Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Multi-view Camera synthesis using Convolutional Neural Network

Valeria Olyunina B.Sc. P.Dip.

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Augmented and Virtual Reality)

Supervisor: Matthew Moynihan, Prof Aljosa Smolic

August 2019

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Valeria Olyunina

August 14, 2019

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Valeria Olyunina

August 14, 2019

Acknowledgments

I wish to express my sincere gratitude to my supervisors - Matthew Moynihan and Prof Aljosa Smolic - for their attention, support and valuable advice while working on this dissertation.
I would also like to thank my husband, Mark, and my mother for allowing me time to work on this thesis.

Valeria Olyunina

University of Dublin, Trinity College
August 2019

Multi-view Camera synthesis using Convolutional Neural Network

Valeria Olyunina, Master of Science in Computer Science

University of Dublin, Trinity College, 2019

Supervisor: Matthew Moynihan, Prof Aljosa Smolic

This dissertation trained a neural network capable of producing an intermediate image between two spatially distributed images - effectively creating a novel point of view. This research is based on the article by Niklaus et al, 2017 "Video Frame Interpolation via Adaptive Separable Convolution" where a convolutional neural network is deployed to generate interpolated frames in a *video sequence*. The same approach is successfully applied in this research to *multi-view camera images*. The neural network is re-trained on a dataset of synthetically produced multi-view camera images. The resulting images are evaluated both for their quality in 2D and as a tool for improving the photogrammetery method of Shape-from-Silhouette in 3D reconstruction. The neural network trained on multi-view camera images produced by this research can generate visually correct interpolated multi-view images. When compared to ground truth, PSNR of these images is above 40 and SSIM is above 92% for the distance between multi-view cameras of less than 60cm (distance from camera to subject between 3-5m) when tested on a synthetic test set. This is higher than the corresponding results for the original video interpolation article. For 3D reconstruction, the cameras needed to be further apart (1-2m) and the silhouettes were not always pixel-accurate. Within 60cm only 1% of pixels were lost, however at distance between cameras over 1 m over 3% of pixels are lost, resulting in loss of voxels in extremities.

Summary

This dissertation addresses a problem of image interpolation in multi-view camera setting. It explores the possibility of applying temporal video interpolation techniques to spatial image interpolation and further using the generated images in 3D reconstruction methods, such as Shape-from-Silhouette.

This document first reviews 3D geometry and photogrammetry reconstruction methods, then it overviews current spatial and temporal image interpolation literature, focusing in particular on research where deep-learning neural networks were applied to the task. It also reviews neural network architectures suitable for the task.

In the Methodology chapter, the applied network architecture is examined in detail, particularly with regards to the *loss functions* applicable to interpolation of multi-view camera images. The synthetic dataset created specially for this research with the view of training a multi-view neural network is described. Then suitable methods for evaluating the interpolated images are examined, including 2D and 3D methods.

The last chapter contains the evaluation of the images produced by the neural networks trained for this research. The networks are compared with each other and with the benchmark network that was created by the reference article for temporal interpolation [35].

Contents

List of Tables

List of Figures

Abbreviations

2D Two dimensional
3D Three dimensional
AR - Augmented Reality
CG - Computer Graphics
CNN - Convolutional Neural Network
FVT - Free-viewpoint Television
FVV - Free-Viewpoint Video. Same as FVT.
GAN - Generative Adversarial Network
GPU - Graphics Processing Unit
GT - Ground Truth
HSV - Human Visual System
IBR - Image-Based Rendering
IBMR - Image-Based Modeling and Rendering
LSTM - Long Short-Term Memory
MVS - Multi-view stereo
NN - Neural Network
PSNR - Peak Signal to Noise Ratio
PSR - Poisson Surface Reconstruction
RNN - Recurrent Neural Network
SfM - Structure from Motion
SfS - Shape-from-Silhouette
SSIM - The Structural Similarity Index
VR - Virtual Reality

Chapter 1

Introduction

This dissertation explores the possibility of generating novel points of view - synthetic cameras - from input images from spatially distributed cameras. The main focus of the research is on videos of human motion obtained from multi-view camera setups. The primary aim is therefore the geometric accuracy of the generated images. The synthetic images can then be used for improved reconstruction of the 3D model of the recorded subject or for video interpolation and display of arbitrary points of view in 360 degree/ Free Viewpoint Video (FVV) scenarios.

1.1 Motivation

There is a growing requirement in the current digital world for 3D digital models of objects and people and full 3D videos of their motion. These can be used in entertainment industry, but also in business, medical and scientific applications to help visualisation of any problem. In the entertainment industry computer games, Augmented Reality (AR), Virtual Reality (VR) and FVV are on the rise [1] and need accurate 3D models. Until recently, most content available was synthetic, created by artists and designers, but there is a growing demand for true to life assets [103].

Additionally, there is a growing demand in new visual experiences in terms of 3D TV, 360-degree video and Free-viewpoint video. These are normally filmed using multiple cameras, but additional views can be produced using interpolation techniques [3] which

allow artificially created viewpoints. The techniques described in this dissertation aim to improve the quality for these synthesised images.

FVV and Multi-view camera systems for performance capture offer a VR/AR experience with the spatio-temporal fidelity of a live performance. However, the quality of 3D reconstruction is dependant on the technology used to capture and process the input videos. Where depth cameras are not available, the current technology relies on Structure-from-Motion (SfM) and Shape-from-Silhouette (SfS) techniques. These techniques vary in accuracy, for example, where a lot of cameras are available in a specially setup green room environment, these can be very accurate. But the accuracy tends to deteriorate as the number of cameras decreases. Recently, some research attempts reconstruction in difficult scenarios from as little as eight mobile phone cameras [64], where there are problems of background removal and camera synchronization for SfM reconstruction as cameras are not stationary. The quality of the SfM reconstruction suffers from 'holes' where occlusions occur [44], inaccuracies from lack of reference points in sparse camera setup, lack of texture, transparent or reflective features, due to camera lens, noise, camera angle [94]. SfS reconstructions alone generally lack the 'completeness' to produce accurate, fully-volumetric reconstructions.

This thesis presents an approach to improve the accuracy of photogrammetry-based methods in case of sparse reconstructions by providing additional synthetic views in between the real camera views. The approach is based on Image-Based Rendering (IBR) techniques, particularly view interpolation. IBR techniques create new images directly from the existing set of images without doing a full 3D reconstruction [103].

Neural networks (NNs) have revolutionised image processing in the recent years. They have been shown to have better performance at computer vision tasks than previously designed procedural approaches [38]. Deep neural networks are able to extract and combine tens of thousands of features from images, where a human approach may typically only find dozens. Deep-learning networks have been used for classification, segmentation and creation of new images and video. IBR was previously combined with deep-learning to create arbitrary points of view when given a collection of images [20]. [61] is able to produce high-quality images for video-frame interpolation with a

convolutional neural network (CNN).

This dissertation aims to apply the same approach to multi-view camera images interpolation and explore the possibility of using a neural network to produce multi-view images.

1.2 Objectives

The *primary objective* of this dissertation is to train a neural network capable of outputting accurate interpolated images in a multi-view camera scenario. As the primary motivation for producing such images was to help 3D reconstruction, part of the primary task is to perform a 3D multi-view reconstruction using photogrammetry methods to check if the reconstruction can be improved by adding the interpolated images.

The following *secondary objectives* were also part of this research:

1) Preparation of the *multi-view camera dataset* suitable for training the neural network. Both real available multi-camera datasets and synthetic datasets were considered, including the option of self-generated synthetic dataset for the training.

2) Exploration of the neural networks suitable for the task and their hyperparameters.

3) Exploration of the measurements to evaluate the quality of produced spatially interpolated images. Accurate measurement can also be used as a loss function for the neural network training.

As a starting point an existing NN implementation was chosen. This was designed by [61]. The NN was chosen as it was shown to be successful at interpolation of images for the task of video interpolation by increasing the framerate. Also, NN code was available as re-implemented by [35]. The NN needed to be adapted for the different task of generating multi-camera spatial images, the difference being that in multi-view camera scenarios the images are generally much further apart than the frames in a video.

1.3 Overview of the Dissertation

The layout of the dissertation is as follows:

Chapter 2 State-of-the-Art looks at the current research in the field. The first section reviews in detail the photogrammetry methods used in 3D reconstruction - SfM, Multi-view stereo (MVS) and SfS. The next section examines IBR, in particular view-interpolation techniques both temporal and spatial. The third section looks at neural networks and different neural network designs. The last section combines the first three - looking at view-interpolation research that uses deep-learning, but also checking if deep-learning approach was previously applied to spatial images.

Chapter 3 Methodology examines the methodology in detail. It covers three main areas:
- Design of the neural network
- Dataset generation. Particularly the technique used to generate the synthetic multi-view dataset of human motion that was deployed in training the neural network.
- Evaluation techniques, including 2D evaluation and 3D reconstruction (SfS methodology) and evaluation are discussed.

Chapter 4 Results and Discussion presents the results of this research and provides the discussion. First, the resultant interpolated multi-view images are presented and discussed. Second, 2D evaluation techniques and results are discussed. And lastly the results of the 3D reconstruction using the interpolated images (SfS) are presented and discussed.

Chapter 5 Conclusion and Future Work summarises the main outcome of this research and proposes ways in which the outcome can be improved in the future.

Chapter 2

State of the Art

The following chapter presents the state-of-the-art for this research. As multi-view imagery has a large dependence on 3D geometry and this research is concerned with improving photogrammetry techniques for 3D reconstruction, the first section delves into 3D geometry and describes photogrammetry methods - SfM, Multi-view stereo (MVS) and SfS.

The second section covers the state-of-the-art in view interpolation. This desrcibes traditional methods without deep-learning.

The third section explores neural network architectures for the purposes of this research. Is looks at CNNs, CapsuleNets, Autoencoders, RNNs and GANs. Autoenconders are included as they form the basis of generative networks. The generation of the new images (interpolated frames) would frequently include an encoder and a decoder.

The last section combines the previous two sections and attempts to analyse the current research most relevant to this project - view interpolation using deep-learning.

In short this chapter explores:

1. 3D Reconstruction, in particular the algorithms of structure-from-motion, mutli-view synthesis and Shape-from-Silhouette.

2. View Interpolation

3. Neural networks and deep-learning

4. Neural networks in view interpolation

2.1 3D reconstruction and Photogrammetry

3D Reconstruction of object shapes from still images and video stream is an ongoing research topic that challenged researchers for decades. Early research first addressed the simplified problem of orthographic projection, then perspective projection was researched that resolves the uncertainty of perspective homographies. 3D reconstruction methods of SfM, MVS and SfS are still under research to suggest the most efficient and precise way to extract 3D geometry from a set of 2D images.

2.1.1 Orthographic projection

As early as 1992, [88] obtained good results from a stream of images using orthographic, rather than perspective, projection and introduced *Factorization method*. Orthographic projection simplified processing, removing the depth dimension [1] . Examples of projective, affine and Euclidean projections are given in Fig. 2.1. [88] worked with affine and orthographic projections only. They decomposed the measurement matrix W (F frames, P tracked points forms $2 * F * P$ matrix in 2D) into 2 matrices - R and S - representing the camera rotation and the object shape respectively plus the projection of the camera translation t along the image plane.

$$W = RS + te_P^T$$

[88] were able to process input with *noisy measurements* by introducing 3x3 matrix Q ($R = \hat{R}Q$, $S = Q^{-1}\hat{S}$) and metric constraints to solve for Q. They are also able to cope with *occlusions* by recovering position of feature points from 3 other positions of the feature.

[1] Orthographic projection is applicable when the distance from the object to the camera (Z_{avg}) is more than 10 times the object's width d_{avg}: $Z_{avg} \geq 10 * d_{avg}$ (from [5], chapter 9)

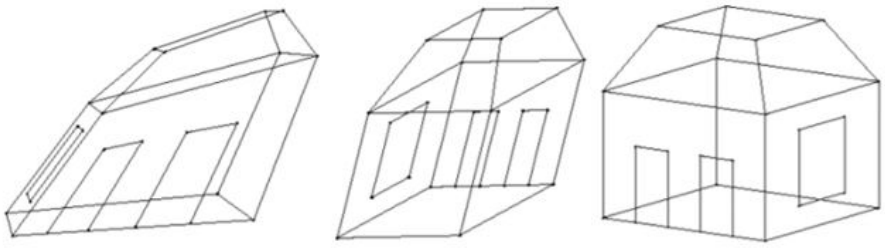


Figure 2.1: Examples of Perspective, Affine and Orthographic projections. [5], chapter 9.

2.1.2 Perspective projection - epipolar geometry

Perspective projection adds extra complexity to obtaining 3D geometry from images.

Calibrated cameras:
First, a special case of calibrated cameras is described.
The *essential matrix* E for correspondence between 2 images was introduced by [48]:

$$E = [t] \times R$$

So, the same point in 2 images correspond as:

$$\hat{x}_1^T E \hat{x}_0 = 0$$

where $\hat{x}_1$is the position of the point in the second image, $\hat{x}_0$ is the position of the same point in the first image and E is the essential matrix - forming *epipolar constraint*. Point $\hat{x}_0$ in the first image is transformed using the essential matrix E into a line in the second image - $l_1 = E\hat{x}_0$, which is called *epipole* [85].

Both translation and rotation of the 2^{nd} camera - and traditionally, any subsequent cameras in a sequence of images - are taken with reference to the camera position of the first image in the sequence, i.e. the camera of the first image is the origin of the world co-ordinates and its orientation R_o equals identity matrix.

If more than one feature point is available between the 2 images, the *Essential Ma-*

trix can be determined from a series of equations: $[x_{i1}x_{i0}^T] \otimes E = 0$, where $\otimes$ denotes point-wise multiplication, and i is the index of the feature. The series of equations can be resolved with SVD (singular-value decomposition) algorithm. It has been shown by several researchers ([28] ; [89]; [27]) that *7 point correspondences* (i.e. features) is sufficient to find the elements of the essential matrix [85].

In addition, [28] suggested that the point co-ordinates need to be translated and scaled to the centre of the object, so that the sum of x and y co-ordinates is 0 and the squared sum of both co-ordinates equals twice the number of points ($\sum_i \tilde{x}_i = \sum_i \tilde{y}_i = 0$, $\sum_i \tilde{x}_i^2 + \sum_i y_i^2 = 2n$)

Uncalibrated cameras
The above equations describe *an ideal case*, where cameras are *perfectly calibrated*. The assumption of un-calibrated cameras adds an additional complexity of the calibration matrix K. The essential matrix becomes the fundamental matrix F:

$$F = K_1^{-T} E K_0^{-1}$$

Where K is the camera calibration matrix. Or, $F = [e] \times \tilde{H}$, where e is the *focus of expansion* and matrix $\tilde{H}$ is one of many possible *homographies* [27], [19].

Calibration Matrix
While it is possible under certain constraints to convert projective reconstruction into a metric one, i.e. recover calibration matrices K_j associated with each image (*self-calibration* [27]), most 3D reconstructions assume *pre-calibrated cameras* or images taken with a single camera with fixed intrinsic parameters.

$$\begin{bmatrix} F \cdot d_u & s & u_0 \\ 0 & F \cdot d_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where F is focal length, d_u and d_v - size of the camera sensor per pixel, u_0 and v_0 - translation of the camera centre with regard to the image [81].

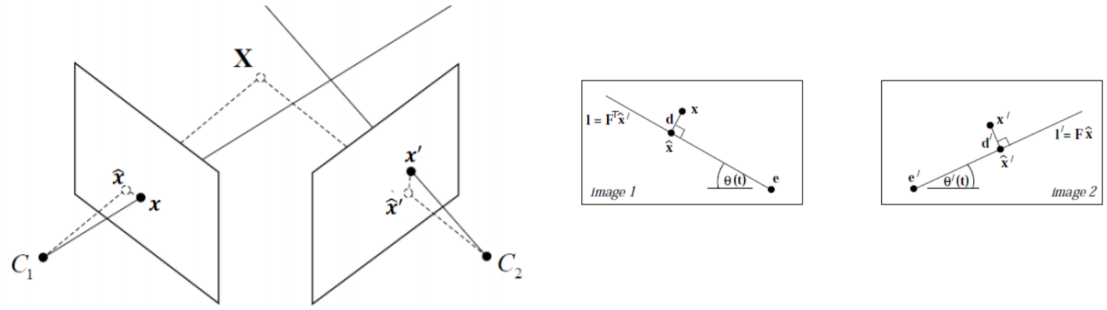


Figure 2.2: Triangulation illustration.

Bundle adjustment

Two images with 7 point correspondences is sufficient to estimate the *Fundamental matrix*, so each new image or each new point-correspondence overdetermines the system. A cost function can be introduced that aims to minimize the re-projection error. The system of equations can be solved with a non-linear method. There are two options for bundle adjustment: this can be done incrementally - as each new image is added - or at the end of the process with all images.

Triangulation

The last topic to discuss in the basics of 3D reconstruction is triangulation. This is a method to estimate depth to the object after the Fundamental matrix is known.
The method aims to minimize the projected error

$$cost(X) = dist(x, \hat{x})^2 + dist(x', \hat{x}')^2$$

while satisfying $\hat{x}'^T F \hat{x} = 0$, where F is fundamental matrix and $\hat{x}$, $\hat{x}'^T$ are projections of 3D points x and x' onto the epipolar line. Demonstrated in Fig. 2.2.

The fundamental matrix and epipolar correspondence lie at the heart of SfM algorithm, as any estimation will start from finding the matrix correspondence between the two images.

2.1.3 Structure from Motion

The following two sections describe Structure-from-Motion (SfM) and Multi-view stereo (MVS) algorithms. These correspond to obtaining sparse 3D point cloud reconstruction and camera positions (SfM) and dense 3D surface reconstruction (MVS).

The typical workflow of a SfM algorithm is as follows [75]:

1. *Feature extraction and feature descriptors.*

   Correspondence between images is found based on distinctive points, so the first step of SfM is to identify feature points and their descriptors for each image in the stream. One of the early methods for finding the interesting points is Autocorrelation function (ACF) [85], which finds if the point is unique in its surroundings. The following authors further expanded on ACF: [26], [52], [78] etc.

   Suitable features are then described in terms of their neighbourhood. This is to ensure invariance to rotation, scaling, perspective distortions, lighting changes etc.

   The proposed algorithms create a description of the point's neighbourhood:

   - SIFT [51]
   - SURF [4]
   - BRIEF [6]
   - ASIFT [59]
   - LDAHash [83]

   For this work, the interpolated images are intended to be used in 3D reconstruction, and because the images are produced with a neural network an issue of blurriness and ghost artefacts can occur in generated images and. These may affect feature detection and make it difficult for SfM to place features correctly.

2. *Feature Matching*

   The above features are matched. The simplest approach is to test every image pair and for every feature in the first image to find the most similar feature in the second image (*similarity metric*). The computational complexity of this approach is $O(N^2_{IMAGES}N^2_{FEATURES})$, so cannot be applied to large image collections [75]. There is research to improve the efficiency of the matching, for example, k-dimensional trees and ANN (Approximate Nearest Neighbour) [80].

3. *Identifying geometrically consistent matches*

   Some feature matches may be excluded when they are checked for possible geometric transformations (homography, fundamental and essential matrices). If a valid transformation maps a sufficient number of features between the images, they are considered geometrically verified. RANSAC algorithm is usually used for the outlier detection [75].

4. *Initialisation before reconstruction and image registration*

   SfM chooses the appropriate initial pair of cameras that would represent the origin of the world co-ordinates. Typically, these will have many common features and a wide baseline [80].

   The order in which the images will be added is important. New images can be registered to the current model by solving the Perspective-n-Point (PnP) problem. The PnP problem estimates the pose of the camera for new image and, for uncalibrated cameras, camera's intrinsic parameters. Every new image provides additional 2D-3D correspondences. [75].

5. *Triangulation*

   Triangulation method is used to compute 3D space point X from feature point correspondence ($x \leftrightarrow x'$). Again, several different methods are proposed. [27] describe triangulation suitable for different types of transformations (affine, projective, etc). They discuss the differences between linear triangulation method (DLT, inhomogeneous), error minimisation, Sampson approximation and solving a 6-degree polynomial.

