used for creating the dataset and the network we used.

3.1 Dataset

The frames used were RGB images with dimensions 640x360 that were downscaled to half the resolution: 320x180. The ground truth homographies were represented using the 4-pt parametrization described by Baker, Datta, and Kanade [2], flattened into a 1D vector for increased ease of use with the CNN.

The dataset was split into three parts: one part each for training, validating and testing. The validation and test sets both consisted of frames from games that were not seen by the network during training to ensure that low losses would indicate a model that generalized well. The validation set was used to pick hyper-parameters and to adjust the model, while the test set was only to generate results for the work. To avoid cherry picking results, we only used the test set once and did not modify the model after the test set was used. The frames in the dataset were taken from video of 18 games. 15 games were used for the training set, 1 game for validation and 2 games for the test set.

Many frames in the dataset were outliers. Frames near the end-zones required large transformations to align, and other frames were close up shots with very few visual features useful for registration. This had a large impact on the training performance of the network. Some annotations were also faulty. This threw off the optimizer even more. To remedy these issues, we enforced that that each frame should
contain at least 10 intersection points between yard lines and hash lines. However, we did lower this threshold for the training set to 8 after experiments showed that the extra frames contained features that were beneficial to lowering the validation loss. We also removed frames requiring unreasonably large transformations by filtering frames with transformations where the maximum offset between the target point and reference point exceeded a threshold value found by visually inspecting the transformations. This second step mostly removed incorrectly annotated samples.

The size of the datasets after removing the difficult outliers was 2828 for the training set, and 218 for the validation set, where 1179 frames were removed from the training set and 90 frames were removed from the validation set due to not satisfying the above mentioned conditions. The test set was processed similarly. It contained 347 frames after cleaning.

### 3.1.1 Annotation

A web based tool was used for annotating the All-22 frames. Both the tool and the actual annotation was provided by the principal.

Each frame was annotated with the position of the yard lines and hash lines visible in the frame. The line positions were stored as pairs of points, i.e. each line was defined by the tuple \((x_1, y_1), (x_2, y_2)\). The hash lines and yard lines were stored separately.

### 3.1.2 Generating ground truth homographies

Each frame \(F\) was represented as a matrix with dimensions \(w_f \times h_f \times c\), where \(w_f\) was the width of the frame, \(h_f\) the height and \(c\) the number of channels. The following steps were performed to find the ground truth homographies.

1. The set of intersection points \(I\) between the yard lines and hash lines was calculated using the annotations. The Line-Line intersection algorithm of Weisstein \cite{37} was used to find the points of intersection. That is, given a yard line \(L_y\) defined by the points \((x_1, y_1), (x_2, y_2)\) and a hash line \(L_h\) defined by the points \((x_3, y_3), (x_4, y_4)\), the intersection point \(p\) of \(L_y\) and \(L_h\) is defined by
2. The homography matrix $H$ that maps $I$ onto the set of reference points in the top-down view was found using the `findHomography` function of the OpenCV library \[20\]. The function can use either least-squares, RANSAC or least-median to find the homography. We used the default method: least-squares.

3. The homography was applied to the fixed points. The fixed points were the 4 corners of the patches, given in homogeneous coordinates

$$R = (0, 0, 1), (w_p, 0, 1), (0, h_p, 1), (w_p, h_p)$$

where $w_p$ and $h_p$ were the width and height of the cropped patches (We expand upon the patches in the next section on data augmentation). That is, for each point $P_i$, $i = 1, ..., 4$ in the set of corner positions $R$, the transformed corner position $P^h_i = (x^h_i, y^h_i, z^h_i)$ was calculated:

$$P^h_i = HP_i$$
4. Let $P^c_i$ denote point $P^h_i$ in Cartesian coordinates. Then,

$$P^c_i = (x^c_i, y^c_i) = \left( \frac{x^h_i}{z^h_i}, \frac{y^h_i}{z^h_i} \right)$$

