Normal map prediction from light field data through deep learning

Matthijs van Veen
S0133299

Preface

Since the resurrection of VR a few years ago, one of the things that caught my interest is that of light fields which, at least in theory, allow for the capture and playback of captured scenes with 6 degrees of freedom in VR. Although this promise is still long off due to technical constraints, light fields can also be used for other applications like predicting depth maps and scene dissection. This research then would allow me to explore this topic and learn more about light fields in general. I must say that although it took me longer than expected, it was an interesting experience that brought new insights and knowledge of a topic of great personal interest.

In this preface I would like to thank all people who supported me in completing this research. First I would like to thank my employer, Saxion University of Applied Sciences for allowing me the opportunity to pursue my interest. Secondly I would like to thank my graduation coach Gwenn Englebienne for the support and feedback during the process. I must honestly say that I think I could not have finished it without him and wished we had met earlier during my studies. Thirdly I would like to thank NEP and especially Robert van Merweland for hosting me during a part of my research. Lastly I would like to thank my beautiful and lovely wife Paula, without whose support this would never have been possible.

Enschede, 31-12-2019

Summary

Recently the increasing use of augmented reality has demanded for a more realistic integration of synthetic and real world images. One of the ways in which this integration could be improved is by attaining a normal map of the real world which in turn theoretically allows for relighting and physic interaction with computer generated objects. This research aims to explore if deep learning in combination with a light field data can be used to solve this problem by predicting normal maps directly from RGB inputs.

To do this two methods for the construction normal maps from RGB inputs have been researched as a baseline. Since normal maps can only be computed from depth maps this is done in two steps. First two methods to compute normal maps from depth data have been implemented and evaluated. Secondly two methods to compute depth maps from RGB light field data have been implemented and evaluated.

In order to see if machine learning can be utilized for the prediction of normal maps, a dataset has been created. For this purpose 8 synthetic scenes have been built with a 3d programme. From these scenes 288 light fields have been rendered. Image enhancement methods through the use of gamma, saturation and value adjustments as well as flipping and rotating are used to scale up this data set to 24624 light fields for training (6 scenes), 4104 for validation (1 scene) and 1 for testing (from a novel scene).

To determine the deep learning architecture nine experiments have been done to test a large number of network architectures. In these experiments various parameters have been tested as well as different activation and loss functions. Based on these experiments a final architecture has been chosen.

In order to evaluate the quality of the network the predicted results are compared with two other methods mentioned above. Here two depth maps are made using stereoscopy and EPInet. Based on these depth maps the Hinterstoisser method is applied to create two normal maps. These maps are compared with the prediction result visually and using a metric called mean of difference in angles (MDA).

For the comparison it can be concluded that although the network architecture used in this research produces better normal maps than the combined methods above, it is still far from the ground truth. This would make them difficult to use in any real world application. For improving upon this result it is suggested to look at a higher quality synthetic dataset, possible precomputations and parameter tweaking of both the neural network and light field camera setup.

Table of contents

1. Introduction

Given the recent progression in graphics hardware, computer vision and machine learning, augmented reality (AR) has become a part of our everyday life. Applications for this technology can be found in various areas and on various devices. Examples include AR face augmentation for mobile (Snapchat, Snap Inc.), television shows that mix real people and virtual sets (Ziggo Sports, NEP) and remote assist applications for wearable AR glasses (Remote Assist, Microsoft).

Although the applications available have various uses and run on different devices, all share the same underlying computer vision problems. Here a set of algorithms try to understand the real world through an input device. One of these problems involves the understanding of the geometric properties of the real world. If a correct representation of the scene can be known, virtual objects can be scaled and mixed accordingly. Another problem that needs to be solved involves the understanding of the physical aspects of the world. Here one would like to determine properties of materials, which in turn allows for realistic relighting and physics interaction.

To provide input for these algorithms, various input devices are currently being used. Ranging from simple mobile cameras to complex solutions like laser scanner and light field cameras. Complex input devices like laser scanners and light field cameras provide rich structural data that allow for slower but more precise scene reconstruction. On the other hand simple input system allow for quick but less accurate reconstruction.

Recently machine learning in combination with light field data resulted in a huge step forward in determining the geometric quantities of real-world scenes. By using a convolutional network Shin, Jeon, Yoon, So Kweon, and Joo Kim (2018) computed reasonably accurate disparity maps with little trade-off with respect to speed. Given this result it seems obvious to explore if these same algorithms can also be used to compute the physical quantities of scenes. Especially the computation of normal maps would be interesting, since this would allow for the additional lighting to be added to real world scenes.

The report consist of the following sections. In section two the problem will be described and analysed. Here several methods for the creation of normal maps are explored. In section three the research questions and its sub questions are laid out. In section four the research methodology is described. Section five describes the results. At last section six and seven respectively describe the conclusion and recommendations.

2. Problem definition

Given the need for more realistic and precise augmented reality applications, the understanding of the physical reality of the world allows for the addition of computer-generated lights.

In order to add a virtual light to a real-world scene realistically, two problems need to be solved. First a normal for each pixel needs to be determined. Here the normal represents the orientation of the pixel in the real world. A correct normal is required to compute the reflected lighting towards the capture device. The second property required is that of the reflective properties of the material captured by the pixel. Both problems are challenging to solve and this research will focus on the former and not on the latter.

2.1. Problem analysis

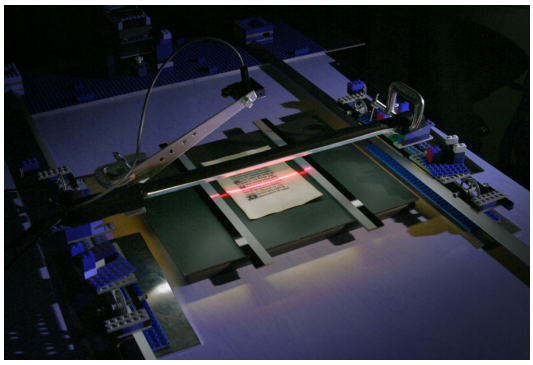
Computing normal maps from real world scenes is a challenging and difficult task. Various methods exist that either use RGB image data or a depth measurements. Below an overview of the current state-of-the art of how normal maps can be computer through hardware device and the utilization of depth maps.

2.1.1. RGB image based normal map creation methods

Methods that compute normal maps from RGB data usually employ some kind of hardware device in order to control the environment, lighting or both.

One such method is that of Gardner, Tchou, Hawkins, and Debevec (2003). Their solution uses a scanning device as depicted in Figure 1 that is drawn on a semi flat surface. Through measuring the change in light, different material properties can be computed including the normal map.

Some other methods utilise the UCS light stage depicted in Figure 2 that requires a person to sit inside a sphere with cameras. These methods (Ma et al., 2007; Weyrich et al., 2006) all use multiple photographs where a person or object is relit from multiple angles. The problem however with the methods mentioned above is that they are very dependent on complex hardware, controlled environments and they are limited to objects that can be placed in their respective hardware setups.



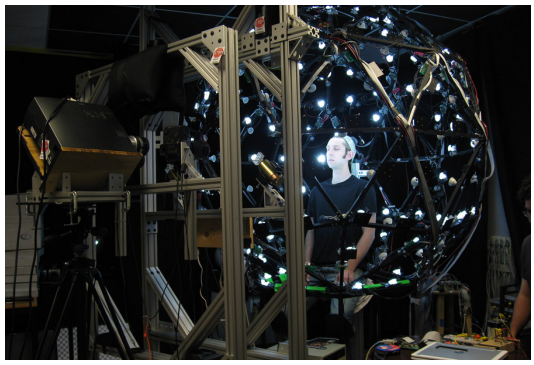


Figure 1: The Linear Light Source Apparatus of Gardner et al. (2003).

Figure 2: Newer version of the UCS light stage used in the research of Ma et al. (2007).