6. *Bundle Adjustment*

   Bundle adjustment minimizes the reprojection error as more 2D-3D correspondences are added to the system. It performs a joint non-linear refinement of parameters P_c (Camera position and intrinsic parameters) and point positions - X. The following formulae defines bundle adjustment, where E - reprojection error, x_j - co-ordinates of the point in image j, ρ_j - loss function to down-weight the outliers and π symbolises the function that converts scene points into image space [75].

$$E = \sum_j \rho_j(\|\pi\left(P_c,\ X\right) - x_j\|_2^2)$$

   The output of the SfM stage is a *sparse, unscaled 3D point cloud in arbitrary units along with camera models and poses*. This can be resolved into metric reconstruction if camera calibrations are known, or if metric parameters of some of the points are known (for example, ground-control points in case of georeferencing [80]) .

2.1.4 Multi-view stereo

MVS provides a complete 3D reconstruction or *dense modelling* of the object from a known sparse 3D cloud and known camera positions and intrinsic matrices. [80] summarises the review of MVS methods by As detailed by [75] with reference to [77], there is a wide variety of MVS algorithms, which can be classified into:

1. Voxel-based methods which are 3D grids that are occupied to define the scene (for example, [76]).

2. Surface evolution-based methods that use iteratively evolved polygonal meshes (for example, [22]).

3. Depth-map merging methods where individual depth maps showing the distance between the camera viewpoint to the 3D scene objects are combined into a single model (for example, [43])

4. Patch-based methods where collections of small patches or surfels represent the scene (for example, [68]).

The last two steps in MVS are to generate a polygonal 3D mesh from dense point cloud. This can be achieved with *Poisson Surface Reconstruction* (PSR) [37]. Texture is then added with *Texture mapping*.

It can be seen that 3D reconstruction is a multi-step process, where different methodologies can be selected at every step. The particular choice of algorithms for each stage will strongly affect the accuracy of 3D reconstruction.

2.1.5 Shape-from-Silhouette

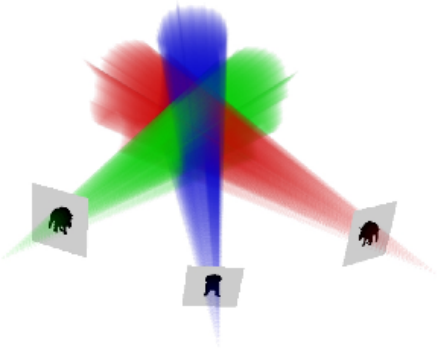


Figure 2.3: SfS - intersection of silhouette cones [57]

A number of techniques have been developed to reconstruct a 3D volumetric model from the intersection of the binary silhouettes projected into 3D. The resulting model is called a visual hull [16]. "Suppose that some original 3D object is viewed from a set of reference views R. Each reference view r has the silhouette s_r with interior pixels covered by the object. For view r one creates the cone-like volume vh_r defined by all the rays starting at the image's point of view p_r and passing through these interior points on its image plane. It is guaranteed that the actual object must be contained in vh_r. This statement is true for all r; thus, the object must be contained in the volume $vh_R = \cap_{r \in R} vh_r$. As the size of R goes to infinity, and includes all possible views, vh_R converges to a shape known as the *visual hull* vh_∞ of the original geometry. The visual hull is not guaranteed to be the same as the original object since *concave* surface regions can never be distinguished using silhouette information alone" - from [57]. See Fig. 2.3.

SfS requires known camera positions, but once these are available this reconstruction method has many advantages: silhouettes are easy to obtain, especially in a green room scenario, the implementation of SfS methods is relatively simple (*voxel carving* is one of the techniques) and the reconstruction always *contains the convex hull of the object* [10]. For this research this method is the selected photogrammetry method as silhouettes do not need the exact color details and are less sensitive to blurriness, as

compared to SfM.

2.2 View interpolation

[34] proposes the following classification of Image-Based Rendering (IBR) techniques. Fig. 2.4 shows a graph of IBR techniques classifications. *View interpolation* and *view morphing* are on the left of the continuum as relying on rendering with implicit geometry and acting on pixel-per-pixel basis.

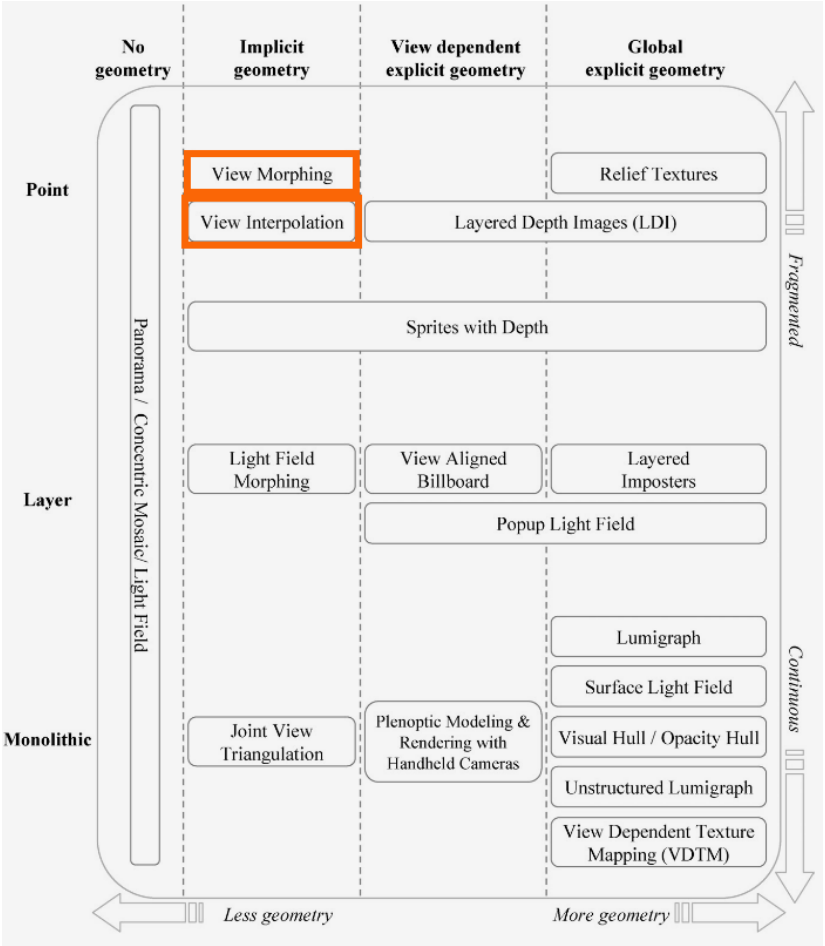


Figure 2.4: IBR techniques classification [34]

2.2.1 Single image interpolation techniques

This dissertation is primarily concerned with view interpolation between two views, but interpolation techniques are also applicable to *a single image*. These belong to a field of *digital image processing* and are required when images are resized, rotated or transformed. It is worth mentioning single image interpolation techniques as they can also be applied to the result of the two frame interpolation. The interpolation techniques for a single image include:

- 2D nearest-neighbour interpolation
- Bilinear interpolation (2x2 neighbourhood)
- Bicubic interpolation (4x4 neighbourhood)
- Spline and Sinc interpolation
- Natural neighbour interpolation using Voronoi cells
- Kriging based on Gaussian distribution

When applied these can produce artefacts in the interpolated images: aliasing, blurring, edge halo (McHugh, 2018). These can be rectified with anti-aliasing, interpolation that is "edge-aware" or "weighted edge-aware" (Paluri 2012).

Interpolating between two views
Interpolation between two views belongs to two broad categories:

1. *Spatial interpolation* of image sequences, when camera position changes and the objects are static.
2. *Temporal interpolation* of image sequences (video interpolation) - when objects in the images/ frames can move. In practice, video interpolation may combine both the moving camera and non-static objects hence incorporating spacial interpolation.

The simplest technique for interpolation between two images is *linear interpolation* where the intermediate pixel can be calculated at any intermediate point $\alpha \in [0,\ 1]$ with:

$$\alpha I_2 + (1 - \alpha)\, I_1$$

Where I_1 and I_2 are pixel values in image 1 and 2. This method produces blurry result where both of the original images can still be distinguished.

2.2.2 Spatial interpolation

Feature based image morphing

Early *spatial interpolation* was introduced by [9]. They worked with computer graphics (CG) images in order to improve the speed of generating CG views. Their technique first determined *pixel-by-pixel correspondences* between images and stored morph maps for further calculation. When required positions and colors of the points were linearly interpolated. As the authors worked with synthetic images, range data and the camera transformations were readily available. They were able to synthesize arbitrary intermediate points of view with bi-directional mapping. The new views only had view-independent shading.

[76] worked with natural images and assumed known camera projections. They expanded on view morphing techniques aiming to keep the shape of 3D objects. The authors first resolve the case of parallel views and prove that for parallel views with orthographic projection linear interpolation between feature points produces the correct result. For non-parallel views, image re-projection is used - this allows to move the image to a different plane using homography matrix.

Their algorithm is composed of 3 steps:
1) Pre-warping - applies reverse homography camera matrices to the 2 images, to bring the images to a single plane and all 3 cameras to a single line.
2) Morph - linearly interpolate position and colors between 2 images.
3) Post-warping - apply homography of the target image camera to obtain the final view.

Both works by [9] and [76] had a big influence on the subsequent research.

Before proceeding to the more recent interpolation techniques, two methods applicable to spatial view interpolation need to be described here: forward mapping and backward (inverse) mapping.

Forward Mapping
Forward mapping maps each pixel on the reference view(s) to the target view using some form of geometry, e.g., depth map (explicit geometry) or correspondences between views (implicit geometry)[34]. If x_t - 2D point in the target image, x_r - 2D point in the reference image, X - point in 3D space, C_r and C_t - camera positions for reference and target images, P_r and P_t - camera projections, ρ_r and ρ_t - scaling factors.

$$\rho_t x_t = {P_t}^{-1} (C_r - C_t) + \ \rho_r {P_t}^{-1} P_r x_r$$

The resultant pixel x_t in the target image can be evaluated from the above equation. There is a problem with this approach - not all pixels in the target image may be populated and therefore will need to be interpolated, or at the same time as many pixels may land on the same pixel in the target image. Even after the interpolation there may still be holes in the image due to magnification and disocclusion. [34]

Therefore, the more traditional approach to use is the reverse of forward mapping:

Inverse mapping
In inverse mapping the pixel mapping in the target is found by tracing the ray from the target view back to the reference view [34]:

$$\rho_r x_r = {P_r}^{-1} (C_t - C_r) + \ \rho_t {P_r}^{-1} P_t x_t$$

Or, expressed in terms of homography $H = {P_r}^{-1} P_t$:

$$x_r = H x_t + d\ e$$

Where H defines the 2D planar perspective transformation from target screen to reference camera, e is the epipole, d is a scale factor and $d\ e$ therefore defines *epipolar*

line.
Inverse mapping ensures that there are no gaps in the target image. However, if x_t is occluded in the reference view the search yields no result [34].

More recent references on view synthesis are able to work with un-calibrated cameras. [21] expands the work of [77], work with uncalibrated cameras and proposes a new algorithm based on interpolating homographies rather than pixel positions and colours. [25] combines spatial interpolation based on feature matching with temporal interpolation based on optical flow (see below). They assert that their approach is suited for wide-baseline setups, where dense stereo matching cannot be applied.

2.2.3 Temporal / Video interpolation

For the *temporal interpolation* the subject may be moving at the same time as the camera, which creates complicated motion, occlusions etc. It may not be possible to simply interpolate based on homographies, also the camera movement is un-known. According to [95] the general problem of image morphing techniques is that the warping and blending might introduce errors when dealing with complex motions between the known images, especially in presence of (dis-)occlusions (caused by moving objects) the approach might exhibit artefacts in these regions.

Optic flow.
A few researchers worked on the image interpolation with optical flow - wrapping the optical flow with input frames to get the interpolated frames - [54], [7], [95] etc. [95] computes the optical flow between 2 interpolated images - this is a bi-directional process as both forward and backward flows are computed. He uses [96] for estimating the optical flow. As the result, because the path is defined at every pixel, no holes are produced when generating interpolated frames.

Another approach is to employ *dense image correspondences*. These are partially based on the above spatial techniques and homographies: [82],[46].

As an alternative approach, a method based on *Fourier transform* is recently used

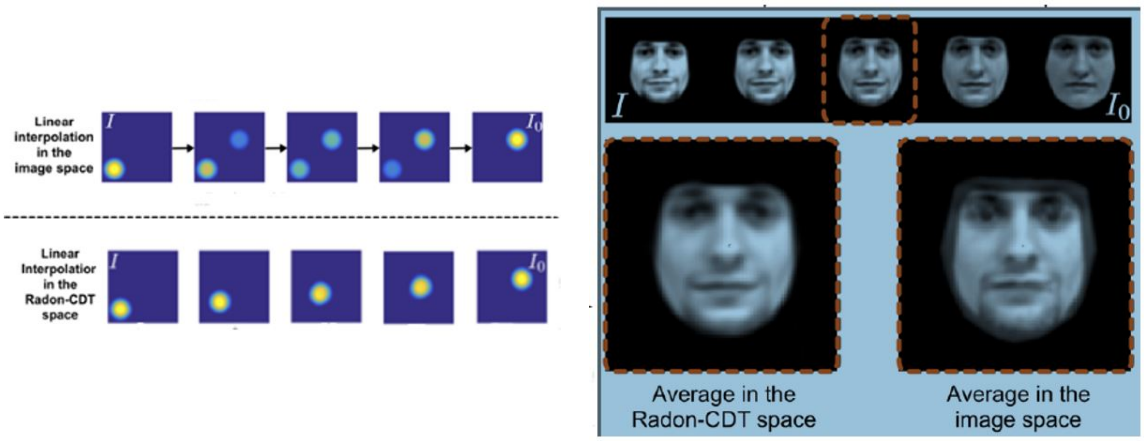


Figure 2.5: Comparing linear interpolation results with non-linear interpolation using Radon-CDT space. Linearly interpolated images in left example top raw and in right example bottom-right [40].

by [40]. He proposes linear interpolation in *Radon Cumulative Distribution Transform* space, where the interpolated image is still linearly separable into the 2 original images. The pixel location information is encoded in transport flows (*optimal transport metric*), so each pixel and neighbourhood are considered from 'Lagrangian' point of view. The transform captures translation and scaling, as well as more complicated transformations - see Fig. 2.5 for an example of the method applied to capturing movement and face interpolation.

Interpolation with dimensionality reduction (*Isomap*) is proposed by as a technique that is able to keep the 3D shape of the object, but seems to only be applicable in the case of repetitive motion, i.e. camera rotating around a rigid object, person waving hand etc [72]. The method finds feature point correspondences between the interpolated images and interpolates a curve between the data points in the feature space, before fitting the intermediate images to the curve.

As a conclusion for this section, we can summarize that that *important properties of the interpolated frame* for 3D reconstruction are*:*

- Keeping features and edges
- Correct location of features and edges

- Sharp
- No ghost artefacts
- No holes

2.3 Neural networks and Deep learning

2.3.1 Convolutional Neural Networks (CNNs)

CNNs have revolutionised image processing in the last decade. They were the first successful application of deep-learning architectures.

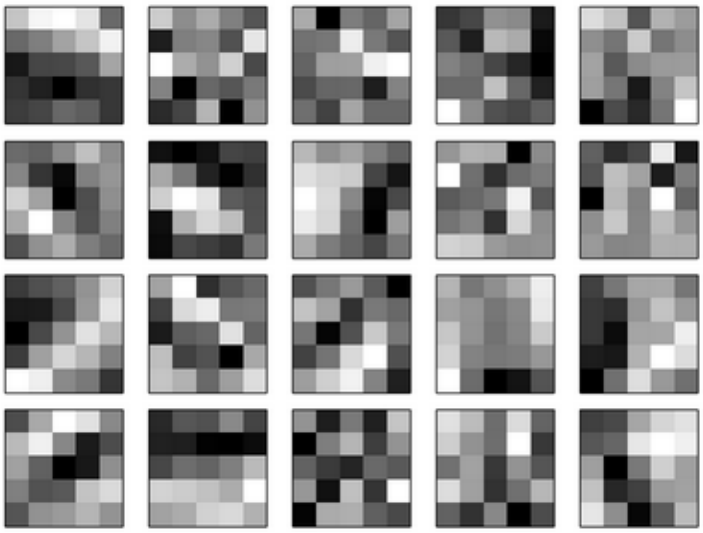


Figure 2.6: Example 5x5 feature maps for different hidden layers of a CNN trained to classify hand-written digits [60].

Their success in image processing is attributed to their *sparse connectivity*, which make processing images more *computationally efficient*. Also, *parameter sharing* - unlike traditional neural net where each weight is only applied once, the convolution kernel is applied to every pixel in the image, which makes it possible to extract the feature independent of the location in the image. CNNs are also *equivariant to translation*

meaning that translation in the input affects the output in the same way [23].

While CNNs have achieved impressive results in image processing, there is still lack of understanding of the internal operation and behaviour of complex models. [102] propose a new approach to visualisation of individual *feature maps* at any layer in the CNN model. They use a multi-layered Deconvolutional network (deconvnet) to project the feature activations back to the input pixel space. [60] uses this technique to visualise feature maps of a middle layer in a CNN for hand-written digit classification - see Fig. 2.6. Each of the 20 maps in the figure represent a 5x5 block image, corresponding to the 5x5 weights in the *local receptive field*. Whiter blocks mean a smaller weight, darker blocks mean a larger weight. It can be seen that the features are quite complicated and not random - all the neurons in a single hidden layer will detect the same feature, just at different locations in the input image.

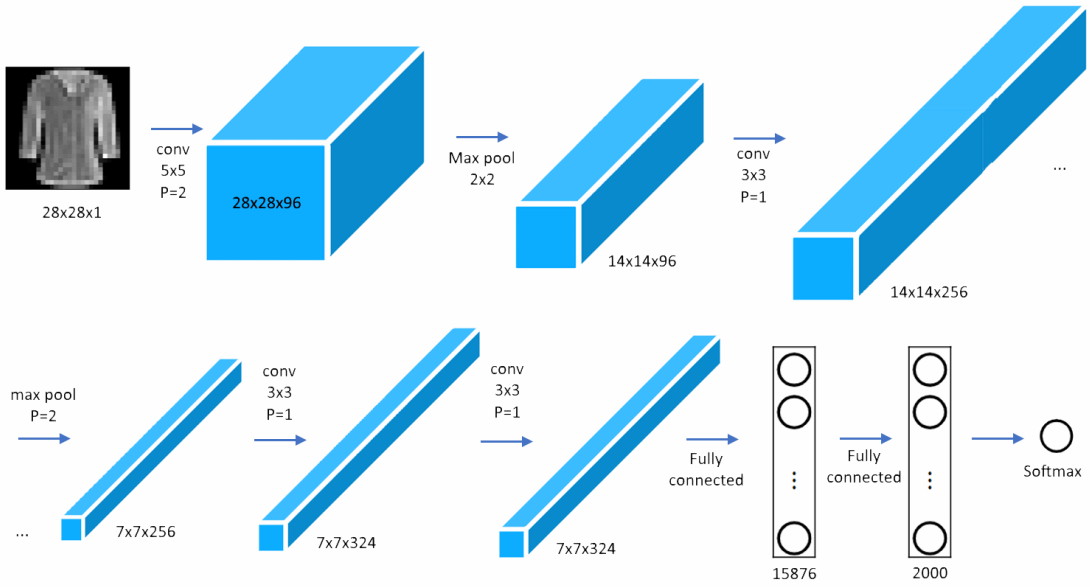


Figure 2.7: Example CNN architecture with 3 convolutional layers. Size of the square shows the size of input images, length of the cuboid - number of hidden layers (source: author)

Fig. 2.7 displays an example of an architecture of a CNN deployed for image classification - clothes items in this case. This networks consists of 4 convolutional layers

interlaced with pooling layers (activation layers not shown). As is typical for a feature extractor - the image size gets smaller (from 28x28 to 7x7 pixels) as the number of hidden layers in the network grows (to 324 hidden layers). The last 2 layers in the network are *fully connected* allowing for the classification of the images.