5. The points were then arranged to generate a 1D vector representing a flattened version of the 4-pt parametrized homography:

$$H_{4\text{-pt}} = \begin{pmatrix} P^c_1 & P^c_2 & P^c_3 & P^c_4 \end{pmatrix} = \begin{pmatrix} x^c_1 & y^c_1 & x^c_2 & y^c_2 & x^c_3 & y^c_3 & x^c_4 & y^c_4 \end{pmatrix}$$

$H_{4\text{-pt}}$ could then be used as the ground truth for the network. See figure 1.1 for an example of a frame and its registration to the reference view using the ground truth homography.

### 3.1.3 Data augmentation

The dataset was relatively small compared to other datasets used for training CNNs. To prevent the network from overfitting the training data, we performed two types of data augmentation: augmentation by cropping and color augmentation.

#### Augmentation by cropping

The dataset was augmented by cropping patches from the image. The patches had dimensions $w_p \times h_p = s w_f \times s h_f$, where the scale parameter $s$ was set to 0.9. The ground truth homographies were stored using the 4-pt parametrization with fixed reference points. The fixed reference points were the dimensions of the patches, represented using the homogeneous coordinates of the patch corners:

$$r_1 = (0, 0, 1)^T$$
$$r_2 = (w_p, 0, 1)^T$$
$$r_3 = (0, h_p, 1)^T$$
$$r_4 = (w_p, h_p, 1)^T$$

The original ground truth homography, $H$, was defined by
\[ p_i = H r_i \]

for \( i = 1, 2, 3, 4 \). Since the points \( r_i \) remained constant for all patches, \( H \) had to be updated for every patch. Thus, for every new patch, an updated homography \( H' \) needed to be computed so that

\[ p'_i = H' r_i \]

with \( p'_i \) representing the updated target points for the patch.

We generated patches by randomly choosing the top-left corner, \((s_x, s_y)\), with \( s_x \in [0, w_f - w_p] \) and \( s_y \in [0, h_f - h_p] \). The corner coordinates of the patch relative to the frame coordinates were defined as

\[
q_1 = (s_x, s_y, 1)^T \\
q_2 = (s_x + w_p, s_y, 1)^T \\
q_3 = (s_x, s_y + h_p, 1)^T \\
q_4 = (s_x + w_p, s_y + h_p, 1)^T
\]

so

\[
p'_i = H q_i \\
= H' r_i
\]

Note that it was sufficient to compute \( p'_i \) during training. \( H' \) was computed post-training using the `findHomography` function of the OpenCV library [20], but this was not necessary during training because the 4-pt parametrization was used.
Figure 3.1: Both homographies map the corners of the patch to the target positions, but $H$ uses the coordinates relative to the frame corners. The solid line polygons depict the full frame, while the dashed polygons depict a cropped patch.

The entire training set of frames was used every epoch of training by creating a maximum of 20 random patches per frame. Randomly cropped patches were discarded if they contained less than the threshold set for the minimum number of intersection points. Each frame was also used to create a center cropped patch, in addition to the random patches. The center crop was also used for the frames in the validation and test sets. The set of generated patches was shuffled before being fed to the network.

**Color augmentation**

We performed color augmentation in the manner of Krizhevsky, Sutskever, and Hinton [23]. The technique is based on performing Principal Component Analysis (PCA) on the set of RGB pixel values from the training set. That is, we treated every pixel in the training set as a data point. We then calculated the $3 \times 3$ covariance matrix of the set of data points and performed PCA using the matrix to get the eigenvalues $\lambda_i$ and eigenvectors $p_i$ where $i = 1, 2, 3$. Then the principal components were added to pixel values of the images, scaled by a random Gaussian $\alpha_i$. 
and the square root of the corresponding eigenvalues $\lambda_i$. That is, to every pixel in the training set we added the following 3D vector:

$$(p_1, p_2, p_3)(\alpha_1\sqrt{\lambda_1}, \alpha_2\sqrt{\lambda_2}, \alpha_3\sqrt{\lambda_3})^T$$

Krizhevsky, Sutskever, and Hinton [23] sampled their $\alpha_i$ with zero mean and standard deviation 0.1, but we found that slightly larger deviations resulted in more useful augmentations. We used standard deviation 0.7. Another difference from their work is that we used the square root to bring the eigenvalue into the range of the standard deviation, as opposed to the variance, while [23] used the eigenvalues without modification.

