,

Master's Thesis

RGB-D Sceneflow with Deep Convolutional Neural Networks

Muazzam Ali

1st Examiner: Prof. Dr. Thomas Brox

2nd Examiner: Prof. Dr. Matthias Teschner

Advisers: Huizhong Zhou
Benjamin Ummenhofer

Albert-Ludwigs-University Freiburg

Faculty of Engineering

Department of Computer Science

July 18th, 2018

Writing period

12. 05. 2018 – 18. 07. 2018

Examiner

Prof. Dr. Thomas Brox

Advisers

Huizhong Zhou, Benjamin Ummenhofer

DECLARATION

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work. I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Freiburg,

Place, Date	Signature

I would like to thank my family, my parents in particular, who always supported me when I needed them. And many thanks to Huizhong Zhou and Benjamin Ummenhofer for their supervision throughout my work, their insights were of tremendous help during my thesis. Special thanks to Muneeb Shahid for proof reading my report. Lastly, I am thankful to my friends who have been of incredible support throughout, especially towards the end of my thesis.

Abstract

Extensive research in the field of *Computer Vision* focuses on the problem of analyzing images. One of these problems is the estimation of depth and motion between frames of a video sequence. In this thesis, we focus on solving this problem also called sceneflow estimation. Instead of approaching this problem in a classical way, we use *Deep Convolutional Neural Networks* to estimate more accurate optical flow values by leveraging the color and depth information from RGBD cameras. We introduce different unsupervised losses to ensure minimum dependence on ground truth datasets and maximum generalization. We also present three different architectures that can be used to train the deep networks for sceneflow, we call them from the original flownet paper as *FlowNetSimple*, *FlowNetSimpleStacked* and *FlowNetGAN*. We discuss in detail on how different network architectures and unsupervised losses affect our network predictions and make detailed comparisons.

Zusammenfassung

Forschung im Bereich Rechnersehen beschaftigt sich hauptsachlich mit dem Problem der Bildanalyse. Eines dieser Probleme ist die Bestimmung der Tiefe und Bewegung zwischen Bildern aus Videosequenzen. In dieser Arbeit beschaftigen wir uns mit genau diesem Problem, das auch "sceneflow estimation" genannt wird. Anstatt uns klassischer Methoden zu bedienen um das Problem zu lasen, werden wir tiefe "faltende neuronale Netze" (engl.: convolutional neural networks) verwenden, um genaueren optischen Fluss zu bestimmen mit Hilfe der Farb und Tiefeninformationen von RGBD-Kameras. Wir stellen verschiedene Kostenfunktionen fur unaberwachtes lernen auf, um die Abhangigkeit von Ground-Truth Daten zu minimieren und die Generalisierung zu verbessern. Wir prresentieren drei Architekturen, die geeignet sind um die Bestimmung von Sceneflow zu lernen. Wir nennen diese, angelehnt an die originale FlowNet Arbeit, FlowNetSimple, FlowNetSimple stacked und FlowNetGAN. Wir besprechen wie diese verschiedenen Architekturen und Kostenfunktionen die Vorhersagen der Netzwerke beeinflussen qualitativ und quantitativ.

Chapter 1

Introduction

Analyzing movement of objects in a 3D space is a very important problem of *Computer Vision*. Its applications range from motion estimation, 3D shape acquisition, object detection, pose Estimation and many more. Now, with the availability of cheap RGB-D cameras, depth information can also be leveraged to attain even higher accuracy in these given problems. There has been quite a lot of progress in estimating sceneflow recently.

In order to understand what sceneflow is. In order to understand how *Sceneflow* works, we'll first go about formally defining optical flow.

1.1 Optical Flow

Optical flow is defined as the motion of pixels from image frame I_1 to I_2 at time t_1 and t_2 respectively. Formally, if we consider the brightness constancy assumption we can define optical flow as

$$I(\vec{x}, t) = I(\vec{x} + \vec{u}, t + 1)$$

Where I($\vec{x}$,t) is the image intensity as a function of space $\vec{x}$ = (x, y)$^{\mathrm{T}}$ and time t, and $\vec{u}$ = (u, v)$^{\mathrm{T}}$ is the 2D velocity.

1.2 Sceneflow

In *scene flow* along with the information as in *optical flow* we additionally consider depth information of the given frames. Hence, instead of just estimating the (u,v) values as in optical flow, we end up estimating $(u, v, w)^T$ in scene flow. given the camera intrinsics, scene flow can be formally defined in the following manner.

$$\mathbf{M} = \begin{pmatrix} \frac{Z}{f_x} & 0 & \frac{X}{Z} \\ 0 & \frac{Z}{f_y} & \frac{Y}{Z} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

Where the equation can be deducted directly from the well-known "pin-hole model", **M** represents the motion field with respect to the camera, $\mathbf{f_x}$ and $\mathbf{f_y}$ represents the focal length while **X,Y,Z** represents the spatial coordinates of the observed point and **u,v,w** represent the estimation of flow values in u(motion in x coordinate),v(motion in y coordinate) and w(depth change) respectively.

1.3 Methods for Estimating Sceneflow

Several methods have been proposed in the past to solve the problem of sceneflow estimation, the focus of this work was to approach this problem using deep learning instead. Lets' take a look at some previous work done using both classical and deep learning methods.

1.3.1 Sceneflow Using Classical Methods

Traditionally, sceneflow has been estimated using stereo cameras using multiple images. The first paper with the idea of sceneflow was proposed by Vedula et al. [1] where they calculated Lucas Kenade optical flow first followed by the range flow constraint equation , obtaining a local solution which does not exploit the geometric data to estimate the optical flow.

There have been other classical methods like Jaimez et. al [2] used a primal-dual approach in calculating dense sceneflow in real time. In their approach they have used a method to refine optical flow and range flow constraints together in a

coarse to fine scheme. Furthermore, 3D surface regularization is imposed for the smoothness of flow fields .

While Jaimez et. al [2] used a primal-dual approach, M. Menze and A. Geiger [3] used segmentation to reason about the objects in the scene by dividing the scene into planasceneflowr patches and reason about the flow of those patches instead of individual pixels. This also helped them to reduce the parameter size making the solution more efficient.

1.3.2 Sceneflow Using Deep Learning

With the recent research and success of deep learning especially in the field of computer vision. Researchers have been more inclined to take this approach as their go to strategy to solve more complex problems. With the success of FlowNet[4], DenseNet[5], ResNet[6] and other famous architectures it seems apparent that many of the problems in the field of computer science can be solved using deep learning giving us the power to solve more complex tasks.

In fact, there have been some research in estimating sceneflow using Deep learning, one such method uses deep networks to predict RGBD sceneflow for action recognition proposed by Wang et al. [7] where they encode RGBD video sequences based on scene flow into one motion map, called Scene Flow to Action Map (SFAM). SFAM is then used to learn depth and RGB features using ConvNet as a single entity.

While for training deep networks, there have been good amount of work done in creating large synthetic datasets with proper ground truth values but things get rather difficult to make the same models work on more realistic scenarios due to higher complexity in motion boundaries, brightness constancy, occlusions and other important attributes. Hence, with that in mind, we can either take the approach of creating more datasets resulting in more time and cost or go for semi-supervised approaches to tackle this problem, making the maximum out of the available resources and the independence of network to work in any scenario.

In this thesis, we're going to build on top of an existing strategy used for estimating Optical flow using deep convolutional neural networks. We'll take our approach further by extending FlowNetS proposed by Fischer et al.[4] to include unsupervised methods for the network to be able to adopt to unseen scenarios. A more realistic dataset without labels will be used along with synthetic dataset

to train the network in a semi-supervised manner with unsupervised losses. One additional advantage of this approach is the reduction of complexity of the task by not considering camera parameters in the training and prediction process unlike other classical sceneflow estimation approaches.

In chapter 2 and 3, we will first define our core network architectures and the steps we took to start training our experiment. In chapter 4, also add links to the chapters) we'll be explaining our core contributions in this thesis while in chapter 5 we'll be discussing in detail about our experiments and results. We'll be concluding our report in chapter 6.

Chapter 2

Training FlowNet

FlowNet was first introduced by Philipp et al. [4] in their famous paper called "*FlowNet: Learning Optical Flow with Convolutional Networks*". The paper introduces two different supervised network architectures FlowNetSimple(FlowNetS) and FlowNetCorrelation(FlowNetC) to predict end-to-end optical flow values. Due to lack of ground truth datasets, they trained their networks on synthetic datasets and proved the networks to generalize well on more realistic datasets.

FlowNet is a state of the art method for estimating optical flow as it was the first of it's kind to formulate the optical flow task end-to-end as a learning problem. As mentioned in the paper, Long et al. [8] used a technique to directly refine the coarse predictions using *UpConvolutional* layers. While flownet uses the same idea of refinement, they also refine complete feature maps in their network using concatenation with previous layers.

2.1 FlowNet Architecture

The network also used an encoder-decoder architecture. The encoder part is composed of multiple convolution layers stacked together while the decoder part uses skip connections in combination with deconvolution layers to achieve a higher spatial resolution with accurate flow predictions. It's rather interesting to look more into detail on how flownet uses deconvolutions to refine the flow predictions with higher spatial resolution.