The operation of convolution and *local receptive field* is demonstrated in Fig. 2.8. This shows a local receptive field of 5x5 pixels corresponding to a single neuron in the next hidden layer.

Convolution is an operation on two functions of real-valued argument. Usually in image processing the first function is a 2D image or a 3D tensor (including time parameter) - in case of a video. The second function is the convolution *kernel*, or sometimes it's called *feature map*. The latter is usually much smaller in size than the image.

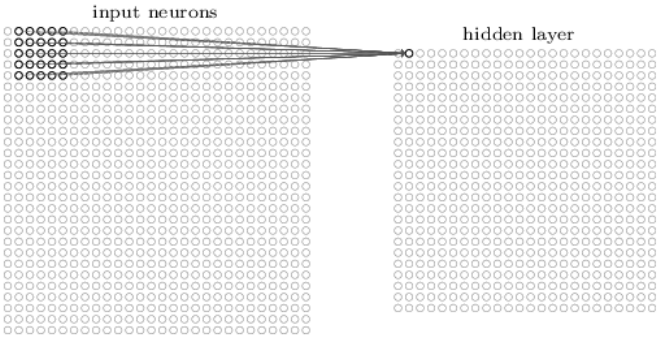


Figure 2.8: Example of a convolution operation applied to a local receptive field [60].

In the discrete domain the formulae for convolution can be written as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m,j-n)$$

Sometimes for implementation this is re-written as equivalent

$$S(i,j) = (K * I)(i,j)$$

as operation is commutative. Also, alternatively, the calculation can be done for *cross-correlation* where the kernel is not flipped. This is used in implementation by many neural network libraries.

The layers in convolutional network are sparsely connected. This is illustrated in Fig.

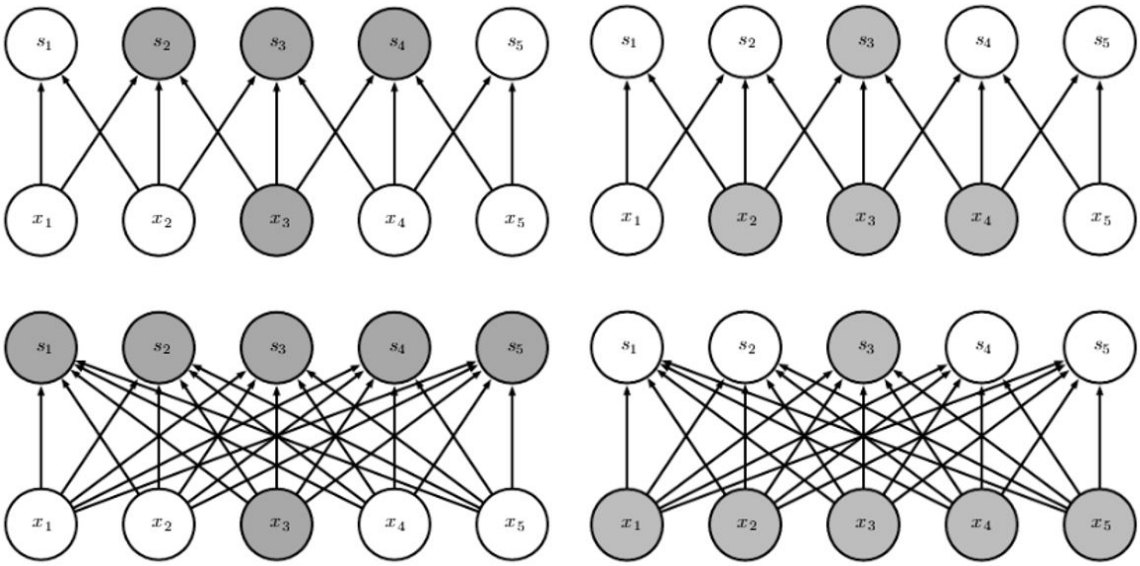


Figure 2.9: Comparison of connectivity in a traditional NN (bottom row) and CNN (top row). Image on the left shows the effect of a single input pixel $x3$, image on the right show the receptive field of a single output pixel $s3$ [23].

2.9 - input $x3$ only affects some of the output pixels, where in a traditional neural network - all outputs are affected by all inputs. Same for the receptive field - output pixel $s3$ is only affected by 3 inputs rather than all.

The connectivity can be even sparser if *stride* bigger than 1 is used, i.e. kernel is not applied to every pixel, but to every 2^{nd} pixel, 3^{rd} etc. This is equivalent to *downsampling* in full convolution function and is used for computational efficiency and low-rate sampling.

Usually, the convolution operation is combined with *pooling*. This can be:

- A maximum value in the neighbourhood (*max pooling*)
- Average of a neighbourhood (*average pooling*)
- L^2-norm of the neighbourhood
- Weighted-average based on a distance from the central pixel

Usually neighbourhoods are square of size 2x2 or 4x4. Pooling operation removes small changes, makes features invariant to small translations, or can introduce an arbitrary invariance to other transformations depending on the parameters [23].

So, a typical *layer of CNN* comprises of the following - see Fig. 2.10:

1. Convolution
2. Activation function (ReLU for example)
3. Pooling

The CNN benefits from having *multiple convolutional layers* - which is the "deep" part in deep-learning. For example, 16-19 layers in VGGNet, 22 layers in GoogleLeNet/ Inception, and up to 152 layers in ResNet - all of these neural networks were developed in 2014 - 2015 [13].
CNNs are used in computer vision extensively - they have a proven application in *image classification and object recognition*, but they can also be used for *segmentation*, *super-resolution* and *image generation*. The last application is used in this research in the form of encoder-decoder architecture, where initial convolution layers are used to extract features, then *up-sampling* layers are used to create a new image using the extracted features.

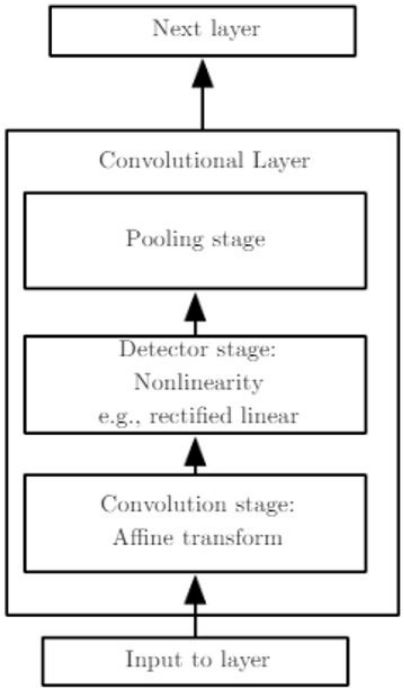


Figure 2.10: Typical layer of a CNN [34]

2.3.2 Capsule Networks

There are certain drawbacks of CNNs. Spatial relationships between components are not very important to a CNN. Fig. 2.11 illustrates this concept - both images in the figure are recognised as "face" by CNN. This is because in CNN features are combined into higher level features based on a weighted sum, so the positional parameter is lost [66].
[30] developed CapsuleNets to counteract this problem. CapusleNets use iterative pro-

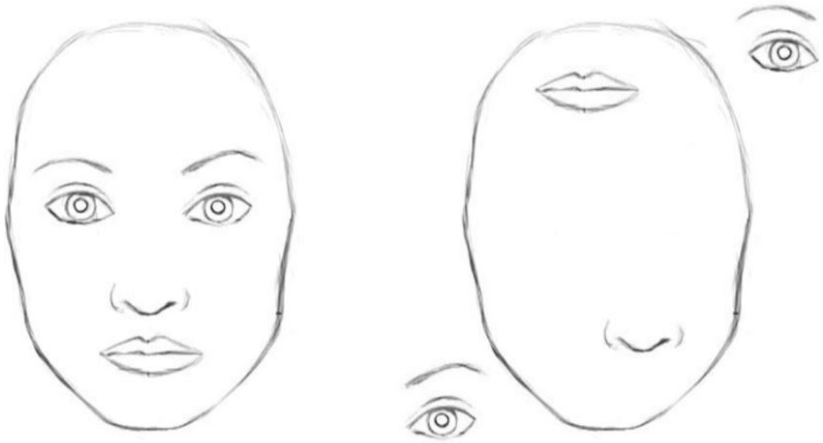


Figure 2.11: Both images are classified as a "face" by CNN

cess called "routing-by-agreement", that "updates the probability with which a part is assigned to a whole based on the proximity of the vote coming from that part to the votes coming from other parts that are assigned to that whole" [30]. This allows knowledge of familiar shapes to derive segmentation.

A capsule network has several layers of capsules. Each capsule consists of a 4x4 pose matrix, M, and an activation probability, a. CapsuleNets convert the set of activation probabilities and poses of the capsules from one layer into the next layer using *Expectation-Maximisation procedure* (EM) [30].

[30] trained CapsuleNet and CNN for comparison on the same small set of 3D objects (5 classes, 5 objects for each class) imaged from different azimuth, elevations etc. Each object had 18 different azimuth, 9 elevations and 6 different lighting conditions. The author shows that although there is no advantage over CNNs on familiar viewpoints. CapsuleNets performed considerably better on *novel viewpoints* (13% error vs 20% on different azimuth, 12% vs 18% error on different elevation).

This research is very new and so far, was applied to classification and object segmentation tasks: ([8], [41]). It seems however that it should have good potential for video interpolation as this presents the subject from different novel points of view.

2.3.3 Recurrent Neural Networks

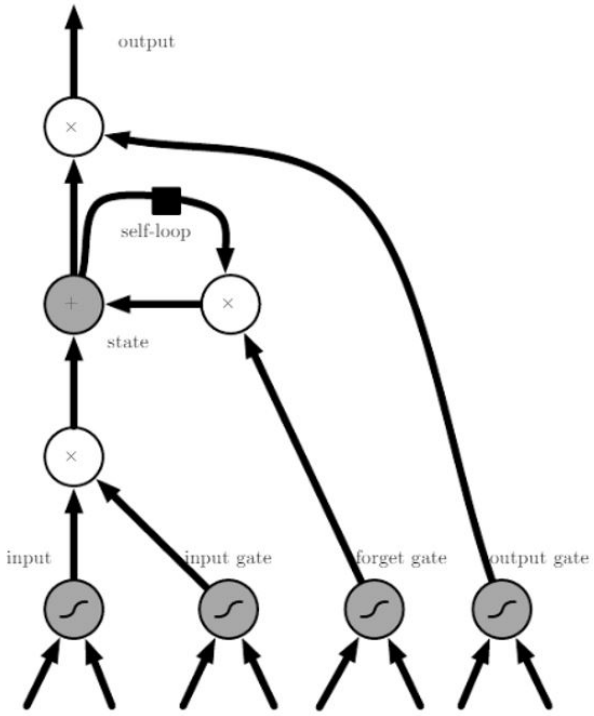


Figure 2.12: Architecture of LSTM recurrent network "cell" [23].

Recurrent Neural Networks (RNNs) are designed for processing sequential data, this can be a sequence of human limb positions in a video or a sequence of words in a sentence. If we had τ positions (states) in a sequence, the output from processing the position $t \in [0, \ \tau \ ]$ is passed for processing to position t+1. RNNs and in particular *gated RNNs* (Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)) are known to be very effective at natural language processing, handwriting recognition, speech recognition, image captioning and parsing [23].

There are many different architectures of RNNs, the simplest once having one layer, but they have been shown to be more effective with multiple layers, i.e. deep RNNs [65]. Some allow input not just from the previous state, but from all the previous states, some allow self-input.

Below is the architecture of LSTM recurrent network "cell". LSTM NNs allow not

only the input from the previous states in the sequence, but also self-input which is *gated* by the previous states. "Cells" are connected recurrently to each other.
While natural language processing is the most known use of RNNs, there are some relevant applications for video processing and video generation which are discussed in the next chapter.

2.3.4 Autoencoder

Autoencoders (AE) are one of the oldest types of neural networks that has existed for decades. This neural network can generate new images from input images and is employed in generative neural networks discussed below.

The network consists of two parts:

1. encoder function $h = f(x)$, where x is the input (image) and
2. decoder function $r = g(h)$ that produces a reconstruction based on the input.

h - is the *code* used to represent the input, for example, the features of the input image [23].

At its simplest $x = g(f(h))$ meaning that the original image is copied, which is not very useful. The network therefore needs to be constraint by *regularizers*. *Undercomplete and sparse autoencoders* can learn features and be used for *dimensionality reduction*. *Denoising autoencoders* (DAE) were used to de-noise images. *Contractive autoencoders* (CAE) produce tangent vectors similar to PCA, so they learn a more powerful nonlinear generalization of PCA [23].
While autoencoders themselves are used for dimensionality reduction and information retrieval tasks, they are the theoretical foundation for the more advanced generative networks.

2.3.5 Generative Adversarial Networks (GANs)

GANs have received recent attention. Pioneered in 2014 by [24], this network consists of two neural networks:

Figure 2.13: Examples of images generated by GAN. Rightmost column shows examples of the original images [24].

1. a generative model G that generates the images or video, and
2. a discriminative model D that attempts to distinguish the generated images from real-samples. It outputs the probability that a sample came from the training data rather than the generating model G

The training procedure for G is to maximize the probability of D making a mistake. If $G(z,\ \theta_g)$ is the differentiable function represented by a multilayer *generator* perceptron with parameters θ_g. And $D(x,\ \ \theta_d)$ is the second multilayer *discriminator* perceptron that outputs a scalar. The value function is then calculated as:

$$\min_G \max_D\ V(D,G)\ = E_{x\sim P_{data}(x)}\left[\log D\left(x\right)\ \right] - E_{z\sim P_z(z)}[1-D(G(z))\ ]$$

The work of [24] was further improved on by [69] who created a deep convolutional GAN (DCGAN). DCGAN performs very well in the image synthesis tasks - see examples in Fig. 2.14.
[15] expanded on GANs allowing these to generate images from a requested category - conditional GANs. His network is called LAPGAN.

2.4 Neural networks in view interpolation

Several researchers proposed view interpolation methods using neural networks. This section reviews the advances in this field.

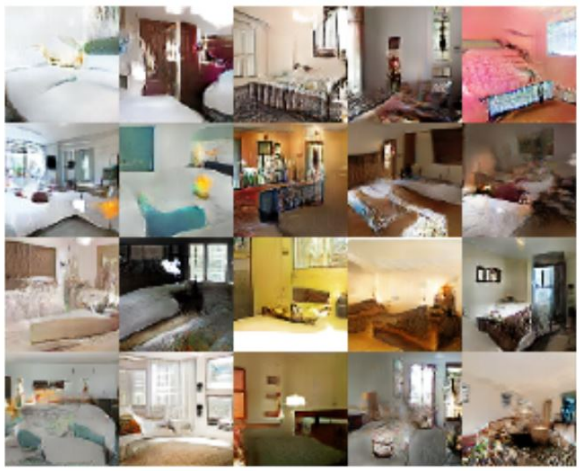


Figure 2.14: Images of bedrooms generated by DCGAN [23].

2.4.1 Deep-learned optic flow

The optic flow approach for image interpolation described above relies on the accurate estimation of the optic flow. Several researches proposed optic flow neural networks. For example, FlowNet by [17] used CNN for flow estimation, FlowNet2.0 by [33] improved the quality of the estimation by stacking several CNNs. EpicFlow by [71], task-oriented flow by [98] and Spatial Pyramid Network SPyNet by [70] are other examples. [61] uses PWCNet optical flow neural network by [84] in his work on view interpolation - brings the frames closer together, before producing the final image using a different neural network.

The challenges in training deep neural network for optical flow estimation - different scale of motion, changes in brightness etc - are similar to view interpolation and therefore are relevant to this work.

2.4.2 View interpolation based on deep-learned phase

[58] proposes a new CNN - PhaseNet - for view interpolation. Their work is "based on the intuition that motion of certain signals can be represented by the change of their phase" [58](Fourier transform). PhaseNet "mirrors the hierarchical structure of the phase decomposition which it takes as input. It then predicts the phase and amplitude values of the in-between frame level by level. The final image is reconstructed

from these predictions at different levels". PhaseNet CNN directly estimates phase decomposition of the intermediate frame without optical flow. The authors propose a new loss function - "phase loss" - based on the phase difference between the prediction and the ground truth. To cope with different scales and orientations the input images are first decomposed into complex-valued sub-bands $R_{\omega,\theta}(x, y)$ using steerable pyramid filters $\psi_{\omega,\theta}$. This forms the input to their decoder network where resolution is increased level by level. The training was based on 10k triplets from the DAVIS video dataset, from which 256x256 patches were selected randomly. Their work is compared to early implementation by Niklaus [63], where it performs better only under certain scenarios, like changing light conditions.

2.4.3 Pixel-by-pixel view interpolation

[105] propose video synthesis approach based on deep voxel flow (DVF). They build up on previous paper on Deep Voxel Flow by [47]. The architecture of the neural network (DVF-RCL) is based on encoder-decoder - the first one uses CNN, the second one RCL (Recurrent Convolutional Layers). The network ouput is 3D-voxel flow, which is passed to a volume sampling layer to generate the intermediate frame guided by the flow. 3D voxel flow field $F = \{\Delta x, \Delta y, \Delta t\}$ is separated into $F_{motion} = \{\Delta x,\ \Delta y\}$ and $F_{mask} = \{\Delta t\}$. The spatial component F_{motion} represents optic flow and the temporal component F_{mask} defines the weights for tri-linear interpolation. The authors use l_1-norm and l_2-norm losses for training, however they find that l_1-norm produces blurry results. l_1 was not able to capture perceptual differences well and high-frequency details. They then used *perceptual loss*, which is accepted as useful for artistic style-transfer and image super-resolution. [47] found that perceptual loss generates visually more realistic looking results as compared to pixel-wise loss. They used UCF-101 dataset for training with approx. 13K videos in 101 action categories and used triplets of frames from randomly cropped 128x128 patches. DeepFlow2 is used to predict optical flow [93]. The result was compared among other research to that of early Niklaus [63], also [47] and PhaseNet [58]. They find that PSNR numerical metric is better for perceptual loss L_f.

Next, we discuss in detail 3 implementations by S. Niklaus et al in detail:

- "Video Frame Interpolation via Adaptive Convolution" [62]
- "Video Frame Interpolation via Adaptive Separable Convolution" [63]
- "Context-aware Synthesis for Video Frame Interpolation" [61]

In their early work [62] proposed to estimate the color of the pixel in the interpolated frame by convolving the two input frame patches with individually estimated special adaptive convolution kernel. They called this method *adaptive convolution (AdaConv)*. Kernel estimation is done with CNN. This method replaces the traditional two step approach to video frame interpolation using optical flow - first motion estimation, then pixel synthesis - with a single step of local convolution over two frames. "The convolution kernel captures both the local motion between the input frames and the coefficients for pixel synthesis"[62], so no separate optical flow estimation step is required. See Fig. 2.15 for explanation of the *Adaptable convolution* method.

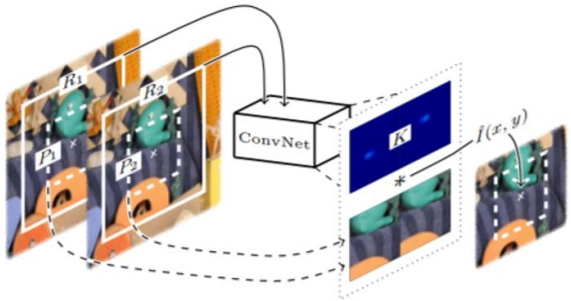
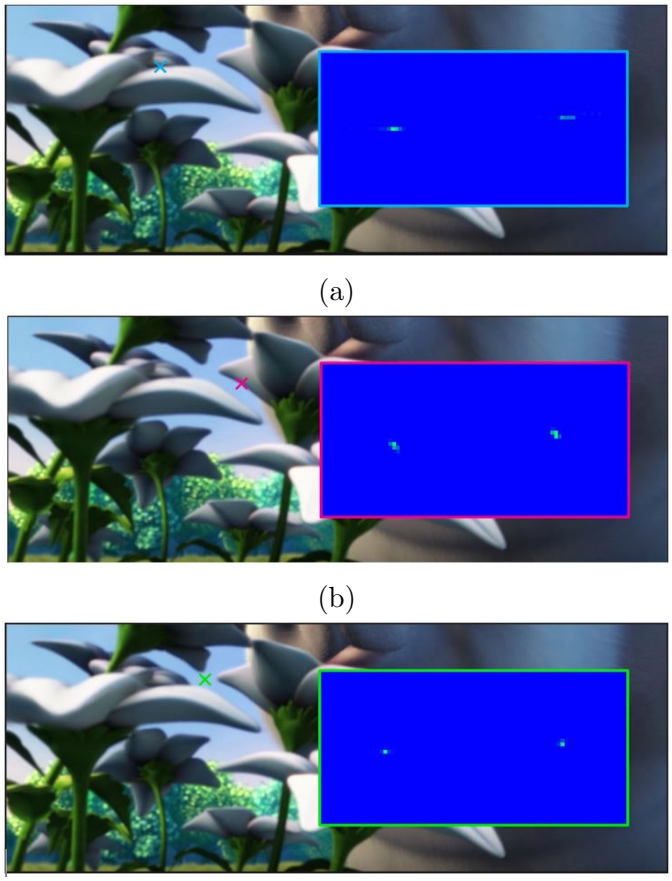


Figure 2.15: Illustration of Adaptable convolution method. The neural network receives 2 receptive field patches - $R1$ and $R2$. The NN estimates convolution kernel K for the selected pixel. Convolution of patches $P1$ and $P2$ is performed with kernel K to synthesise the output pixel [62].