2.1.2. Depth based normal map creation methods

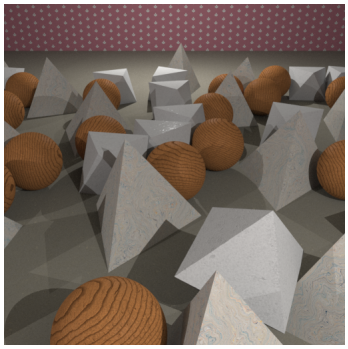
As mentioned earlier, other methods utilise a depth map to compute normals. These methods can be used when a depth maps is available. A simple method is to calculate normal based on local x and y derivatives through formula 1 and 2. Then by using the cross product the normal direction can be computed through 3.

$$\frac{dz}{dx} = \frac{D(x+1,y) - D(x-1,y)}{2} \tag{1}$$

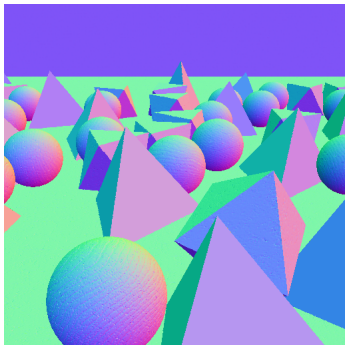
$$\frac{dz}{dy} = \frac{D(x,y+1) - D(x,y-1)}{2} \tag{2}$$

$$n = \begin{bmatrix} 1 \\ 0 \\ dz/dx \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ dz/dy \end{bmatrix} \tag{3}$$

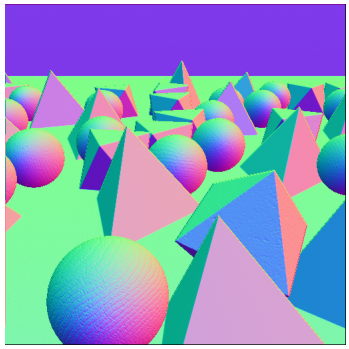
To test the accuracy of this method an implementation has been made and compared with a ground truth depth map. As depicted in Figure 3, this method produces correct normal on continued surfaces. However, normals are incorrectly calculated around sharp edges and discontinuities. This can be seen in the third image where the absolute difference of the two normal maps is shown.



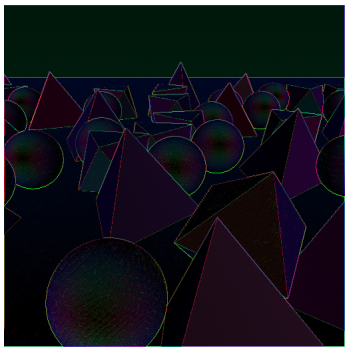
(a)



(b)



(c)

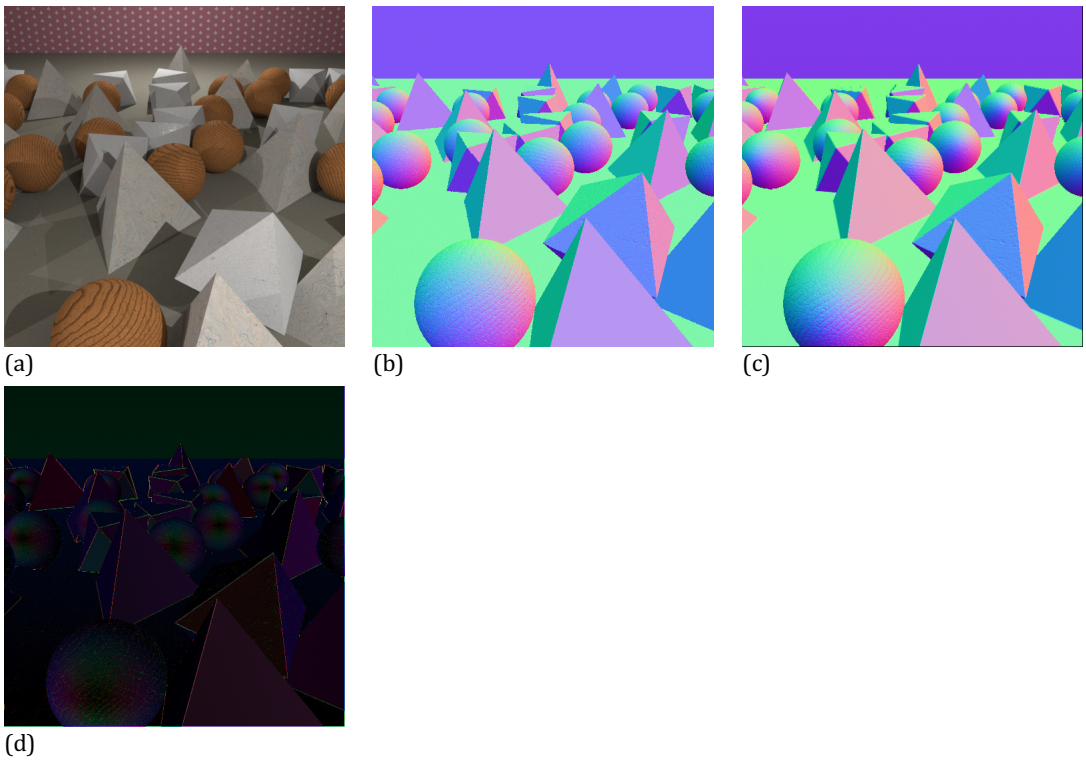


(d)

Figure 3: Comparison of computed normal map through image derivatives with a ground truth. (a) original RGB image. (b) Ground truth, (c) Calculated normal based on image derivatives, (d) Absolute difference between the computed normal map and the ground truth.

Another more complex method is that of Hinterstoisser et al. (2011). This solution computes a normal based on its 8 surrounding neighbours. By using a threshold value, neighbours with a large depth disparity are discarded when computing the normal.

An implementation of Hinterstoisser has been made and its results are depicted in Figure 4. Here again a ground truth depth map has been used to compute a normal map. A threshold value is chosen through trial and error. As can be seen the results on continued surfaces and sharp edges are comparable to that of the method above. However, when it comes to discontinuities, improvements are visible.



(a) (b) (c)

(d)

Figure 4 Comparison of a computed normal map with the Hinterstoisser method and a ground truth. A threshold value of 0.05 is used to ignore depth disparities. (a) original RGB image. (b) Ground truth, (c) Calculated normal on its eight surrounding neighbours, (d) Absolute difference between the computed normal map and the ground truth.

In Table 1 an overview of both methods and their performance in different scene properties.

Table 1: Overview of normal map from depth map methods and their strong and weak points.

	Continues surfaces	**Discontinuities**	**Edges**
Image derivatives	Accurate	Inaccurate	Inaccurate
Hinterstoisser	Accurate	Accurate but only for the right threshold.	Inaccurate

2.1.3. Normal maps from light field camera's

Both implementations described above use ground truth depth maps which are computer generated by a 3D modelling programme. However, when utilising real world scenes, these ground truth depth maps are not available. A normal map has to be based on a depth map, which itself has to be generated based on RGB inputs.

However, looking at the current state of depth map prediction from RGB inputs, this itself is not a solved issue. Monocular and stereo based methods are poor in performance. This can be seen in Figure 5 where a stereo matching algorithm is used to compute a depth and normal map for a scene. Both normal map generation methods result in noise and incomplete normal maps.