FlowNetSimple is an encoder-decoder network that was trained by stacking two RGB images into 6 channel input and retrieving an output estimating the

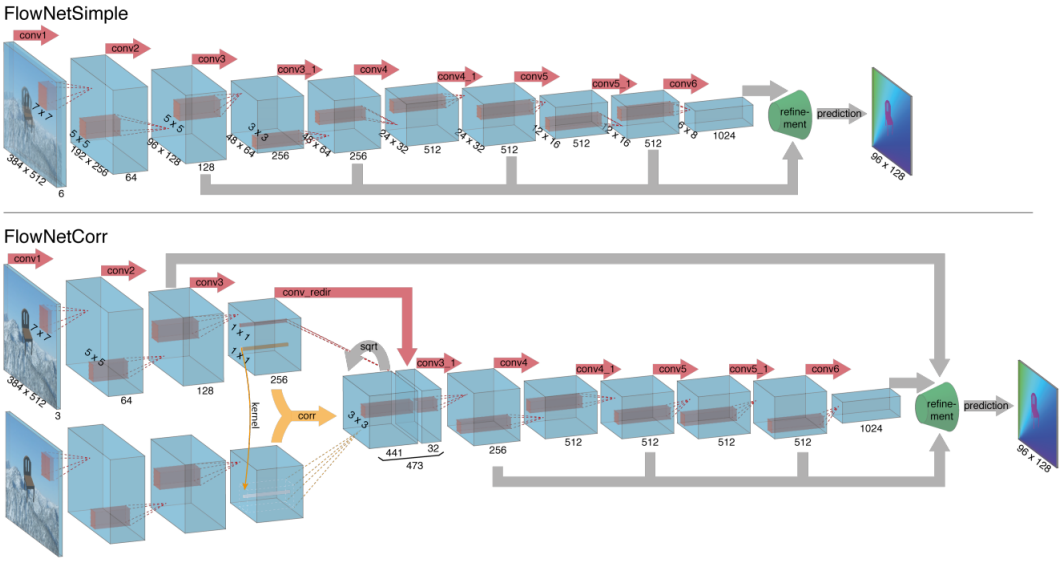

Figure 1: *The two network architectures: FlowNetSimple (top) and FlowNetCorr (bottom). Image from flownet paper[4]*

per pixel flow in a similar but slightly smaller image spatial dimension. The architecture of the network can be seen in figure 1.

While *FlowNetSimple* was giving pretty good results on its own, the researchers also tried another architecture for the network in which they focused on first learning the high level features from individual RGB images by passing them separately into two separate series of convolution layers and than used a *correlation layer* which acted as another *convolution layer* but instead of convolving with a filter, it convolves data with data i.e. the two feature maps obtained from the individual images convolutions. Although this has a higher cost but achieves higher accuracy.

2.1.1 Flow Refinement

Since flownet is an encoder-decoder network, we pass an image into the network which is spatially convolved to a smaller dimension. From there a refinement method is used as mentioned in the original flownet[4] paper under the refinement section.

To perform the refinement, we apply the 'upconvolution' to feature

maps, and concatenate it with corresponding feature maps from the 'contractive' part of the network and an upsampled coarser flow prediction (if available). This way we preserve both the high-level information passed from coarser feature maps and fine local information provided in lower layer feature maps

As seen in figure 1(top), the green funnel at the end of the network represents multiple refinement layers. The grey arrows at the bottom of the network, pointing towards the green funnel, represent the layers which are later concatenated with the deconvolved layers, resulting in higher resolution features.

This turned out to be a quite useful method to for achieving flow values with higher accuracy. As mentioned further in this report. We were able to introduce some useful unsupervised losses on these layers to help learn useful features more accurately.

2.1.2 Forward & Backward Flow

Since sceneflow is a problem where we have a continuous stream of images (video sequences), we can use the backward motion of the scene to our advantage to ensure our network predictions are more accurate. This gives us even broader range of opportunities to integrate unsupervised losses by ensuring the forward and backward flow of the scenes are coherent in nature.

So instead of using flownet in a traditional way, we feed in swapped sandwiched RGBD images to the network and make the network take into account the forward and backward consistency of the flow values.

Note that the images used in Figure 2 and 3 are only RGB images. Respective depth images are also concatenated as a channel, while inputting the images to the network.

2.2 Datasets

There are various RGBD datasets available, McCormac et al. [10] provide a large scale RGBD photorealistic dataset with indoor scene trajectories with ground truth values. The dataset is a combination of 5 million RGBD images from over 15K trajectories in synthetic layouts with random but physically simulated object

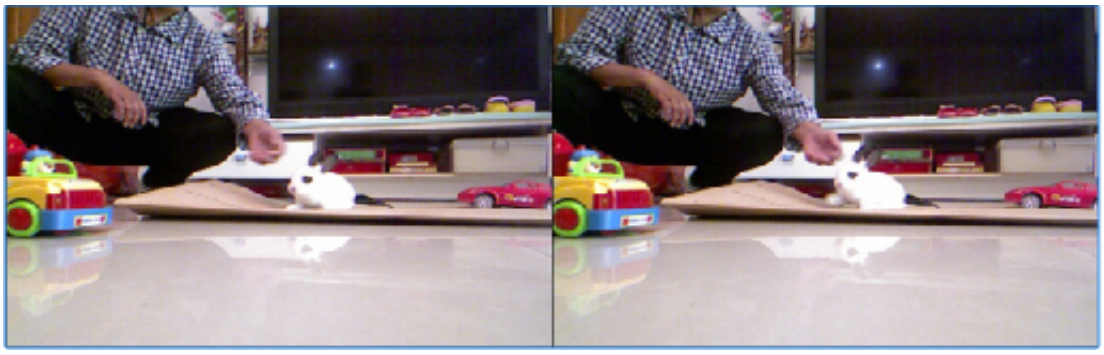

Figure 2: *Input images fed forward. Taken from Princeton Tracking Benchmark [9]*

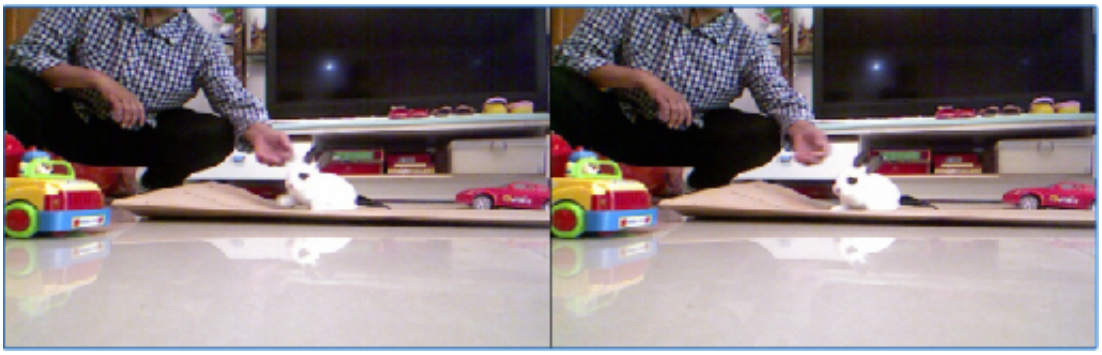

Figure 3: *Input images fed backward. Taken from Princeton Tracking Benchmark [9]*

poses. Jaeyong Sung et al. [11, 12] came up with a dataset of human activities with video sequences in different environments. They offered two versions of the dataset called CAD-60 and CAD-120 with 60 and 120 videos of daily activities respectively.

Although there have been a lot more RGBD datasets with and without ground truth values. We settled for an approach training the network with *Synthetic Dataset* and *Princeton Tracking Benchmark Dataset*. Both these datasets seemed to be the best combination of with and without ground truth values to prove our network has the ability to generalize well. This better generalization will be achieved by using unsupervised learning to make things fit better for realistic scenarios.

2.2.1 Princeton Tracking Benchmark

The Princeton Tracking Benchmark is a dataset containing a combination of 100 different RGBD videos with manually annotated ground truth bounding boxes. The dataset was developed by Shuran Song and Jianxiong Xiao [9] to perform tracking using 2D and 3D pointcloud methods.

The dataset is a good fit for our work due to high variance and different motion scenarios. The dataset is divided into motion of human, animals and rigid objects; as human movements will have higher degrees of freedom compared to the movement of rigid objects. This helps us cater to all kinds of easy and difficult situations in sceneflow estimation with our unsupervised method. Although there are additional ground truth values which give us information regarding the bounding box of moving objects inside the images, but this was not considered as it was out of scope for our work. Hence we treated this dataset as our core dataset for unsupervised losses. Figure 4 shows a distribution of the dataset with respect to different attributes on the left(y-axis) and video numbers at the bottom(x-axis).