Fig. 2.16 shows some example convolution kernels estimated by the NN. It can be seen that the kernel takes pixel surroundings into the account - the examples are for pixels on the horizonal edge, diagonal edge and in a uniform area without texture. The authors assert that sharp results can be achieved with edge-aware convolution kernels.

The CNN used 6 convolution layers with down-convolutions in between and produced kernel images of size 41 x 82 pixels. Authors chose receptive field patches of 72 x 72 pixels and convolution patches of 41 x 41 pixel. The patches are larger than the *largest*



(a)

(b)

(c)

Figure 2.16: Kernels estimated by CNN for pixels on the horizonal edge (a), diagonal edge (b) and in a texture-less area (c). The latter has isotropic kernel. From [62].

motion estimated in the images of 38 pixels. To handle the aperture problem, receptive field patches are a little larger than the convolution patches. CNN uses a combination of L1 and gradient loss functions. This initial implementation used spatial *softmax* layer for generating the kernel images. Same kernel is applied to all 3 colour channels. The training dataset came from carefully selected triple-patches from *youtube* videos - this approach therefore has large advantage over training the neural networks for optic flow as this requires ground-truth optic flow data.

[63] improved performance of their original method by using 1D separable kernels instead of 2D kernels and estimating kernels for all image pixels at once. The latter improvement allows to add *perceptual loss* to training the neural network. The network was changed to employ *encoder-decoder architecture* that acts on 2 pairs of 1D kernels. Down-convolution layers were replaced with average pooling. The authors tested several options for *up-sampling*: transposed convolution; sub-pixel convolution; nearest-neighbour or bi-linear interpolation. 1D kernels were 51 pixels in size.

If the above method estimates the pixel colour at (x, y) as:

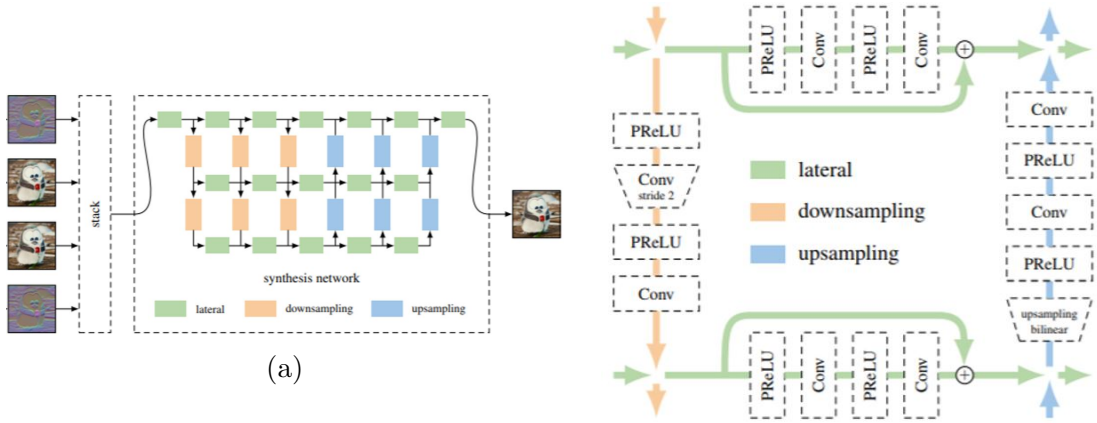
$$\hat{I}\,(x,y) = K_1\,(x,y) * P_1\,(x,y) + \; K_2(x,y) * P_2(x,y)$$

Where $K_1,\ K_2$ are convolution kernels for image 1 and image 2 at (x,y) and $P_1,\ \ P_2$ are patches around (x,y) in the corresponding images. *Adaptive separable convolution (SepConv)* approximates K_1 as $k_{1,h} * k_{1,v}$ and K_2 as $k_{2,h} * k_{2,v}$. Thus, the number of kernel parameters reduces from n^2 to $2n$ [63].

Perceptual loss is based on the comparison of ground-truth and NN output high-level features of images. The authors had several options for finding the features in the images and they used *feature reconstruction loss* based on the relu4_4 layer of the VGG-19 network. The perceptual loss is defined as:

$$L_F = \left\| \varphi\left(\hat{I}\right) - \varphi(I_{gt}) \right\|_2^2$$

Where φ function extracts features from an image, $\hat{I}$ is the interpolated frame and I_{gt}



(a) (b)

Figure 2.17: Grid Net architecture from [61]. Both pre-warped images and per-pixel contextual information images are fed to NN (a). GridNet encoder-decoder NN processes images at 3 different scales. Down-sampling, up-sampling and lateral building blocks are details in (b).

is the ground truth image.

In their latest work [61] aimed to be able to produce an intermediate frame at an arbitrary point in time as well as aiming to incorporate *contextual information* for each pixel. The model consists of the following steps:

1. Estimation of bi-directional flow with PWC-Net neural network for estimating optical flow (author's own adaptation from [84])

2. Extract per-pixel contextual information - using response of the conv1 layer from ResNet-18 [29]. The contextual vector describes pixel's 7×7 neighborhood.

3. Pre-warp both images and contextual information to time t between the two images using bi-directional flow information.

4. Generate interpolated frame $\hat{I}$ using *encoder-decoder* neural network called *GridNet*. The NN processes images at 3 scales: per-row channel-sizes of 32, 64, and 96. See Fig. 2.17 for configuration of the NN.

The authors introduce new loss function that performs better than perceptual loss

qualitatively - *difference between Laplacian pyramids*:

$$L_{Lap} = \sum_{i=1}^{5} 2^{i-1} \left\| L^i\left(\hat{I}\right) - L^i(I_{gt}) \right\|_1$$

The results of the video frame interpolation are compared to there two previous works (AdaConv, SepConv), optical flow approach [97], deep learning-based optical flow algorithm [93], phase-based approach [58].

2.4.4 View interpolation based on GANs

Several researches also attempted to use GANs for video interpolation - [42], [74]. These can employ the above approaches to generate the interpolated images using CNN or RNN, but have an additional *discrimator* neural network for ensuring the generated images look natural. The discriminator network attempts to tell if an image is synthetically generated or is a real-world image - and will reject images that look synthesised.

[42] used multi-scale CNN and WGAN-GP (Wasserstein GAN with gradient penalty) for video interpolation. Their GAN is based on a version of GAN proposed by [2] called Wasserstein GAN which was shown to be more stable than traditional GAN, the authors use DCGAN for comparison. [42] use *pyramid structure* for capturing the motion information of objects in images based on *feature pyramids* [45], as opposed to *image pyramids* [31]. They proposed a new CNN for generating the interpolated image and use WGAN adversarial training to keep generated frames natural. CNN has a simple structure for down-sampling and deep neural network for up-sampling. They use three part loss function for training: L1 loss, the generative adversarial loss and the perceptual loss, and UCF-101 and HMDB-51 datasets. They compare their results in PSNR and SSIM to [74], [56] and [47].

[31] propose an extrapolation/ interpolation framework called Multi-Scale Frame - Synthesis Network (MSFSN) where the frame can be generated at any point in time, not just in the middle of 2 frames. Their network represents a multi-scale pyramid of GAN networks. Temporal aspect is achieved by feeding the temporal ratio as well as the images for training, the training is based on triplets of images. Loss is represented

as a combination of 4 terms: pixel reconstruction loss; feature reconstruction loss to encourage similarity in feature representation; adversarial loss for matching the distribution of generated images to the data distribution in the target feature domain; and a new transitive consistency loss to enhance their custom *mapping function* G with more constraints. The mapping function G is defined as the function predicting the frame of interest: $y_{t_p} = G(x_{t_1}, x_{t_2}, t_p)$.

2.4.5 Combination approaches - based on RNNs and pose estimation

The above approaches addressed generic view interpolation for any 2 images from a video input. Combination approaches are also possible - for example, for interpolation of human motion videos human pose estimation can improve the quality of the interpolation.

As RNNs are good at extending sequences, there is current research into modelling human motion with RNNs. This can be used for synthesising new frames based on the modelled motion. For example, [32] extract human pose from video data from YouTube using Part Affinity Fields (PaF). Each part affinity is a 2D vector field for each limb, encoding location as well as orientation information across different body parts. This data can then be fed to RNN (3-Layer LSTM neural network in this case) to generate human-like motion for a given class of motion from annotated video data.

This approach seems relevant to view interpolation for this project as human motion is the primary focus of this research. If poses are estimated between 2 images - it should be possible to predict the pose in the interpolated image. Also, it may be useful for obtaining the training data as it can identify the videos with certain class of motion.

Chapter 3

Methodology

The following subjects are covered in this chapter:

- Neural Network design, incl. the NN configuration, choices for loss function and hyperparameters.
- Multi-view cameras dataset: quick review of available real datasets and methodology deployed for the generation of synthetic multi-view dataset for the training of NN.
- Evaluation techniques, including 2D evaluation of the quality of the interpolated images and 3D evaluation using Shape-from-Silhouette reconstruction.

3.1 Neural Network Description

3.1.1 Network Architecture

The interpolation of the images was performed using a deep CNN. The configuration of the neural network is adapted from [35] and is unchanged. The following stages are applicable to the network:

1) **Downsampling:**

6 blocks of 3 convolutional layers each (so altogether Downsampling stage is 18 convolutional layers deep) gradually increasing the number of hidden layers from 32 to 512.

These are normal convolutions of size 3 x 3, the image is padded each time and retains the original size within each block. There is a ReLU activation after each convolution. Each block except the last one finishes with a pooling layer of size 2 x 2. Initially the image size is 128 x 128, so after 5 pooling layers there are 512 hidden layers of size 4 x 4.

2) **Upsampling:**

Downsampling is followed by Upsampling stage with skip connections back to the Downsampling layer's output. There are 4 upsampling blocks each consisting of 1 upsampling layer, one convolutional layers and activation (ReLU). After each block the result is matched with the corresponding in size downsampling block and the latter is added to the result (Skip connection). Next a convolutional block integrates the added downsampling output. There are fewer upsampling blocks than downsampling blocks, as the last block is replaced by 1D kernel upsampling and convolutions.

3) **Kernel estimation:**

Upsampling is followed by estimation of vertical and horizontal 1D kernels for each pixel. The neural network is split into 4 sub-nets - vertical and horizontal 1D kernels for each image ($k_{1,v}$, $k_{1,h}$, $k_{2,v}$, $k_{2,h}$). Each subnet has 3 convolutional layers with activation, 1 upsampling layer and 1 more convolution.

4) **Creation of the final interpolated image with separable convolution**

Lastly the output image is generated by applying the vertical and horizontal kernels per patch in a convolution to produce each pixel.

$$\hat{I}(x,y) = K_1(x,y) \otimes P_1(x,y) + K_2(x,y) \otimes P_2(x,y)$$

where $P_1(x,y)$ and $P_2(x,y)$ are the patches centered at (x,y) in image 1 and image 2 correspondingly, and $K_1(x,y)$ and $K_2(x,y)$ are approximations of 2D convolutional kernels with two 1D kernels: $k_{1,v} \otimes k_{1,h}$ and $k_{2,v} \otimes k_{2,h}$.

Fig. 3.1 provides details of the size of each layer and summarises the architecture - image from [63].

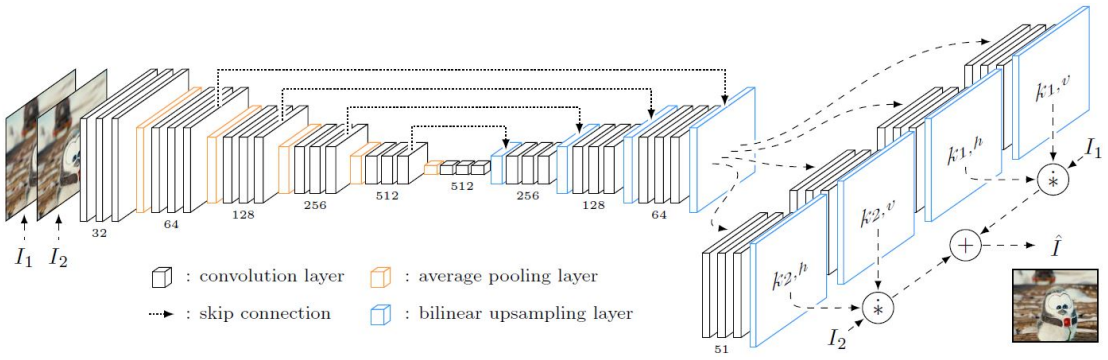


Figure 3.1: Overview of Neural Network architecture. Image from [63]

Data format:
While the dataset is described below, it is good to describe the data format that the neural network accepts here. The network is trained on the triplets of images - left camera image, ground truth (GT) camera, right camera image. In the preparation of the dataset for the training random patches are selected centered around the same point (x, y) for each triplet. The patches are augmented for the purposes of training and arrive at size 128 x 128. The left and right images are combined into a single tensor with 6 channels of size 128 x 128. The neural network accepts batches of these tensors as input to produce the interpolated images in the batch. The GT images are only used for loss estimation.

3.1.2 Network Loss

After the interpolated images are produced in the batch, the neural network estimates the loss of each image. The loss function is the effective driver of the networks learning. [106] have shown that with the same network architecture the quality of the results can be significantly better even with the same network architecture.
[106] compare the effect of different loss functions in the case of image restoration and determine that L_2 loss (Mean Squared Error) frequently does not produce the best result for the images, which they attribute to the way Human Visual System (HVS)

senses the images. For example, L_2 error imposes larger penalty on large errors and is more tolerant to small errors independent of the underlying structure of the image. HVS treats texture-less regions differently - in these regions it is more sensitive to luminance and color variations. Similarly, [63] suggests that L2 loss produces blurry results.

Loss for a patch P for an error function E can be written as

$$L^E(P) = \frac{1}{N}\sum_{p\in P} E(p)$$

where N - number of pixels p in the patch (from [106])

Below is the list of loss functions suggested for training the neural networks for image generation and their mathematical expressions:
1) L_2 loss (Mean Squared Error, quadratic)
MSE maximises the likelihood of Gaussian random variables.

$$L^{l_2} = \frac{1}{N}\sum_{p\in P}(x(p) - y(p))^2$$

where $x(p)$ and $y(p)$ are the values of the pixels in the generated and GT patches.
2) L_1 loss (Mean Absolute Error)
MAE is harder to compute than MSE and it is not differentiable at 0, but it does not give greater influence to larger errors, so is suitable for image processing.

$$L^{l_1} = \frac{1}{N}\sum_{p\in P}|x(p) - y(p)|$$

3) SSIM loss
SSIM is a perception-based evaluation that considers image degradation as perceived change in structural information, while also incorporating important perceptual phenomena, including both luminance masking and contrast masking terms [104].
SSIM is defined for a neighbourhood of pixel p as:

$$SSIM(p) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}\frac{2\sigma_x y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} = l(p)c(p)s(p)$$

where l is luminance, c is contrast and s is structure at pixel $p(x, y)$. These values depend on a set of parameters - μ_x and μ_y are the average values of x and y respectively, σ_x^2 and σ_y^2 are the variances, σ_{xy} is the covariance of x and y on a window of selected size (11 x 11 pixels in this dissertation). $c_1 = (k_1 L)^2$,$c_2 = (k_2 L)^2$,where L - the dynamic range of the pixel-values. 3 $k_1 = 0.01$, $k_2 = 0.03$.

The loss function for SSIM is:

$$L^{SSIM} = \frac{1}{N} \sum_{p \in P} 1 - SSIM(p)$$

[106] suggest that different values of variance σ influence the visual qualities of the output patch - for example, smaller values of σ correctly represent edges (larger values create a halo of noise around the edge). At the same time smaller values of σ fail to keep the local structure and introduce splotchy artifacts, while larger values of σ do not produce splotchy artifacts.

4) MS-SSIM loss (Multi-scale SSIM)
Similar to SSIM but computed at multi-scale given a diadic pyramid of M levels:

$$MS - SSIM(p) = l_M^{\alpha}(p) \prod_{j=1}^{M} c(p) s_j^{\beta_j}(p)$$

where l, c and s are luminance, contrast and structure at pixel p as defined previously at scale M and j.

5) Laplacian pyramids loss
Suggested by [61] this loss measures the difference between Laplacian piramids. This loss separates local and global features at different scales, depending on the number of considered levels.

$$L^{Lap} = \sum_{i=1}^{5} 2^{i-1} \left\| Lap^i(\hat{I}) - Lap^i(I_{gt}) \right\|$$

where $Lap^i(I)$ is the i-th level of Laplacian pyramid of image I. $\hat{I}$ and I_{gt} are predicted and GT images respectively.

The authors in [61] suggest that Laplacian loss produces more pleasing visual results then L_1 loss.
6) Feature loss
Also suggested by [61], this loss is based on features ϕ extracted by a certain layer from VGG-19. It is calculated as L_2 distance between features in two images.

$$L^F = \left\| \phi(\hat{I}) - \phi(I_{gt}) \right\|_2^2$$

For this research two loss functions were deployed:
- L1 loss
- Perceptual loss based on SSIM
While it would be interesting to try the other loss functions, time limitations of this dissertation and GPU computing power did not permit such analysis.

Combining the loss functions or switching the loss function during training

Several researchers ([106], [35], [63]) suggested either combining loss functions with a weighted sum or training some number of epochs with one loss and then switching to a different loss function. For this dissertation the second approach was adopted for one of the tests - switch from L1 to SSIM loss after 50 epochs.

Implementation details for NN
The neural network is implemented in Python and PyTorch. The training was performed on NVIDIA GEFORCE GTX 1050 GPU. Training took 2.5 hours per epoch for 51-kernal pixel and 5.8 hours per epoch for 71-kernel pixel.

3.1.3 Hyperparameters

There were two hyperparameters changed for the network training: kernel size and batch size. The rest of the parameters were un-changed from [35], but are stated here for completeness.

1) Activation Function

Convolutional layers are interlaced with ReLU activations. Activation function ReLU - Rectified Linear Unit - is formulated as :

$$f(x) = max(0, x)$$

Generally this activation is used for training the convolutional neural networks as it is fast to compute, compared to other activation options: Sigmoid, TahH, LeakyReLU, etc, while the other options have not been shown to have better performance.

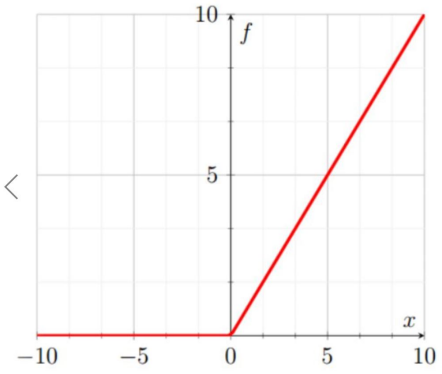


Figure 3.2: ReLU activation function

2) Optimizer and learning rate

Optimizer used in the training is `torch.optim.adamax` function with learning rate of 0.001. Adamax is Pytorch implementation variant of Adam algorithm based on infinity norm - there is also a straightforward Adam implementation (`torch.optim.adam`). Adam as a method of stochastic optimization proposed by [39]. The name Adam is derived from *adaptive moment estimation*. According to the authors of the method: "Adam only requires first-order gradients and therefore has little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients."

