



**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

July 26, 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

July 26, 2019

# Acknowledgments

...ACKNOWLEDGMENTS...

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

...ABSTRACT...

# Summary

...SUMMARY...

# Contents

Acknowledgments	iii
Abstract	iv
Summary	v
List of Tables	ix
List of Figures	x
Chapter 1 Introduction	1
1.1 Motivation . . . . .	1
1.2 Objectives . . . . .	3
1.3 Plan of the Dissertation . . . . .	4
 Chapter 2 State of the Art	5
2.1 3D reconstruction and Photogrammetry . . . . .	6
2.1.1 Orthographic projection . . . . .	6
2.1.2 Perspective projection - epipolar geometry . . . . .	6
2.1.3 Structure from Motion . . . . .	9
2.1.4 Multi-view stereo . . . . .	11
2.2 View interpolation . . . . .	12
2.2.1 Single image interpolation techniques . . . . .	13
2.3 Deep learning . . . . .	18
2.3.1 Convolutional Neural Networks (CNNs) . . . . .	18
2.3.2 Capsule Networks . . . . .	21

2.3.3	Recurrent Neural Networks . . . . .	23
2.3.4	Autoencoder . . . . .	24
2.3.5	Generative Adversarial Networks . . . . .	25
2.4	Neural networks in view interpolation . . . . .	26
2.4.1	Deep-learned optic flow . . . . .	26
2.4.2	View interpolation based on deep-learned phase . . . . .	27
2.4.3	Pixel-by-pixel view interpolation . . . . .	27
2.4.4	View interpolation based on GANs . . . . .	32
2.4.5	Combination approaches - based on RNNs and pose estimation . . . . .	33
<b>Chapter 3</b>	<b>Methodology</b>	<b>34</b>
3.1	Neural Network Description . . . . .	34
3.1.1	Network Architecture . . . . .	34
3.1.2	Network Loss . . . . .	36
3.1.3	Hyperparameters . . . . .	39
3.2	Multi-view camera dataset . . . . .	42
3.2.1	Real-life datasets . . . . .	42
3.2.2	Synthetic dataset . . . . .	43
3.3	Evaluation . . . . .	47
3.3.1	MSE and PSNR . . . . .	47
3.3.2	SSIM . . . . .	48
3.3.3	False Negative and False Positive Silhouette Pixels . . . . .	49
3.3.4	Hausdorff Distance . . . . .	50
<b>Chapter 4</b>	<b>Results and Discussion</b>	<b>54</b>
4.1	SSIM . . . . .	54
4.2	PSNR . . . . .	54
4.3	Silhouettes - False Negative Ratio and False Positive Ratio . . . . .	59
4.4	Hausdorff distance . . . . .	59
4.5	Discussion . . . . .	61
<b>Chapter 5</b>	<b>Conclusion and Future Work</b>	<b>62</b>
5.0.1	Conclusion . . . . .	62
5.0.2	Future Work . . . . .	62



Bibliography	63
Appendices	71

# List of Tables

3.1	Online multi-view datasets . . . . .	43
-----	--------------------------------------	----

# List of Figures

2.1	Examples of Perspective, Affine and Orthographic projections. From [1], chapter 9. . . . .	7
2.2	IBR techniques classification [2] . . . . .	13
2.3	Comparing linear interpolation results with non-linear interpolation using Radon-CDT space. Linearly interpolated images in left example top row and in right example bottom-right [3]. . . . .	17
2.4	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 $x_3$ , image on the right show the receptive field of a single output pixel $s_3$ [4].	19
2.5	Typical layer of a convolutional neural network [2] . . . . .	20
2.6	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) . . . . .	21
2.7	Both the image on the right and the image on the left are classified as a “face” by CNN . . . . .	22
2.8	Architecture of LSTM recurrent network “cell” [4]. . . . .	23
2.9	Examples of images generated by GAN. Rightmost column shows examples of the original images [5]. . . . .	25
2.10	Images of bedrooms generated by DCGAN [4]. . . . .	26
2.11	Illustration of adaptable convolution method. The neural network receives 2 receptive field patches - $R_1$ and $R_2$ . The NN estimates convolution kernel $K$ for the selected pixel, this kernel is used to convolve with patches $P_1$ and $P_2$ to synthesise the output pixel. Image from [6]. . . .	28

2.12	Kernels estimated by CNN for pixels on the horizontal edge (a), diagonal edge (b) and in a texture-less area (c). The latter has isotropic kernel. From [6]. . . . .	29
2.13	Grid Net architecture from [7]. 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). . . . .	31
3.1	Overview of Neural Network architecture. Image from [8] . . . . .	36
3.2	ReLU . . . . .	40
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 [9]. . . . .	40
3.4	Illustration of the algorithm working in Blender Python <b>bpy</b> module. Over 800 different random scenes were generated to produce 8506 triplets of images for training the neural network and further 1322 triplets for testing. . . . .	44
3.5	Illustration of the camera setup in a Blender file. Green arrow show the person's range of movement. Blue circle - the positions possible for the ground truth camera. The left and right camera are always on a tangent to the ground truth camera focus line. . . . .	45
3.6	Illustration of the cameras pitch - possible improvement to the dataset.	45
3.7	Illustration of the cameras off the tangent line - possible improvement to the dataset. . . . .	46
3.8	Examples of images generated. . . . .	47
3.9	Example of False Negative (FN) and False Positive (FP) pixels (a). Ground truth only shown (b, c) as visually the interpolated image and its silhouette are very similar to ground truth - both silhouettes are required to produce (a). . . . .	50
3.10	Illustration of the camera setup for SfS reconstruction. 12 cameras are real (on the circle), 12 are "synthetic" (on the green segments). . . . .	51
4.1	TODO . . . . .	55
4.2	TODO . . . . .	56

4.3	TODO . . . . .	57
4.4	TODO . . . . .	58
4.5	TODO . . . . .	59
4.6	TODO . . . . .	60
4.7	TODO . . . . .	60
4.8	TODO . . . . .	60
4.9	TODO . . . . .	61
4.10	TODO . . . . .	61

# List of Abbreviations

AR - Augmented reality

CG - Computer Graphics

CNN - Convolutional Neural Network

Depth estimation - set of techniques and algorithms aiming to obtain a representation of the spatial structure of a scene.

Disparity map - refers to the apparent pixel difference or motion between a pair of stereo image.

FVT - Free-viewpoint television. System for viewing natural video, allowing the user to interactively control the viewpoint and generate new views of a dynamic scene from any 3D position

FVV - Free-viewpoint video. Same as FVT.

HSV - Human Visual System

IBMR - image-based rendering

IBMR - image-based modeling and rendering. Methods rely primarily on the original or trained set of images to produce new, virtual views, rather than 3D model.

MS-SfS - Multi-source Shape-from-Silhouette

MVS - Multi-view stereo. Aims to reconstruct disparity maps from a collection of images with known camera poses and calibration, possibly estimated using Structure from Motion (SfM) algorithms. Optical flow - independent flow estimation for each pixel.

Photogrammetry - science of making measurements from photographs, especially for recovering the exact positions of surface points.

SfM - Structure from Motion. Photogrammetric range imaging technique for estimating 3D structure from 2D image sequences that may be coupled with local motion signals.

SfS - Shape-from-Silhouette. Shape reconstruction method which constructs a 3D shape estimate of an object using silhouette images of the object (for example, where silhouettes are obtained by object segmentation).

View Synthesis - aims to create new views of a specific subject starting from a number of pictures taken from given point of views.

VR - Virtual Reality

# Chapter 1

## Introduction

This dissertation explores the possibility to generate novel points of view - synthetic cameras - from an input of images from spatially distributed cameras. ~~It aims to create an image interpolated spatially.~~ The main focus of the research is on videos of human motion obtained from multi-view camera setup. The primary focus is on ~~geometric correctness~~ of the generated images. The synthetic image 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 Free-Viewpoint Video (FVV) are on the rise [TODO:citation] and need accurate 3D models. Until recently, most content available was synthetic, created by artists and designers, but there is a growing demand for ~~real-live models~~ [10].

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 the additional views are produces with interpolation techniques [TODO:citation] to allow an arbitrary point of view.~~ The techniques described in this dissertation can improve the quality for these synthesised images.

Free Viewpoint Video (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 the 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 ~~where a lot of cameras is 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, in real-life scenarios with imperfect lighting.~~ Recently, some researches attempt reconstruction in difficult scenarios from as little as 8 mobile phone cameras ~~see [11], 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 [12], inaccuracies from lack of reference points in sparse camera setup, lack of texture, transparent or reflective features, due to camera lens, noise, camera angle [13]. SfS reconstructions generally suffer from producing a mesh that is too volumous.~~

~~Our approach suggests 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 [10].

Neural networks (NNs) have revolutionised image processing in the recent years (as recent as 2014). They have been shown to have better performance at computer vision tasks than previously designed procedural approaches [14]. Deep neural ~~network~~ ~~is~~ able to extract and combine tens of thousands of features from images, where a human approach ~~would~~ only find dozens. ~~The~~ deep-learning networks have been used for classification, segmentation and creation of new images/ video. IBR was previously combined with deep-learning to create arbitrary points of view when given a collection of images [15]. [7] 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 - and existing NN implementation was chosen - this was designed by [7]. 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 [16]. The NN needed to be adapted for the different task of multi-camera spatial images the difference being that in multi-view camera scenario the images are generally much further apart than the frames in a video.

## 1.3 ~~Plan 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 - Structure-from-Motion, Multi-view stereo...and (TODO: 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 - it mostly looks at view-interpolation research that uses deep-learning, but also checks if deep-learning approach was previously applied to spatial images.

**Chapter 3 Methodology** examines the methodology in detail. It covers 3 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 ( Shape-from-Silhouette) 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 - Structure-from-Motion (SfM), Multiview stereo (MVS) and Shape-from-Silhouette(SfS).

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

The third section covers explores neural network architectures for the purposes of being used for this research. It looks at CNNs, CapsuleNets, Autoencoders, RNNs and GANs. Autoencoders are included as they form the basis of the generative networks, as 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 ~~looks at:~~

1. 3D Reconstruction, in particular the algorithms of structure-from-motion (SfM) and multi-view synthesis (MVS)
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. This section will cover the basics as well as SfM and MVS. It will also justify the choices for the particular 3D reconstruction software used for the evaluation of this project.

### 2.1.1 Orthographic projection

As early as 1992 [17] obtained good results from a stream of images using orthographic, rather than perspective, projection and introduced *Factorization method*. The orthographic projection simplified processing, removing the depth dimension<sup>1</sup>. Examples of projective, affine and Euclidean projections are given in Fig. 2.1. [17] 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.

$$W = RS + te_P^T$$

They 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.

### 2.1.2 Perspective projection - epipolar geometry

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

---

<sup>1</sup>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 [1], chapter 9)

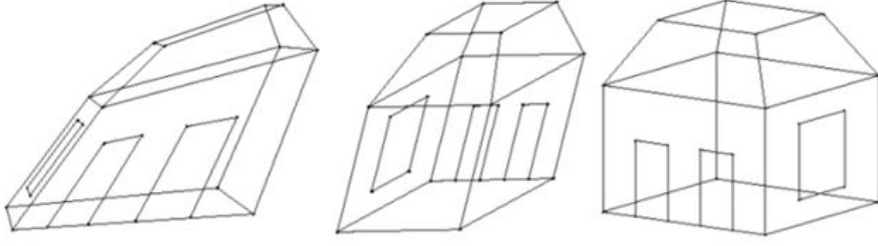


Figure 2.1: Examples of Perspective, Affine and Orthographic projections. From [1], chapter 9.