2.2.2 Synthetic Dataset

While there is still a problem of having more realistic datasets with proper ground truth values for training deep networks, Mayer et al. [13] came up with three synthetic datasets with proper disparity and flow ground truths. The

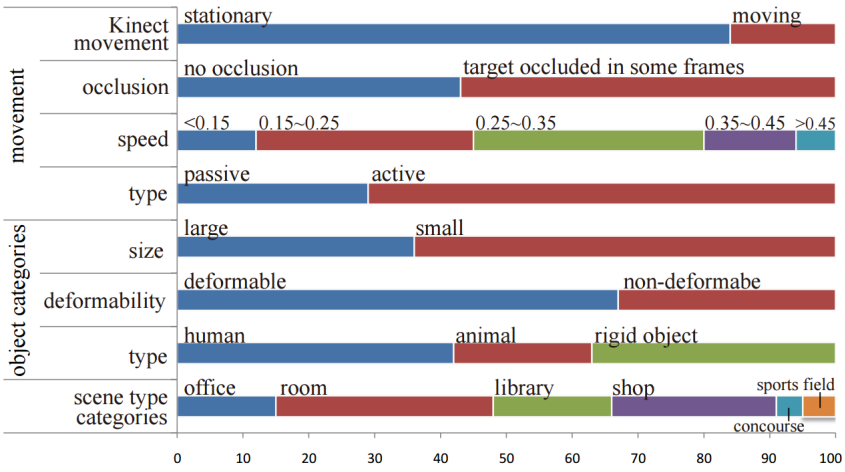

Figure 4: *Statistics from Princeton Tracking Benchmark*

Figure 5: *(left)Images from Princeton Tracking Benchmark dataset collected using Kinect motion sensor in different realistic environments. (right)A sample from synthetic dataset [13] showing original image,flow, disparity and depth images.*

datasets offer images from three separate environments with sufficient realism, these environments are named Driving, FlyingThings3D and Monkaa. All three datasets are used in training our deep convolutional network. The datasets are not used completely and part of it was left out intentionally to make sure the network makes up for it using its unsupervised losses. The three datasets are rendered using a focal length of 35mm on a 32mm wide simulated sensor. The driving dataset is also using a focal length of 15mm. The images of the datasets are available in two versions namely final-pass and clean-pass images. Clean-pass shows colors textures and scene lighting but without image degradations. While the final pass additionally includes postprocessing effects such as simulated depth-of-field blur, motion blur, sunlight glare, and gamma curve manipulation.

Driving

The driving dataset is composed of images of video sequences with a first person view of moving around a city in a car. The dataset was built with KITTI benchmark suite [3] in mind.

FlyingThings3D

The flyingthings3D dataset is the largest of all three datasets with 25000 stereo frames. The dataset is composed of images with daily household objects flying around in different scenes.

Monkaa

The dataset is built with open source blender assets of the animated short film Monkaa. The dataset was built with MPI sintel [14] dataset in mind with much resemblence. Monkaa contains nonrigid and softly articulated motion as well as visually challenging fur.

Middlebury

We also use the Middlebury benchmark dataset in order to test how our networks react to a completely new and unseen data.

We use Middlebury 2003 [15] and 2005[16] for our experiments. The datasets consists of disparity maps and RGB images taken with a structured-lighting

approach to construct a database of stereo pairs with ground-truth disparities.

Chapter 3

Training Generative Adversarial Networks

Generative Adversarial Networks (GANs) have been talked about a lot since the past couple of years. I. Goodfellow [17] in 2014 proposed the idea of GANs which was the starting point of a new direction of research in deep learning. The idea got highly popular in the deep learning community with *Yann LeCun*, Facebook's chief AI scientist, calling GANs *"the coolest idea in deep learning in the last 20 years"*.

The research in GAN grew very quickly with the introduction of many variants of GANs including DCGAN[18],cGANS[19],InfoGANs[20] and a lot others.

Currently, GANs are being used in different areas of research like Adobe implemented IGANs[21] to generate beautiful drawings with minimum user edits as shown in figure 6. While at Nvidia, researchers used variational autoencoders(VAE) along with GANs to convert images to different times of the day which were also successful in changing the weather of the environment[22].

3.1 How It Works

Usually, the best way to understand how GANs work is considering an example of a criminal and a cop where the objective of the criminal is to be a successful money counterfeiter, the criminal wants to fool the cop such that the cop can't tell the difference between counterfeited money and real money while the objective of the cop is to find out if the money is counterfeited or real.

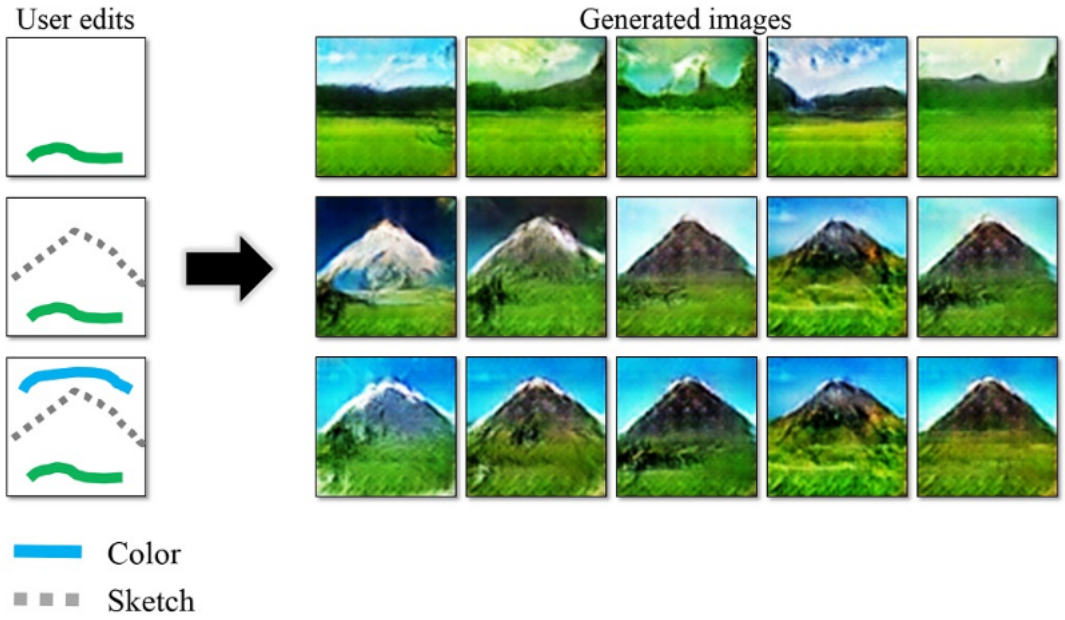

Figure 6: *The left chart shows the user edits while the right images are the resulting images generated by the network with those edits [21]*

Figure 7: *Using GANs to translate images to different times of the day or changing weather [22]*

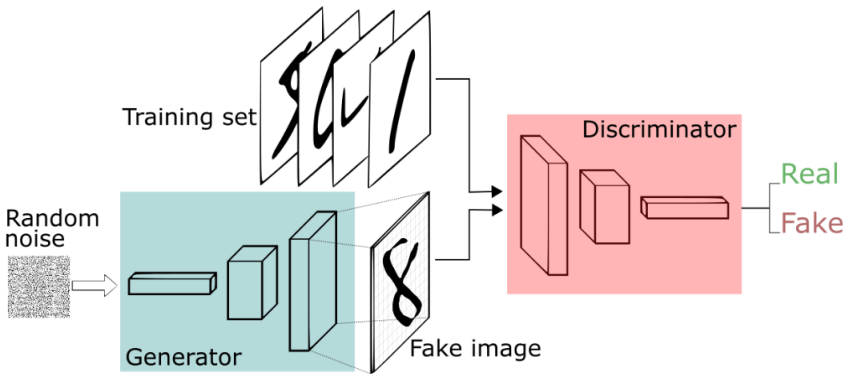

Figure 8: *Architecture of GAN using Convolutional layers. Taken from https://deeplearning4j.org/generative-adversarial-network*

Similarly, the generator network acts like the criminal while the discriminator network acts like a cop. The generator tries to generate data from random noise trying to generate data that resembles closely with the ground truth while the discriminator's job is to identify the real or generated data and assign a probability of as real or fake. The discriminator network forces the generator to generate even more realistic looking data as we progress through the training.

GANs define an adversarial loss which the networks are trying to minimize and maximize simultaneously.

$$\min_G \max_D \quad V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[log\ D(x)] + \mathbb{E}_{z\sim p_{data}(z)}[log\ D(G(z))]$$

Where $p_{data}(x)$ represents the distribution of the real data, $p_{data}(z)$ represents the distribution of the generator, z reprsents a sample from $p(z)$ while x represents a sample from $p(x)$, $G(x)$ represents the generator network and $D(x)$ represents the discriminator network. In our function V(D, G) the first term is entropy that the data from real distribution $p_{data}(x)$ passes through the discriminator. The discriminator tries to maximize this to 1. The second term is entropy that the data from random input $p_{data}(z)$ passes through the generator, which then generates a fake sample which is then passed through the discriminator to identify the fakeness. In this term, discriminator tries to minimize it to 0 (i.e. the log probability that the data from generated is fake is equal to 0). So overall, the discriminator is trying to maximize our function V.