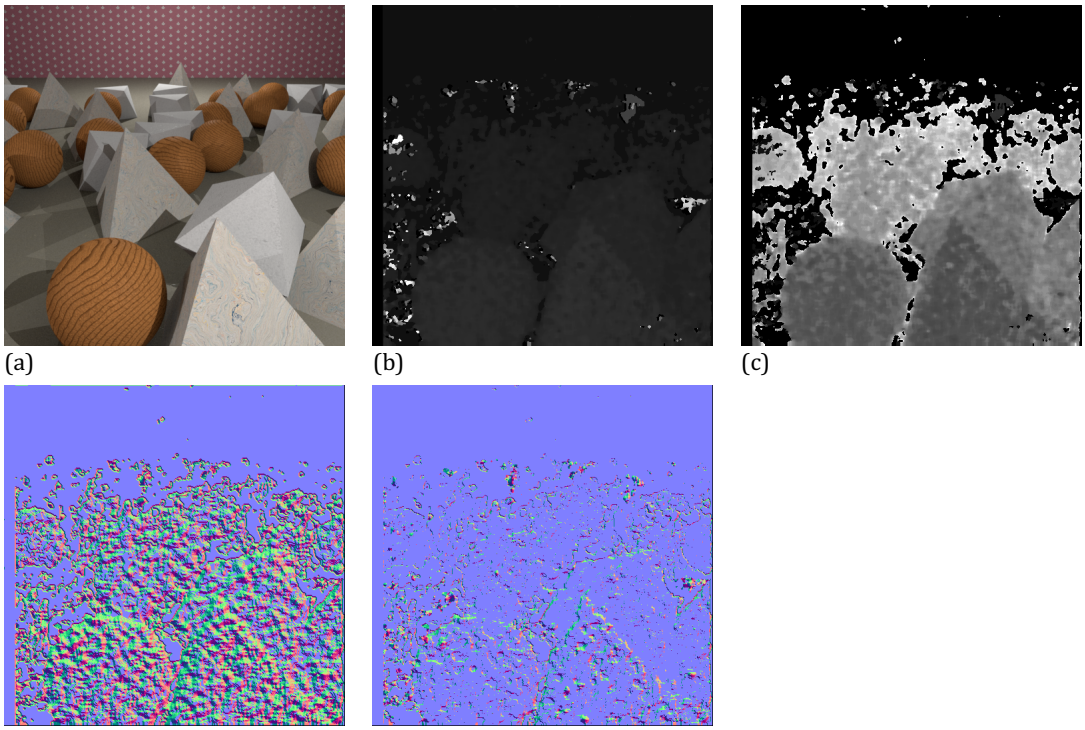
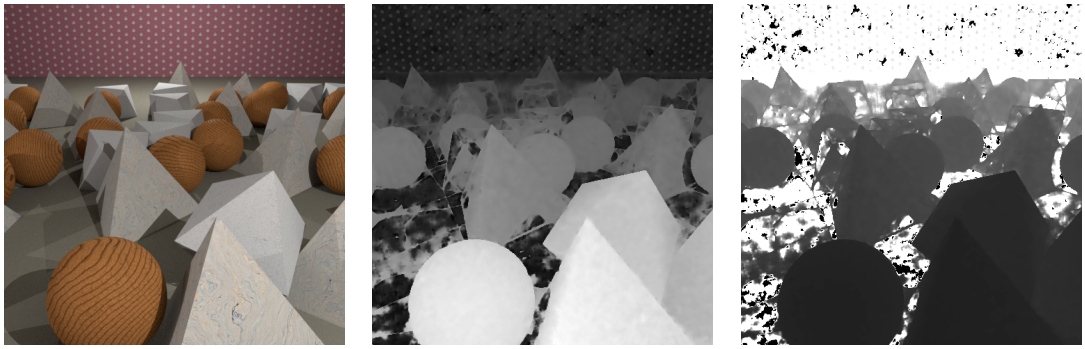


Figure 5: Normal maps computed based on a stereo matching algorithm. (a) Right input for the algorithm. (b) disparity map generated by stereo matching. (c) Depth map computed from the disparity map. (d) Normal map computed with image gradients. (e) Normal map generated by the Hinterstoisser method.

Recently depth maps from light field data algorithms in combination with deep learning have made huge accuracy improvements ("4D Light Field Benchmark," 2019). Here light fields provide rich structural data that can be used to more accurately predict the depth of a scene.

One such method is that of Shin et al. (2018) which uses a 5 by 5 light field camera array. The algorithm utilises deep learning to find a mapping from RGB to disparity. Shown in Figure 6 this method generates a depth map based on 17 RGB inputs. Based on these depths a normal map is computed both by the image derivatives and the Hinterstoisser method.


(a) (b) (c)

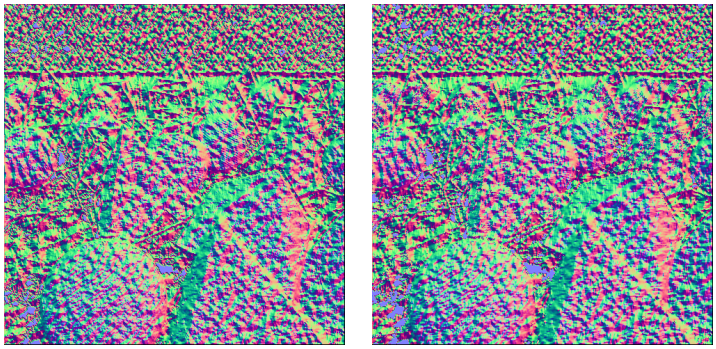

(d) (e)

Figure 6: Normal maps indirectly computed from a light field image. (a) The centre image of the 5x5 cross light field input. (b) disparity map generated by the EPInet solution. (c) Depth map computed from the disparity map. (d) Normal map computed with image gradients. (e) Normal map generated by the Hinterstoisser method.

2.1.4. Conclusion

Computing normal maps from depth maps remains an unsolved problem. Even in ideal situations depth disparities and sharp edges will result in incorrect normals. However, when using computed and predicted depth maps the problem becomes even more problematic due to noisy and incorrect depth maps.

3. Research Question

Given recent advances in using machine learning for depth map prediction from light fields, this research investigates if this same method can be used to directly predict normal maps. By training an neural network to predict normals from RGB inputs, errors propagated from depth map generation might be avoided. This leads to the following research question:

How can machine learning be used to compute normal maps based on light field data and how does this method compare with other normal map generation methods based on depth maps?

With the following sub questions:

1. *How can machine learning be used to predict normal maps?*
2. *How does a normal map generated with machine learning compare with other depth based normal map generation methods based?*

4. Research Methodology

In order to answer the research question an experiment will be conducted. Here a learning algorithm will be trained on synthetic lights fields and ground truth depth maps.

After the network is trained several scenes will be analysed and compared to already existing normal map creation methods.

4.1. Scene setup

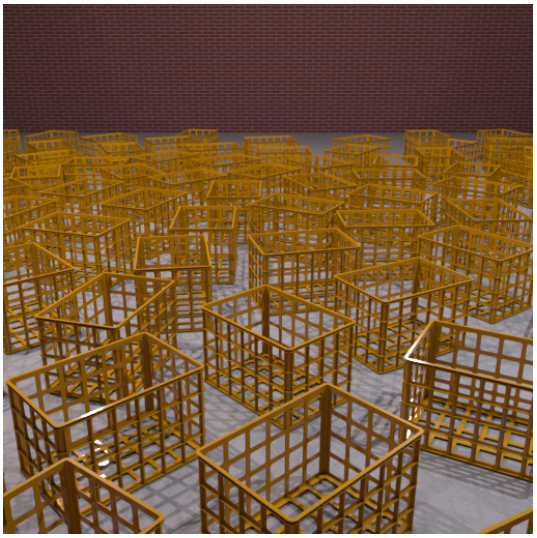
In order to train the neural network for the prediction of normal maps 8 synthetic 3D scenes have been created. Here the computer generated environment is chosen because it allows for the generation of perfect ground truths. Furthermore, according to the work of Lee and Moloney (2017) and Prakash et al. (2018) synthetic images would be a valid alternatives real world data, hereby not reducing training quality or prediction on real world examples.

All scenes are situated in a closed indoor environment of different sizes. Within these environments a range of objects have been added. This includes geometric primitives and objects like doors, tables, statues, etc. A wide range of synthetic materials have been created to mimic real world materials. Materials are all procedurally generated and physically based. Displacement maps are used to add detailed surface variations.

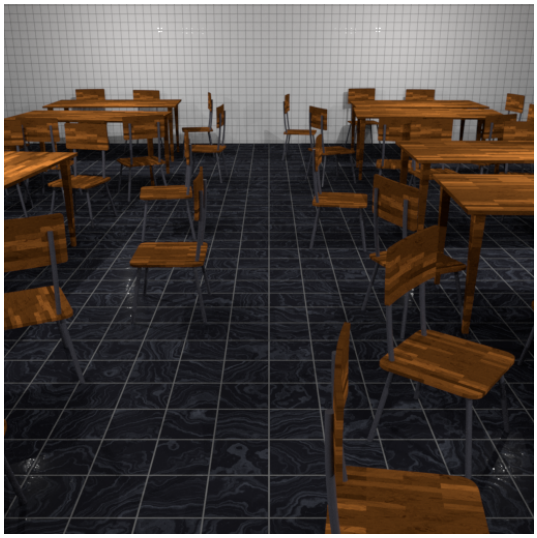
Due to the possible complexity of real-world scenes the synthetic environments have been limited to not include transparent, highly reflective or anisotropic materials. This is considered out of the range of the research and will also limit the data set needed for the learning algorithm.

Furthermore, lights added to the scenes are placed in such a way that the light source is out of the camera viewport and does not show up in the light field renders. Also lighting and materials are tweaked to avoid over and under exposure and strong reflection. All scenes are created in the Houdini software environment. In Figure 7 a render from each of the created scenes can be seen.