*Calibrated cameras:*

First, a special case of calibrated cameras is described.

The *essential matrix*  $E$  for correspondence between 2 images was introduced by [18]:

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

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*. The essential matrix  $E$  maps a point  $\hat{x}_0$  in the first image into a line in the second image -  $l_1 = E\hat{x}_0$ , which is called *epipole* [19].

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 Matrix 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 ([20] ; [21]; [22]) that *7 point correspondences* (i.e. features) is sufficient to find the elements of the essential matrix [19].

In addition, [20] suggests that the point co-ordinates should 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 ( $\sum_i \tilde{x}_i = \sum_i \tilde{y}_i = 0$ ,  $\sum_i \tilde{x}_i^2 + \sum_i \tilde{y}_i^2 = 2n$ )

### *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* - [22], [23].

### *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*[22], 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 [24].

### *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 ~~basic stones~~ of 3D reconstruction is - triangulation. This is a method to estimate depth to the object after the Fundamental matrix is known. ~~[describe method?]~~

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 2 images.

## **2.1.3 Structure from Motion**

The following two sections describe SfM (Structure from Motion) and MVS (Multi-view stereo) algorithms. These correspond to obtaining sparse 3D point cloud reconstruction and camera positions (SfM) and dense 3D surface reconstruction (MVS). The project will use COLMAP for 3D reconstruction, so particular implementation specific to COLMAP is specified.

The typical workflow of a SfM algorithm is as follows [25]:

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) [19], which finds if the point is unique in its surroundings. The following authors further expanded on ACF - [26], [27], [28] etc.

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

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



- SIFT [29]
- SURF [30]
- BRIEF [31]
- ASIFT [32]
- LDAHash [33]

COLMAP uses *RootSIFT* for its feature extraction, which is an improved version of SIFT algorithm. For this work, which intends to produce additional images based on deep-learning, it is possible to have blurriness in generated images and ghost artefacts, which may make feature detection more difficult or place features incorrectly.

## 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*). This approach has computational complexity  $O(N^2_{IMAGES}N^2_{FEATURES})$  and is prohibitive for large image collections [25]. There is research to improve the efficiency of the matching, for example, k-dimensional trees and ANN (Approximate Nearest Neighbour) [34].

## 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 [25].

## 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 [34].

Also, 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 involves estimating the pose of the camera for the new image and, for uncalibrated cameras, camera's intrinsic parameters. Every new image provides additional 2D-3D correspondences. [25].

### 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. [22] 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. COLMAP developers [25] propose their own version of the triangulation method.

### 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 -  $X$ . COLMAP uses the following formulae, where  $E$  - reprojection error,  $x_j$  - co-ordinates of the point in image  $j$ ,  $\rho_j$  - loss function to down-weight the outliers and  $\pi$  symbolises the function that converts scene points into image space [25].

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

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 [34] ) .

## 2.1.4 Multi-view stereo

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. [34] summarises the review of MVS methods by As detailed by [25] with

reference to [35], 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, [36]).
2. Surface evolution-based methods that use iteratively evolved polygonal meshes (for example, [37]).
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, [38] )
4. Patch-based methods where collections of small patches or surfels represent the scene (for example, [39]).

The latter method is also called PMVS and is the one utilised by COLMAP [25]. Schonberger suggests a new PMVS algorithm based on [40].

The last 2 steps in Multi-view stereo is to generate a polygonal 3D mesh from dense point cloud. This can be achieved with *Poisson Surface Reconstruction* (PSR) [41]. Texture is then added with *Texture mapping*, which is another field of research in itself.

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 reconstruction, we have therefore specified the particular algorithms applicable to COLMAP, which will be used for 3D testing.

## 2.2 View interpolation

[2] proposes the following classification of Image-based rendering (IBR) techniques - Fig. 2.2. *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.

Motion interpolation and view synthesis are popular terms more recently that are also of interest.

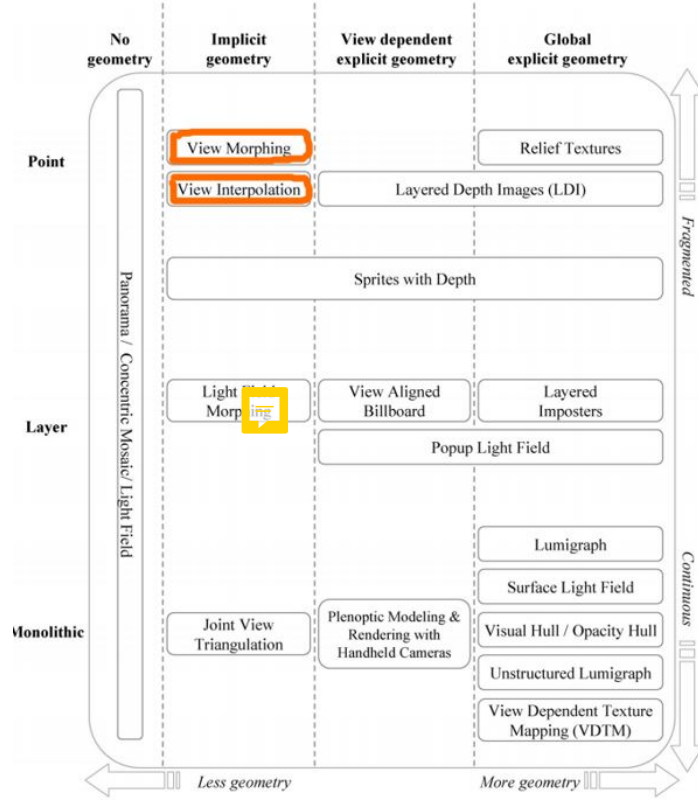


Figure 2.2: IBR techniques classification [2]

### 2.2.1 Single image interpolation techniques

This dissertation is primary concerned with view interpolation between 2 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 the single image interpolation techniques - as they can also be applied to the result of the 2 frame interpolation. The interpolation techniques for a single image include (Wikipedia):

- 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:

$$\alpha I_1 + (1 - \alpha) I_0$$

Where  $I_0$  and  $I_1$  are pixel values in image 1 and 2. This method produces blurry result where both of the original images can still be distinguished.

## Spatial interpolation

### *Feature based image morphing*

Early *spatial interpolation* was introduced by [42]. They worked with computer graphics 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.

[36] 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 reprojection was 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 [42] and [36] 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). 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,  $\rho_r$  and  $\rho_t$  - 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 due to magnification and disocclusion. [2]

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:

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

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. [2]

More recent references on view synthesis are able to work with un-calibrated cameras. [43] expands the work of [35], work with uncalibrated cameras and proposes a new algorithm based on interpolating homographies rather than pixel positions and colours. [44] combines spatial interpolation based on feature matching with temporal interpolation based on optical flow (see below). They assert that their approach is suited even for wide baseline setups, where dense stereo matching approaches reach their limits.

### **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 [45] the general problem of image morphing techniques is that the warp-

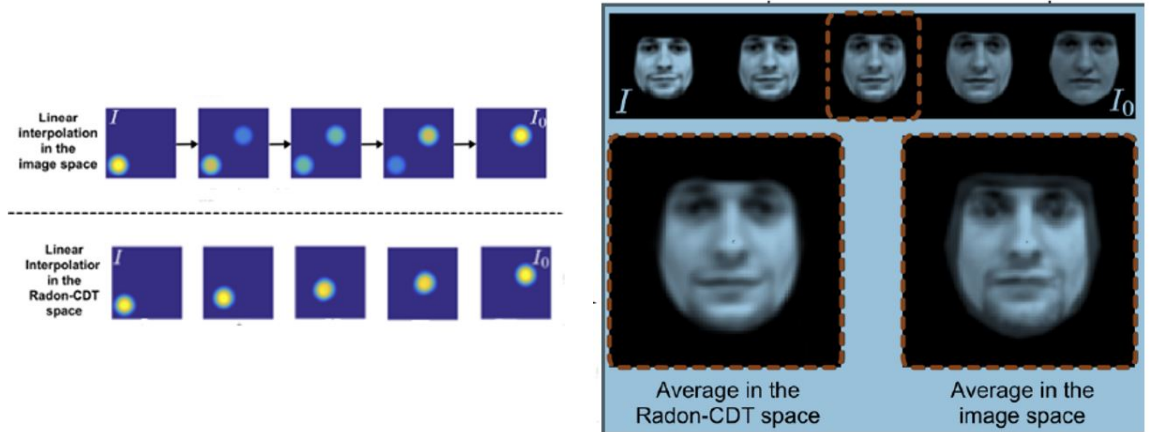


Figure 2.3: Comparing linear interpolation results with non-linear interpolation using Radon-CDT space. Linearly interpolated images in left example top row and in right example bottom-right [3].

ing 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 - [46], [47], [45] etc. [45] computes the optical flow between 2 interpolated images - this is a bi-directional process as both forward and backward flows are computed. He uses [48] 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* - is partially based on the above spatial techniques and homographies: [49],[50].

As an alternative approach, a method based on *Fourier transform* is recently used by [3]. 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. 2.3 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 [51]. 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 Deep learning

### 2.3.1 Convolutional Neural Networks (CNNs)

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

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

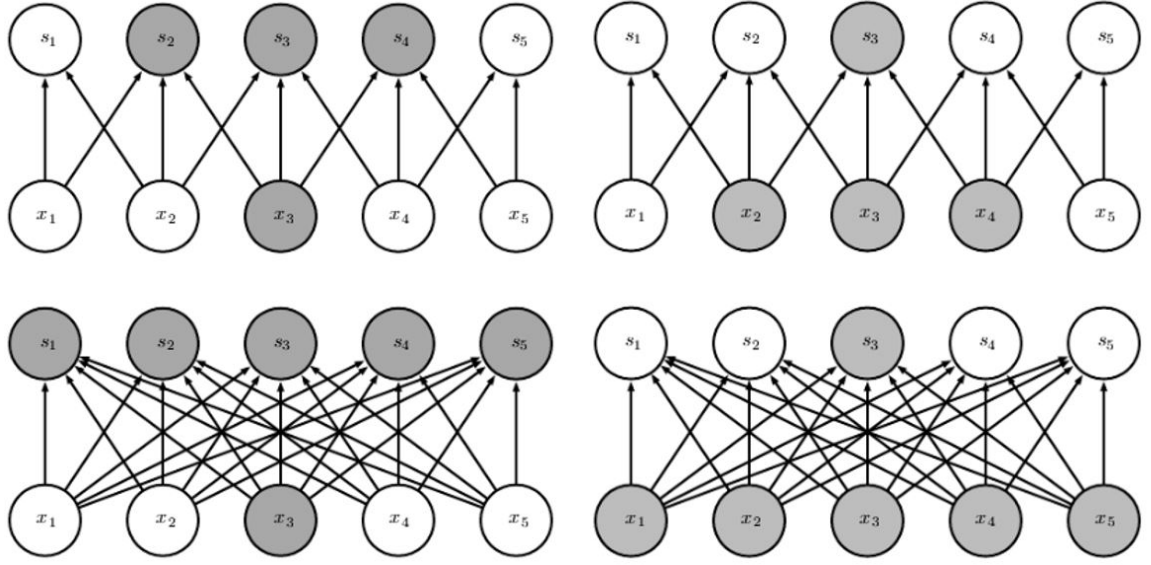


Figure 2.4: Comparison of connectivity in a traditional NN (bottom row) and CNN (top row). Image on the left shows the effect of a single input pixel  $x_3$ , image on the right show the receptive field of a single output pixel  $s_3$  [4].