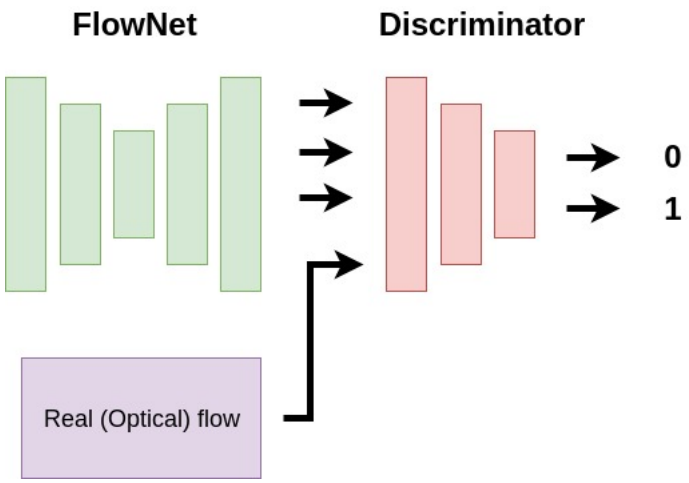

Figure 9: *The GAN architecture used to train flownet with an adversarial loss.*

On the other hand, the task of generator is exactly opposite, i.e. it tries to minimize the function V, so that the differentiation between real and fake data is bare minimum. This, in other words is a cat and mouse game between generator and discriminator.

3.2 Flownet as a GAN

Using the same idea defined in the previous section, we came up with an approach to introduce a new discriminator network that helps us implement an additional unsupervised adversarial loss. The idea is to train the discriminator network in such a way that it forces the flownet to generate even better predictions. As shown in figure 9, we input the output flow predictions from the flownet to the discriminator which tries to output a probability close to 0 if the input image is from the flownet otherwise close to 1.

The discriminator network is a simple 5 layer convolution network which outputs a single number representing the probability of real or fake. Both the networks were trained with the same learning rate and single step each. There were other options tried like *feature matching* suggested in [23] where the idea is to select a mid layer in the discriminator network from the fake input and real input to the discriminator network and minimize the difference. This forces the generator to generate more realistic images in order for the discriminator to be not able to identify part of the images using which it assigns a low probability to

the fake image.

Trying this method somehow made the discriminator more powerful and lose the balance of GAN due to which we gave up on this method.

There were several other methods used to train the GANs, but the best combination was reached when we trained a normal GAN with the adversarial loss defined above. Since the output from the GAN network was still blurry, we decided to add an extra FlowNetS as a stacked network to smooth out our predictions from the FlowNetGAN. This is further discussed in the experiments section.

Chapter 4

Unsupervised Losses

One of the most important part of deep networks is to define effective losses for the network to minimize them for better accuracy. But these losses are designed by keeping in mind certain aspects of the problem we are looking at, helping the network to focus on minimizing the difference between those features of the target and estimated values.

The two most common forms of losses that are generally used are *least absolute deviation(L1)* and *least square errors(L2)* losses. More formally L1 loss is defined as follows.

$$\sum_{i=0}^{n} |\hat{y_{\mathrm{i}}} - h(\hat{x_{\mathrm{i}}})|$$

Where $\mathrm{y_i}$ represents the target values while $\mathrm{h(x_i)}$ represents the predictions. As it is apparent from the above equation that L1 loss will treat the outliers in the same manner as closely predicted values, but in case of squared deviation loss is more sensitive to outliers which are penalized higher than the normal values. More formally squared L2 loss is defined as.

$$\sum_{i=0}^{n} (\hat{y_{\mathrm{i}}} - h(\hat{x_{\mathrm{i}}}))^2$$

We take a square of the difference between the target and estimated value. The higher the difference the higher the penalty.

Although there have been many losses defined in the deep learning research for solving different type of problems the two most commonly used losses while

training deep neural networks from optical flow are called *Endpoint Error(EPE)* and *Angular Error(AE)*.

4.1 Image Warping

When estimating the optical flow values, it is always useful to visualize how good are the estimated results. If we recall from chapter one, optical flow is the apparent motion of objects from image I_1 to image I_2, hence we can easily use the estimated flow values to generate one image by applying the flow values to the other.

As shown in figure 11, image I_2 from figure 10 is warped with the ground truth forward flow to obtain the resulting warped image looking close to I_1. Under the hood, warping in optical flow can be implemented in three simple steps. The pseudocode is.

1. Creating Meshgrid.

2. Adding the flow values.

3. Resampling to generate the resulting image. ...

For each step, we use the corresponding tensorflow functions in our implementation.

Figure 10: *Sample images I_1(left) and I_2(right) from the synthetic driving dataset.*

Figure 11: *Warped image for the images in figure 10, using the ground truth flow values provided by the dataset.*

4.1.1 Meshgrid

When trying to warp the image, the first step we do is to create a meshgrid of the image in separate horizontal and vertical dimensions, these meshgrid matrices are of the same spatial dimension as the image we'll be performing the resampling operation(step 3) on.

Let's suppose the images we're dealing have a spatial dimension of 3x3 (width-height). The meshgrids will help us to get a matrix of the same dimensions represented by the pixel index as shown in table 1.

1	2	3
1	2	3
1	2	3

1	1	1
2	2	2
3	3	3

Table 1: *Matrices generated by meshgrid in horizontal(left) and vertical(right) dimensions.*

4.1.2 Adding flow values

The intuition behind this step is to just add the flow values to the given meshgrid matrices in order to get the resulting pixel positions where we would like to sample

3.23	9.53	1.71
32.98	23.63	2.44
44.11	1.09	7.56

Table 2: *Flow matrix in horizontal direction.*

4.23	11.53	4.71
33.98	25.63	5.44
45.11	3.09	10.56

Table 3: *Index matrix used for resampling.*

from, to generate the warped images. Just for the sake of simplicity, we take the meshgrid and flow values for only the horizontal dimension now. Imagine our flow matrix has only integer values and the *u-component* of the flow looks like as in table 2 while the resulting index matrix after adding the given flow values to the meshgrid is given in table 3.

4.1.3 Resampling

Let's call the image we are performing resampling on as I_x. Now we can simply use bilinear interpolation or any other suitable method to perform resampling on I_x at the positions mentioned in table 3. This will result in us getting a warped image, like the one shown in figure 11.

The black values shown in figure 11 are due to the occlusions occuring because of the large flow values at the corner of the image resulting in the target pixel being outside the spatial dimension of the image.

4.1.4 Bilinear Interpolation

The sampling technique we used for warping the images is called Bilinear Interpolation. As shown in figure 12, the four red dots Q_{12},Q_{22},Q_{11},Q_{21} represents the points we want to interpolate from the green dot **P** will be the resulting point. In order to get **P**, we first linearly interpolate points Q_{12}, Q_{22} horizontally to find R_2 and Q_{11}, Q_{21} to find R_1. We then further interpolate vertically points R_1 and R_2 to find the resulting point **P**.

More formally, we perform the following steps to calculate $\mathbf{R_1}$, $\mathbf{R_2}$ and **P**.

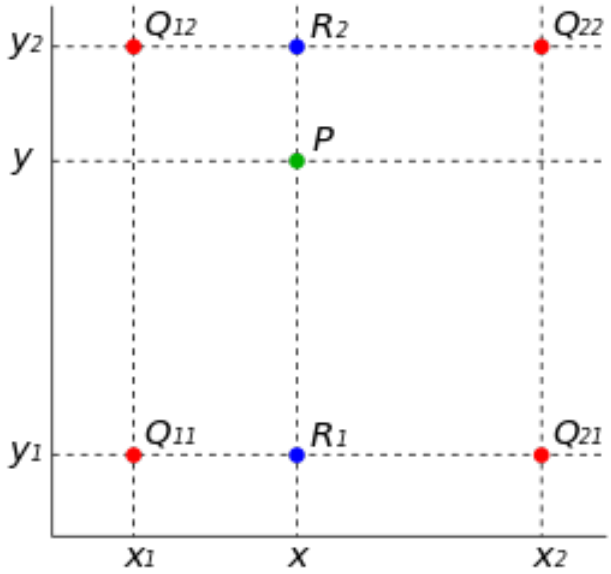

Figure 12: *Image explaining Bilinear Interpolation, taken from Wikipedia.*

$$R_1 = f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$R_2 = f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

$$P = f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2)$$

Where $f(Q_{11})$, $f(Q_{12})$, $f(Q_{21})$, $f(Q_{22})$ are assumed to be known values.

4.2 Endpoint Error

Endpoint error is the only loss used in all cases throughout our experiments. The loss is defined as the non-squared version of L2 norm defined above. The loss is also used in benchmarks to measure optical flow performance. More formally, the loss can be defined as,

$$\frac{\sum_{i=0}^{n}((\hat{y_{\mathrm{u}}} - h(\hat{x_{\mathrm{u}}}))^2 + (\hat{y_{\mathrm{v}}} - h(\hat{x_{\mathrm{v}}}))^2)}{n}$$

where $(\hat{y_u} - h(\hat{x_u}))^2$ represents the squared difference between the u-component of the ground truth and predicted flow values and same goes for the v-component. We divide it by n i.e the number of items giving us the average endpoint error.

4.3 Photoconsistency Loss

As explained in the previous section, image I_1 can be generated by warping the image I_2 with the forward flow $\mathrm{f}_{\mathrm{forward}}$. We define a photoconsistency loss to minimize the difference between the input image I_1 and warped image $\mathrm{I}_{\mathrm{warped_f}}$ with the estimated forward flow. The loss can be formally defined as.

$$PC(I_1, I_2) = I_1 - I_2(x + u, y + v)$$

To extend this further, we not only focus on defining the forward photoconsistency loss, but also backward photoconsistency loss. This can be achieved by simply warping I_1 with the backward flow $\mathrm{f}_{\mathrm{backward}}$ to minimize the difference between $\mathrm{I}_{\mathrm{warped_b}}$ and I_2.

$$PC(I_2, I_1) = I_2 - I_1(x + u, y + v)$$

4.4 Forward Backward Loss

In addition to the photoconsistency loss. We also define a forward backward loss. As shown in figure 13. If we consider the two grids to be our two images I_1 and I_2, you can see in the flow forward image, we move from pixel **P11** in I_1 to **P23** in I_2. But in order for our flow values to be consistent, once we're at **P23** we should be able to easily get back to the original position **P11** by using our backward flow prediction.