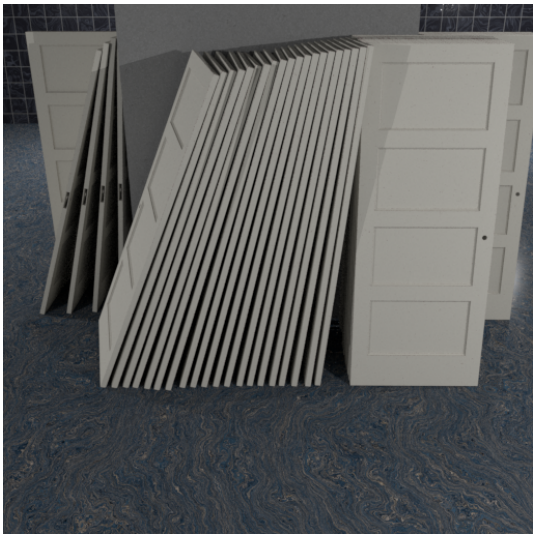
(a) Scene 1



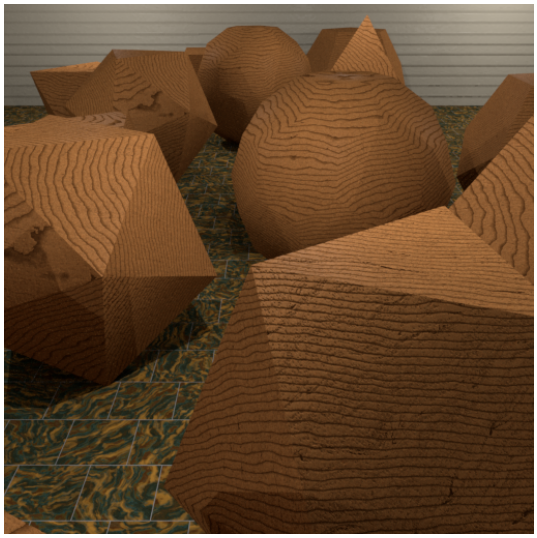
(b) Scene 2



(c) Scene 3

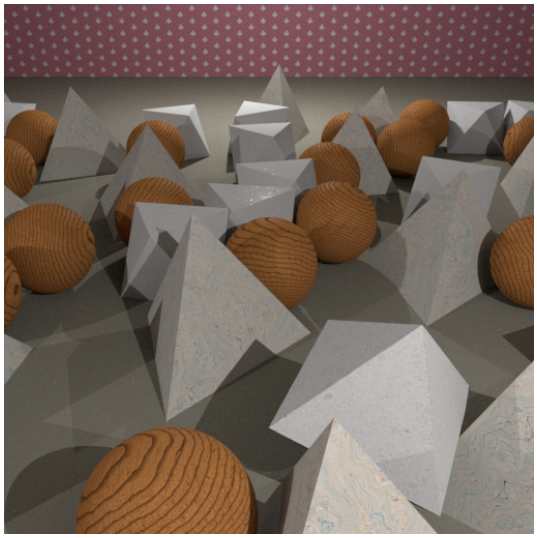


(d) Scene 4



(e) Scene 5

(f) Scene 6



(g) Scene 7

(h) Scene 8

Figure 7: Overview of the 8 scenes created in Houdini. For each scene 36 light fields have been rendered.

4.2. Synthetic light field camera

For the light field camera rig configuration, a 3 by 3 rig is chosen. This decision has two reasons. First, a smaller rig would allow for cheaper and easier camera construction when the result would need to be tested with real hardware. Secondly, a smaller rig would allow for quicker computation times. This would make real-time procession possible in the near future.

Although the distance between cameras can be varied, for this experiment a fixed camera baseline of 9 mm is chosen. Hereby mimicking a light field camera system that has its camera in a close setup. This other camera parameters can be found in the table below.

Table 2: Light field camera parameters.

Focal range	50 mm
Resolution	512 x 512 pixels
Aperture	41.4214 mm
Focus distance	5 m
F-stop	5.6

To create a large data set for training, for each scene 36, 3 by 3 light fields are rendered at a resolution of 512 by 512 pixels. This is done by placing the virtual light field camera rig inside each scene and rotate it in steps of 10° inside the environment. Each step a light field is rendered including a depth and normal map for each camera. In Figure 8 a screenshot of scene 3 with a light field camera pointing at its centre.

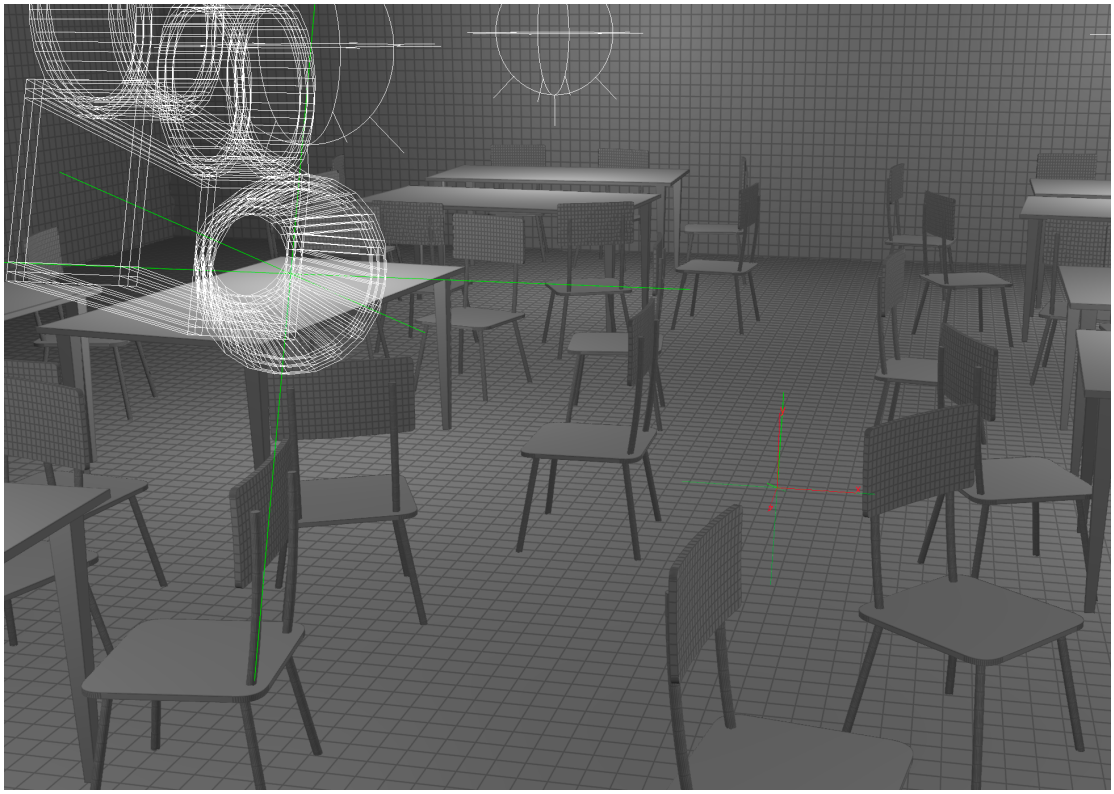


Figure 8: Scene setup in Houdini. A light field camera can be seen in the top left corner pointing at the centre of the room. The light field camera can be rotated around the centre point to generate new light field renders.

4.3. Experiments setup

For the learning algorithm the EPInet (Shin et al., 2018) solution is used as a starting point. This framework uses grayscale images of a light field to predict a disparity map. The stacked horizontal, vertical and diagonal cross sections of a light field are used as an input as depicted in Figure 9. By convoluting over these stacked inputs the neural network tries to learn the depth of a scene by analysing the Epipolar Plane Images (hence EPInet).