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 if input changes, the output changes in the same way, so convolution can create a 2D map of where certain features appear in the input (based on [4]).

*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.

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

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

Or sometime 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. 2.4 - input  $x_3$  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  $s_3$  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 operation (*max pooling*) - usually neighbourhoods are size  $2 \times 2$  or  $4 \times 4$ . Or average of a rectangular neighbourhood (*average pooling*),  $L^2$ -norm of the rectangular neighbourhood or a weighted average based on a distance from the central pixel. Pooling operation removes small changes, makes features invariant to small translations, or can introduce an arbitrary invariance to other transformations depending on the parameters [4].

So, a typical *layer of convolutional neural network* comprises of the following - see Fig. 2.5:

1. Convolution

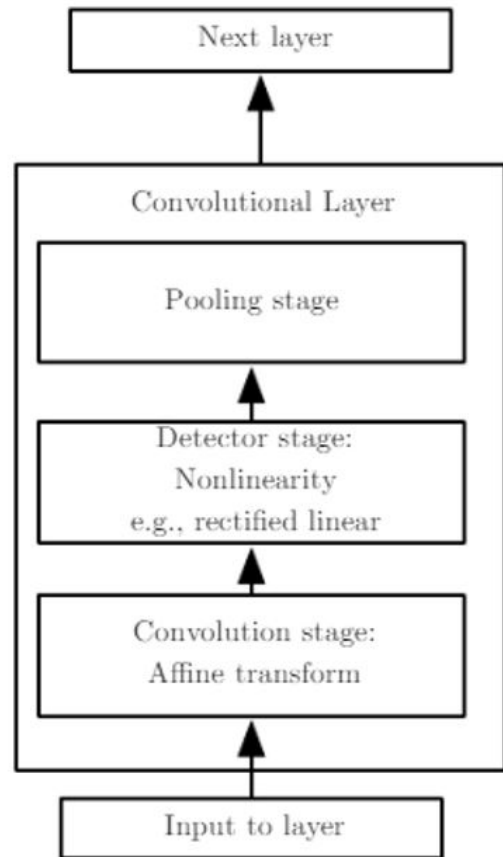


Figure 2.5: Typical layer of a convolutional neural network [2]

2. Activation function (ReLU for example)

3. Pooling

The convolutional neural network 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 [52].

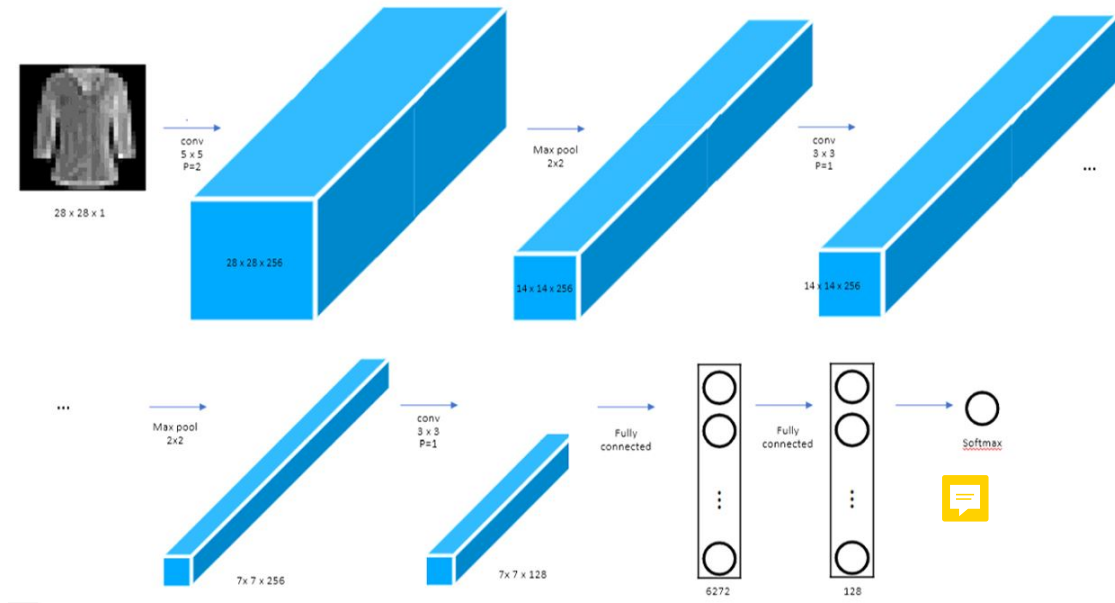


Figure 2.6: Example CNN architecture with 3 convolutional layers. Size of the square shows the size of input images, length of the cuboid - number of hidden layers (source: author)

CNNs are used in computer vision extensively mainly for *image classification and object recognition*, but they can also be used for *segmentation* (no pooling is employed for segmentation). Their application to video interpolation is discussed in the next section.

### 2.3.2 Capsule Networks

There are certain drawbacks of CNNs - spatial relationships between components are not very important to a CNN. Fig. 2.7 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 [53].

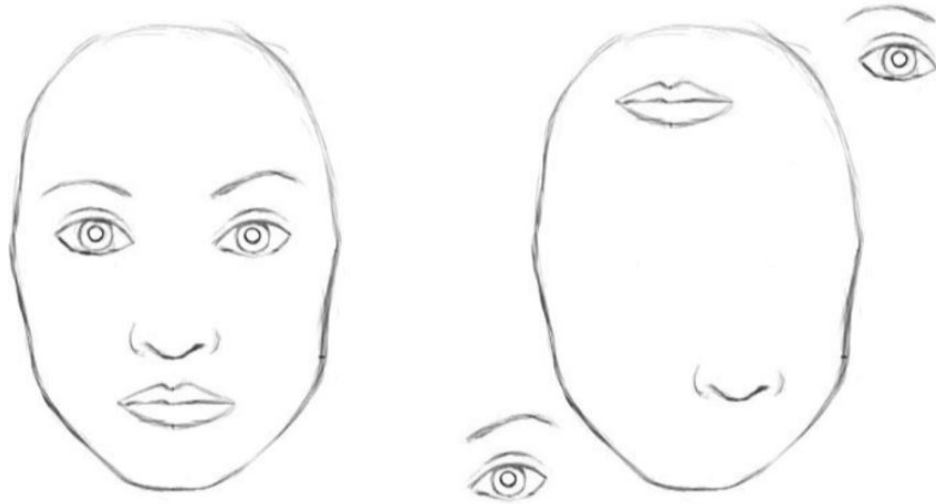


Figure 2.7: Both the image on the right and the image on the left are classified as a “face” by CNN

[54] developed CapsuleNets to counteract this problem. CapsuleNets use iterative process 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. This allows knowledge of familiar shapes to derive segmentation.

A capsule network consists of several layers of capsules. Each capsule has a  $4 \times 4$  pose matrix,  $M$ , and an activation probability,  $a$ . CapsuleNets converts the whole set of activation probabilities and poses of the capsules in one layer into the activation probabilities and poses of capsules in the next layer using Expectation-Maximisation procedure (EM) [54].

The author 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 condi-

tions. 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: ([55], [56]). 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

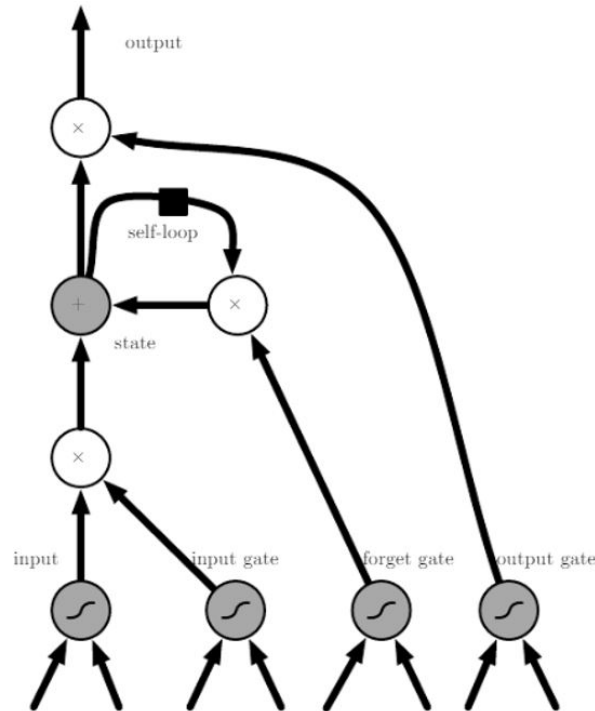


Figure 2.8: Architecture of LSTM recurrent network “cell” [4].

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  $\tau$  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) and Gated Recurrent Unit (GRU) - have been shown to be very effective at natural language processing, handwriting

recognition, speech recognition, image captioning and parsing [4].

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 [57]. 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) is one of the oldest types of neural networks that exists 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 [4].

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 [4].



Figure 2.9: Examples of images generated by GAN. Rightmost column shows examples of the original images [5].

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

Generative Adversarial Networks (GANs) have received recent attention. Pioneered in 2014 by [5], this network consists of two neural networks:

1. a generative model  $G$  that captures the data distribution, and
2. a discriminative model  $D$  that estimates the probability that a sample came from the training data rather than  $G$

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  $\theta_g$ . And  $D(x, \theta_d)$  is the second multilayer *discriminator* perceptron that outputs a scalar. The value function is then calculated as:

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

The work of [5] was further improved on by [58] who created a deep convolutional GAN (DCGAN). DCGAN performs very well in the image synthesis tasks - see examples in Fig. 2.10.





Figure 2.10: Images of bedrooms generated by DCGAN [4].

[59] 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.

### 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 [60] used convolutional neural network for flow estimation, FlowNet2.0 by [61] improved the quality of the estimation by stacking several CNNs. EpicFlow by [62], task-oriented flow by [63] and Spatial Pyramid Network SPyNet by [64] are other examples. [7] uses PWCNet optical flow neural network by [65] 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

[66] proposes a new convolutional neural network - 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 (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” - which is 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 subbands  $R_{\omega,\theta}(x,y)$  using steerable pyramid filters  $\psi_{\omega,\theta}$ . This forms the input to decoder-only network increasing resolution 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 [8], where it performs better only under certain scenarios, like changing light conditions.