This is not the case, as we'll end up somewhere close or maybe far off the original position during the flow backward step resulting in us getting to position **P31** in this case instead. Hence we try to minimize this loss to make our flow estimations more consistent. More formally, the loss can be defined as,

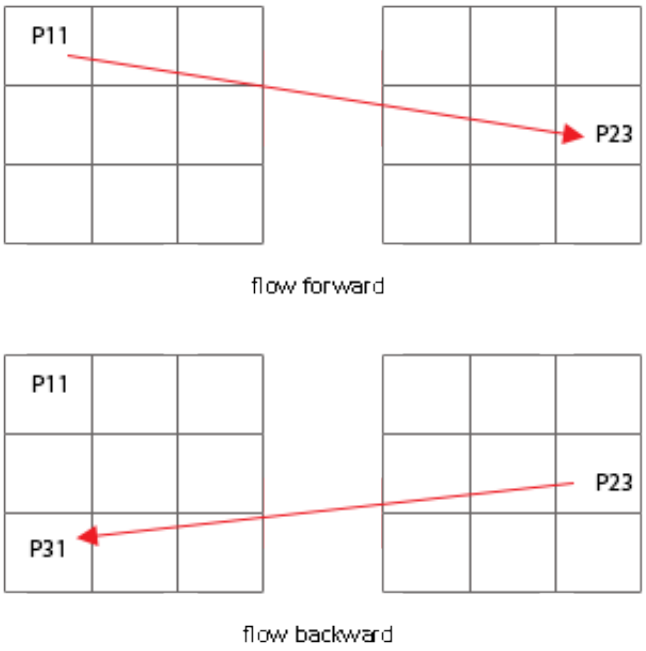

Figure 13: *Forward backward loss between two images. I_1 on left, I_2 on right.*

$$FB(f, b) = f - b(x + f_{\mathrm{u}}, y + f_{\mathrm{v}})$$

Where $\boldsymbol{f}$ represents the predicted forward flow while **b** represents the backward flow with respect to the predicted forward flow.

4.5 Scale Invariant Gradient Loss

One of the major problems when training networks for estimating flow values is that the predicted flow is somewhat blurry resulting in less accurate predictions. In order to tackle this problem we used scale invariant gradient loss (SIGL) introduced in the *DeMoN: Depth and Motion Network for Learning Monocular Stereo[24]* where the writers defined a loss function on the neighboring image pixels to smooth out relative depth discontinuities. We first calculate scale invariant gradients for the given true and predicted flow values on 5 different spacings (h) with respect to a particular pixel.

$$g_h[f](i,j) = \left(\frac{f(i+h,j) - f(i,j)}{|f(i+h,j)| + |f(i,j)|} + \frac{f(i,j+h) - f(i,j)}{|f(i,j+h)| + |f(i,j)|} \right)$$

We define a loss over the calculated scale invariant gradients penalizing relative depth errors between the neighboring pixels.

$$\mathcal{L}_{grad\,\xi} = \sum_{h \in 1,2,4,8,16} \sum_{i,j} \| g_h[\xi](i,j) - g_h[\hat{\xi}](i,j) \|_2$$

Where $\hat{\xi}$ represents our ground truth flow values, ξ represents the predicted flow. We apply the SIG loss to both flow components to smooth out the resulting predicted flow.

Chapter 5

Experiments

In this chapter, we will continue to use the concepts explained in the previous chapters and see how different combinations of our losses and network architectures worked out on synthetic, realistic as well as unseen data for the network.

We will only publish results for the optical flow u-v components without the depth values. Since our focus was to maximize accuracy on the flow components, prediction of better depth is left as a future work.

The three major network architectures we will be discussing are a simple variant of FlowNetS, a stacked version of FlowNetS and a GAN network architecture.

For the first architecture, we train the simple FlowNetS as a baseline. In contrast to the original FlowNetS from [4] we also input the depth information for each image.

For the second architecture we have a stacked version of FlowNetS where we use two instantiations of the FlowNetS with separate parameters and pass the output of the first network to the second network to improve the smoothness in our predictions.

The third architecture will feature a GAN network where we train our FlowNetS as a generator along with a discriminator network that will be used to create an additional unsupervised adversarial loss.

Along with the three major architectures we used a combination of different supervised and unsupervised losses explained in the previous chapters. We used different combinations of these losses throughout our experiments to see how the losses play their part in better predictions.

Finally, we also train a simple version of FlowNetS where we train the network

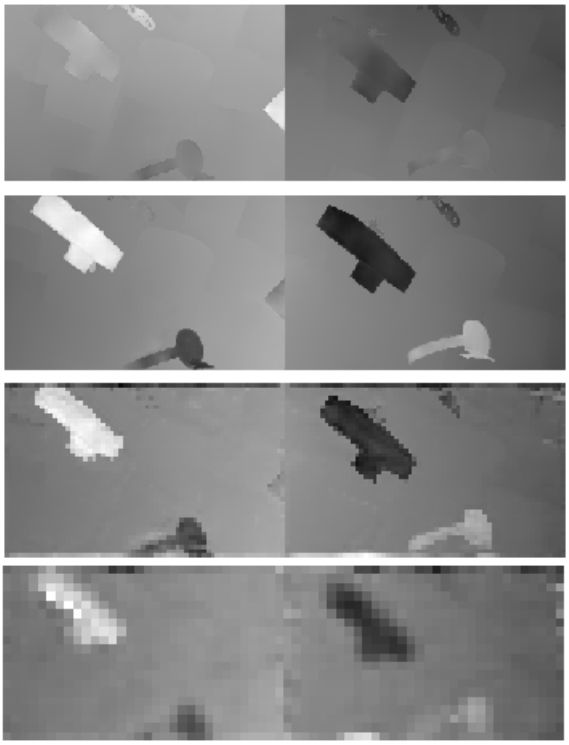

Figure 14: *Flow u-component Forward Backward predictions from the network on flying dataset. The topmost picture represents the prediction on the highest resolution(final prediction). The images in row 2,3 and 4 represents the predictions on the smaller resolutions of the network with the last one being the smallest.*

with all the mentioned losses without giving in the depth information. In short, we train the network as a classical FlowNetS to have a baseline method for better comparisons.

A chart is shown at the end of this chapter to provide a summarized understanding about the final achieved minimum average endpoint errors on different test datasets with different methods. All the images of predictions from the network provided throughout this chapter are taken from the test datasets.

5.1 FlowNetS

Our first approach was to simply train FlowNetS by training our network individually on each environment of synthetic dataset. Since **flyingthings dataset** is the largest among the three environments, we first train our network on the

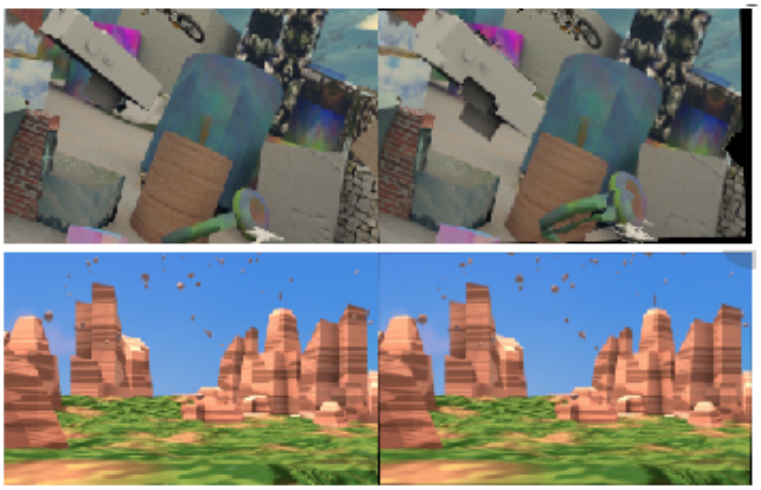

Figure 15: *Warped flow images with the predicted flows. The upper one is from flying dataset while the lower one from monkaa.*

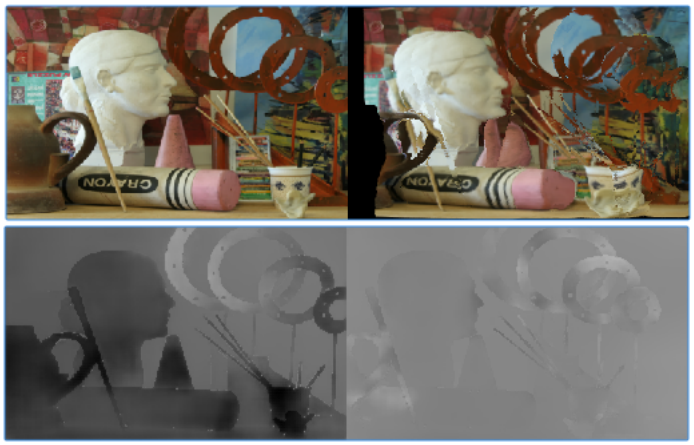

Figure 16: *(top left)Image 1. (top right)Warped image 2 with predicted forward flow. Predicted forward and backward flow at bottom left and right respectively. These images are from the Middlebury dataset 2005. The results are from FlowNetS-ST.*