The method combines the advantages of two previous methods: AdaGrad [18] and RMSProp [87]. The first method works well with sparse gradients, while the second is applicable in on-line and non-stationary settings [39].

The learning rate used in the training is related to the choice of the optimizer and is set as 0.001. For example, [53] analyses the performance of different optimizers depending on the learning rate only - see Fig. 3.3. The authors run the test on MNIST dataset training CNN with Tensorflow. It can be seen (see grey line in the figure) that the learning rates of 0.0001-0.01 are suitable for Adam optimizer, albeit the training times vary. Similar results were obtained by the author of this dissertation when testing with FCN (fully connected network) on MNIST digits dataset.

Figure 3.3: Performance of different optimizers depending on the learning rate (time of more than 120 seconds means the network failed to train. Image from [53].

3) Batch size

The experiment deployed batch size of 32 for training the networks with 51-pixel size kernel and batch of size 16 for training the networks with 71-pixel size kernel. This was primary due to the memory limitations - 71-pixel size kernel requires over 2GB of memory (854pixels x 480 pixels x 3 colors x 71 kernel x 32 batch size = 2,794Mb), while 51-pixel kernel requires just under 2GB. The former could not be processed with the graphics card available.

4) Kernel size

This hyperparameter is specific to the *Adaptive Separable convolution* method used in this research. The output of upsampling the left and right images is split into 1D horizontal and vertical kernels of specific size. This kernels are utilised to calculate the value for a single pixel in the image. Larger kernel size will mean that a larger area around the pixel is analysed and used for the reconstruction of the pixel. [63] used kernel size of 51-pixel, as there largest motion was just over 40 pixels. The images used for training and testing this neural network were sometimes as far as 160 pixels apart, so a bigger kernal size was attempted - 71-pixels. The results of the two neural networks are compared in Chapter 4. It is possible to attempt to work with kernel size

of 160-pixels. The batch size would have needed to be reduced to 8 because of memory limitations. This is something that can be done in the future iterations.

Other hyperparameters included: number of hidden layers/ channels (left unchanged from the original model), weight initialization (gradients were initialised to zero) and data augmentation (random rotation between -90°and +90°, random vertical/ horizontal flip, random temporal order). The models were trained to 50 epochs.

3.2 Multi-view camera dataset

The main contribution of this dissertation was to investigate multi-view camera datasets suitable for training the neural network or alterntively to design a synthetic dataset.

3.2.1 Real-life datasets

Real-life datasets were analysed: see below table of online multi-view datasets that were considered 3.1. There are several requirements on the contents of the dataset: firstly, *many subjects* are required. Several datasets had a large number of people participating - e.g. OU-ISIR Gait database had 10307 people [86], HUMBI had 164 people [100], Casia Gait filmed 124 people [107].
Secondly, the *cameras needed to be close together* and needed to have , *3 cameras within about 30 °angle*, where the middle camera could be used as the GT. This is because the neural network is trained to create a pixel from the available pixels in left and right image - so the scale of motion is a constraint.
4 datasets had enough cameras to satisfy this requirement - OU-ISIR Gait, HUMBI, Dyna [67] and Casia Gait, the rest of the datasets had cameras too far apart to be suitable for the training. OU-ISIR Gait dataset had 7 cameras within 90 °angle, and the same amount of the other side of the circle, so the angle between cameras is exactly 15 °. HUMBI had 72 cameras focused on the body (the rest were pointing at the face of the subject) situated in 2 rows, so 10 °between each two cameras in each row. Casia Gait had 11 cameras at 18 °to each other on one side of the room only, so cameras are a little wider than required.
Lastly the *quality of the images* was of concern: HUMBI and Casia Gait only had

Database name	Contents	Number of cameras	Number of people
OU-ISIR Gait Database [86]	People walking Silouettes only	14	10307
AVAMVG [50]	People walking Full PNG images	6	20
HumanEva [79]	6 human motions incl walking	7	4
HUMBI [100]	Human poses. Available size 192x108 pxls for 72 cameras	107	164
KTH [36]	Football players	3	2
Human 3.6M [101]	Human poses	4	11
Dyna [67]	Human poses	22	10
Casia Gait [107]	People walking Image size 323x242 pxls	11	124

Table 3.1: Online multi-view datasets

images of small size available - 192x108 and 323x242 respectively. The latter also had very low quality of the images - in low lighting and blurry. OU-ISIR Gait dataset provides images of large size - 1280 x 960, but only image silhouettes are available for the download.
Also, none of the available footage was *filmed in a green room*, which was the suggested benefit to the dataset.
Another limitation of the available datasets was - *the camera angle is fixed within the dataset*, even when many subjects are filmed. So, any neural network would be over-trained on the particular camera setup and may not work as well in other camera setups

After having considered the available real multi-view datasets it was deemed appropriate to design a synthetic dataset to overcome all of the above limitations. In the future it may be beneficial to add the real data to the synthetic dataset and re-train the network.

3.2.2 Synthetic dataset

Synthetic multi-view dataset was created for the purposes of this research.

The human models for this dataset are based on research by [49]. They present Skinned Multi-Person Linear model (SMPL) - "a learned model of human body shape and pose-dependent shape variation". The learned shape is based on 1700 registrations for males and 2100 for females, the learned poses are based on 891 pose registrations spanning 20 females and 895 pose registrations spanning 20 males. As the result the authors offer a female and a male model that come with 10 shape blendshapes and 207 pose blendshapes.

The human models, their motion and multi-view camera setup around these are modelled in Python module in Blender. The workflow for generating the initial moving models comes from [91]. The authors create a synthetic dataset called SURREAL that they use to train a neural network for estimating human pose, shape, and motion from images. We adapt the same approach to generate a synthetic dataset for producing multi-view camera images.

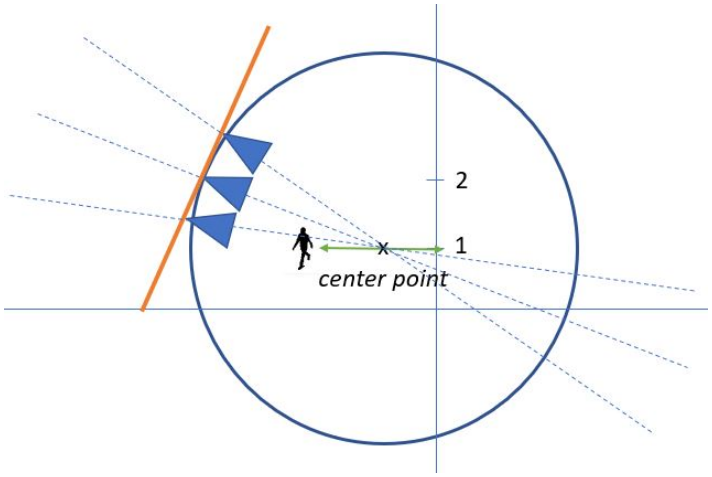


Figure 3.4: Illustration of the camera setup in a Blender file. Green arrow show the person's range of movement. Blue circle - the positions possible for GT camera. The left and right camera are always on a tangent to GT camera focus line.

The following is the workflow for generating the multi-view camera dataset (steps 1-4, 7 are following from SURREAL dataset, steps 5-6, 8 are custom):

1) Select random gender (male/ female)
2) Select random clothes texture (930 female, 925 male options)
3) Apply cloth texture to body parts (24 body parts)
4) Select random motion capture file (5338 motions)
5) Generate random position for the GT camera and add the camera.
6) Generate positions for left and right cameras and add the cameras.
7) Apply motion capture at selected key frames (for this dataset - 10 frames per file taking every 30th frame, so that there is a range of motion captured from each motion file, but only 10 frames, so that the network can look at many randomly generated models)
8) Store image captured at every key frame for the 3 cameras.

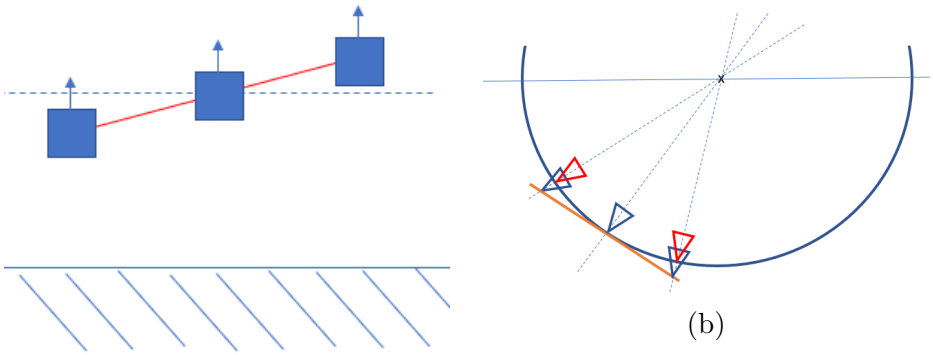
All images have the same background - a cut from the real green-room footage. Over 800 different random scenes were generated to produce 8506 triplets of images for training the neural network and further 1322 triplets for testing.
Steps 5-6 are described below in detail.

Generating random positions for the 3 multi-view cameras
The position of the *GT camera* has the following parameters - see Fig. 3.4 for the illustration of the camera setup:

- Distance from camera to "center" point (radius): 3 to 5 m
- Camera yaw around the vertical axis through the "center" point: 0 to 360°
- Camera height: 0.9 to 1.9 m
- Camera roll random - from pointing to the "center" point: -10°to +10°

In addition the *left* and *right* camera have the following parameter:

- Left and Right camera distance from the central camera (same distance for both): 0.05m to 50 m



(a) (b)

Figure 3.5: Illustration of the possible improvements to the dataset: (a) Left and Right camera pitch to horizontal line, (b) Cameras away from the tangent line.

The parameters are random and picked for each scene with a random character. About 10-20 key frame images are taken with the current camera setup and the current character, then the next scene will have a different camera setup/ character.

All 3 cameras are pointed towards the "center" - which is a point between the position of the person in the first frame and the center of their movement. This is to maximize the likelihood of the person always being fully in the shot. Sometimes this was not the case and the algorithm removed all empty triplets from the training set. So, while most samples have the person fully in the shot, there are some cases where the person is too close to the camera or walking out of the shot - these "half-shots" were deemed suitable for the training and remained in the dataset.
To state some of the limitations of the current dataset that will need to be addressed in the future:

1. All 3 cameras are on the same level. It would be beneficial to change the "pitch" of the left and right cameras - as this would be relevant to realistic setups. See Fig. 3.5 (a) for illustration of the concept.

2. 3 cameras are pointing towards the "center" point. Sometimes multi-view setup would have cameras setup in parallel, so a range of values from pointing to "center" point to "parallel" should be introduced in the setup. The author attempted

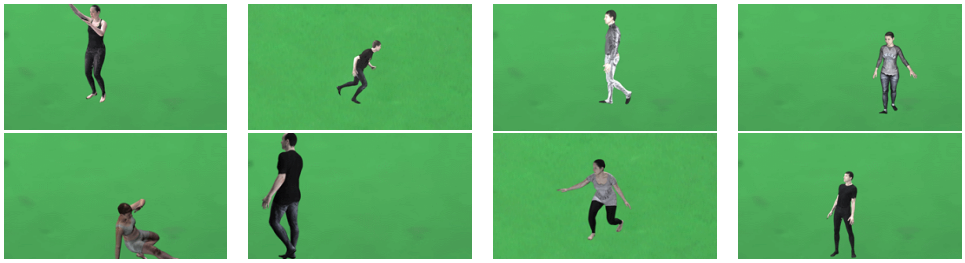


Figure 3.6: Examples of images generated.

to have the cameras "parallel", but found that this meant that the character in the left and right image were further in terms of pixels, so created a harder case for interpolation. On the other hand, having cameras focus on a single point improved the interpolation results.

3. All 3 cameras are on a single tangent line. In reality cameras would frequently be setup on a circle around the subject, so this case should be addressed. See Fig. 3.5 (b) for illustration of the concept.

4. Background is always green and therefore only useful for the green-room setup. This is not useful for real-life scenarios, unless the background is removed first. The dataset can be generated "in a cube" with real 3D backgrounds to overcome this limitation. SURREAL dataset, for example, used a set of backgrounds that came from LSUN dataset [99] - however these were 2D backgrounds.

See Fig. 3.6 for examples for images produced. 8504 triplets of images were generated (approx. 850 different scenes, characters and motions). From this - triplets of patches were generated randomly. Upto 20 patches were allowed to come from the same image. All triples were checked for the character being present in the shot - any with at least 2 green images were removed. 140505 triples were included in the final patches dataset used for the training. In addition 1322 triplets were produced for testing.

3.3 Evaluation

This section describes the evaluation methods applicable to assessing the quality of the generated images. First, 2D image comparison methods are discussed: PSNR, SSIM

and Silhouette assessment. Next, the steps taken for 3D evaluation are discussed - including Shape-from-Silhouette reconstruction details and Hausdorff distance measure.

3.3.1 MSE and PSNR

Quality metric traditionally used for assessing the image quality is the mean squared error (MSE). This calculates the average square intensity differences between pixels in a reference image and a "distorted" image. A version of this metric frequently applied to measuring image quality is: Peak Signal to Noise Ratio (PSNR). While MSE represents the cumulative squared error, PSNR is a measure of the peak error. The lower the value of MSE, the lower the error and higher the quality of the image. PSNR is measured in decibels (dB). Higher PSNR means better quality of the reconstructed image.

First, MSE is computed as:

$$MSE = \frac{\sum_{M,N}[I_1(m,n) - I_2(m,n)]^2}{M \cdot N}$$

where M is number of rows and N is number of columns in the input images. Next PSNR is computed as:

$$PSNR = 10log_{10}(\frac{R^2}{MSE})$$

where R is the maximum fluctuation in the input image data type, so for the color images used in this research its a number between 0 and 255.

The calculations are implemented using PyTorch `torch.Tensor` to speed up the calculations.
For this research, because the image background is green and the same for all images, the green pixels in the range of the background image were not considered. The results in Chapter 5 therefore only consider significant pixels (non-green).

3.3.2 SSIM

According to [92] PSNR and MSE do not convey perceived visual quality. They developed measure of structural similarity (SSIM) based on a hypothesis that *Human Visual System* (HVS) is highly adapted for extracting structural information. *Structural information* is the idea that the pixels have strong inter-dependencies especially when they are spatially close. These dependencies carry important information about the structure of the objects in the visual scene [55]. SSIM also incorporated important perceptual phenomena - luminance masking and contrast masking terms. *Luminance masking* is a phenomenon whereby image distortions tend to be less visible in bright regions, while contrast masking is a phenomenon whereby distortions become less visible where there is significant activity or "texture" in the image [73].

The formulae for calculating SSIM was provided in section 3.1.2 Network Loss (see SSIM Loss).

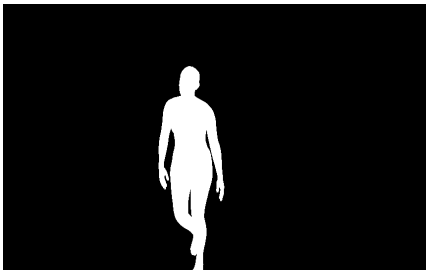
SSIM is also calculated using PyTorch `torch.Tensor`, which is particularly important when using SSIM as a loss, as all evaluations are batched. For this research, green pixels were excluded from SSIM evaluations - all graphs below are for significant pixels only.

3.3.3 False Negative and False Positive Silhouette Pixels

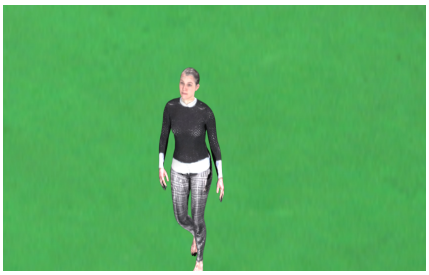
As part of the evaluation of the quality of images included a 3D reconstruction with Shape-from-Silhouette (SfS), for the task of SfS only *silhouettes* in the images are considered. While the correctness of the colour of the pixels may be important for video interpolation and FVV, for SfS the color information is discarded as long as the shape (of the person) stands out from the background. It was therefore deemed reasonable to evaluate the quality of the silhouettes only. The 2 measures proposed are:

- Ratio of False Negative (FN) pixels to the number of pixels in the shape
- Ratio of False Positive (FP) pixels to the number of pixels in the shape

(a) FN (red) and FP (green) pixels



(b) Silhouette of the GT image



(c) GT image

Figure 3.7: Example of False Negative (FN) and False Positive (FP) pixels (a). GT only shown (b, c) as visually the interpolated image and its silhouette are very similar to GT - both silhouettes are required to produce (a).

False negative pixels are counted where GT image has silhouette, and the interpolated image does not. False positive pixels are the opposite - where the GT does not have the silhouette, but the interpolated image does. See Fig. 3.7 for demonstration of the concept - red pixels are False Negative, green pixels are False Positive.

In the context of SfS, False Negative pixels would mean that voxels are removed during the reconstruction, so these are more dangerous than False Positive pixels, that have the potential to add voxels. Referring to Fig. 3.7 again - there will be missing voxels around the hands and leg, and a lot of voxels added around the shape. This bad example of interpolation was selected for the purposes of demonstration.

3.3.4 Hausdorff Distance

The last part of the evaluation addressed the hypothesis that interpolated multi-view camera images can be used in photogrammetry to aid 3D reconstruction. Only Shape-from-Silhouette reconstruction was attempted.

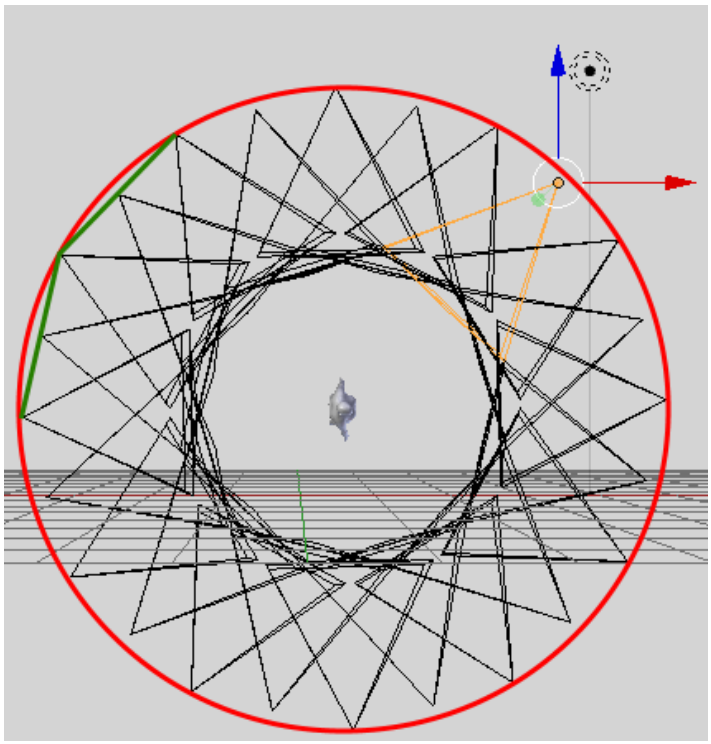


Figure 3.8: Illustration of the camera setup for SfS reconstruction. 12 cameras are real (on the circle), 12 are "synthetic (on the green segments).

The evaluation steps were as follows:

1. *Generate 12 real camera images around the test subject.*

   When the initial synthetic test dataset was generated, the resultant Blender files (.blend) containing the generated models were saved by Python. Another Python script loads these and replaces the original 3 cameras with 12 cameras in a *perfect circle* around the model. Unlike cameras in the test dataset, the 12 cameras for

SfS reconstruction were always focused on the person. This created ideal conditions - which would not be the case in the green room. So, a further test with the person being away from the single focal point of all cameras is required in the future.

The following parameters were varied for SfS Test set:

- Radius of the circle of cameras (3.5m - 5.5m) - to fit the entire model.
- Camera height - 0 - 1m up from person's waist height.
- Starting position on the circle. Then each next camera was situated at a 30°angle.