### 2.4.3 Pixel-by-pixel view interpolation

[67] propose video synthesis approach based on deep voxel flow (DVF). They build up on previous paper on deep voxel flow by [68]. 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 predicts 3D voxel flow, and then a volume sampling layer synthesizes 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  $F_{mask}$  serves as sampling weights for trilinear interpolation. The authors use  $l_1$ -norm and  $l_2$ -norm losses for training, however they find that  $l_1$ -norm produces blurry results.  $l_1$  had limited ability to capture perceptual differences, such as high-frequency details. They then used perceptual loss that has been found effective in artistic style-transfer and image super-resolution - and found that this generates visually more appealing

results than pixel-wise distance metrics. 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 [69]. The result was compared among other research to that of early Niklaus [8], also [68] and PhaseNet [66]. 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 [6]
- Video Frame Interpolation via Adaptive Separable Convolution [8]
- Context-aware Synthesis for Video Frame Interpolation [7]

In their early work [6] 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 - local convolution over two input frames. The convolution kernel captures both the local motion between the input frames and the coefficients for pixel synthesis, so no separate optical flow estimation step is required. See Fig. 2.11 for explanation of the *adaptable convolution* method.

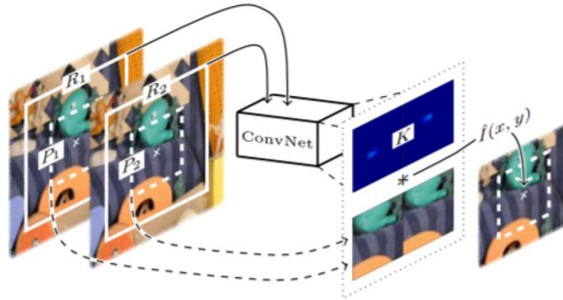


Figure 2.11: Illustration of adaptable convolution method. The neural network receives 2 receptive field patches -  $R_1$  and  $R_2$ . The NN estimates convolution kernel  $K$  for the selected pixel, this kernel is used to convolve with patches  $P_1$  and  $P_2$  to synthesise the output pixel. Image from [6].

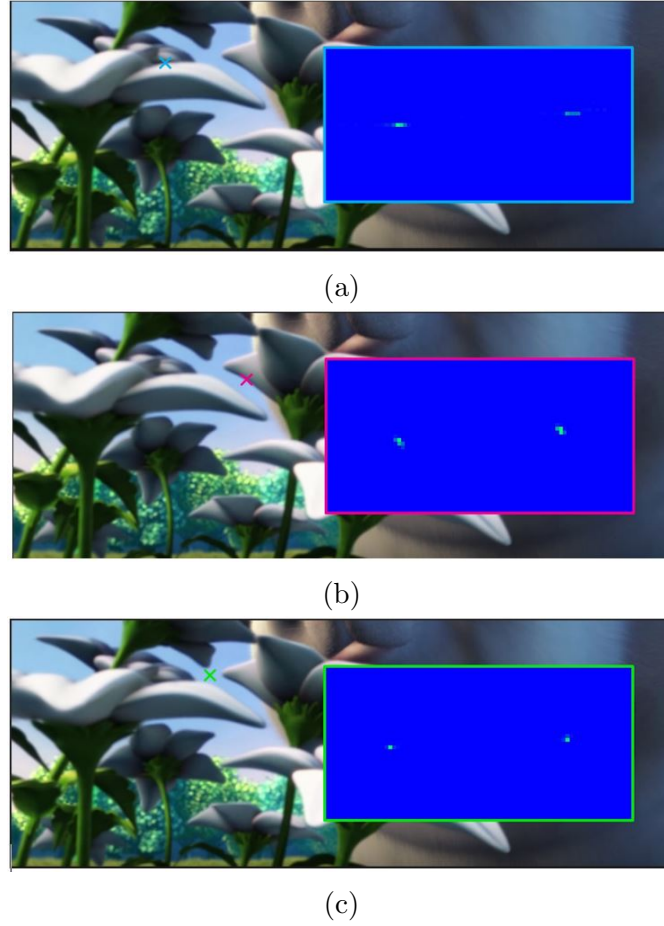


Figure 2.12: Kernels estimated by CNN for pixels on the horizontal edge (a), diagonal edge (b) and in a texture-less area (c). The latter has isotropic kernel. From [6].

Fig. 2.12 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 horizontal edge, diagonal edge and in a uniform area without texture. The authors assert that edge-aware convolution kernels lead to sharp results.

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 motion estimated in the images* of 38 pixels. Receptive field patches are larger than convolution patches to handle the aperture problem. 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.

[8] 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 and bilinear 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  $2n$ .

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:

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

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

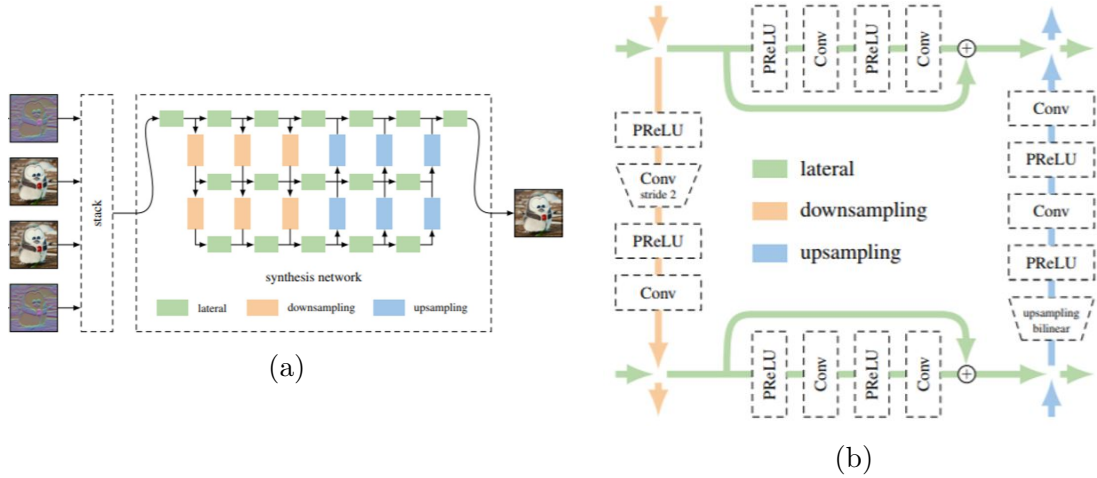


Figure 2.13: Grid Net architecture from [7]. Both pre-warped images and per-pixel contextual information images are fed to NN (a). GridNet encoder-decoder NN processes images at 3 different scales. Down-sampling, up-sampling and lateral building blocks are details in (b).

In their latest work [7] 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 [65] )
2. Extract per-pixel contextual information - using response of the conv1 layer from ResNet-18 [70]. The contextual vector describes pixel's  $7 \times 7$  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 *Grid-Net*. The NN processes images at 3 scales: per-row channel-sizes of 32, 64, and 96. See Fig. 2.13 for configuration of the NN.

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

qualitatively - *difference between Laplacian pyramids*:

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

The results of the video frame interpolation are compared to there two previous works (AdaConv, SepConv), optical flow approach [71], deep learning-based optical flow algorithm [69], phase-based approach [66].

#### 2.4.4 View interpolation based on GANs

Several researches also attempted to use GANs for video interpolation - [72], [73]. These can employ the above approaches to generate the interpolated images using CNN or RNN, but have an additional *discriminator* 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.

[72] used multi-scale convolutional neural network (CNN) and WGAN-GP (Wasserstein generative adversarial networks with gradient penalty) for video interpolation. Their GAN is based on a version of GAN proposed by [74] called Wasserstein GAN which was shown to be more stable than traditional GAN, the authors use DCGAN for comparison. [72] use *pyramid structure* for capturing the motion information of objects in images based on *feature pyramids* [75], as opposed to *image pyramids* [76]. 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 3 part loss 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 [73], [77] and [68].

[76] 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, [78] 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 configuration, choices for loss for training and hyperparameters - Multi-view cameras dataset: quick review of available real datasets and methodology deployed for the generation of synthetic multi-view dataset for training - 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 convolutional neural network (CNN). The configuration of the neural network is adapted from [16] 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  $3 \times 3$ , 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  $2 \times 2$ . 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 up-sampling 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 [8].

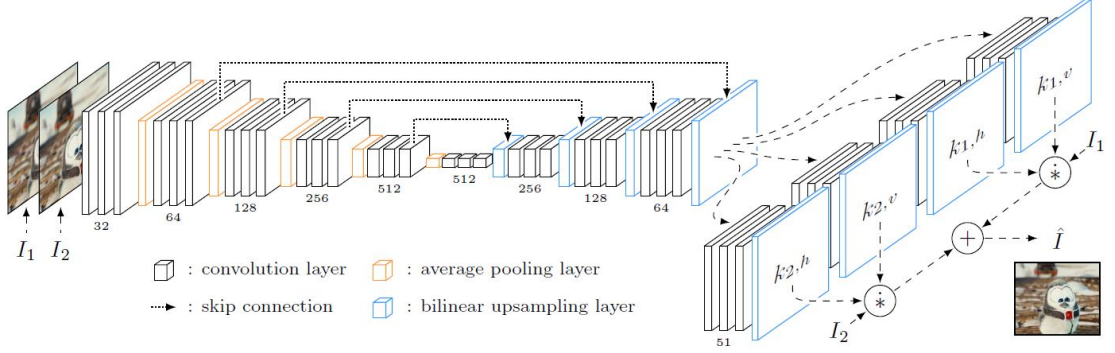


Figure 3.1: Overview of Neural Network architecture. Image from [8]

*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 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  $128 \times 128$ . The left and right images are combined into a single tensor with 6 channels of size  $128 \times 128$ . The neural network accepts batches of these tensors as input to produce the interpolated images in the batch. The ground truth 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. [79] have shown that with the same network architecture the quality of the results improves significantly with better loss functions, even when the network architecture is left unchanged.

[79] 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 penalizes larger errors, but is more tolerant to small error, regardless of the underlying structure of the image, while HVS is more

sensitive to luminance and color variations in texture-less regions. Similarly, [8] 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  $N$  is the number of pixels  $p$  in the patch (from [79])

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 predicted patch and the ground truth respectively.

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 (Wikipedia). SSIM is defined for a neighbourhood of pixel  $p$  as:

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

where  $l$  is luminance,  $c$  is contrast and  $s$  is structure at pixel  $p(x, y)$ . These values depend on a set of parameters -  $\mu_x$  and  $\mu_y$  are the average values of  $x$  and  $y$  respectively,  $\sigma_x^2$  and  $\sigma_y^2$  are the variances,  $\sigma_{xy}$  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  $L$  is the dynamic range of the pixel-values.  $k_1 = 0.01, k_2 = 0.03$ .

The loss function for SSIM is:

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

[79] suggest that different values of variance  $\sigma$  influence the visual qualities of the output patch - for example, smaller values of  $\sigma$  correctly represent edges (larger values create a halo of noise around the edge), but fails to preserve the local structure and introduces splotchy artifacts, while larger values of  $\sigma$  help reduce 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 [7] this loss measures the difference between Laplacian pyramids. 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 ground truth images respectively.

The authors in [7] suggest that Laplacian loss produces more pleasing visual results than  $L_1$  loss.

#### 6) Feature loss

Also suggested by [7], this loss is based on features  $\phi$  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 ([79], [16], [8]) 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 30 epochs.