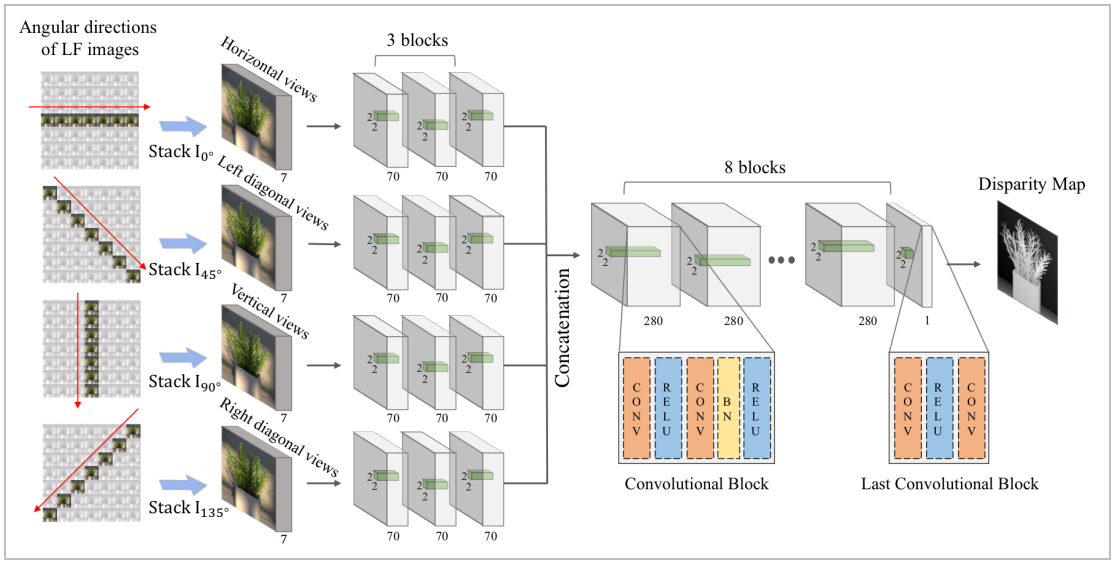


Figure 9: EPInet architecture (Shin et al., 2018).

4.4. Training setup and data enhancement

From the 8 scenes described in 4.3. Experiments setup, 7 are used for machine learning. 1 scene (scene 8) is used for the evaluation of the results in chapter 6. For all experiments 7-fold cross validation is used to determine the mean and standard deviation for each network architecture. In each fold the data is split between a training and validation data sets. For all experiments (with the exception of experiment 5) 6 light field scenes are used for training and one for validation.

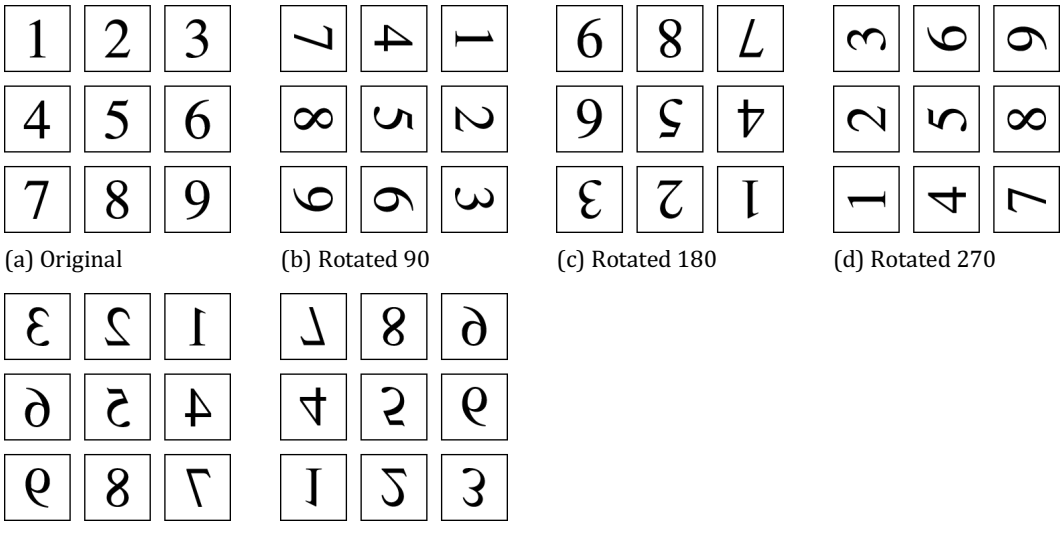
For the software pipeline Keras (Tensorflow) is used. Data generators are used for generating and enhancing light field data. During training Keras *EarlyStopping* (min-delta = 0) is used to determine to maximum numbers of epochs needed. In most experiments a T-test is used to see if the results are significant.

For all experiments data enhancement methods are used to enlarge the datasets. This is done by adjusting the gamma, saturation and value changed in various degrees. In Table 3 the three methods, their minimum, maximum values and step size are given. Using these enhancement methods increases the dataset 19 times resulting in a dataset of 4104 light fields.

Table 3: Light field enhancement methods and values.

Method	Minimum	Maximum	Step
Gamma	1.6	2.5	0.1
Saturation	-0.2	0.3	0.1
Value	-0.2	0.3	0.1

In the last experiment light fields are also flipped over the x and y axis as well as rotated 90, 180 and 270 degrees to increase the number of inputs even further to 24624. As depicted in Figure 10 this not only includes rotating and flipping of images but updating the configuration as well.



(a) Original (b) Rotated 90 (c) Rotated 180 (d) Rotated 270

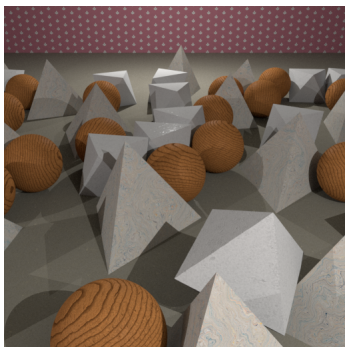
(e) Flipped X (f) Flipped Y

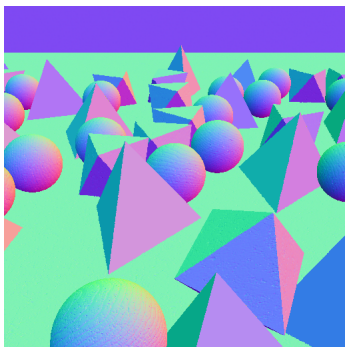
Figure 10: Overview of rotation and flipping of light field for data enhancement.

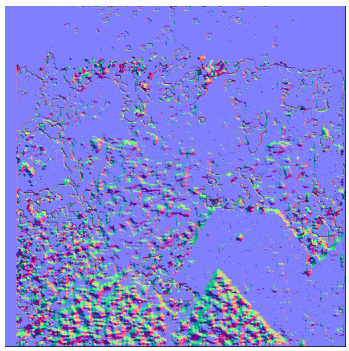
5. Experiments

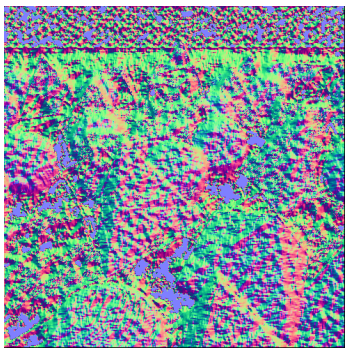
5.0. Experiment 0: Baseline measurement

To measure the result of any solution found in the experiments below it is necessary to compare the result with the state of the art from chapter 2. Here the mean of difference in angles (MDA) is used as a metric. To establish a baseline two light fields from scene 7 and 8 are used (rotation 0). In Figure 11 an overview of the two light fields, their ground truths, normal maps and computed normal maps from stereo depth and EPInet, combined with the Hinterstoisser method.

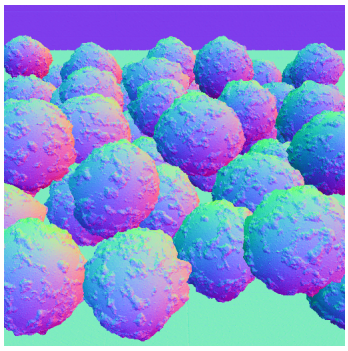

(a)

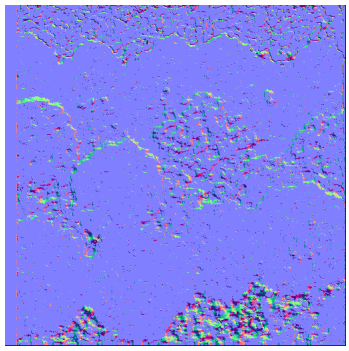

(b)

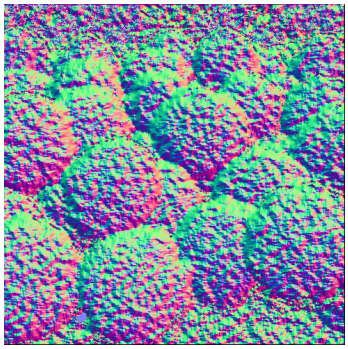

(c)


(d)


(e)


(f)


(g)



(h)

Figure 11: Inputs and normal maps for two light fields scenes. (a) RGB middle input image of the light field, (b) ground truth normal map, (c) normal map generated by Hinterstoisser and naive stereo depth, (d) normal map generated by Hinterstoisser and EPInet (5x5 input). (e-h) Similar to *a-e* for scene 8.