We used the color augmentation technique by applying the PCA perturbations to every patch, every epoch. New random values $\alpha_i$ were sampled for each patch.

### 3.1.4 Multiple ground truths

The problem was not well formed when ignoring the horizontal alignment of the frames. Two almost identical frames could potentially have very different ground truth homographies if one of the frames contained an extra yard line at the very edge of frame. This would make the network difficult to train as a correct registration that was horizontally shifted by five yards could potentially yield a high loss. The problem became even bigger when adding data augmentation by cropping as the cropping often removed intersection points.

To decrease the severity of this problem, 3 ground truth homographies where used for each frame, where the only difference between the homographies was that they registered the frame with a horizontal shift equivalent to 5 yard lines.
The multiple ground truths were used both during training and testing. After each epoch, the ground truth vectors were updated for each input to be set to the ground truth homography that yielded the smallest loss for the prediction.

While the technique of modifying the ground truth vectors during training might appear unorthodox, it is in effect very similar to the unified loss proposed by Lee et al. [25] and other MCL-based [12] techniques. Where Lee et al. [25] used the oracle loss of MCL to encourage ensembles to diversify, we used a similar formulation to improve the robustness of our single network.

Instead of minimizing the loss

$$L_a(x, y) = \min_{m \in [1, M]} l(\theta_m(x), y)$$

over multiple predictors $\theta_m$, $m \in [1, M]$ like [25], we essentially minimized the loss

$$L_b(x, \{y_1, ..., y_K\}) = \min_{k \in [1, K]} l(\theta(x), y_k)$$

over the multiple ground truths $y_k$. The problem with using $L_b$ directly in the network is that the min-operation is not differentiable, which is a requirement for gradient descent.

To overcome this, the optimizer used the differentiable loss $l(\theta(x), y_K)$ where we updated $y_K$ between epochs such that
$K = \arg \min_{k \in [1, K]} l(\theta(x), y_k)$

This method required an additional predict per epoch for every training sample, a computationally cheap operation.

A benefit of using 4-pt parameterized homographies is that they made the multiple ground truths simple to implement. Instead of explicitly computing and storing the three ground truths, only the center aligned homography $H_c$ was required.

$$H_c = (x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$$

The left and right aligned homographies, $H_l$ and $H_r$, could be computed when necessary by subtracting respectively adding a horizontal shift.

$$H_l = H_c - \Delta x$$

$$H_r = H_c + \Delta x$$

where $\Delta x = (\alpha, 0, \alpha, 0, \alpha, 0, \alpha, 0)$ and $\alpha$ equaled the distance in the reference view equivalent to five yards.

### 3.2 Network architecture

We started our experiments using the mean squared error (MSE) between the prediction and the ground truth as the loss function:

$$l_{\text{MSE}}(\theta(x), y) = \frac{1}{8} \sum_{i=1}^{8} (\theta(x) - y)^2$$

where $\theta(x)$ is the predicted homography and $y$ the ground truth homography for every patch $x$.

The MSE greatly penalizes outliers, and we found that the training set contained several outliers not representative of actual data, mostly due to incorrectly annotated frames. While the cleaning operations described in section 3.1 removed many such frames, some remained and disturbed the optimizer noticeably. Therefore, we switched to the mean absolute error (MAE) for optimization.
\[ l_{\text{MAE}}(\theta(x), y) = \frac{1}{8} \sum_{i=1}^{8} |\theta(x) - y| \]

However, we kept the MSE as a metric for some experiments to enable comparison between the methods. We found that minimizing the MAE instead of the MSE increased both the speed of convergence, as well as minimum validation loss achieved.