#### *Implementation details for NN*

The neural network is implemented in Python and PyTorch. The training was performed on NVIDIA GEFORCE GTX 1050 GPU. Timings....

### **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 [16], 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, TanH, LeakyReLU, etc, while the other options have not been shown to have better performance.

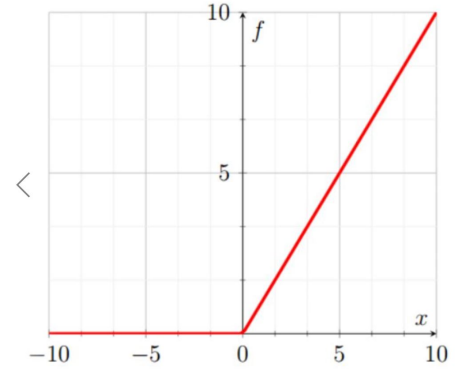


Figure 3.2: ReLU

## 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 [80]. 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 was designed to combine the advantages of two popular methods: AdaGrad [81], which works well with sparse gradients, and RMSProp [82], which works well in on-line and non-stationary settings [80].

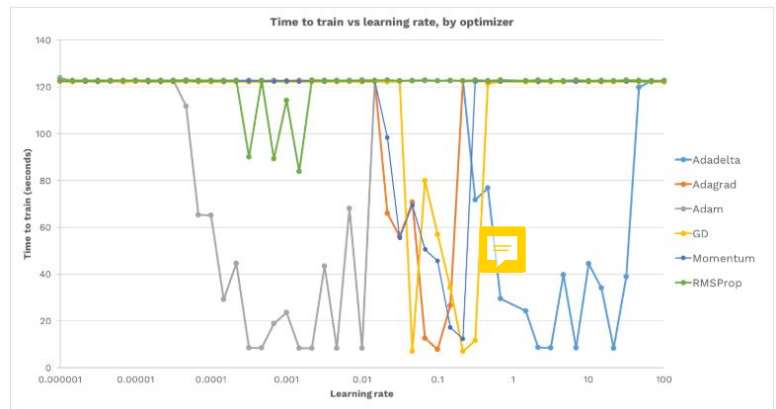


Figure 3.3: Performance of different optimizers depending on the learning rate (time of more than 120 seconds means the network failed to train. Image from [9].

The learning rate used in the training is related to the choice of the optimizer and is set as 0.001. For example, [9] 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.

### 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 ( $854\text{pixels} \times 480\text{ pixels} \times 3\text{ colors} \times 71\text{ kernel} \times 32\text{ batch size} = 2,794\text{Mb}$ ), 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. These 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. [8] used kernel size of 51-pixel, as their 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 kernel 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.

(TODO) Add table with memory requirements for each level.

*Other hyperparameters* included: number of hidden layers/ channels (left unchanged



from the original model) and weight initialization - gradients were initialised to zero. The models were trained to 30 epochs to determine the best model, than the best model was trained to (TODO)... epochs.

## 3.2 Multi-view camera dataset

The main contribution of this paper was to investigate multi-view camera datasets suitable for training the neural network or alternatively 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, HUMBI had 164 people, Casia Gait filmed 124 people.

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 ground truth. 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.

(TODO: add links to the datasets in references)

4 datasets had enough cameras to satisfy this requirement - OU-ISIR Gait, HUMBI, Dyna 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 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

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

Table 3.1: Online multi-view datasets

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 [83]. 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.

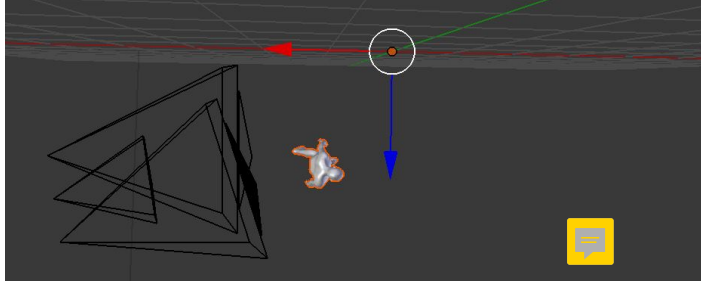


Figure 3.4: Illustration of the algorithm working in Blender Python `bpy` module. Over 800 different random scenes were generated to produce 8506 triplets of images for training the neural network and further 1322 triplets for testing.

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 [84]. 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.

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 (TODO: size ab 5K)
- 5) Generate random position for the ground truth 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. Steps 5-6 are described below in detail.

#### *Generating random positions for the 3 multi-view cameras*

The position of the *ground truth camera* has the following parameters - see Fig. 3.5 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

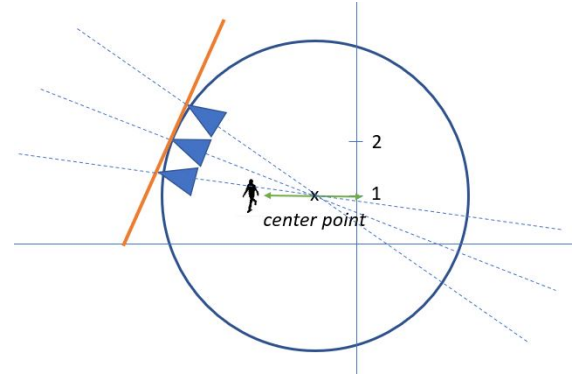


Figure 3.5: Illustration of the camera setup in a Blender file. Green arrow show the person's range of movement. Blue circle - the positions possible for the ground truth camera. The left and right camera are always on a tangent to the ground truth camera focus line.

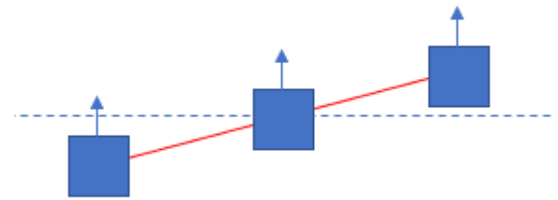


Figure 3.6: Illustration of the cameras pitch - possible improvement to the dataset.

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. 3.6 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 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

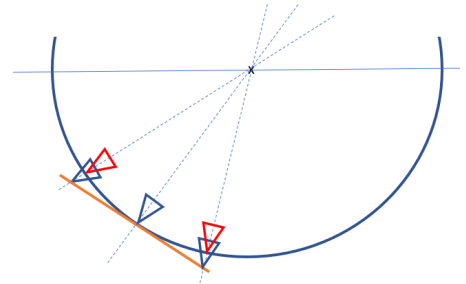


Figure 3.7: Illustration of the cameras off the tangent line - possible improvement to the dataset.



Figure 3.8: Examples of images generated.

be setup on a circle around the subject, so this case should be addressed. See Fig. 3.7 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 (TODO: ref) - however these were 2D backgrounds.

See Fig. 3.8 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

The simplest and most widely used full-reference quality metric is the mean squared error (MSE), computed by averaging the squared intensity differences of distorted and

reference image pixels, along with the related quantity of peak signal-to-noise ratio (PSNR) [85].

The Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR) are the two error metrics used to compare image compression quality. The MSE represents the cumulative squared error between the compressed and the original image, whereas PSNR represents a measure of the peak error. The lower the value of MSE, the lower the error [86].

To compute the PSNR, the block first calculates the mean-squared error using the following equation:

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

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

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

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.

PSNR is measured in decibels (dB) - the higher the PSNR, the better the quality of the compressed, or reconstructed image.

The calculations are implemented using PyTorch `torch.Tensor` to speed up the calculations. (TODO: ? Code samples)

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 [85] PSNR and MSE are not very well matched to 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(Wikipedia). 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 (Wikipedia).

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 color 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:

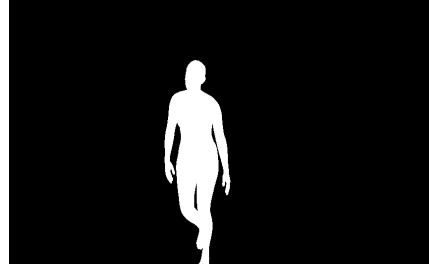
- Ratio of False Negative (FN) pixels to the number of pixels in the shape
- Ratio of False Positive (FP) pixels to the number of pixels in the shape

False negative pixels are counted where the ground truth image has silhouette, and the interpolated image does not. False positive pixels are the opposite - where the ground truth does not have the silhouette, but the interpolated image does. See Fig. 3.9 for demonstration of the concept - red pixels are False Negative, green pixels are False Positive.

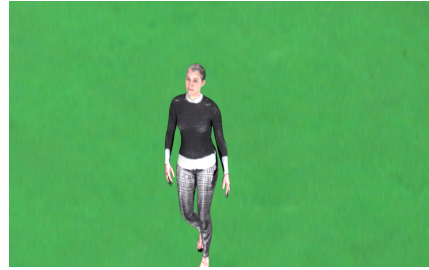




(a) FN (red) and FP (green) pixels



(b) Silhouette of the ground truth image



(c) Ground truth image

Figure 3.9: Example of False Negative (FN) and False Positive (FP) pixels (a). Ground truth only shown (b, c) as visually the interpolated image and its silhouette are very similar to ground truth - both silhouettes are required to produce (a).

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. 3.9 again - there will be missing voxels around the hands and leg, and a lot of voxels added around the shape (bad example of interpolation 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.

The evaluation steps were as follows:

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

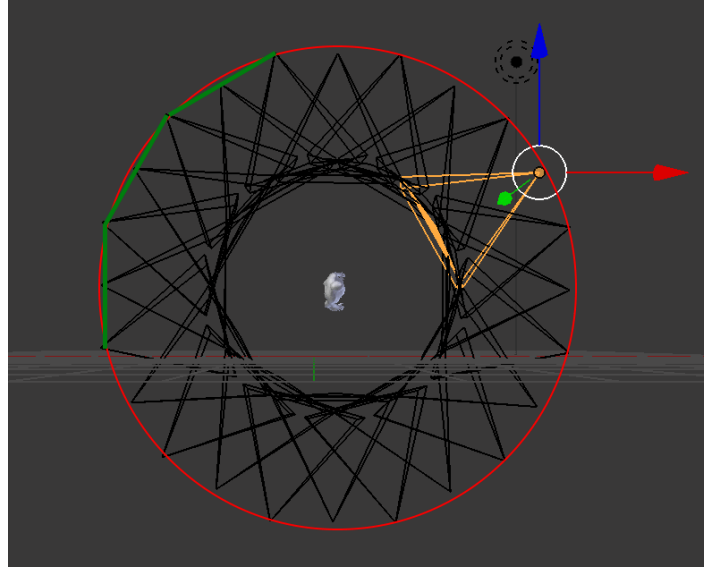


Figure 3.10: Illustration of the camera setup for SfS reconstruction. 12 cameras are real (on the circle), 12 are "synthetic" (on the green segments).