Both camera images and the camera extrinsic and intrinsic parameters were exported from Blender.

2. *Generate theoretical camera positions for the interpolated images.*
   12 "interpolated" cameras were generated in addition - for the purpose of obtaining the *camera parameters* for the interpolated images. There were no images generated from these cameras.
   The cameras are situated in the middle of the segment connecting each 2 of the neighbouring real cameras - exactly simulating the position of the "synthetic" camera that would have produced the interpolated images. See Fig.3.8 - synthetic cameras are situated on the green segments.

3. *Generate 12 interpolated images using the selected neural network model*
   A separate Python script generated the interpolated images using the NN model for each pair of the real cameras.

4. *Extract Silhouettes*
   Next, silhouette images were extracted from 24 images (12 real and 12 interpolated). The extraction of the silhouettes was done based on the color of the pixel - all pixels in the "green" range determined from the background image were set to 0. The clothes texture of the synthetic people rarely had green color, so pixels in the model were not removed.

5. *Shape -from-Silhouette reconstruction*
   The SfS reconstruction used 3^{rd} party code (Vacancy) which is a voxel carving implementation in C++ [90]. The code takes camera intrinsic and extrinsic parameters and the silhouette images, and outputs both the voxel representation and the surface reconstruction.

   The reconstruction saves all stages - the following two were used:

   - The result of the reconstruction with 24 images - with the interpolated images.
   - The result of the reconstruction with 12 images (real camera images only) to use for comparison.

6. *Align and simplify meshes*
   Three meshes (2 reconstructed meshes from the previous step and GT mesh from Blender) were imported into MeshLab [12]. MeshLab's "Align" functionality was applied to align the meshes.
   Next, as the number of vertices in the reconstructed meshes (around 50K) were much higher than the number of vertices in GT mesh (around 7K). The two reconstructed meshes were simplified using MeshLab's "Simplification: Clustering Decimation" function with the default parameters. As the result the reconstructed meshes now had around 6K vertices.

7. *Hausdorff distance*
   Hausdorff Distance between two meshes is a the maximum between the two one-sided Hausdorff "distances" [11]:

$$d_H(X,Y) = \max\{\sup_{x\in X}\inf_{y\in Y} d(x,y), \sup_{y\in Y}\inf_{x\in X} d(x,y)\}$$

   Distance $d(x,y)$ is computed by taking samples (vertices or faces) from mesh X and searching for each x the closest point y on a mesh Y. So the choice of the "first" mesh is important.

   MeshLab's "Hausdorff distance" function lets the user choose the one-sided distance rather than computing two distances and taking a maximum. For this

evaluation the distance was always computed *from the reconstructed mesh to GT mesh*. Also, *faces*, rather than *vertices* were sampled for the reconstruction - this was chosen as the reconstruction technique (SfS) is surface-based.

MeshLab outputs the Hausdorff distance in mesh units and also as a percentage of the bounding box diagonal. The results below are in mesh units.

Chapter 4

Results and Discussion

4.1 Comparison of trained networks

Several modifications of the multi-view neural network were attempted. Below is the table listing the attempted options - see Table 4.1. Also, the original network proposed by [35] was trained as described by the author using DAVIS videos dataset [14] (non-multiview) (DAVIS L1 51) to be used as a benchmark.

Model name	Kernel Size	Loss	Batch size	Max epochs trained
MV L1 51	51	L1	32	50
MV SSIM 51 (started with MV L1 51 model)	51	SSIM	32	10
MV L1 71	71	L1	16	50
DAVIS L1 51 (Benchmark)	51	L1	32	100

Table 4.1: Attempted Multi-view neutral network configurations

The hyperparameters that were changed are:

- Training loss: L1 or SSIM
- Size of 1D vertical and horizontal kernel: 51 or 71 pixels
- Batch size: 16 or 32

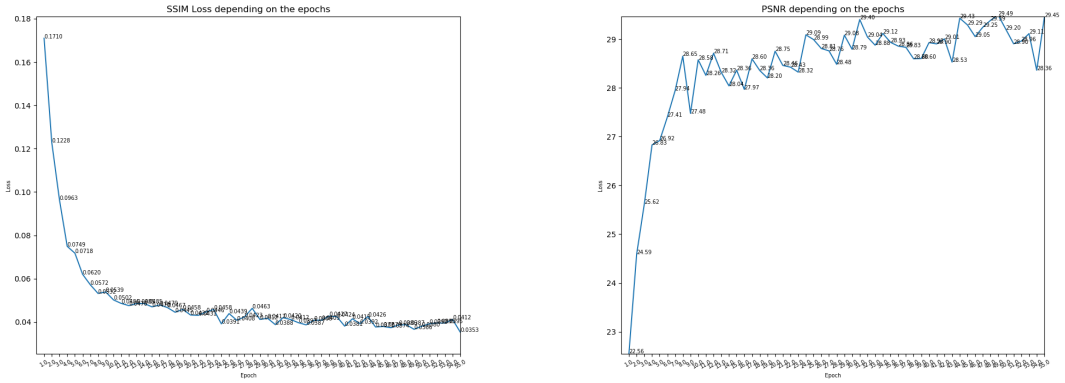


Figure 4.1: Improvement in SSIM Loss and PSNR (both include non-significant pixels) with the number of training epochs - MV L1 51 example.

- Different number of epochs: see Table 4.1

For the detailed description of the NN architecture please see section 3.1 in Methodology.

The models were trained to 50 epochs. As can be seen from Fig. 4.1 the improvement after 30 epochs is quite slow, albeit there is still some improvement in SSIM, so potentially further training (outside of the scope of this dissertation) can improve the quality.

Fig. 4.2 displays the result of applying the networks for interpolation. More examples can also be found in Appendix A and Appendix B. Working and not-working examples are given. As can be seen the results vary from perfect quality, to where the difference is only noticeable with a pixel-by-pixel subtraction, to visually obvious imperfections and ghost artifacts.

Table 4.2 compares results for the trained neural networks and the benchmark network - based on the average for a test sample of 1322 triplets of synthetic images. The best results with regards to SSIM and PSNR were achieved with the network trained with SSIM loss - MV SSIM 51 - see Table 4.2. However, the network trained with just L1 loss achieves better silhouette shapes - MV L1 51 - and has the lowest FN Ratio of 1.1%. So, although the pixel color may be more correct for the network with SSIM

loss, the shapes are better with L1 loss.

Model name	SSIM	PSNR	FN Ratio	FP Ratio trained
MV L1 51	0.8830	28.13	**0.0110**	0.0246
MV SSIM 51 (started with MV L1 51 model)	**0.8860**	**40.63**	0.0129	**0.0244**
MV L1 71	0.8611	31.28	0.0158	0.0278
DAVIS L1 51 (Benchmark)	0.8124	26.78	0.0164	0.0601

Table 4.2: Comparison of results for the trained Multi-view neutral networks

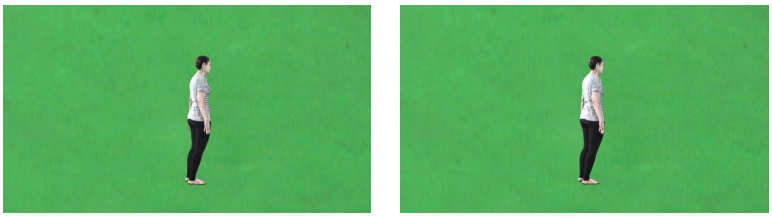
The next few sections examine the results in detail and discuss when the models work well and when they fail to produce visually correct results.

4.2 SSIM

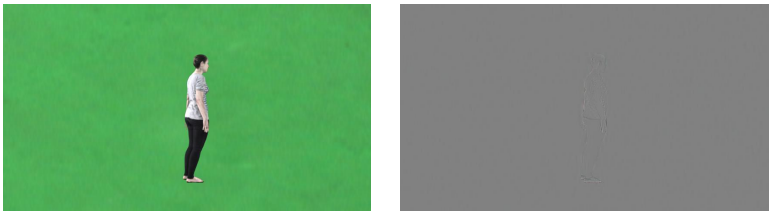
First evaluation criteria to discuss is SSIM (see section 3.3.2 for the description of this metric). The network trained on non-multi-view data (DAVIS L1 51) has SSIM of 0.8124, which is considerably lower than the rest of the trained networks - 0.8830, 0.8860 and 0.8611 - respectively for MV L1 51, MV SSIM 51 and MV L1 71.

Fig. 4.3 (a) displays *SSIM depending on the distance between left and right cameras*, so distance to ground truth is half this amount. SSIM is higher for small distances as can be expected. For models MV L1 51 and MV SSIM 51 is higher than 0.91 for distances between cameras upto 60 cm. It is above 0.81 for distances upto 90 cm between the interpolated cameras. The quality decreases with larger distances. Compared to non-MV network, where SSIM is above 0.91 for the first 2 intervals only - upto 40 cm between cameras - and is below 0.8 for distances above 60 cm.

The networks were also evaluated depending on the *distance to subject* - Fig. 4.3 (b). The cameras are located around 3-5 m from the "center" point, but because the subject

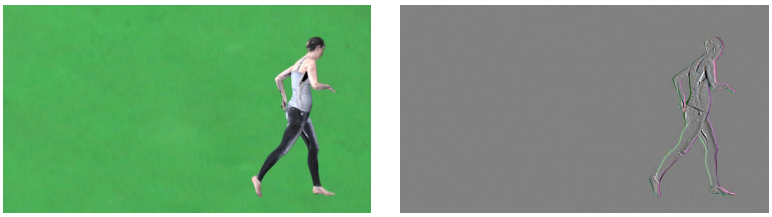


(a) Left (b) Right

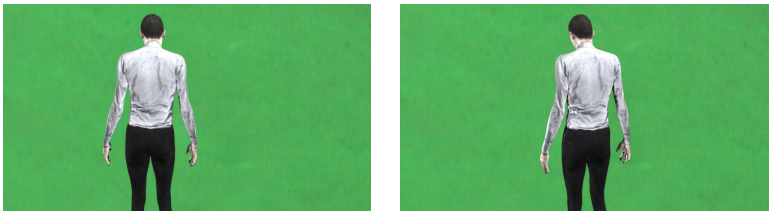


(c) Interpolated image (d) Visual Difference

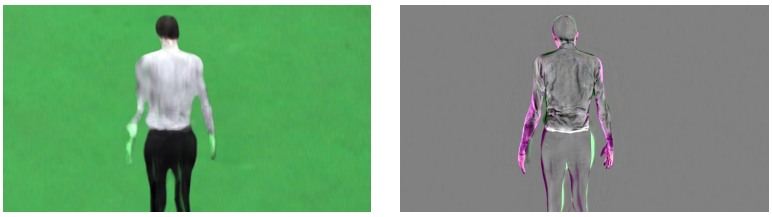
(e) Left (f) Right



(g) Interpolated image (h) Visual Difference



(i) Left (j) Right



(k) Interpolated image (l) Visual Difference

Figure 4.2: Examples of interpolated images (c, g, k) and visual difference with the ground truth image (model MV L1 51). See also Appendix A and B.

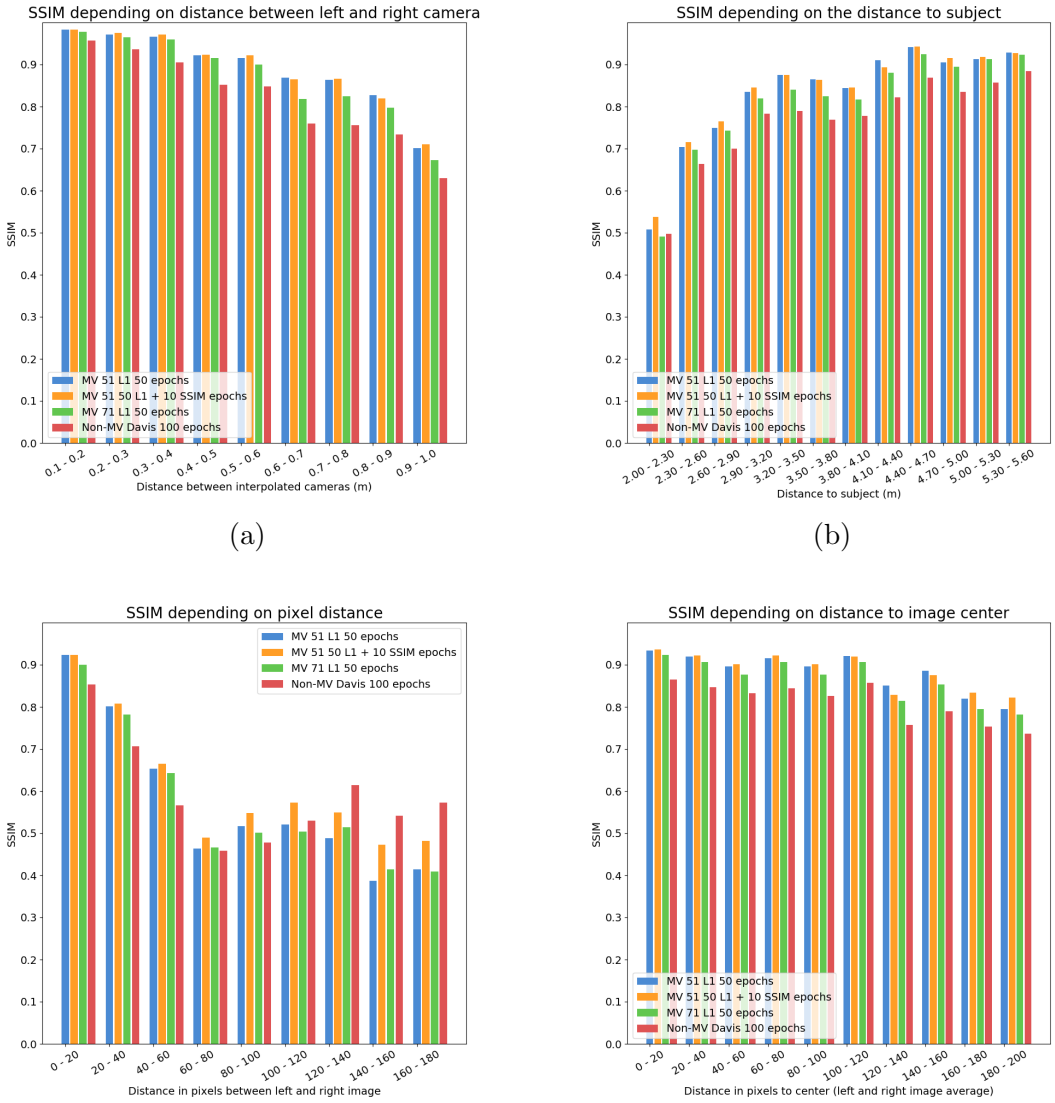
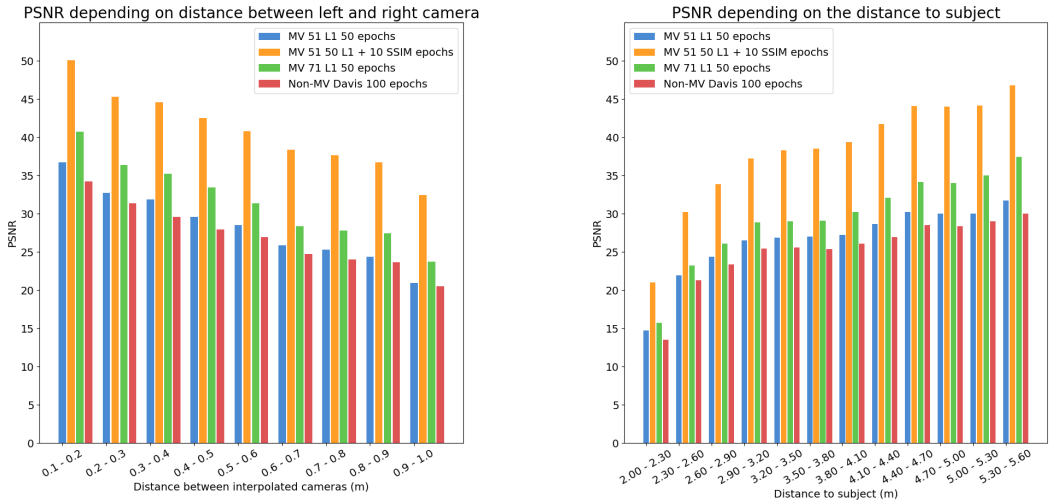


Figure 4.3: Comparison of NNs - SSIM depending on the distance between the interpolated cameras (a), distance to the subject (b), pixel distance between left and right image (c) and pixel distance of the models to the center of the image (d).

could move both towards and away from the camera, the distance to the subject can be outside of this range. However most samples in both training and test set still lie within 3-5 m, so the middle categories in the graph recieve more samples.
SSIM increases as the subject gets further from the camera - as both the amount of pixels being evaluated and the movement in pixels between left and right camera gets smaller. Potentially the results are better because the entire movement fits into the 51/71-pixel kernel. SSIM is above 0.91 for distances over 4m.
As most training data would be within 3-5 m range, the networks do not produce a good results when the subject gets closer than 3 m as the network has not seen enough of the training samples. Also, when the subject is close there is a larger distance in pixels between left and right image, so the entire movement is not potentially covered by the kernel. At distances <3m the person would normally not fit into the image entirely, albeit because training is based on patches the network should still be able to cope. It should be possible to train the network for closer distances by adjusting the samples in the training dataset and increasing the size of the kernel.

An interesting metric is *SSIM depending on the pixel distance between left and right cameras* - Fig. 4.3 (c). Pixel distance in this case is measured between the horizontal centers of the bounding boxes of the silhouettes in left and right image. This metric is a combination of the above two metrics - distance to the subject and the distance between left and right cameras: the pixel distance gets larger as the subject gets closer to the camera and the distance between the interpolated cameras grows.
As the result - the closer the subject is in pixels between left and right image, the higher is SSIM. SSIM is 0.92 for the 51-pixel kernel models when left and right image are less than 20 pixels apart. This measure falls abruptly with the distance increasing - it is around 0.8 for 20-40 pixel distance and is under 0.7 for larger distances, so is not satisfactory.
The kernel size of 51/71-pixels would explain such results. But also there is a different number of samples in each category - around half the test dataset has pixel distance within 10 pixels, and around 3/4 have the pixel distance below 20 pixels. So, the training would also be focused on smaller pixel distances.

Lastly, SSIM is plotted against the *distance of the subject to the center of the image*



(a) (b)

(c) (d)

Figure 4.4: Comparison of NNs - PSNR depending on the distance between the interpolated cameras (a), distance to the subject (b), pixel distance between left and right image (c) and pixel distance of the models to the center of the image (d).

- Fig. 4.3 (d). This distance is calculated as the average horizontal distance for left and right image from the subject to the center of the image. The models are not particularly effected by this parameter - there is a small decrease in quality as the subject gets further from the center of the image.

4.3 PSNR

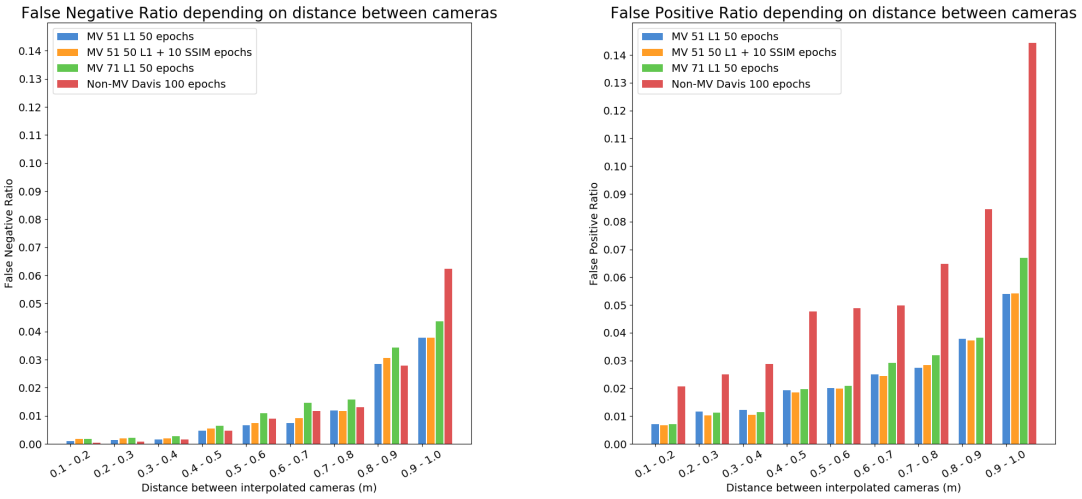
Similar trends are applicable to PSNR (see Fig. 4.4). Here the model trained with SSIM loss performs by far superior, producing results of over 40 on average. 71-pixel kernel network also performs well on PSNR - 31.28. All models perform better on PSNR than the benchmark non-multi-view model - at 26.78.