To put the losses into perspective, we note that one unit in the top-down reference view equaled 1/6 yard. Thus, an MAE of 6 would indicate an average corner error of a yard.

Tensorflow [1] was used to implement the network. The optimizer used was the Adam optimizer described by Kingma and Ba [21] with default parameters. The batch size was 32.

The network was based on the VGG-like regression network described by DeTone, Malisiewicz, and Rabinovich [5]. The convolutional layers contained 3x3 filters and the number of filters, \( f = 16 \), doubled following each pooling layer. All convolutional and fully connected layers used ReLUs for activation, except the output layer that used linear activation. The pooling layers used 2x2 max pooling with stride 2.

Several variations of the network were evaluated. The baseline network is pictured in figure 3.3.

We evaluated a variation where dropout was added after layers FC1 and FC2, like in VGG [33], with probability \( p = 0.5 \) of retaining each unit during training, and \( p = 1.0 \) during testing. To avoid extra hyperparameters, we elected to only employ the "standard" way of using dropout with CNNs, i.e. only on the fully-connected layers. We also doubled the number of units in the layers affected by dropout when dropout was enabled to ensure that the dropout networks possessed approximately the same predictive power as the networks without dropout.

We also evaluated a version of the network with batch normalization. The batch normalization was applied to the convolutional layers, pre-activation, like the network described by DeTone, Malisiewicz, and Rabinovich [5]. As suggested by Ioffe and Szegedy [19], dropout was not used in whenever batch normalization was used.
Figure 3.3: The baseline network based on [5] and [33].
Chapter 4

Results

We use this chapter to present the results of our experiments. Unless otherwise specified, all experiments where made using the parameters described in section 3.2. The default dataset used by the networks was generated using max 20 random cropped patches per frame. The multiple ground truths were used for both training and validation unless otherwise specified.

To put the error values into perspective, we note that the reference view was defined such that each unit equaled $\frac{1}{6}$ yard. Thus, absolute error values lower than 6 indicate a registration with an error smaller than a yard.

4.1 Data augmentation

Here we present our results on the effectiveness of data augmentation.

4.1.1 Augmentation by cropping

We compare three datasets. The baseline used no augmentation. The other models used datasets consisting of a maximum of 20 and 30 random crops per frame. We will refer to these datasets as no-aug, max-20 and max-30. However, we note that this formulation did not increase the size of the augmented datasets by factors 20 and 30 as many patches were disqualified for removing too many intersection points. On average, the dataset of max-20 was 15 times larger than no-aug, and max-30 was 21 times larger than no-aug.
### Table 4.1: Loss at epoch 100 for the models evaluating the effectiveness of data augmentation.

<table>
<thead>
<tr>
<th></th>
<th>no-aug</th>
<th>max-20</th>
<th>max-30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training loss (MAE)</td>
<td>1.6</td>
<td>0.9</td>
<td>0.7</td>
</tr>
<tr>
<td>Validation loss (MAE)</td>
<td>9.1</td>
<td>4.8</td>
<td>4.7</td>
</tr>
</tbody>
</table>

Figure 4.1: Validation loss of the models evaluating the effectiveness of augmentation by cropping. no-aug refers to the model without augmentation, while max-20 and max-30 refer to the models where a maximum of 20 respectively 30 patches were generated for each frame in the training set per epoch.

#### 4.1.2 Color augmentation

We present our results on the effectiveness of color augmentation.