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 upwards from person's center.
- 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 ex-

ported 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.3.10 - 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<sup>rd</sup> party code - Vacancy - which is a voxel carving implementation in C++ [87]. 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 2 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 the ground

truth mesh from Blender) were imported into MeshLab. 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 the ground truth 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":

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

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 the ground truth 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

(TODO: Add discussion)

### 4.1 SSIM

Model with 51-px kernel L1-Loss trained for 50 epochs:

Fig. 4.1

(TODO: Text)

Model with 71-px kernel SSIM-loss trained for 10 epochs (+30 epochs L1 loss):

Fig. 4.2

(TODO: Text)

### 4.2 PSNR

Model with 51-px kernel L1-Loss trained for 50 epochs:

Fig. 4.3

(TODO: Text)

Model with 71-px kernel SSIM-loss trained for 10 epochs (+30 epochs L1 loss):

Fig. 4.4

(TODO: Text)

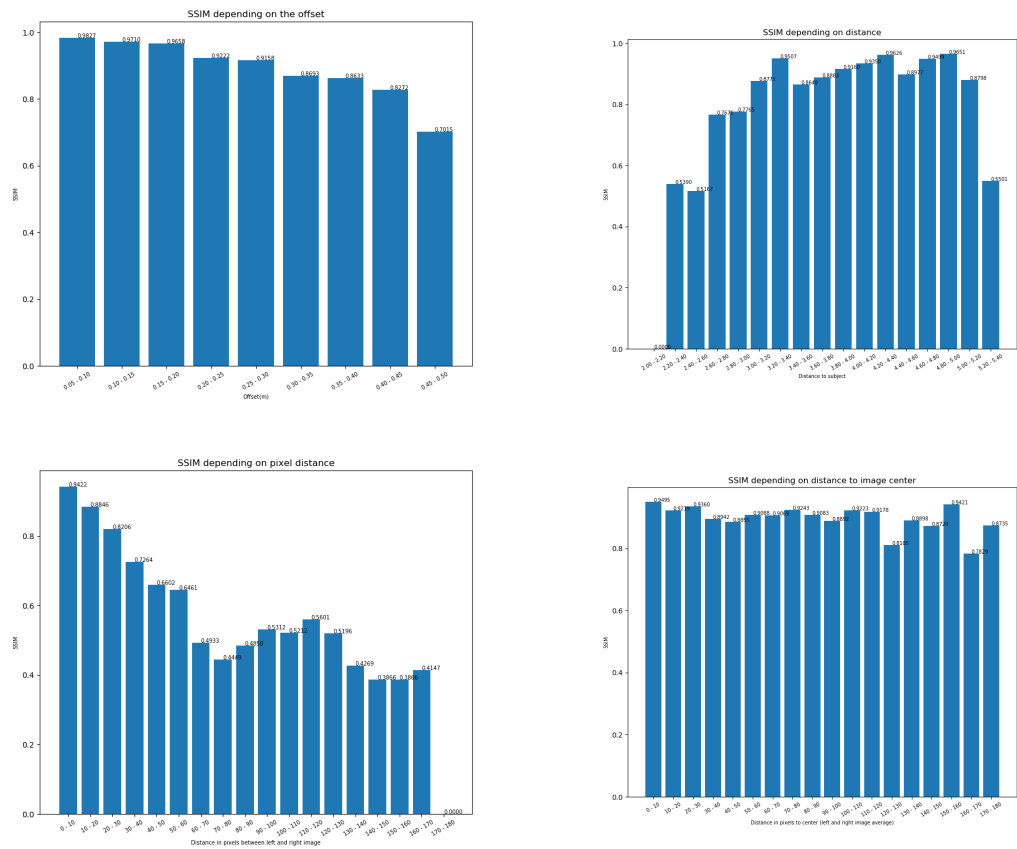


Figure 4.1: TODO

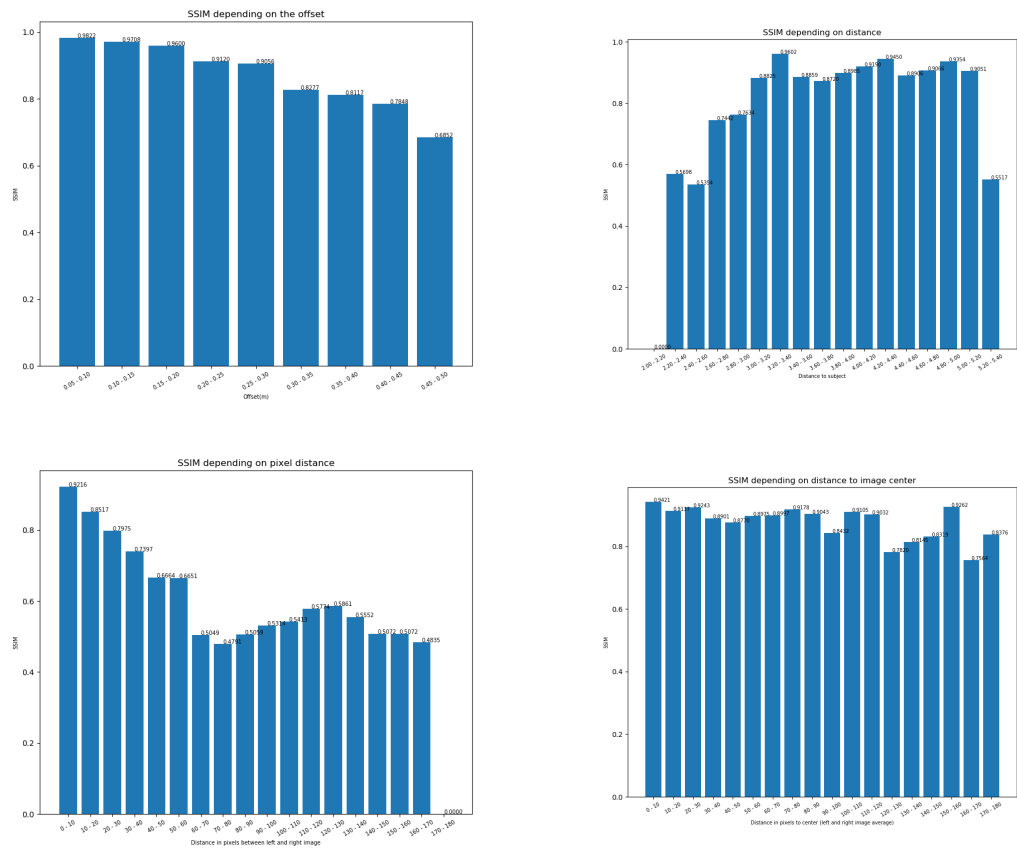


Figure 4.2: TODO

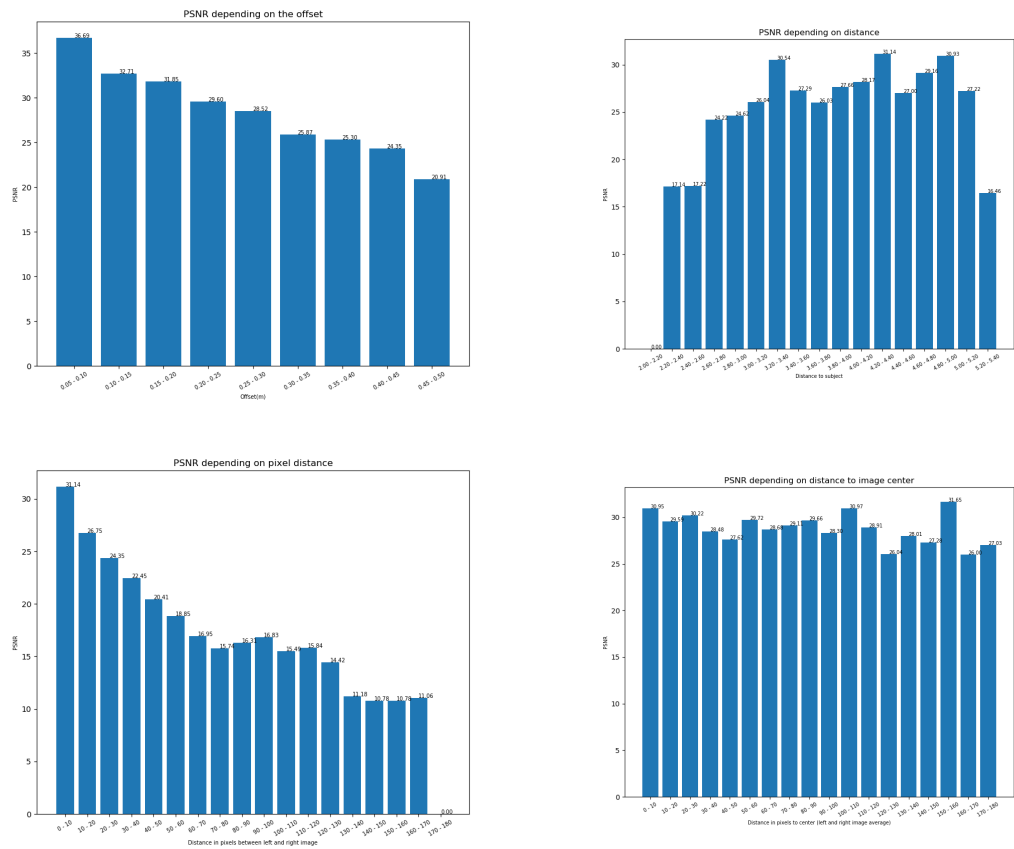


Figure 4.3: TODO



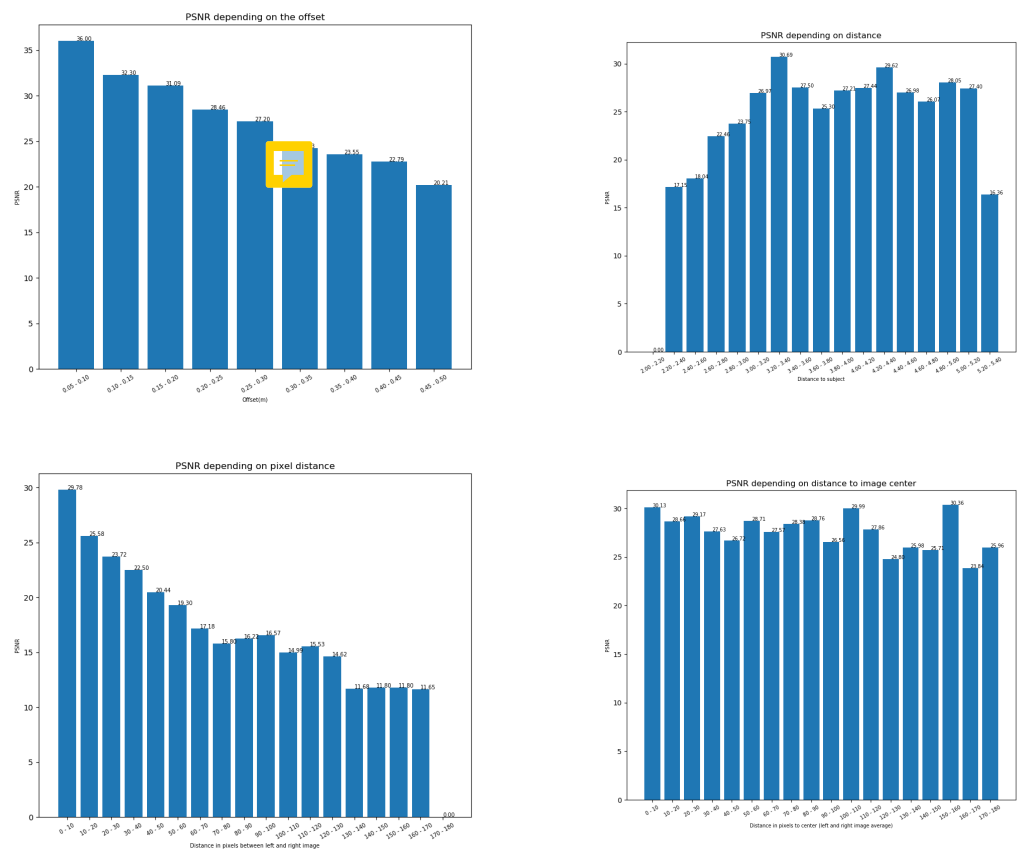


Figure 4.4: TODO

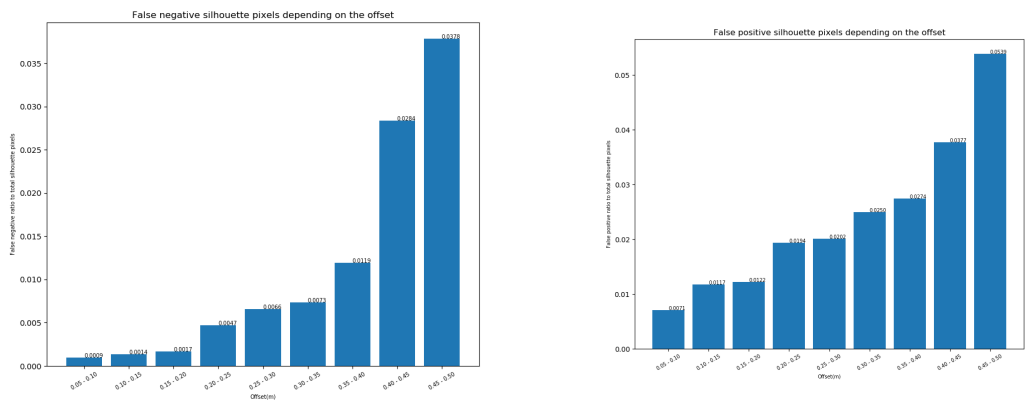


Figure 4.5: TODO

### 4.3 Silhouettes - False Negative Ratio and False Positive Ratio

Model with 51-px kernel L1-Loss trained for 50 epochs:

Fig. 4.5

(TODO: Text)

Model with 71-px kernel SSIM-loss trained for 10 epochs (+30 epochs L1 loss):

Fig. 4.6