For the creation of the stereo depth map the left and middle image from the middle row of a 3x3 light field is used (see Figure 5 for a depth map created through this method). The StereoSGBM algorithm (number of disparities = 16, block size = 9) from OpenCV is used to compute a disparity map which in turn is computer to a depth map using the camera rigs parameters.

Similarly EPInet is used to create a disparity map based on a 5x5 light field input (see Figure 6 for a depth map created through this method). Based on this map a depth map is created with the camera rig parameter. Here a similar rig setup is used as that in the experiments below; with the difference being the number of cameras (25 instead of 9). As described above the Hinterstoisser method is then used to create the normal maps in Figure 11.

Based on the computed normal maps above the MDA from formula 4 is used to establish the difference between the computed and ground truth normal maps. An overview of the MDA for both methods can be found in Table 4. The computed images above and the table below shows that currently the computed normals are noisy and quit far from the ground truth.

$$Mean\ of\ different\ angles = \frac{1}{n}\sum_{i}^{n} \cos^{-1}\left(\frac{a(i)\cdot b(i)}{|a(i)|\cdot|b(i)|}\right) \qquad (4)$$

Table 4: Baseline measurement of the mean of different angles for two state of the art workflows.

Normal map	**Mean Different Angle**
Scene 7	
Stereo + Hinterstoisser	60°
EPInet + Hinterstoisser	73°
Scene 8	
Stereo + Hinterstoisser	52°
EPInet + Hinterstoisser	66°

5.1. Experiment 1: Normal only versus normal and depth

5.1.1. Experiment setup

The first experiment is done to determine if there is a difference in training only on normal map data or a combination of normal and disparity data.

As inputs the 512x512 images from the cross sections of the 3 by 3 rendered light fields are used. The RGB images are all converted to greyscale using formula 5 as defined by the CCIR 601 standard. The inputs are then stacked according to the configuration of the EPInet architecture.

$$Grey = 0.2989 * R + 0.5870 * G + 0.1140 * B \quad (5)$$

For the outputs the normals are converted to normal maps space. Here the X and Y coordinates of the normals are mapped from their original range of -1 to 1, to a range between 0 and 1. The Z coordinate which always points towards the camera is mapped to a range between 0.5 and 1. To create the disparity map the depth value for each pixel is converted to a disparity value using the parameters of the light field rig through formula 6.

$$disparity = \frac{(baseline * focal)}{depth} \quad (6)$$

For trai ning 2x2 kernel filters are used. The number of filters is set at 45, the maximum that is allowed for training on the GPU (2080ti, 8Gb). In accordance with EPInet, MSE and Relu are used as loss and activation function as well the root means squared propagation optimizer with a learning rate of 0.00001. An overview of the networks used during training can be found in Table 5.

For the training a maximum of two epochs are used to evaluate the difference. This to reduce training times. 7-fold cross validation and data enhancement are used as described in the previous chapter.

Table 5: Two networks used to see testing training loss for different outputs.

Input	Activation	Filter size	# Filters	Colour mode	Dropout	Output
Cross	Relu	2x2	45	Grey	No	Normal Disparity
Cross	Relu	2x2	45	Grey	No	Normal

5.1.2. Experiment results

The mean squared error validation loss of the first experiment can be found in Table 6. From a t-test it can be concluded that the difference between both networks is not significant; after one epoch (P = 0.719) and two epochs (P = 0.612). Although there is no significant different between the two network we can however still see that the network trained with depth slightly better. Therefore this setup will be used in the next experiments.

Table 6: Validation loss (lower is better) of two trained networks after the first and second epoch. The first network only trains on normal map data. The second network on normal and depth data.

	cross, 2x2, 45, grey normal only		cross, 2x2, 45, grey normal & depth	
Fold	**1st epoch**	**2nd epoch**	**1st epoch**	**2nd epoch**
1	0.0511	0.0560	0.0549	0.0550
2	0.0498	0.0543	0.0505	0.0502
3	0.0511	0.0509	0.0513	0.0521
4	0.0519	0.0589	0.0567	0.0525
5	0.0933	0.0866	0.0767	0.0723
6	0.0591	0.0581	0.0596	0.0575
7	0.0511	0.0557	0.0767	0.0618
Mean	**0.0582**	**0.0601**	**0.0609**	**0.0573**
SD	**0.0158**	**0.0120**	**0.0112**	**0.0076**

5.2. Experiment 2: Dropout versus non-dropout

A second experiment is done to measure the effects of a dropout layer in the middle section of the neural network.

5.2.1. Experiment setup

The setup of the experiment is similar to that of experiment 1. The only difference being the network itself. Here two network configurations are tested based on the outcome of experiment 1. In one of the networks dropout layers with 50% dropout are added to the middle section of the neural network (after the concatenation in Figure 9). The other network is unadjusted. Since dropout layers also add the GPU memory usage the number of filters is reduced from 45 to 24. An overview of the networks used can be found in Table 7.

Table 7: Two networks used to test the effects when applying dropout during network training. Output (normal and depth) is based on the outcome of experiment 1.

Input	Activation	Filter size	# Filters	Colour mode	Dropout	Output
Cross	Relu	2x2	24	Grey	No	Normal Disparity
Cross	Relu	2x2	24	Grey	Yes, middle layer (50% dropout)	Normal Disparity

5.2.2. Experiment results

The result from the training can be found in Table 8. Although the result for training on a network using a dropout layer is slightly better, the result are not significant (P = 0.1455). Furthermore, based on the number of epochs needed to conclude training (due to Earlystopping) it can be seen that there are not much epochs needed for attaining the optimal result. As to the reason why this is, it can be argued that each scene generates 36 very similar light fields. Since these light fields are very familiar, the number of epochs needed for an optimal training result might be on the lower than expected. Based on these results the

subsequent networks, with the exception of the next experiment will include dropout layers.

Table 8: Validation loss (lower is better) of two trained networks and the number of epochs needed to conclude training. The first epoch is trained without a dropout layer. The second network uses a dropout layer (50% dropout).

	cross, 2x2, 45, grey no dropout		cross, 2x2, 45, grey dropout (50%)	
Fold	**Last epoch**	**Epochs trained**	**Last epoch**	**Epochs trained**
1	0.0441	3	0.0358	3
2	0.0456	3	0.0353	4
3	0.0435	2	0.0396	4
4	0.0497	2	0.0401	6
5	0.0556	2	0.0456	3
6	0.0444	3	0.0545	3
7	0.0437	5	0.0431	4
Mean	**0.0467**	**2.86**	**0.0420**	**3.86**
SD	**0.0045**		**0.0066**	

5.3. Number of filters

A third experiment is done to test how the number of filters influences training results.

5.3.1. Experiment setup

For this test the setup and outcome of experiment 1 is used. Dropout (experiment 2) is not used to allow for training on more than 24 filters. 9 networks are used with a range in the number of filters from 5 to 45. An overview of the networks trained for the third experiment can be found in Table 9.

Table 9: Nine networks trained to test the influence of the number of filters on the training result. The output is based on the outcome of experiment 1. Although based on experiment 2, dropout will be applied in the subsequent test. For this experiment it is disabled to reduce training time.

Input	Activation	Filter size	# Filters	Colour mode	Dropout	Output
Cross	Relu	2x2	5	Grey	No	Normal Disparity
Cross	Relu	2x2	10	Grey	No	Normal Disparity
Cross	Relu	2x2	15	Grey	No	Normal Disparity
Cross	Relu	2x2	20	Grey	No	Normal Disparity
Cross	Relu	2x2	25	Grey	No	Normal Disparity
Cross	Relu	2x2	30	Grey	No	Normal Disparity
Cross	Relu	2x2	35	Grey	No	Normal Disparity
Cross	Relu	2x2	40	Grey	No	Normal Disparity

Cross	Relu	2x2	45	Grey	No	Normal Disparity

5.3.2. Experiment results

The results for experiment 3 can be found in Table 10. From the table it can be seen that a lower number of filters results in a lower validation loss. When comparing the best results (5 filters) with the second best (15 filters) the difference is not significant (P = 0.2271). However given that it is still the best result 5 filters will be used in the next experiment.