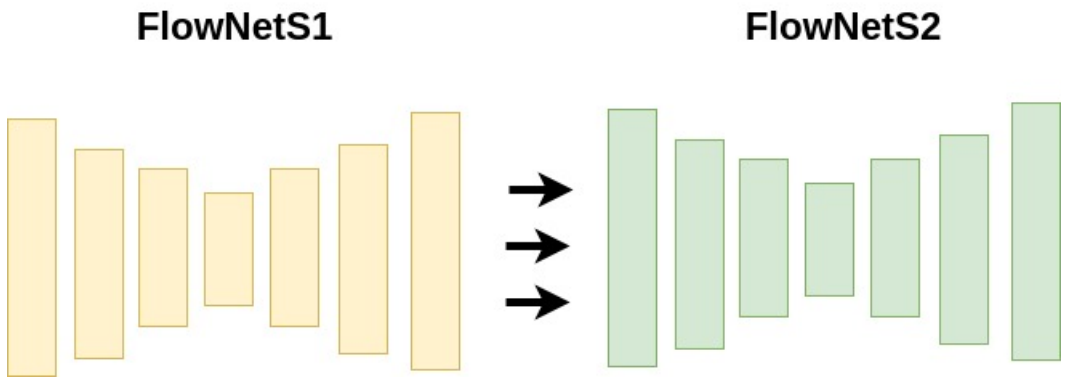

Figure 17: *Two FlowNetS stacked. The predicted flow from the first network is passed to the second network.*

flyingthings dataset and later on add **monkaa dataset** for fine tuning. We further train our network on PTB dataset. This is a dataset with highly sparse depth images and our idea was to ensure the network is able to learn to generalize well to the kinect depth sparse images. After training the network for a couple of days, we concluded that the sparsity in the dataset is very high due to which synthetic dataset was not enough to generalize on PTB. The final predictions for PTB were very noisy.

Although the predictions on the PTB datasets were not good enough, the network worked pretty well with the synthetic datasets themselves.

In order to further verify if our network can generalize well on a less sparse unseen environment, we used Middlebury dataset. On this dataset our results looked much better compared to PTB. This is because of less sparsity in the disparity images in the dataset. Although we believe the network can do even better in generalizing over Middlebury if we augment the synthetic dataset by introducing some sparsity. This will help the network learn how to interpolate the sparse pixels in the given synthetic images which in turn can be helpful on unseen datasets like PTB and Middlebury. This is left as a future work.

We train our network by minimizing the endpoint error and scale invariant gradient loss as the supervised losses, photo-consistency loss and forward backward loss as the unsupervised losses from the start of the training.

The endpoint error has been assigned 500 times more importance on the highest resolution while 100 times more importance on the lower resolutions making it the most important loss for our experiments. Both supervised and

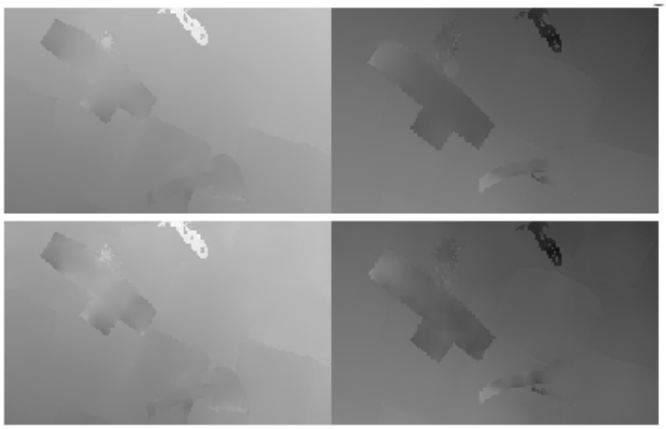

Figure 18: *Top row represents the FlowNetS2 forward and backward predictions while the bottom one shows the prediction of FlowNetS1. The left image is the forward flow while the right one is the backward flow. The shown images are from the u-component of the flow prediction.*

unsupervised losses apart from SIGL are also minimized on the middle layers of the network to ensure better convergence. In figure 14, one can take a look at the predictions on different layers(resolutions) of the network.

While in figure 15, we can look at the warped images for both the monkaa and flying dataset with the final predicted flows. The image from the flying dataset is the same for which the predicted flows are shown in figure 14.

At the topmost row we show the final prediction while predictions on the previous middle layers as we go down further. In the final row of the figure, the image looks blurry due to the prediction being in the lowest spatial dimension in the middle layers of the network. We can see the predictions look quite close to what should be expected with the sharp edges and objects being identified clearly.

But still there is some more place for improvement as can be seen from the topmost image a slight gradient like effect on the object. This will be smoothed out further when we use the stacked architecture discussed later.

5.2 FlowNetS-Stacked(ST)

Another version of FlowNetS was trained by stacking two FlowNetS one after the other. This helped increase further accuracy of the network. By using the first network to minimize a loss between the predicted and ground truth flow. While

the input to the second network is the output of the first network which helps us fine tune our predictions.

Unsupervised losses were not used on the second network giving it the objective to minimize the supervised endpoint error and SIGL for better prediction of sharp edges. As shown in figure 18, if we look more closely, the top predictions (predictions from the second network) are more smoothed out because the network has learned the details from the prediction of the first network and tried to smooth out the results to look closer to the ground truth flow. The image in figure 18 has a greyish dark to light gradient color effect on the forefront object. And if we compare the same object in the second image, one can notice the effect is smoothed out.

A much better example of the same image can be seen in figure 19 where we used the prediction from one of the middle layers (refinement layers) of the network. The difference is quite clear.

Although using a stacked network is a good way to achieve better accuracies, this also leads to slower processing due to larger number of parameters. It's always a trade off between speed and accuracy.

We also ran the networks on the benchmark Middlebury 2003 and 2005 datasets to make a comparison between the stacked and normal FlowNetS versions. The loss for comparison was *root mean squared error (RMSE)*. We chose this dataset as it is considered a benchmark for normal optical flow prediction tasks in different research papers. We can see clearly from table 4, the stacked version performed better on all images.

Note that the difference between the losses mentioned for each image of the dataset has a higher difference with respect to each other. This is due to the diversity of images taken, representing different scenes.

5.3 FlowNetS-GAN

In the FlowNetS-GAN version we use an adversarial loss in addition to a FlowNetS. Additionally, we use a stacked version of FlowNetS as mentioned in section 5.2 in order to ensure the resulting images are smoothed out, when introducing the adversarial loss, the predictions are not accurate enough and somehow distorted. This is because the network is then instead of minimizing multiple losses, it is

Dataset Sample	FlowNetS	FlowNetS Stacked	FlowNetGAN
Art	6.002	5.77	5.05
Moebius	3.38	3.31	2.73
teddyF	4.29	4.12	5.24
Reindeer	3.15	2.41	3.25
Books	2.38	2.18	4.40
conesF	5.11	4.92	4.84
Dolls	7.56	7.26	3.37
Laundry	3.45	3.08	3.17
Average	**4.41**	**4.13**	**4.01**

Table 4: *RMSE applied to Middlebury datasets 2005 and 2003 using the FlowNetS and FlowNetS stacked architectures..*

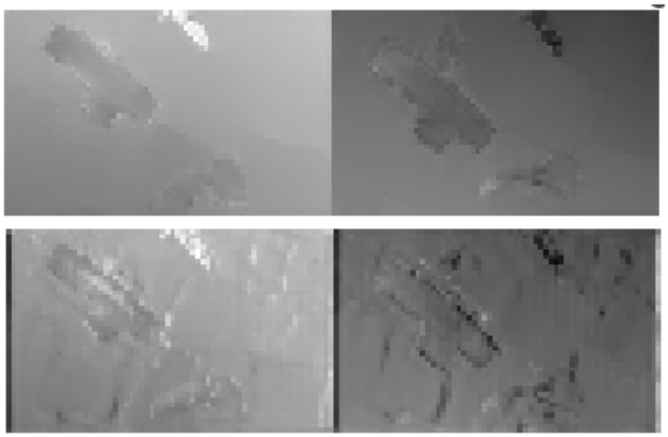

Figure 19: *Same predictions as in figure 18 on smaller resolution for the u-component of flow prediction.*

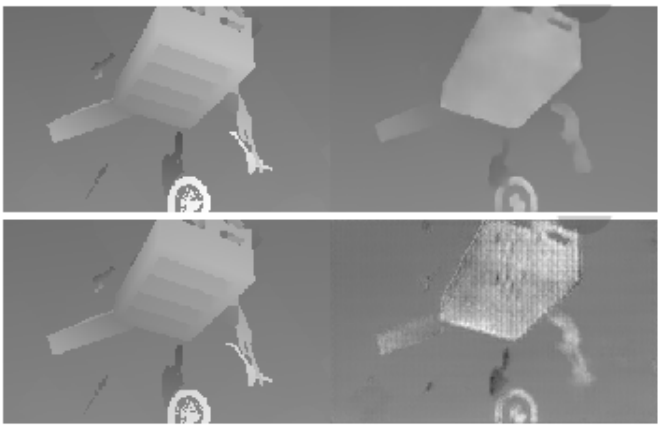

Figure 20: *Prediction of FlowNetGAN evolution 1(bottom) and evolution 2(top). The left image represents the ground truth flow while the right image represents the predicted flow. The mentioned images are the u-components of the flow values.*

also maximizing one of the losses which somehow affects the performance of the other losses being minimized. While minimizing the adversarial loss the network is trying to learn a distribution over the given dataset, hence it results in more distorted images with time.