(TODO: Text)

### 4.4 Hausdorff distance

Model with 51-px kernel L1-Loss trained for 50 epochs:

Fig. 4.7

Fig. 4.8

(TODO: Text)

Model with 71-px kernel SSIM-loss trained for 10 epochs (+30 epochs L1 loss):

Fig. 4.9

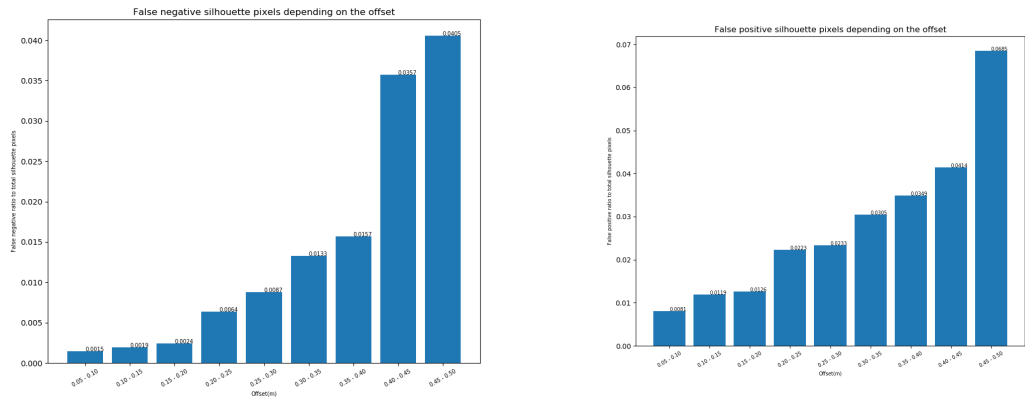


Figure 4.6: TODO

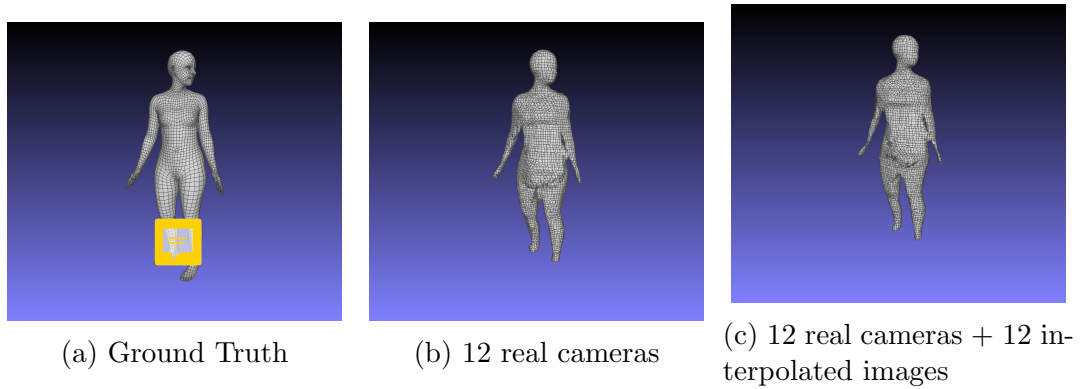


Figure 4.7: TODO

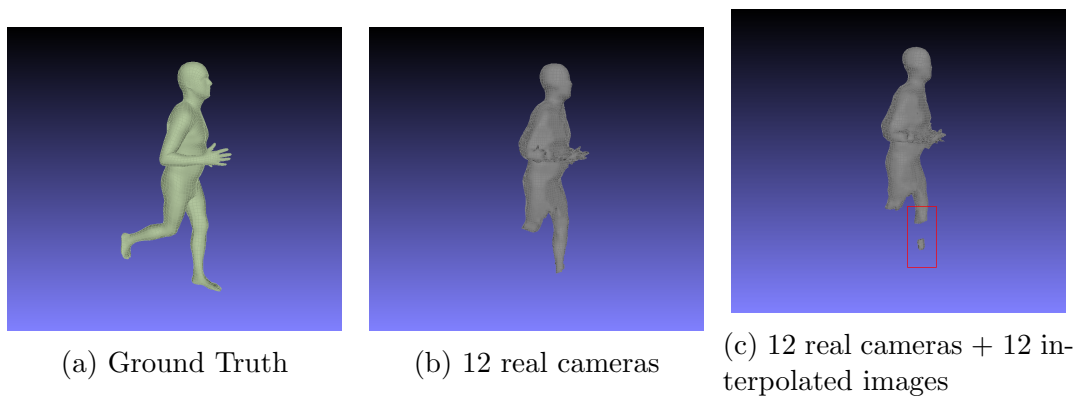


Figure 4.8: TODO

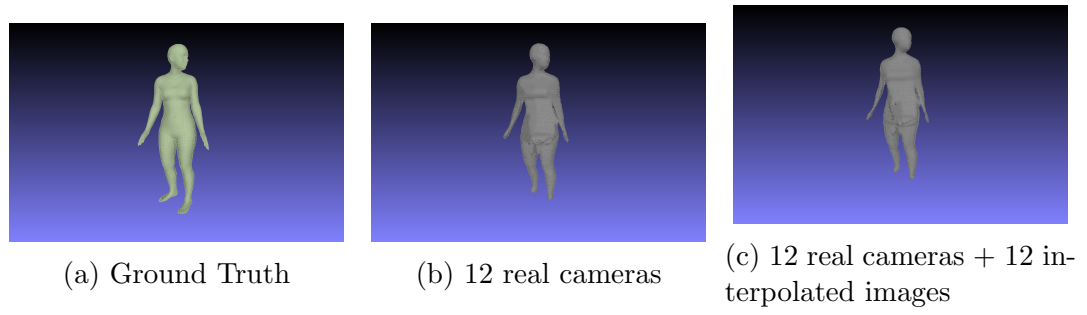


Figure 4.9: TODO

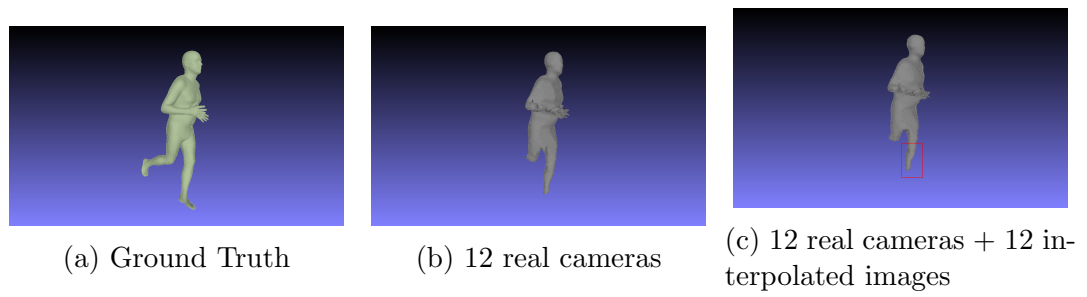


Figure 4.10: TODO

Fig. 4.10  
(TODO: Text)

## 4.5 Discussion

# Chapter 5

## Conclusion and Future Work

...

### 5.0.1 Conclusion

TODO

### 5.0.2 Future Work

TODO

1) DS - see above + SfS reconstruction test with person not in the focus point of cameras

# Bibliography

- [1] S. Belongie, “CSE 252B: Computer Vision II.” <https://cseweb.ucsd.edu/classes/sp04/cse252b/notes>, 2009.
- [2] S. B. Kang, Y. L. Xin Tong, and H.-Y. Shum, “Image-based rendering,” *Foundations and Trends in Computer Graphics and Vision*, 2007.
- [3] S. Kolouri, S. R. Park, and G. K. Rohde, “The radon cumulative distribution transform and its application to image classification,” *Ieee Transactions on Image Processing*, vol. 25, no. 2, pp. 920–934, 2016.
- [4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks..” <https://arxiv.org/abs/1406.2661>, 2014.
- [6] S. Niklaus, L. Mai, and F. Liu, “Video frame interpolation via adaptive convolution,” *CoRR*, vol. abs/1703.07514, 2017.
- [7] S. Niklaus and F. Liu, “Context-aware synthesis for video frame interpolation,” *CoRR*, vol. abs/1803.10967, 2018.
- [8] S. Niklaus, L. Mai, and F. Liu, “Video frame interpolation via adaptive separable convolution,” *CoRR*, pp. 261–270, 2017.
- [9] D. Mack, “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>, 2018.