<table>
<thead>
<tr>
<th></th>
<th>no-color-aug</th>
<th>color-aug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation loss (MAE)</td>
<td>4.6</td>
<td>4.1</td>
</tr>
<tr>
<td>Validation loss (MSE)</td>
<td>50.0</td>
<td>49.7</td>
</tr>
</tbody>
</table>

Table 4.2: Loss at epoch 75 for the models evaluating the effectiveness of color augmentation.
4.2 Multiple ground truths

We present our results comparing models using the multiple ground truths. Model $S$ used only the center aligned homography. Model $M_V$ used the center, left, and right homographies, but only updated the ground truth vector during validation. Model $M_{T+V}$ also used the three homographies, but updated the ground truths during both training and validation. As models $S$ and $M_V$ were identical with regards to training, we have omitted the training results of $M_V$.

<table>
<thead>
<tr>
<th></th>
<th>$S$</th>
<th>$M_V$</th>
<th>$M_{T+V}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training loss (MAE)</td>
<td>0.8</td>
<td>-</td>
<td>0.8</td>
</tr>
<tr>
<td>Validation loss (MAE)</td>
<td>6.0</td>
<td>4.9</td>
<td><strong>4.8</strong></td>
</tr>
</tbody>
</table>

Table 4.3: Loss at epoch 100 for the models evaluating the effectiveness of multiple ground truths.

![Training loss graph](image)

Figure 4.2: The training loss of models $S$ and $M_{T+V}$. Model $S$ used only the center aligned ground truth, while model $M_{T+V}$ used the left, center and right alignments and updated the ground truth to be set to the one providing the minimum loss after each epoch.
Figure 4.3: The validation loss of models $S$, $M_V$ and $M_{T+V}$. Model $S$ used only the center aligned ground truth, while models $M_V$ and $M_{T+V}$ used the left, center and right alignments and updated the ground truth to be set to the one providing the minimum loss after each epoch. The difference between $M_V$ and $M_{T+V}$ is that $M_V$ only updated the ground truths for the validation set, while $M_{T+V}$ updated the ground truths for both the training and validation sets.

### 4.3 Dropout

This section contains the results of the models evaluating dropout. Both models were identical except the dropout model contained twice as many units in the layers affected by dropout.

<table>
<thead>
<tr>
<th></th>
<th>No dropout</th>
<th>Dropout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training loss (MAE)</td>
<td>0.8</td>
<td>3.5</td>
</tr>
<tr>
<td>Validation loss (MAE)</td>
<td>4.8</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Table 4.4: Loss at epoch 100 for the networks with and without dropout. The network without dropout contained 256 units in the fully-connected layers. The network with dropout contained twice as many units in the fully-connected layers.
4.4 Batch normalization

We compared two models using batch normalization with our baseline model. The batch normalization models differed only in batch size.
BN-16 used batch size 16 and BN-32 used batch size 32. The baseline model no-BN also used batches with size 32 and was identical to BN-16 and BN-32 except it did not use batch normalization.

<table>
<thead>
<tr>
<th></th>
<th>no-BN</th>
<th>BN-16</th>
<th>BN-32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training loss (MAE)</td>
<td>0.8</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>Validation loss (MAE)</td>
<td>4.8</td>
<td>5.0</td>
<td>5.7</td>
</tr>
</tbody>
</table>

Table 4.5: Loss at epoch 100 for the baseline and batch normalized models.

Figure 4.6: Training loss of models evaluating batch normalization. Model no-BN was the baseline without batch normalization, while BN-16 and BN-32 contained batch normalization layers between the convolutional layers and their non-linearities. The batch normalization models used batch sizes of 16 and 32 respectively.
Figure 4.7: Validation loss of models evaluating batch normalization. Model no-BN was the baseline without batch normalization, while BN-16 and BN-32 contained batch normalization layers between the convolutional layers and their non-linearities. The batch normalization models used batch sizes of 16 and 32 respectively.

### 4.5 Choice of loss function

We present the results of the experiments we performed to select our loss function.

<table>
<thead>
<tr>
<th>Loss Function</th>
<th>MAE</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation MSE, mean</td>
<td>60.9</td>
<td>60.6</td>
</tr>
<tr>
<td>Validation MSE, median</td>
<td><strong>17.8</strong></td>
<td>21.9</td>
</tr>
</tbody>
</table>

Table 4.6: Metrics at epoch 100 when using the different loss functions. The column names state which loss the optimizer minimized, while the rows contain different metrics.