PSNR decreases as the *distance between left and right camera* increases. PSNR is over 40 for cameras within 60 cm, and above 30 for all categories for MV SSIM 51 model. For MV L1 71 model it is 40 for the first 10 cm distance, and above 30 within 60 cm.

For the *pixel distance between left and right image*, PSNR is good (>30) for MV SSIM 51 model for distances in pixels below 60 pixels. For MV L1 71 model this number is considerable lower (>20) for distances upto 60 pixels.

4.4 Silhouettes - False Negative Ratio and False Positive Ratio

For the photogrammetry method of Shape-from-Silhouette reconstruction, the minor errors in colour of the generated pixels have no significance for the silhouette extraction - as long as the silhouette stands out from the background. For this reason, the models were evaluated for the quality of the silhouettes. This evaluation criteria is described in the methodology chapter - section 3.3.3. The subtraction of the background was exact with the known values of colour for the background pixels, so the silhouettes could be evaluated with high accuracy.
The ratios were only evaluated for the distance between the left and right cameras.

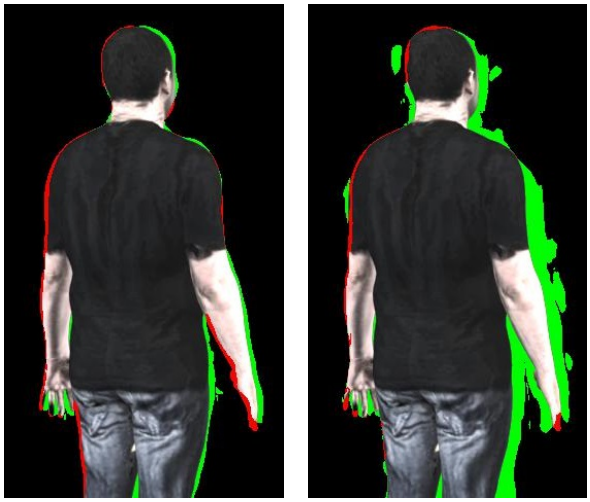


(a) (b)

Figure 4.5: Comparison of NNs - False Negative (FN) - (a) - and False Positive (FP) - (b) - ratios depending on the distance between the interpolated cameras.

Fig. 4.5 (a) shows that all networks, including the non-multi-view network, have few false negative pixels when distance between cameras is below 60 cm (less than <1%). The network trained with L1 loss and 51-pixel kernel performs best - the error rate is <0.5% for distances <50 cm, and is less than 1% for distances <70 cm. But at distance between cameras of 1m the best network can loose 3.7% of silhouette pixels, which can result in the corresponding loss of voxels during 3D reconstruction from just one interpolated image. This is a high amount considering that several interpolated images would be used during in the reconstruction.

The number of False Positive pixels is generally larger - Fig. 4.5 (b). Here the disadvantage of the non-multi-view trained network - DAVIS L1 51 - becomes apparent, strongly over-inflating the generated silhouettes. This can add 2% of pixels to the silhouettes at 20 cm between cameras and over 14% for distances between cameras of 1 m. This is compared to 0.7% and 5.4% for the best performing network. The example of how the non-multi-view network over-inflates a silhouette is shown in Fig. 4.6.



(a) (b)

Figure 4.6: Comparison of multi-view (MV L1 51) network - (a) - and non-multi-view (DAVIS L1 51) - (b) - network interpolation results - silhouettes false positive (red) and false negatives (green) are displayed. Non-MV network adds significantly more false positive pixels.

4.5 Visual comparison

It is interesting to compare the output of the networks visually - this difference is *only noticeable in results with low quality*, so the below *failed* sample was specially picked for demonstrating the network differences - Fig. 4.7.

The top row of the images contains images with *51-pixel* kernel at 30 epochs, 50 epochs and 50 epochs + 10 epochs with SSIM loss. The second row displays images with *71-pixel* kernel at 30 epochs, 50 epochs and 30 epochs + 10 epochs with SSIM loss (training was not extended to 50 + 10 SSIM loss epochs for 71-pixel kernel). The 3rd row displays the result for non-multi-view network and the ground truth.

It can be seen that albeit the network with SSIM loss has the highest SSIM/ PSNR, it is less accurate visually as compared to networks trained with L1 loss. The person has lost half of the left arm in case of MV SSIM 51 network - Fig. 4.7 (c). The sharpest result is produced by the network with 71-pixel kernel - Fig. 4.7 (e). For example, the network has kept the wrinkles on the shirt. And while 71-kernel network performs worse than 51-kernel networks both on SSIM/ PSNR and silhouettes, visually the result is appealing as the person has both arms reasonably shaped.

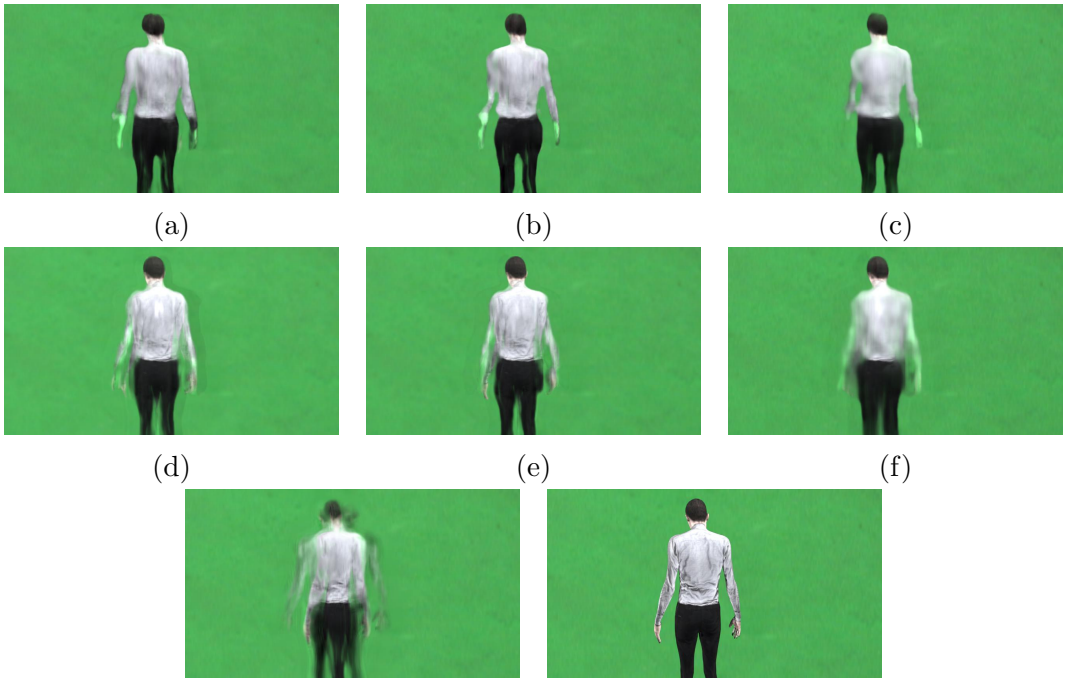


Figure 4.7: Visual comparison of the trained neural networks on a *failed* result. Last image - (h) - is *ground truth*. First row (a, b, c) has networks with 51-pixel kernel: MV L1 51 network 30 epochs, MV L1 51 network 50 epochs, MV SSIM 51 network (50 L1 + 10 SSIM epochs). Second row (d, e, f) has networks with 71-pixel kernel: MV L1 71 network 30 epochs, MV L1 71 network 50 epochs, MV SSIM 71 (30 L1 + 10 SSIM epoch). Lastly, (g) has the result for non-multi-view network DAVIS L1 51.

The non-multi-view trained network (DAVIS L1 51) duplicates the figure and fails to produce a single silhouette.

4.6 Hausdorff distance

For this dissertation a 3D test was created to check the hypothesis that the interpolated images created with a multi-view neural network could assist in 3D reconstruction. A number of cameras was placed in a perfect circle around the subject at distance of 3.5 to 5.5 m. Then the equivalent number of synthetic images were created by interpolating every pair of neighbouring cameras. The appropriate number of cameras for this

test was selected as 12, as this reflects a realistic green room setting. Further details of SfS reconstruction and Hausdorff distance are described in Methodology - section 3.3.4.

It needs to be said that the distance between the cameras relate to their number and the radius of the circle as:

$$D = 2\sin(\pi/N_{cam})r$$

where D - distance between left and right camera, N_{cam} - the number of cameras in the circle, r is the radius of the circle. So, for 12 cameras the distance between cameras is 1.55m at 3.5 m radius and 2.84m at 5.5 m radius - this is considerably larger than the distances that the network was trained for.

All of the below results relate to model with 51-px kernel trained with L1-Loss for 50 epochs (MV L1 51) - as this model achieved the best silhouettes and false negative ratio.

Table 4.3 displays the results of evaluting SfS reconstructions from 12 real cameras and 12 real cameras plus interpolated images. When evaluating from the ground truth mesh to the target reconstructed mesh - the results are better for the reconstruction from 12 real cameras. RMS for 12 cameras is 0.0189 which is a smaller error than RMS for 24 cameras - 0.0231. Mean Bounding box error is also smaller for 12 cameras - 0.0089 as compared to 0.0115.

Reconstruction from	BB min	BB max	BB mean	**BB RMS**
24-cameras (12 real+ 12 interpolated)	0.0	0.1270	0.0115	**0.0231**
12-cameras	0.0	0.1182	0.0089	**0.0189**

Table 4.3: Hausdorff distance. Average Bounding Box (BB) percentage

Fig. 4.8 and Fig. 4.9 display some examples of SfS reconstruction. It can be seen that 24-cameras produces a more refined model. However if interpolated images have some false negative pixels there is a risk of loosing voxels - see Fig. 4.9Fig. 4.9 (c). In this example, the limb is lost in 3D recontruction as compared to reconstruction with only real cameras.

(a) Ground Truth

(b) 12 real cameras

(c) 24 cameras (12 real plus 12 interpolated images)

Figure 4.8: Example output from SfS reconstruction using just real cameras vs real cameras plus interpolated images.

(a) Ground Truth

(b) 12 real cameras

(c) 24 cameras (12 real plus 12 interpolated images)

Figure 4.9: Example output from SfS reconstruction using just real cameras vs real cameras plus interpolated images. Note part of the leg that disappeared due to inaccuracies in the interpolated images - (c).

(a) Left (b) Interpolated image (c) Right

Figure 4.10: Applying the trained NN (MV L1 51) to real-life imagery from Green room.

4.7 Real data

The trained model was applied to real-life images. No ground truth was available for the images, so the only result available is visual - illustrated in Fig. 4.10.

As can be seen in the image - the neural network trained on synthetic images can be applied to real-life data and produces a realistic result. Notice that the network was able to add the gap between the legs and able to get the correct orientation of the body.

Currently the trained network has a limitation - the network can only be applied when the subject in the interpolated images is placed on the green background - the background that was used for training the network. For SfS reconstruction this limitation is not prohibiting as the silhouette of the subject will be extracted. However for FVV the image needs to be interpolated as a whole, so further work is required if this approach is to be used for free-viewpoint video - the network needs to be retrained with

3D background.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This research has investigated the possibility of generating spatially interpolated images to create novel viewpoints. It focused on interpolating images from multi-view cameras in a special setup, e.g. green room, with a single human performer. The approach endeavoured to deploy a neural network for creation of such images. Along with the general idea to generate images from multi-view camera images, there was a specific hypothesis to be answered - if synthetically generated intermediate images can help with photogrammetry. This research only partially addressed this hypothesis as only Shape-from-Silhouette reconstruction was attempted, and not Structure-from-Motion.

As the result of this dissertation a synthetic multi-view images dataset was created that could be used for training the neural network. Several neural network architectures were considered with the view of being used for the task. Also, different evaluation methods were considered for assessing the generated images - as well as a method proposed for assessing the quality specifically for the purpose of SfS reconstruction - silhouette false positive/ false negatives.

The following conclusions were made based on the results of this research:

1. CNN is capable of producing spatially interpolated images when cameras are not too far apart. These images can have the correct angle of view without prior 3D estimation.
   In this research the network was trained with maximum kernel of 71-pixels and the interpolated images had high quality for cameras upto 60 cm apart, or where the subject is no further than 20 pixels horizontally between left and right image.

2. A neural network can be trained on synthetic images and then applied to real-life imagery.
   Multi-view data for training a neural network can be hard to obtain, so using a synthetic dataset can be very helpful. The dataset created for this research had some limitations, i.e. static background image, static camera intrinsics, equal camera height for the interpolated cameras etc, but with a synthetic dataset it is a matter of adding the additional features to improve the dataset.

3. The neural networks trained for this research on multi-view images produced better interpolated results than the original network proposed by [35] for temporal interpolation.
   When applied to images that were far apart pixel-wise the temporal network produced duplicate silhouettes, where spatial networks better able to cope with the range of movement. The temporal network was also behind on both SSIM, PSNR and False Positive silhouette ratio for all categories. It performed well on False negative silhouette ratio, due to the network generously over-inflating the silhouettes.

4. With regards to 3D reconstruction with Shape-from-Silhouette, the reconstructions including the interpolated images were generally more refined than the reconstructions from just real cameras. However they performed worse on Hausdorff distance, usually due to missing parts of extremities. Any inconsistency pixel-wise between the ground truth and the interpolated images will result in loss of voxels during the reconstruction, even where the images have an acceptable quality visually.
   It is unlikely that a neural network can be trained to be pixel-wise precise, so this approach is not recommended for the use in photogrammetry.

5. The interpolated images were realistic looking, so this approach can be used in free-viewpoint video, if the neural network is retrained to be non-sensitive to the background. For this the image need to be taken in 3D setup, where not only the human performer, but the whole background is modelled.

6. Different neural network configurations generated different quality images. The networks trained with SSIM loss generally had blurrier results, albeit they performed best on SSIM and PSNR. The network trained with a larger kernel (71-pixel) and L1 loss was able to produce the sharpest details, i.e. keep wrinkles on a shirt. The network trained with 51-pixel and L1 loss kernel had the best results when silhouettes were examined.

5.2 Future Work

The task of interpolating multi-view images can be further addressed by the following improvements. However, as mentioned above the author does not believe that pixel-level accuracy can be achieved with more training to produce images suitable for photogrammetry.

1. Improve multi-view dataset:

    - Add pitch to left and right camera with regards to horizontal line
    - Add varied 3D backgrounds
    - Change camera intrinsics. This will allow to place the cameras closer to the subject.
    - Improve models - real clothes, more motion scripts
    - Add specifically parts that fail to interpolate - hands, legs, faces etc
    - Add real-life samples to the synthetic dataset

2. Neural network training:

    - Train with larger kernel to allow the network to cope with the larger range of motion. The load on GPU is the main prohibitor of introducing a larger kernel.

- Train with False negative/ false positive loss specifically to get the correct silhouettes.
- Train with GAN architecture, so that a discriminator network can discard images that look "interpolated" rather than "real". This approach should be able to deal with situations were arms were duplicated or faces smudged.

Also, with regards to SfS reconstruction - only the subject in the focus of the cameras was tested, it would be reasonable to test with the subject not being in the center of the images. More testing can also be performed where the distance between the cameras is smaller and the number of cameras is larger.

More testing is also required on real-life images preferably with ground truth from a green room.

Bibliography

[1] ABIResearch (2018). Augmented Reality & Virtual Reality Coverage. Market update. https://www.abiresearch.com/market-research/service/augmented-virtual-reality/.

[2] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. https://arxiv.org/abs/1701.07875.

[3] Ballan, L., Brostow, G. J., Puwein, J., and Pollefeys, M. (2010). Unstructured video-based rendering: Interactive exploration of casually captured videos. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2010)*, pages 1–11.

[4] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). SURF: Speeded up robust features. *Computer Vision and Image Understanding*, 110:346359.

[5] Belongie, S. (2009). CSE 252B: Computer Vision II. https://cseweb.ucsd.edu/classes/sp04/cse252b/notes.

[6] Calonder, M., Lepetit, V., and Fua, P. (2011). Brief: Binary robust independent elementary features. *11th European Conference on Computer Vision*.

[7] Chen, K. and Dirk, A. (2010). Image sequence interpolation using optimal control. *Journal of Mathematical Imaging and Vision*, 41.

[8] Chen, R., Jalal, M. A., Mihaylova, L., and Moore, R. K. (2018). Learning capsules for vehicle logo recognition. *21st International Conference on Information Fusion (FUSION)*.

[9] Chen, S. E. and Williams, L. (1993). View interpolation for image synthesis. *Proceedings of the 20th annual conference on Computer graphics and interactive techniques: SIGGRAPH*.

[10] Cheung, K., Baker, S., and Kanade, T. (2005). *Shape-From-Silhouette Across Time. Part I: Theory and Algorithms*. The Robotics Institute.

[11] Cignoni, P. (2010). Meshlab stuff. practical mesh processing experiments. https://meshlabstuff.blogspot.com/2010/01/.

[12] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. (2008). MeshLab: an Open-Source Mesh Processing Tool. In Scarano, V., Chiara, R. D., and Erra, U., editors, *Eurographics Italian Chapter Conference*. The Eurographics Association.

[13] Das, S. (2017). CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more ... https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5.

[14] DAVIS (2017). DAVIS: Densely Annotated VIdeo Segmentation. Semi-supervised dataset. https://davischallenge.org/davis2017/code.html.

[15] Denton, E., Chintala, S., Szlam, A., and Fergus, R. (2015). Deep generative image models using a laplacian pyramid of adversarial networks. *CVPR*.

[16] Dimitrios, C. and Antonios, G. (2012). *Three-Dimensional Scene Reconstruction: A Review of Approaches*, pages 142–162. IGI Global.

[17] Dosovitskiy, A., Fischer, P., Ilg, E., Husser, P., Hazirbas, C., Golkov, V., Smagt, P. v. d., Cremers, D., and Brox, T. (2015). Flownet: Learning optical flow with convolutional networks. *IEEE International Conference on Computer Vision*.

[18] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

[19] Faugeras, O. D. (1992). What can be seen in three dimensions with an uncalibrated stereo-rig? *Second European Conference on Computer Vision*.

[20] Flynn, J., Neulander, I., Philbin, J., and Snavely, N. (2015). Deepstereo: Learning to predict new views from the worlds imagery. *CoRR abs/1506.06825*.

[21] Fragneto, P. and Fusiello, A. (2012). Uncalibrated view synthesis with homography interpolation. *Second Joint 3DIM/ 3DPVT Conference 3d Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT 2012)*, pages 270–277.

[22] Furukawa, Y. and Ponce, J. (2009). Carved visual hulls for image-based modeling. *International Journal of Computer Vision*, 81:53–67.

[23] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[24] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks. https://arxiv.org/abs/1406.2661.

[25] Gurdan, T., Oswald, M. R., and Cremers, D. (2014). Spatial and temporal interpolation of multi-view image sequences. *Pattern Recognition, GCPR 2014*, 8753:305–316.

[26] Harris, C. and Stephens, M. J. (1988). A combined corner and edge detector. *Alvey-Vision Conference*.

[27] Hartley, R. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge, 2 edition.

[28] Hartley, R. I. (1995). In defence of the 8-point algorithm. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1064–1070.

[29] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*.

[30] Hinton, G., Sabour, S., and Frosst, N. (2018). Matrix-capsules with EM routing. *International Conference on Learning Representations*.

[31] Hu, Z., Ma, Y., and Ma, L. (2017). Multi-scale video frame-synthesis network with transitive consistency loss. *CVPR*.