Table 10: Validation loss (lower is better) of seven trained networks with different number of filters (5 till 45).

	cross, 2x2, 45, grey, no dropout number of filters used in training								
Fold	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**
1	0.0415	0.0385	0.0403	0.0423	0.0443	0.0436	0.0394	0.0386	0.0410
2	0.0420	0.0384	0.0385	0.0425	0.0417	0.0408	0.0466	0.0419	0.0466
3	0.0358	0.0386	0.0398	0.0419	0.0399	0.0437	0.0439	0.0410	0.0459
4	0.0377	0.0414	0.0400	0.0451	0.0437	0.0509	0.0664	0.0741	0.0434
5	0.0385	0.0540	0.0498	0.0570	0.0663	0.0456	0.0524	0.0527	0.0603
6	0.0429	0.0484	0.0421	0.0428	0.0446	0.0460	0.0452	0.0445	0.0491
7	0.0361	0.0409	0.0402	0.0406	0.0410	0.0417	0.0432	0.0432	0.0435
Mean	**0.0392**	**0.0429**	**0.0415**	**0.0446**	**0.0459**	**0.0446**	**0.0482**	**0.0480**	**0.0471**
SD	**0.0029**	**0.0060**	**0.0038**	**0.0056**	**0.0092**	**0.0033**	**0.0089**	**0.0123**	**0.0064**

5.4. Cross versus full, RGB versus grey and different filter sizes

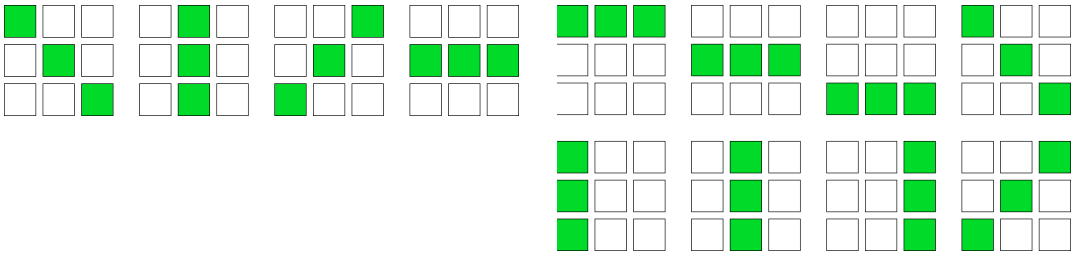
In this experiment various configurations are tested using different input configurations, RGB and a combination of two filter sizes.

5.4.1. Experiment setup

For this experiment a similar setup is chosen as experiment 1 with the following adaptations:

Input mode

Where previous experiments all used the cross section input similar to the EPInet architecture, here two networks are tested that both use the complete light field as input. A schematic display of the difference in input configuration can be seen in Figure 12.



(a) (b)

Figure 12: Different input configurations for the classifier. (a) Horizontal, vertical and diagonal (cross) inputs. (b) Full light field input that utilized 8 input stacks.

Filter size

The EPInet solution uses 2x2 filters in order to predict pixel disparity. As the normals are dependent on their direct neighbourhood, this filter size seems a good choice. However since the light field images can contain larger areas of similar colours, the 2x2 filters are tested in combination with 5x5 and 7x7 filters sizes.

Colour mode
As mentioned above the EPInet solution uses greyscale images as inputs. By converting RGB to greyscale any information stored in the RGB channels is lost. Therefore RGB inputs in combination with 3D convolutions are tested as an alternative to greyscale only.

Based on the variables above different combination have been made for training. An overview of these networks trained can be found in Table 11.

Table 11: Nine networks trained to test the influence of different features on the training result. The output is based on the outcome of experiment 1. The use of dropout is based on experiment 2. The number of filters is reduced to 5 based on experiment 3.

Input	Activation	Filter size	# Filters	Colour mode	Dropout	Output
Cross	Relu	2x2	5	Grey	Yes	Normal Disparity
Cross	Relu	2x2	5	RGB	Yes	Normal Disparity
Full	Relu	2x2	5	Grey	Yes	Normal Disparity
Full	Relu	2x2	5	RGB	Yes	Normal Disparity
Cross	Relu	2x2 5x5	5 5	RGB	Yes	Normal Disparity
Cross	Relu	2x2 7x7	5 5	RGB	Yes	Normal Disparity

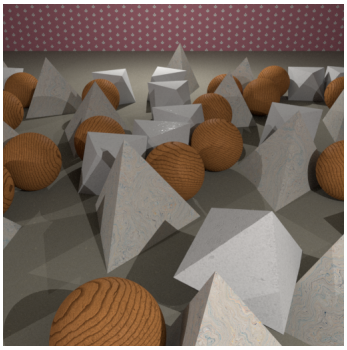
5.4.2. Experiment results

For experiment 4 the training results can be found in Table 12. The best performing network uses a cross section RGB input, a 2x2 and 5x5 filter, 5 filters and a dropout layer. Comparing the best results with the second best it can be concluded that the results are significant (P = 0.0255).

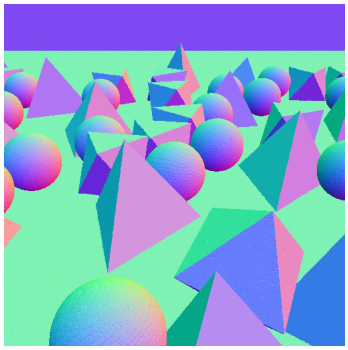
Table 12: Validation loss (lower is better) for six network with various network architectures.

Fold	cross, 2x2, 5, grey, dropout	full, 2x2, 5, grey, dropout	cross, 2x2, 5, rgb, dropout	full, 2x2, 5, rgb, dropout	cross, 2x2, 5x5, 5, rgb, dropout	cross, 2x2, 7x7, 5, rgb, dropout
1	0.0471	0.0384	0.0491	0.0569	0.0394	0.0456
2	0.0445	0.0339	0.0415	0.0531	0.0357	0.0356
3	0.0437	0.0354	0.0958	0.0432	0.0385	0.0392
4	0.0424	0.0975	0.0367	0.0384	0.0362	0.0466
5	0.0473	0.0335	0.0449	0.0473	0.0357	0.0590
6	0.0489	0.0594	0.0451	0.0550	0.0476	0.0687
7	0.0690	0.0433	0.0349	0.0378	0.0360	0.0472
Mean	**0.0490**	**0.0488**	**0.0497**	**0.0474**	**0.0385**	**0.0489**
SD	**0.0091**	**0.0233**	**0.0209**	**0.0079**	**0.0043**	**0.0114**

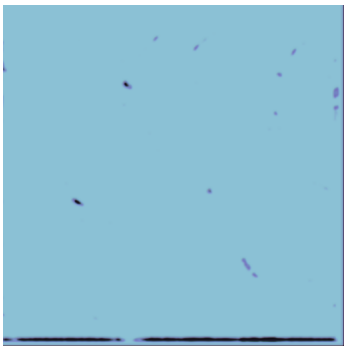
Visual inspection
Based on the training result the best network is used for the prediction of validation (scene 7) and unseen (scene 8) data. After prediction the normal map is normalized and any values below 0 and 1 are min/maxed to 0 and 1. As can be seen in Figure 13 this prediction is far off from the ground truth normal map.


(a) Scene 7 centre image.


(b) Predicted normal map. Scene 7, rotation 0.


(c) Ground truth normal map. Scene 7, rotation 0.


(d) Scene 8 centre image.


(e) Predicted normal map. Scene 8, rotation 0.

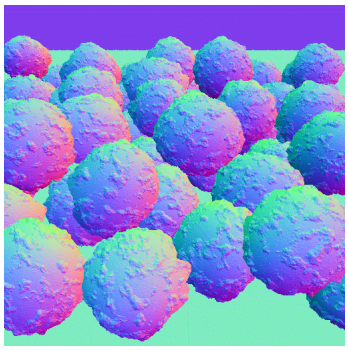

(f) Ground truth normal map. Scene 8, rotation 0.

Figure 13: Predicted normal maps using the best performing network.

Given the unsatisfactory result of the trained network adjustments are necessary. However before changes are made to the network design, first a test is done to see how the number of light field scenes influences training results.

5.5. Dataset size

The fifth experiment done is to test the influence of the number of light field scenes on the training results.

5.5.1. Experiment setup

For the experiment a range of 1 till 6 light field scenes of each 36 light field inputs are selected for training and compared on their training results. Here the experiment setup is similar to that of experiment 1 and just like previous experiments data enhancement is used to increase the training data set. The design of the network used in this experiment can be found in Table 13.