### 4.6 Best model

We present the results of applying our best model to the test set. The best model used both augmentation by cropping (max 20 crops per frame and epoch) and color augmentation. It did not use dropout or batch normalization.
Table 4.7: Metrics of our best model when performing predictions on the test set.

<table>
<thead>
<tr>
<th>Prediction error</th>
<th>MAE</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11.35</td>
<td>448.93</td>
</tr>
</tbody>
</table>

Figure 4.8: The samplewise prediction errors (MAE) of our best model on the test set.

4.6.1 Line estimation

To visualize the quality of the results we present the original images with lines overlaid on top. The lines were estimated by applying the inverse homographies to the canonical view, where the homographies were estimated by the network. See section 5.1 for details on the format.

We use the format to present the 5 most and 5 least accurately predicted homographies.
Figure 4.9: Predicted yard and hash lines on the 5 most accurately predicted frames.
Figure 4.10: Predicted yard and hash lines on the 5 least accurately predicted frames. The bottom image appears to be a good registration, but received the highest loss in the entire dataset due to an incorrectly annotated ground truth.
Chapter 5
Discussion and conclusions

Despite its simplicity, data augmentation by random cropping proved to be the technique most useful in reducing the validation loss. Adding augmentation by cropping to the dataset almost halved the validation MAE. This emphasizes the fact that the size of the dataset is of extreme importance when training networks. However, increasing the number of random crops did not scale infinitely. The model max-30 converged in fewer epochs than max-20, but both models converged at about the same value. The faster convergence is explained by the fact that max-30 processed more frames per epoch than max-20. This also meant that each training epoch of max-30 took longer to complete than max-20. On average an epoch took 440s for max-20 and 640s for max-30. The fact that both max-20 and max-30 converged at approximately the same value suggests that a "sweet spot" exists for data augmentation by cropping, especially when the size of the patches is close to the size of the original frames. Using smaller patches could potentially increase the benefits of generating more patches, but we found that smaller patches often failed to satisfy our requirements on the number of intersection points. The best way to solve the problem would be to simply annotate more videos and increase the size of the dataset, as our dataset was relatively small compared to other deep learning datasets.

By augmenting the color of the training samples we managed to reduce the MAE, but the MSE remained consistent or increased slightly. We suggest that this was caused by outliers. The color augmentation should help the model generalize to frames where the intensity and color of the illumination varies [23], but it does very little that might
help the network predict difficult outliers.

The multiple ground truths proved effective, but not to the same extent as the augmentation by cropping. Using the multiple ground truths during validation reduced the final validation loss by 18%, suggesting that the network correctly registered multiple frames in the validation set with a horizontal shift. Reassigning the ground truths during training also reduced the validation loss, but with diminished effect. We observed few differences when comparing the training loss of $S$ with $M_{T+V}$. Both models managed to converge at losses near zero.

We had little success using dropout. Srivastava et al. [34] suggested that if a layer performs optimally with $n$ hidden units without dropout, then the corresponding dropout network should contain at least $n/p$ units with $p$ as the probability of retaining a unit, as on average the layer will contain only $np$ units after dropout. We followed their recommendation and multiplied the number of units in the fully-connected layers by $1/p$, increasing the number of units to 512. This formulation meant that on average, the networks with and without dropout should contain the same number of units. The results of these experiments are shown in table 4.4. The table shows that the network without dropout performed better than the network with dropout on the validation set. The table also shows that the network with dropout failed to fit the training data as well as the network without dropout did. Because of this, we cannot disprove the hypothesis that our dropout network underfit. Further work should investigate what might have caused the underfitting. As [34] pointed out, the models should on average be equally powerful when the number of units are increased by $1/p$ in the dropout network.