[32] Hwang, J. and Shabbir, D. (2017). Human Motion Reconstruction from Action Video Data Using a 3-Layer-LSTM. http://cs231n.stanford.edu/reports/2017/pdfs/945.pdf.

[33] Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., and Brox, T. (2017). Flownet 2.0: Evolution of optical flow estimation with deep networks. In *CVPR*.

[34] Kang, S. B., Xin Tong, Y. L., and Shum, H.-Y. (2007). Image-based rendering. *Foundations and Trends in Computer Graphics and Vision*.

[35] Kartašev, M., Rapisarda, C., and Fay, D. (2018). Implementing adaptive separable convolution for video frame interpolation. https://arxiv.org/abs/1809.07759.

[36] Kazemi, V., Burenius, M., Azizpour, H., and Sullivan, J. (2013). Multi-view body part recognition with random forests. *British Machine Vision Conference*.

[37] Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. *Eurographics Symposium on Geometry Processing*.

[38] King, D. (2016). Easily create high quality object detectors with deep learning. http://blog.dlib.net/2016/10/easily-create-high-quality-object.html.

[39] Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

[40] Kolouri, S., Park, S. R., and Rohde, G. K. (2016). The radon cumulative distribution transform and its application to image classification. *IEEE Transactions on Image Processing*, 25(2):920–934.

[41] LaLonde, R. and Bagci, U. (2018). Capsules for object segmentation. *Medical Imaging with Deep Learning*.

[42] Li, C., Gu, D., Ma, X., Yang, K., Liu, S., and Jiang, F. (2018a). Video frame interpolation based on multi-scale convolutional network and adversarial training. *IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pages 553–560.

[43] Li, J., Li, E., Chen, Y., Xu, L., and Zhang, Y. (2010). Bundled depth-map merging for multi-view stereo. *CVPR*, pages 2769–2776.

[44] Li, S., Zhu, C., and Sun, M. (2018b). Hole filling with multiple reference views in DIBR view synthesis. *IEEE Transactions on Multimedia*, 20(8):1948–1959.

[45] Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017). Feature pyramid networks for object detection. In *CVPR*.

[46] Lipski, C., Linz, C., Berger, K., Sellent, A., and Magnor, M. (2010). Virtual video camera: Imagebased viewpoint navigation through space and time. *Computer Graphics Forum*, 29:2555 – 2568.

[47] Liu, Z. W., Yeh, R. A., Tang, X. O., Liu, Y. M., and Agarwala, A. (2017). Video frame synthesis using deep voxel flow. In *16th IEEE International Conference on Computer Vision*, pages 4473–4481.

[48] Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from twoprojections. *Nature*, 293.

[49] Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., and Black, M. J. (2015). SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16.

[50] López-Fernández, D., Madrid-Cuevas, F. J., Carmona-Poyato, A., Marín-Jimnez, M. J., and Muñoz Salinas, R. (2014). The AVA Multi-View Dataset for Gait Recognition. In Mazzeo, P. L., Spagnolo, P., and Moeslund, T. B., editors, *Activity Monitoring by Multiple Distributed Sensing*, Lecture Notes in Computer Science, pages 26–39. Springer International Publishing.

[51] Lowe, D. (2001). Object recognition from local scale-invariant features. *Proceedings of the IEEE International Conference on Computer Vision*, 2.

[52] Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application in stereo vision. *Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*.

[53] Mack, D. (2018). How to pick the best learning rate for your machine learning project. https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2.

[54] Mahajan, D., Huang, F.-C., Matusik, W., Ramamoorthi, R., and Belhumeur, P. (2009). Moving gradients: A path-based method for plausible image interpolation. *ACM Trans. Graph.*, 28.

[55] Manikandan, L., Anusha, M., and Fred, A. L. (2014). Structural similarity based efficient multi-view video coding. *IOSR Journal of Engineering*.

[56] Mathieu, M., Couprie, C., and LeCun, Y. (2016). Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations*.

[57] Matusik, W., Buehler, C., Raskar, R., Gortler, S. J., and McMillan, L. (2000). Image-based visual hulls. *ACM SIGGRAPH 2000 Conference Proceedings*, page 369374.

[58] Meyer, S., Wang, O., Zimmer, H., Grosse, M., and Sorkine-Hornung, A. (2015). Phase-based frame interpolation for video. In *CVPR*, pages 1410–1418.

[59] Morel, J. M. and G., Y. (2009). Asift: A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences*, 2:438–469.

[60] Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

[61] Niklaus, S. and Liu, F. (2018). Context-aware synthesis for video frame interpolation. *CVPR*, abs/1803.10967.

[62] Niklaus, S., Mai, L., and Liu, F. (2017a). Video frame interpolation via adaptive convolution. *CVPR*, abs/1703.07514.

[63] Niklaus, S., Mai, L., and Liu, F. (2017b). Video frame interpolation via adaptive separable convolution. *IEEE International Conference on Computer Vision*, pages 261–270.

[64] Pages, R., Amplianitis, K., Monaghan, D., Ondrej, J., and Smolic, A. (2018). Affordable content creation for free-viewpoint video and vr/ar applications. *Journal of Visual Communication and Image Representation*, 53:192–201.

[65] Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013). How to construct deep recurrent neural networks. *International Conference on Learning Representations*.

[66] Pechyonkin, M. (2017). Understanding hinton's capsule networks. part I: Intuition. https://medium.com/ai^3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b.

[67] Pons-Moll, G., Romero, J., Mahmood, N., and Black, M. J. (2015). Dyna: A model of dynamic human shape in motion. *ACM Transactions on Graphics, (Proc. SIGGRAPH)*, 34(4):120:1–120:14.

[68] Quan, M. and Lhuillier, L. (2005). A quasi-dense approach to surface reconstruction from uncalibrated images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):418–433.

[69] Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. *International Conference on Learning Representations*.

[70] Ranjan, A. and Black, M. J. (2017). Optical flow estimation using a spatial pyramid network. In *CVPR*.

[71] Revaud, J., Weinzaepfel, P., Harchaoui, Z., and Schmid, C. (2015). Epicflow: Edge-preserving interpolation of correspondences for optical flow,. In *CVPR*.

[72] Robaszkiewicz, S. and El Ghazzal, S. (2013). Interpolating images between video framesusing non-linear dimensionality reduction. *Proceedings of the 30th International Conference on Ma- chine Learning*.

[73] Sai Prasad Reddy, K. and Nagabhushan Raju, K. (2017). Video quality assessment metrics for infrared video frames using different edge detection algorithms. In *International Conference on Current Trends in Computer, Electrical, Electronics and Communication*, pages 231–236.

[74] Samsonov, V. (2017). Deep frame interpolation. https://arxiv.org/pdf/1706.01159.pdf.

[75] Schonberger, J. L. and Frahm, J. M. (2016). Structure-from-motion revisited. In *CVPR*, pages 4104–4113.

[76] Seitz, S. and Dyer, C. (1999). Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35.

[77] Seitz, S. M., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. *CVPR*.

[78] Shi, J. and Tomasi, C. (1994). Good features to track. *CVPR*.

[79] Sigal, L., Balan, A., and Black, M. J. (2010). Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *International Journal of Computer Vision*.

[80] Smith, M. W., Carrivick, J. L., and Quincey, D. J. (2016). Structure from motion photogrammetry in physical geography. *Progress in Physical Geography*, 40(2):247–275.

[81] Smolic, A. (2019). Augmented reality lectures. https://tcd.blackboard.com/bbcswebdav/pid-1169545-dt-content-rid-6756029_1/courses/CS7434-A-SEM202-201819/02_AR2019_CameraModel.pdf.

[82] Stich, T., Linz, C., Albuquerque, G., and Magnor, M. A. (2008). View and time interpolation in image space. *Comput. Graph. Forum*, 27:1781–1787.

[83] Strecha, C., Bronstein, A. M., Bronstein, M., and Fua, P. (2012). Ldahash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34.

[84] Sun, D., Yang, X., Liu, M.-Y., and Kautz, J. (2018). Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. *CVPR*, pages 8934–8943.

[85] Szeliski, R. (2010). *Computer Vision:Algorithms and Applications*. Springer.

[86] Takemura, N., Makihara, Y., Muramatsu, D., Echigo, T., and Yagi, Y. (2018). Multi-view large population gait dataset and its performance evaluation for cross-view gait recognition. *IPSJ Trans. on Computer Vision and Applications*, 10(4):1–14.

[87] Tieleman, T. and Hinton, G. (2012). Lecture 6.5 - RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

[88] Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography - a factorization method. *International Journal of Computer Vision*, 9(2):137–154.

[89] Torr, P. H. S. and Murray, D. (1997). The development and comparison of robust methods for estimating the fundamental matrix. *International Journal of Computer Vision*.

[90] VACANCY (2019). Vacancy: A voxel carving implementation in c++. https://github.com/unclearness/vacancy.

[91] Varol, G., Romero, J., Martin, X., Mahmood, N., Black, M. J., Laptev, I., and Schmid, C. (2017). Learning from synthetic humans. In *CVPR*.

[92] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13.

[93] Weinzaepfel, P., Revaud, J., Harchaoui, Z., and Schmid, C. (2013). Deepflow: Large displacement optical flow with deep matching. *IEEE International Conference on Computer Vision*, pages 1385–1392.

[94] Wenger, S. M. B. (2016). *Evaluation of SfM against traditional stereophotogrammetry and Lidar techniques for DSM creation in various land cover areas.* Thesis, Stellenbosch University.

[95] Werlberger, M., Pock, T., Unger, M., and Bischof, H. (2011). Optical flow guided tv-l1 video interpolation and restoration. *Proceedings of the 8th international conference on Energy minimization methods in computer vision and pattern recognition*, pages 273–286.

[96] Werlberger, M., Trobin, W., Pock, T., Wedel, A., Cremers, D., and Bischof, H. (2009). Anisotropic Huber-L1 Optical Flow. *British Machine Vision Conference*, 1.

[97] Xu, L., Jia, J., and Matsushita, Y. (2012). Motion detail preservingoptical flow estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[98] Xue, T., Chen, B., Wu, J., Wei, D., and Freeman, W. (2019). Video enhancement with task-oriented flow. *CVPR*.

[99] Yu, F., Zhang, Y., Song, S., Seff, A., and Xiao, J. (2015). LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop. *arXiv*.

[100] Yu, Z., Yoon, J. S., Venkatesh, P., Park, J., Yu, J., and Park, H. S. (2018). HUMBI 1.0: HUman Multiview Behavioral Imaging Dataset. *CVPR*.

[101] Zanfir, A., Marinoiu, E., and Sminchisescu, C. (2018). Monocular 3d pose and shape estimation of multiple people in natural scenes the importance of multiple scene constraints. *CVPR*.

[102] Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks. *CVPR*.

[103] Zhang, C. and Chen, T. (2004). A survey on image-based rendering - representation, sampling and compression. *Signal Processing: Image Communication*, 19:1–28.

[104] Zhang, J., Yuan, C., Huang, G., Zhao, Y., Ren, W., Cao, Q., Li, J., and Jin, M. (2018a). Acquisition of a full-resolution image and aliasing reduction for a spatially modulated imaging polarimeter with two snapshots. *Applied optics*, 57(10):2376–2382.

[105] Zhang, Z., Song, L., Xie, R., and Chen, L. (2018b). Video frame interpolation using recurrent convolutional layers. In *IEEE Fourth International Conference on Multimedia Big Data (BigMM)*, pages 1–6.

[106] Zhao, H., Gallo, O., Frosio, I., and Kautz, J. (2017). Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57.

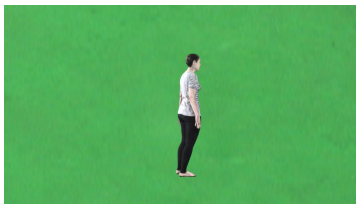
[107] Zheng, S., Zhang, J., Huang, K., He, R., and Tan, T. (2011). Robust view transformation model for gait recognition. *International Conference on Image Processing(ICIP)*.

Appendix A

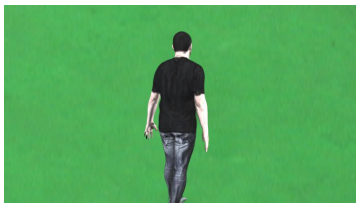
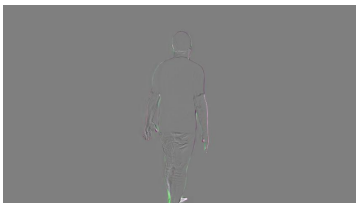
This appendix displays examples of interpolated images and their visual difference to the ground truth in order of **decreasing SSIM**.

Model: MV SSIM 51.

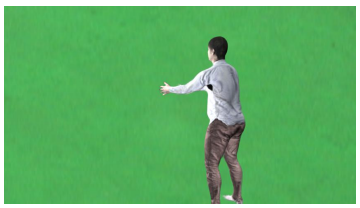
Dist cameras: 0.58m
Distance subject: 4.61m
Pixel distance: 11 pxls
SSIM: 0.9774
PSNR: 47.06



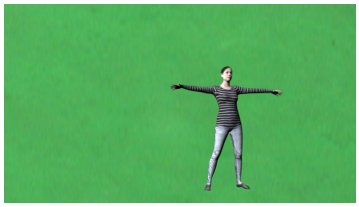
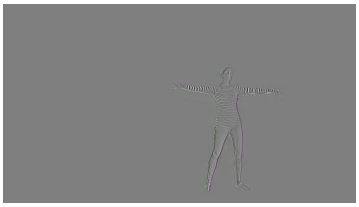

Dist cameras: 0.72m
Distance subject: 3.34m
Pixel distance: 13 pxls
SSIM: 0.9540
PSNR: 38.14

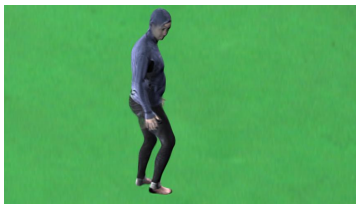
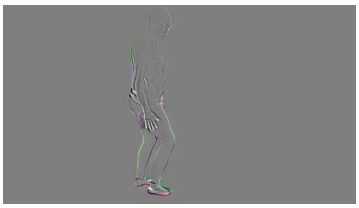
Dist cameras: 0.76m
Distance: 3.60 m
Pixel distance: 24 pxls
SSIM: 0.9041
PSNR: 37.88




Dist cameras: 0.64m
Distance: 5.05 m
Pixel distance: 19 pxls
SSIM: 0.8534
PSNR: 36.29

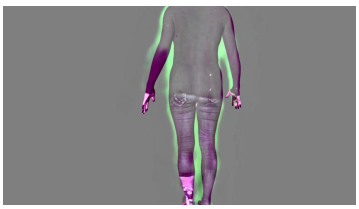



Dist cameras: 0.76m
Distance: 3.21 m
Pixel distance: 23 pxls
SSIM: 0.7591
PSNR: 32.97

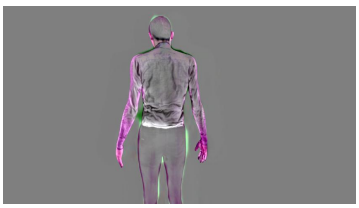



Dist cameras: 0.58m
Distance: 2.96m
Pixel distance: 13 pxls
SSIM: 0.6795
PSNR: 30.35


Dist cameras: 0.94m
Distance: 2.19m
Pixel distance: 108 pxls
SSIM: 0.6011
PSNR: 23.53

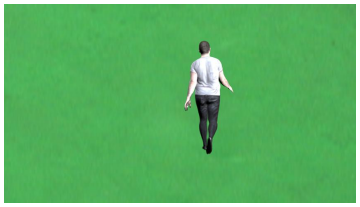
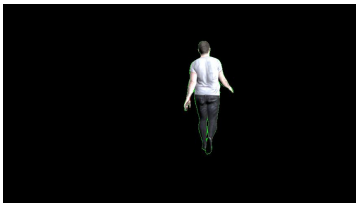
Dist cameras: 0.96m
Distance: 2.35m
Pixel distance: 87 pxls
SSIM: 0.5320
PSNR: 24.60

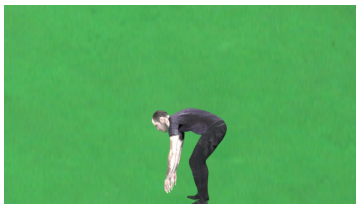
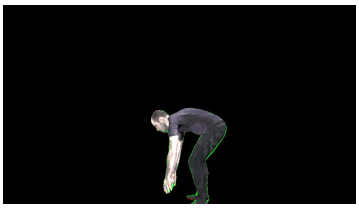
Appendix B

This appendix displays examples of interpolated images and ground truth with highlighted "false negative" (FN) - *red*- and "false positive" (FP) - *green* - pixels in order of **increasing False Negative ratio**. FN pixels denote pixels that are present in the ground truth, but not in the interpolated image. FP pixels denote pixels present in the interpolated image, but not in the ground truth. Model: MV SSIM 51.

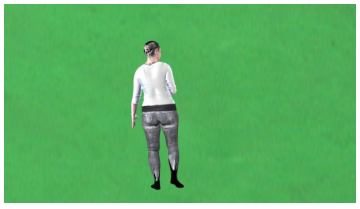
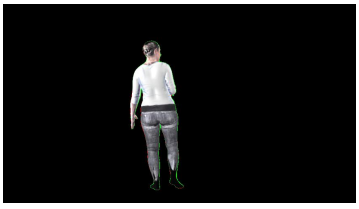
Dist cameras: 0.26m
Distance subject: 4.34m
Pixel distance: 15 pxls
FN ratio: 0.0014
FP ratio: 0.0168

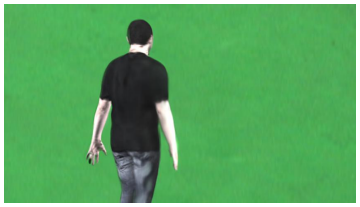
Dist cameras: 0.64m
Distance subject: 3.84m
Pixel distance: 21 pxls
FN ratio: 0.0032
FP ratio: 0.0249

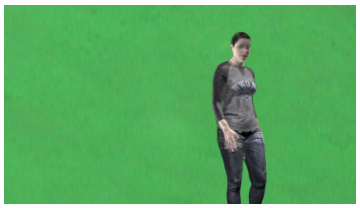
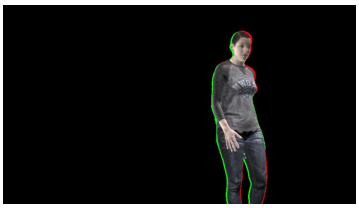
Dist cameras: 0.80m
Distance subject: 3.85m
Pixel distance: 15 pxls
FN ratio: 0.0062
FP ratio: 0.0163

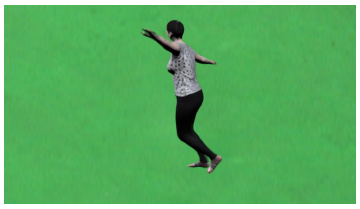
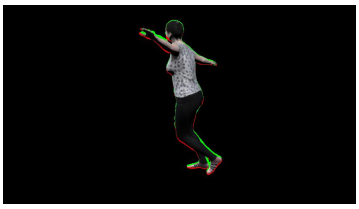
Dist cameras: 0.72m
Distance subject: 2.59m
Pixel distance: 38 pxls
FN ratio: 0.0121
FP ratio: 0.0344



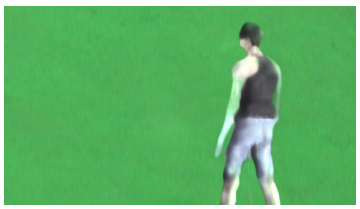

Dist cameras: 0.76m
Distance subject: 2.93m
Pixel distance: 28 pxls
FN ratio: 0.0243
FP ratio: 0.0392

Dist cameras: 0.92m
Distance subject: 3.30m
Pixel distance: 32 pxls
FN ratio: 0.0361
FP ratio: 0.0466

Dist cameras: 0.96m
Distance subject: 2.71m
Pixel distance: 73 pxls
FN ratio: 0.0498
FP ratio: 0.0682




Dist cameras: 0.94m
Distance subject: 2.71m
Pixel distance: 58 pxls
FN ratio: 0.0742
FP ratio: 0.0515