As seen in figure 20, the image is smoothed out after passing the prediction of the first image through the second network.

We also included all the unsupervised and supervised losses throughout our training for the first evolution while only SIG loss and EPE loss for the second evolution as we tried in the stacked version of FlowNet.

Looking back at table 4 we notice that FlowNetGAN was able to achieve the lowest *RMSE* on the unseen Middlebury 2003, 2005 datasets. Although there are some images in which the FlowNetGAN could not perform better than FlowNetS stacked but with some images there is a high difference for e.g in Dolls the minimum error achieved by FlowNetS stacked is 7.26 while FlowNetGAN achieves 3.37. This completely shows the potential of using a discriminator network which has better capabilities for the network to produce better predictions for unseen data.

Furthermore, we can also notice the performance of FlowNetGAN on flying, monkaa and PTB datasets as mentioned in table 5. Although the performance is somewhat similar with a slight improvement in accuracy for flying and PTB

datasets, where PTB dataset was trained with only unsupervised losses along with the adversarial loss.

5.4 FlowNetS With Losses Terms

We also wanted to see how the different unsupervised and supervised losses affect the performance of our networks. For this we use a combination of losses and represent them as FlowNetS-EPE, FlowNetS-EPE-PC, FlowNetS-EPE-FB, where EPE represents the network trained with only endpoint error, EPE-PC represents the network trained with a combination of endpoint error and photoconsistency error(PC) and EPE-FB represents the network trained with the combination of endpoint error and forward backward loss(FB).

5.5 Results

Table 5 represents the performance of each network on the datasets they were trained on. For the never seen datasets we provide the results in table 4 for the FlowNetS, FlowNetS-ST and FlowNetGAN.

FlowNetS* which represents the normal FlowNetS trained for optical flow without depth information performs worse than the other FlowNetS which is trained with depth information as an input. Although, this helps us make better predictions, we can further improve our results with FlowNetS-ST. As discussed earlier, this is a tradeoff between prediction, time and accuracy.

It can be observed clearly that the stacked FlowNet version(FlowNetS-ST) outperforms all the other methods in table 5. Although the difference is minor but we can observe that the unsupervised losses does make a difference when training FlowNet. The photoconsistency loss helps us achieve smaller test loss compared to forward backward loss, these losses were assigned a very low importance, 500 times less than the endpoint error. Hence trying out different importance combinations for this task might be a good experiment in itself for future.

It can also be observed that the optical flow version performs the worse among all the methods due to lack of depth information for the input images. Hence, it can be concluded that depth information (whenever available) will always help in prediction of flow values.

Networks	Flying-AEPE	Monkaa-AEPE
FlowNetS*	0.053	0.040
FlowNetS	0.027	0.011
FlowNetS-ST	0.022	0.010
FlowNetS-EPE	0.027	0.039
FlowNetS-EPE-PC	0.024	0.013
FlowNetS-EPE-FB	0.027	0.030
FlowNetS-GAN	0.025	0.010

Table 5: *Comparison between different network architectures and their achieved averaged endpoint errors on different datasets. The FlowNetS* represents the network trained for optical flow without depth. FlowNetS-ST represents two FlowNetS stacked, while FB, EPE, PC reprsents the network training with forward backward loss, endpoint error loss, photoconsistency loss. FlowNetS-GAN represents the GAN version of FlowNetS with all the other losses included.*

Networks	Flying-AFB	Monkaa-AFB	PTB-AFB
FlowNetS*	0.043	0.036	0.0066
FlowNetS	0.016	0.0078	0.0017
FlowNetS-ST	0.010	0.0063	0.0018
FlowNetS-EPE-FB	0.018	0.0080	0.0024
FlowNetS-GAN	0.018	0.0083	0.0014

Table 6: *Same as table 5 with unsupervised average forward backward(FB) loss for comparisons with PTB (since it's without ground truth). We only consider the networks we trained with FB loss.*

Networks	Flying-APC	Monkaa-APC	PTB-APC
FlowNetS*	0.104	0.069	0.089
FlowNetS	0.098	0.045	0.045
FlowNetS-ST	0.090	0.044	0.040
FlowNetS-EPE-PC	0.094	0.050	0.042
FlowNetS-GAN	0.086	0.042	0.039

Table 7: *Same as table 6 with unsupervised average photoconsistency(PC) loss for comparisons with PTB (since it's without ground truth). We only consider the networks we trained with PC loss.*

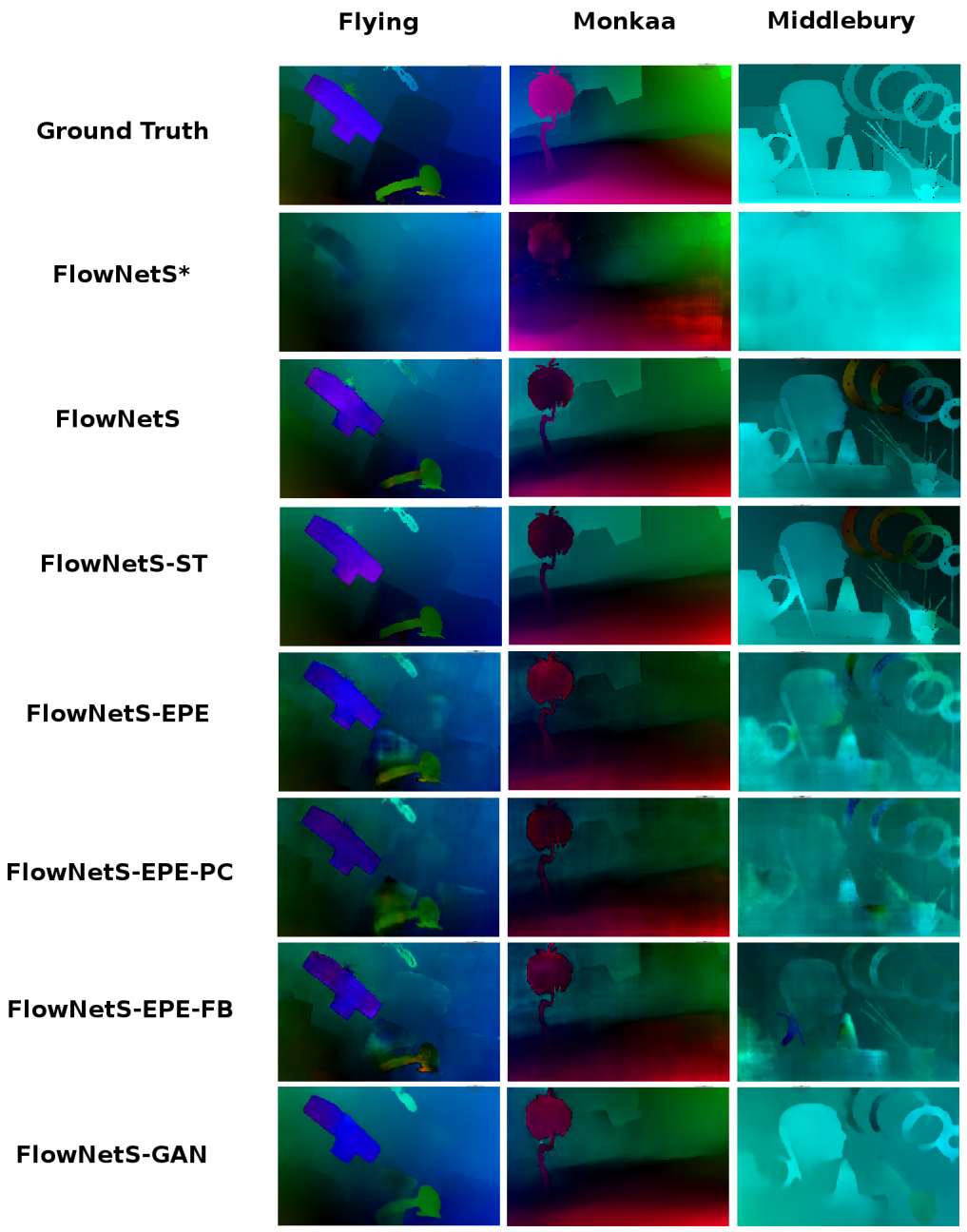

Figure 21: *Comparisons between predicted flows of different network architectures and combination of losses on different datasets. Note that Middlebury dataset was only used for testing*