Batch normalization did not improve our results. The extra parameters introduced by batch normalization made the models using batch normalization converge at higher losses than the networks not using batch normalization. Like with dropout, this might be explained by the small size of our network. Our network is quite shallow when compared to other modern CNNs. The Inception network used by [19] consisted of 22 layers with parameters, while our network was built using only 11 such layers. Because the internal covariate shift grows with depth, the benefits of batch normalization increase as the depth of a network increases. It is possible that our network was so shallow that the drawbacks of batch normalization (i.e. extra parameters) outweighed the benefits of removing covariate shift. Also, another
benefit of batch normalization is that it enables the use of saturating non-linearities for deep networks. But, as we did not use such non-linearities (we used ReLUs throughout), this benefit did not influence our model.

Predictably, table 4.6 shows that minimizing for MSE reduced the mean error while minimizing for MAE reduced the median. However, minimizing MAE reduced the median considerably more than minimizing the MSE reduced the mean: a 18.9% reduction in median and a 0.5% reduction in mean. This result, and the large difference between the mean and median in the same table (the mean was approximately three times larger than the median), suggests that the outliers in the dataset have a large effect on the results. This motivates our choice of MAE as the loss function.

However, an even better way of dealing with the outliers would be to remove them. But, as the dataset contained roughly 3000 frames even after the cleaning operations described in section 3.1, manually removing incorrect outliers is no trivial task. Nonetheless, the dataset should be cleaned before future work is performed.

Some of the frames in the test set were registered well, but received high losses due to incorrectly annotated frames. For example, the bottom frame in figure 4.10 appears to be a quality registration but had the highest loss of all the frames in the dataset. In that particular case the problem was that one of the hash lines had been annotated twice: once as a hash line and once as a yard line. That caused the generated ground truth homography for the frame to be wildly inaccurate, which led to the high loss.

Future work could investigate different network architectures. ResNets [14] in particular could prove potent, as they lack the max pooling layers present in our VGG-like network. Max pooling layers are often used in classification networks to, among other things, increase the robustness of the network with regard to displacement. This is desirable in classification but might have negative effects for networks where the exact position of the detected features matter, e.g. our network.

5.1 Applications

The frames need to be horizontally aligned before being used in the final registration pipeline. The simplest approach would be to manually
extract the yard numbers and use them for alignment. A deep learning approach could also be investigated by training a classification network on the registered frames from our network with the horizontal alignment as the label.

By calculating the homography mapping the target points predicted by the network onto the patch corners (i.e. the “reverse” homography) we could transform the lines in the canonical view to the original frame. This enabled the network to estimate the position of the lines in the frames, basically making the network function as a feature extractor. The estimated lines could be used to speed up further annotation. However, traditional techniques such as the Hough transform [8] are probably better suited if line estimation is the primary application. Nonetheless, overlaying the predicted lines on the frame proved useful in visualizing the quality of the predictions, as seen in section 4.6.1.

One of the limitations of the network is that it performs poorly when few visual features are available. However, once the “easier” frames have been horizontally aligned they could be used together with a stitching tool to register the remaining frames. The stitching tool would register successive frames onto each other, while the frames from our network would anchor the stitched frames to the canonical view. An example of such a stitching tool can be found in [11].

5.2 Conclusions

We implemented a network that estimates homographies mapping video frames from football video to the center of a top-down view of a football field. The network was inspired by [5], but differs in that our network takes a single image as input and generates homographies without a reference image. The network we implemented can register many frames where at least 10 intersection points between yard lines and hash lines are visible. The mean absolute error of the predictions on the test set was approximately equivalent to 2 yards. However, the median error was considerably lower as outliers caused the mean to increase. The results of the network need to be horizontally aligned before being usable as complete image registration.

Because the results of the network consistently improved as the size of the dataset increased, either by augmentation or by annotating
new videos, we suggest that a larger dataset would enable the network to estimate homographies for frames with fewer distinctive features and further improve the quality of the predictions.
Bibliography