- [10] C. Zhang and T. Chen, “A survey on image-based rendering - representation, sampling and compression,” *Signal Processing: Image Communication*, vol. 19, pp. 1–28, 2004.
- [11] R. Pages, K. Amliantis, D. Monaghan, J. Ondrej, and A. Smolic, “Affordable content creation for free-viewpoint video and vr/ar applications,” *Journal of Visual Communication and Image Representation*, vol. 53, pp. 192–201, 2018.
- [12] S. Li, C. Zhu, and M. Sun, “Hole filling with multiple reference views in DIBR view synthesis,” *IEEE Transactions on Multimedia*, vol. 20, no. 8, pp. 1948–1959, 2018.
- [13] S. M. B. Wenger, *Evaluation of SfM against traditional stereophotogrammetry and Lidar techniques for DSM creation in various land cover areas*. Thesis, Stellenbosch University, 2016.
- [14] D. King, “Easily create high quality object detectors with deep learning.” <http://blog.dlib.net/2016/10/easily-create-high-quality-object.html>, 2016.
- [15] J. Flynn, I. Neulander, J. Philbin, and N. Snavely, “Deepstereo: Learning to predict new views from the worlds imagery,” *CoRR abs/1506.06825*, 2015.
- [16] M. Kartaev, C. Rapisarda, and D. Fay, “Implementing adaptive separable convolution for video frame interpolation,” *arXiv*, 2018.
- [17] C. Tomasi and T. Kanade, “Shape and motion from image streams under orthography - a factorization method,” *International Journal of Computer Vision*, vol. 9, no. 2, pp. 137–154, 1992.
- [18] H. C. Longuet-Higgins, “A computer algorithm for reconstructing a scene from twoprojections,” *Nature*, vol. 293, 1981.
- [19] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.
- [20] R. I. Hartley, “In defence of the 8-point algorithm,” in *Proceedings of IEEE International Conference on Computer Vision*, pp. 1064–1070, 1995.

- [21] P. H. S. Torr and D. Murray, “The development and comparison of robust methods for estimating the fundamental matrix.,” *International Journal of Computer Vision*, 1997.
- [22] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, 2 ed., 2004.
- [23] O. D. Faugeras, “What can be seen in three dimensions with an uncalibrated stereo-rig?,” *Second European Conference on Computer Vision*, 1992.
- [24] A. Smolic, “Augmented reality lectures.” [https://tcd.blackboard.com/bbcswebdav/pid-1169545-dt-content-rid-6756029\\_1/courses/CS7434-A-SEM202-201819/02\\_AR2019\\_CameraModel.pdf](https://tcd.blackboard.com/bbcswebdav/pid-1169545-dt-content-rid-6756029_1/courses/CS7434-A-SEM202-201819/02_AR2019_CameraModel.pdf), 2019.
- [25] J. L. Schonberger and J. M. Frahm, “Structure-from-motion revisited,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (NEW YORK), pp. 4104–4113, Ieee, 2016.
- [26] C. Harris and M. J. Stephens, “A combined corner and edge detector.,” *Alvey-Vision Conference*, 1988.
- [27] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application in stereo vision.,” *Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, 1981.
- [28] J. Shi and C. Tomasi, “Good features to track,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1994.
- [29] D. Lowe, “Object recognition from local scale-invariant features,” *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2, 2001.
- [30] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” *Computer Vision and Image Understanding*, vol. 110, p. 346359, 2008.
- [31] M. Calonder, V. Lepetit, and P. Fua, “Brief: Binary robust independent elementary features,” *11th European Conference on Computer Vision*, 2011.



- [32] J. M. Morel and Y. G., “Asift: A new framework for fully affine invariant image comparison,” *SIAM Journal on Imaging Sciences*, vol. 2, pp. 438–469, 2009.
- [33] C. Strecha, A. M. Bronstein, M. Bronstein, and P. Fua, “Ldhash: Improved matching with smaller descriptors,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, 2012.
- [34] M. W. Smith, J. L. Carrivick, and D. J. Quincey, “Structure from motion photogrammetry in physical geography,” *Progress in Physical Geography*, vol. 40, no. 2, pp. 247–275, 2016.
- [35] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, “A comparison and evaluation of multi-view stereo reconstruction algorithms,” *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.
- [36] S. Seitz and C. Dyer, “Photorealistic scene reconstruction by voxel coloring,” *International Journal of Computer Vision*, vol. 35, 1999.
- [37] Y. Furukawa and J. Ponce, “Carved visual hulls for image-based modeling,” *International Journal of Computer Vision*, vol. 81, pp. 53–67, 2009.
- [38] J. Li, E. Li, Y. Chen, L. Xu, and Y. Zhang, “Bundled depth-map merging for multi-view stereo,” *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2769–2776, 2010.
- [39] M. Quan and L. Lhuillier, “A quasi-dense approach to surface reconstruction from uncalibrated images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005.
- [40] E. Zheng, E. Dunn, V. Jovic, and J. M. Frahm, “Patchmatch based joint view selection and depthmap estimation,” *CVPR*, 2014.
- [41] M. Kazhdan, M. Bolitho, and H. Hoppe, “Poisson surface reconstruction,” *Eurographics Symposium on Geometry Processing*, 2006.
- [42] S. E. Chen and L. Williams, “View interpolation for image synthesis,” *Proceedings of the 20th annual conference on Computer graphics and interactive techniques: SIGGRAPH*, 1993.

- [43] P. Fragneto and A. Fusiello, “Uncalibrated view synthesis with homography interpolation,” *Second Joint 3DIM/ 3DPVT Conference 3d Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT 2012)*, pp. 270–277, 2012.
- [44] T. Gurdan, M. R. Oswald, and D. Cremers, “Spatial and temporal interpolation of multi-view image sequences,” *Pattern Recognition, GCPR 2014*, vol. 8753, pp. 305–316, 2014.
- [45] M. Werlberger, T. Pock, M. Unger, and H. Bischof, “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*, pp. 273–286, 2011.
- [46] D. Mahajan, F.-C. Huang, W. Matusik, R. Ramamoorthi, and P. Belhumeur, “Moving gradients: A path-based method for plausible image interpolation,” *ACM Trans. Graph.*, vol. 28, 2009.
- [47] K. Chen and A. Dirk, “Image sequence interpolation using optimal control,” *Journal of Mathematical Imaging and Vision*, vol. 41, 2010.
- [48] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof, “Anisotropic huber-l1 optical flow,” *Bmvc*, vol. 1, 2009.
- [49] T. Stich, C. Linz, G. Albuquerque, and M. A. Magnor, “View and time interpolation in image space,” *Comput. Graph. Forum*, vol. 27, pp. 1781–1787, 2008.
- [50] C. Lipski, C. Linz, K. Berger, A. Sellent, and M. Magnor, “Virtual video camera: Imagebased viewpoint navigation through space and time,” *Computer Graphics Forum*, vol. 29, pp. 2555 – 2568, 2010.
- [51] S. Robaszkiewicz and S. El Ghazzal, “Interpolating images between video frames using non-linear dimensionality reduction,” *ISML*, 2013.
- [52] S. Das, “CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more ....” <https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5>, 2017.

- [53] M. Pechyonkin, “Understanding hinton’s capsule networks. part I: Intuition..” <https://medium.com/ai<sup>3</sup>-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b>, 2017.
- [54] G. Hinton, S. Sabour, and N. Frosst, “Matrix-capsules with em routing,” *ICLR*, 2018.
- [55] R. Chen, M. A. Jalal, L. Mihaylova, and R. K. Moore, “Learning capsules for vehicle logo recognition.,” *ISIF*, 2018.
- [56] R. LaLonde and U. Bagci, “Capsules for object segmentation,” *MIDL*, 2018.
- [57] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, “How to construct deep recurrent neural networks,” *ICLR*, 2013.
- [58] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *ICLR*, 2016.
- [59] E. Denton, S. Chintala, A. Szlam, and R. Fergus, “Deep generative image models using a laplacian pyramid of adversarial networks,” *CVPR*, 2015.
- [60] A. Dosovitskiy, P. Fischer, E. Ilg, P. Husser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox, “FlowNet: Learning optical flow with convolutional networks,” *IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [61] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, “FlowNet 2.0: Evolution of optical flow estimation with deep networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [62] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, “Epicflow: Edge-preserving interpolation of correspondences for optical flow,,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [63] T. Xue, B. Chen, J. Wu, D. Wei, and W. Freeman, “Video enhancement with task-oriented flow,” *Computer Vision and Pattern Recognition*, 2019.

- [64] A. Ranjan and M. J. Black, “Optical flow estimation using a spatial pyramid network,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [65] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, “Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume,” *CVPR*, pp. 8934–8943, 2018.
- [66] S. Meyer, O. Wang, H. Zimmer, M. Grosse, and A. Sorkine-Hornung, “Phase-based frame interpolation for video,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1410–1418, 2015.
- [67] Z. Zhang, L. Song, R. Xie, and L. Chen, “Video frame interpolation using recurrent convolutional layers,” in *IEEE Fourth International Conference on Multimedia Big Data (BigMM)*, pp. 1–6, 2018.
- [68] Z. W. Liu, R. A. Yeh, X. O. Tang, Y. M. Liu, A. Agarwala, and Ieee, “Video frame synthesis using deep voxel flow,” in *16th IEEE International Conference on Computer Vision (ICCV)*, pp. 4473–4481, 2017.
- [69] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, “Deepflow: Large displacement optical flow with deep matching,” *ICCV*, pp. 1385–1392, 2013.
- [70] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2016.
- [71] L. Xu, J. Jia, and Y. Matsushita, “Motion detail preserving optical flow estimation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.
- [72] C. Li, D. Gu, X. Ma, K. Yang, S. Liu, and F. Jiang, “Video frame interpolation based on multi-scale convolutional network and adversarial training,” *IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pp. 553–560, 2018.
- [73] V. Samsonov, “Deep frame interpolation..” <https://arxiv.org/pdf/1706.01159.pdf>, 2017.
- [74] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan.” <https://arxiv.org/abs/1701.07875>, 2017.

- [75] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [76] Z. Hu, Y. Ma, and L. Ma, “Multi-scale video frame-synthesis network with transitive consistency loss,” *CVPR*, 2017.
- [77] M. Mathieu, C. Couprie, and Y. LeCun, “Deep multi-scale video prediction beyond mean square error,” in *International Conference on Learning Representations (ICLR)*, 2016.
- [78] J. Hwang and D. Shabbir, “Human motion reconstruction from action video data using a 3-layer-lstm.” <http://cs231n.stanford.edu/reports/2017/pdfs/945.pdf>, 2017.
- [79] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, “Loss functions for image restoration with neural networks,” *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 47–57, 2017.
- [80] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *ICLR*, 2015.
- [81] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.
- [82] T. Tieleman and G. Hinton, “Lecture 6.5 - RMSProp: Divide the gradient by a running average of its recent magnitude.” COURSERA: Neural Networks for Machine Learning, 2012.
- [83] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, “SMPL: A skinned multi-person linear model,” *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, vol. 34, pp. 248:1–248:16, Oct. 2015.
- [84] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid, “Learning from synthetic humans,” in *CVPR*, 2017.

- [85] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 13, 2004.
- [86] “Matlab tutorials - PSNR.” <https://uk.mathworks.com/help/vision/ref/psnr.html>, 2019.
- [87] “Vacancy: A voxel carving implementation in c++.” <https://github.com/unclearness/vacancy>, 2019.

# Appendix

...