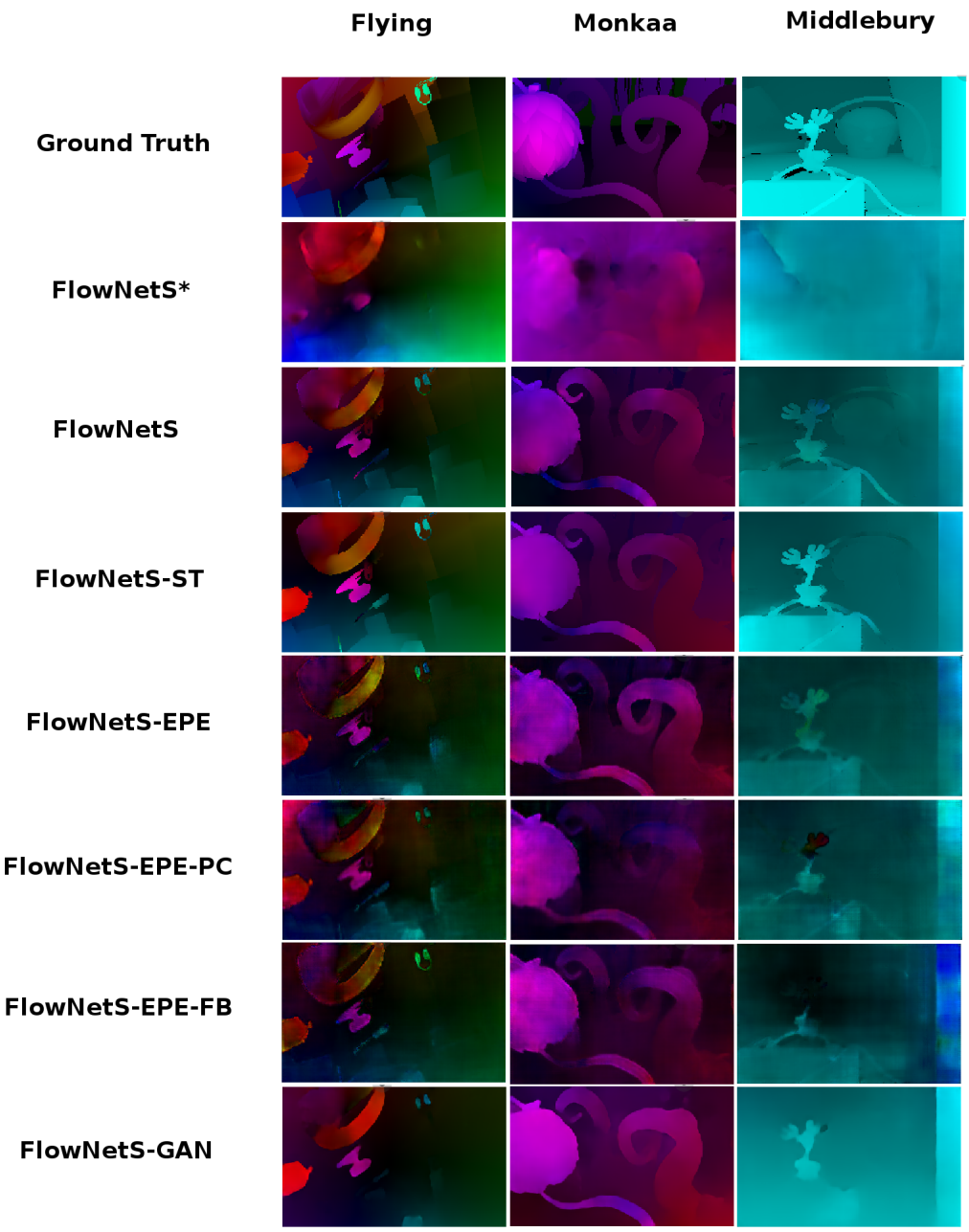

Figure 22: *Another example of comparisons between predicted flows of different network architectures and combination of losses on different datasets. Note that Middlebury dataset was only used for testing.*

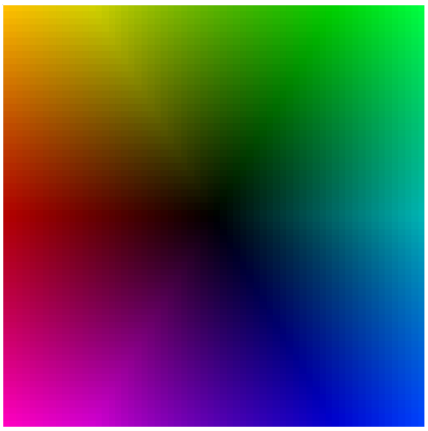

Figure 23: *Color legend for figure 21. Each color represents the direction of the flow field.*

In order to make a comparison of the results with the PTB dataset, we also show results for the forward backward loss on the test data shown in table 6. We only used the network architectures which were trained on the FB loss. FlowNetS-GAN outperforms other networks on the PTB dataset. This is due to the ability of the GANs to learn complex distributions and since PTB is a more complex dataset with sparse values this can be one of the reasons for the network to be performing much better on PTB compared to synthetic datasets while the results on other datasets for FlowNetGAN are also acceptable.

Furthermore, we also tested the datasets with the photoconsistency loss which gives us close to similar results as in table 7 with respect to the performance of different networks. The FlowNetS-GAN once again turns out to be a good competitor of FlowNetS-ST. While FlowNetS* performs the worst.

Finally we provide different examples of flow predictions for the flying, monkaa and Middlebury datasets in figures 21 and 22. Each color represents the direction of the flow field in the images for which the legend is provided in figure 23. The predictions for FlowNetS, FlowNetS-ST and FlowNetS-GAN look closer to the ground truth flow provided at the top while the predictions for other flow values are slightly distorted due to relatively less better performance. The FlowNetS* predictions are worst due to less information(excluding depth) provided to the networks.

Chapter 6

Conclusion

In this masters thesis, we discussed in detail how we can use different unsupervised losses to train a SceneFlowNet using the existing FlowNetS architecture. We presented multiple approaches with respect to the network architecture and different unsupervised losses that can be used to train better networks.

Our main approaches towards network architectures include FlowNet as a GAN, a simple FlowNetS and a stacked version of FlowNetS. In FlowNetS as a GAN we presented our FlowNet as a generative adversarial network along with different unsupervised losses.

Although, FlowNetGAN is an idea coming through the latest updates in the field of deep learning, we still believe this can be improved further in future when trying to combine it with FlowNet. Since GANs are not that easy to train especially when it comes to maintaining the balance between the two generator and discriminator networks, the main target when training GANs along with FlowNet needs to be the right balance of loss weighting for the gradients to flow and maintain the correct size of the networks.

We also used a simple FlowNetS in which we implemented different unsupervised losses for comparisons. We concluded that using unsupervised losses with respect to more realistic datasets that do not provide ground truth values can be useful for better accuracy. It was also concluded, that although using these unsupervised losses is of great help, one cannot completely depend on them, hence it's always better to use supervised losses when possible.

We also used a stacked version of FlowNetS with multiple networks one after the other to conclude this can help us further to increase our network accuracy

with unsupervised losses. The visual and analytical differences were quite clear from the experiments mentioned with the tradeoff between time and accuracy.

In future, different methods needs to be applied to learn how we can manage to train the network in a completely unsupervised manner. Another interesting addition would be to run the experiments with higher number of stacked networks to see how the network behaves when it comes to training with depth images.

Bibliography

[1] S. Vedula, P. Rander, R. Collins, and T. Kanade, "Three-dimensional scene flow," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 475–480, March 2005.

[2] M. Jaimez, M. Souiai, J. Gonzalez-Jimenez, and D. Cremers, "A primal-dual framework for real-time dense rgb-d scene flow," pp. 98–104, May 2015.

[3] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," pp. 3061–3070, June 2015.

[4] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," *CoRR*, vol. abs/1504.06852, 2015.

[5] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[7] P. Wang, W. Li, Z. Gao, Y. Zhang, C. Tang, and P. Ogunbona, "Scene flow to action map: A new representation for rgb-d based action recognition with convolutional neural networks," 02 2017.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014.

[9] S. Song and J. Xiao, "Tracking revisited using rgbd camera: Unified benchmark and baselines," pp. 233–240, Dec 2013.

[10] J. McCormac, A. Handa, S. Leutenegger, and A. J.Davison, "Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation?," 2017.

[11] J. Sung, C. Ponce, B. Selman, and A. Saxena, "Unstructured human activity detection from rgbd images," pp. 842–849, May 2012.

[12] J. Sung, C. Ponce, B. Selman, and A. Saxena, "Human activity detection from rgbd images," pp. 47–55, 2011.

[13] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," *CoRR*, vol. abs/1512.02134, 2015.

[14] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," pp. 611–625, 2012.

[15] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," vol. 1, pp. I–195–I–202 vol.1, June 2003.

[16] D. Scharstein and C. Pal, "Learning conditional random fields for stereo," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, June 2007.

[17] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *ArXiv e-prints*, June 2014.

[18] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2015.

[19] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014.

[20] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," *CoRR*, vol. abs/1606.03657, 2016.

[21] J. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, "Generative visual manipulation on the natural image manifold," *CoRR*, vol. abs/1609.03552, 2016.

[22] M. Liu, T. Breuel, and J. Kautz, "Unsupervised image-to-image translation networks," *CoRR*, vol. abs/1703.00848, 2017.

[23] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *CoRR*, vol. abs/1606.03498, 2016.

[24] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox, "Demon: Depth and motion network for learning monocular stereo," *CoRR*, vol. abs/1612.02401, 2016.