Table 13: The network used to test the influence of the number of light fields on the training result.

Input	Activation	Filter size	# Filters	Colour mode	Dropout	Output

Cross	Relu	2x2	24	Grey	Yes	Normal Disparity

5.5.2. Experiment results

The results of the fifth experiment can be found in Table 14. From the mean and standard deviation it can be seen that there is improvement as soon as more light fields are added to the training dataset. From Figure 14 it can be seen that although the training is still improving after training on more than three light field scenes, the validation loss curve is flatting out. This would mean that adding more light field scenes would probably improve performance but the improvement would be minimal.

Table 14: Validation loss (lower is better) of the last epoch of six trained networks. Each network is trained on a various number of light field scenes.

	cross, 2x2, 24, grey, dropout number of light field scenes used for training					
Fold	**1**	**2**	**3**	**4**	**5**	**6**
1	0.0351	0.0789	0.0349	0.0408	0.0344	0.0358
2	0.0455	0.0493	0.0430	0.0425	0.0397	0.0359
3	0.0563	0.0498	0.0368	0.0431	0.0356	0.0360
4	0.1068	0.0491	0.0577	0.0384	0.0425	0.0376
5	0.0448	0.0332	0.0370	0.0403	0.0321	0.0468
6	0.0627	0.0686	0.0544	0.0590	0.0562	0.0513
7	0.0843	0.0413	0.0386	0.0388	0.0459	0.0394
Mean	**0.0622**	**0.0529**	**0.0432**	**0.0433**	**0.0409**	**0.0404**
SD	**0.0253**	**0.0157**	**0.0092**	**0.0071**	**0.0083**	**0.0062**

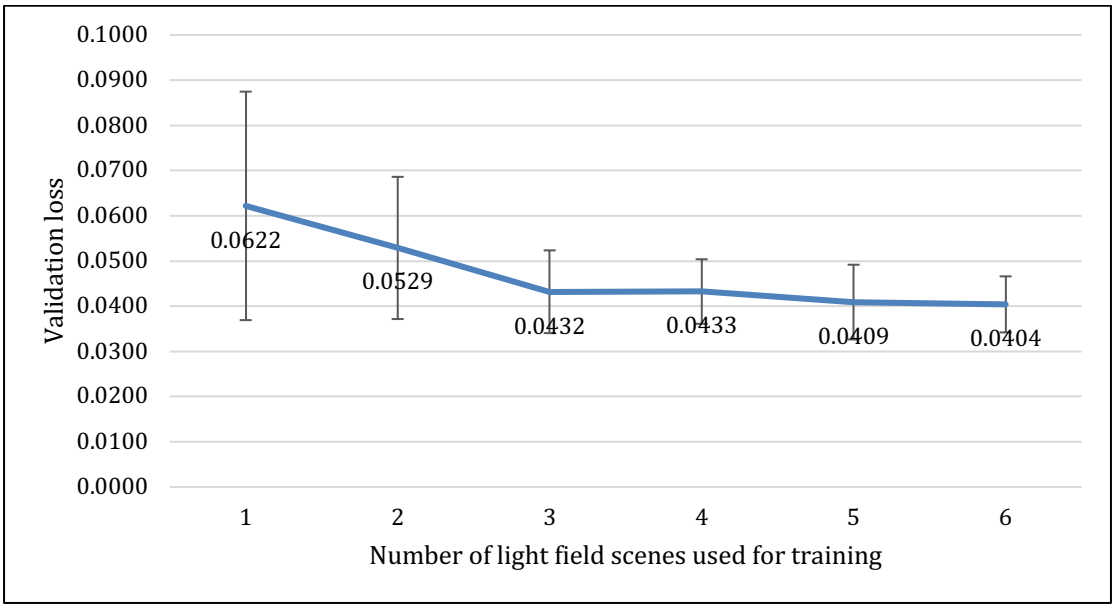


Figure 14: Diagram showing the influence of the number of light field scenes on training quality.

Based on these result it can concluded that adding more training data currently is not useful. The next experiment will focus on different activation functions.

5.6. Alternative activation functions

A sixth experiment is done to test different activation functions. This due to the discrepancy between the range of the Relu function (0 - ∞) and that of the normal map (0-1). As an alternative the Hard sigmoid activation function with a similar range to that of a normal map is tested.

5.6.1. Experiment setup

For this experiment three networks are compared. A network that uses Rely only, a network with Relu and a Hard Sigmoid at the end and a network using only Hard Sigmoid functions.

In order to facilitate the change to Hard Sigmoid the output (ground truth) has to be adjusted as well. The disparity map which uses a range between 0 and infinity (theoretically) has been removed. For the normal map the Z component is adjusted. Here the previous conversion from a range between 0.5 and 1 is replaced to a range between 0 and 1. The number of filters have been raised to 18, the maximum that allowed for training on the GPU. An overview of the networks used for training can be found in Table 15.

For the training all networks have been trained on three epochs. This to reduce training times.

Table 15: The networks used to test the influence of different activation functions on the training result.

Input	Activation	Filter size	# Filters	Colour mode	Dropout	Output
Cross	Relu / Relu	2x2	18	Grey	Yes	Normal
Cross	Relu / Hsig	2x2	18	Grey	Yes	Normal
Cross	Hsig / Hsig	2x2	18	Grey	Yes	Normal

5.6.2. Experiment results

The results of the sixth experiment can be found inTable 16. From the training results in Table 16 it can be seen that the Hsig/Hsig performs better than the other two. Although the difference between the best performing and two other networks is not significant (Relu/Hsig: P = 0.1523 | Relu/Relu: P = 0.1852) it is still the best choice for attaining a good training result. Therefore hard sigmoid will be used as a activation function in the next experiments.

Table 16: Validation loss (lower is better) of the last epoch of the three trained networks. Each network uses different activation functions.

	cross, 2x2, 18, grey, dropout		
Fold	**Hsig / Hsig**	**Relu / Hsig**	**Relu / Relu**
1	0.0726	0.0676	0.0695
2	0.0649	0.0631	0.0647
3	0.0610	0.0671	0.0662
4	0.0621	0.0957	0.0791
5	0.0570	0.1029	0.1132
6	0.0962	0.0864	0.0838
7	0.0624	0.0758	0.0784
Mean	**0.0680**	**0.0798**	**0.0793**
SD	**0.0133**	**0.0155**	**0.0166**

5.7. Custom loss function: Mean Vector Distance

The next two experiment are done to test alternative loss functions. This due to the MSE not representing the relation between the three vector components of the normal map. The two loss functions that are tested are the mean vector distance (MVD) of formula 7 and mean different angles (MDA) of formula 8.

$$Mean\ vector\ distance = \frac{1}{n}\sum_{i}^{n}|a(i) - b(i)| \tag{7}$$

$$Mean\ of\ different\ angles = \frac{1}{n}\sum_{i}^{n}\cos^{-1}\left(\frac{a(i)\cdot b(i)}{|a(i)|\cdot|b(i)|}\right) \tag{8}$$

5.7.1. Experiment setup

The first loss function tested is mean vector distance (MVD) of formula 6, which computes the distance between the predicted and ground truth normal vector.

Due to the resizing of the light fields and the poor performance of experiment 4 various network configurations are tested to evaluate the loss functions performance. Different filter size are tested as well as different colour inputs. An overview of the networks trained can be seen in
Table 17. For comparison all networks have been trained on ten epochs.

To compare both the MVD and the MDA both functions are used in this experiment. The MVD is used as the loss function. The MDA is added as a metric. The MDA metric is used in the next experiment for comparison.

Another change done for this experiment is the resizing of the dataset. Here the original images and normal maps are resized from 512x512 to 128x128 through Lanczos resampling. This allows for reduced training times and testing of more parameters (e.g. number of filters). Due to this the this experiment is done with 70 filters, similarly to that of EPInet.

Table 17: Five networks trained to test the performance of the mean vector distance.

Input	Activation	Filter size	# Filters	Colour mode	Dropout	Output
Cross	Hsig	2x2	70	Grey	Yes	Normal
Cross	Hsig	3x3	70	Grey	Yes	Normal
Cross	Hsig	4x4	70	Grey	Yes	Normal
Cross	Hsig	5x5	70	Grey	Yes	Normal
Cross	Hsig	2x2 5x5	70 70	Grey	Yes	Normal
Cross	Hsig	2x2 5x5	70 70	RGB	Yes	Normal

5.7.2. Experiment results

From the training results in Table 18 it can be seen that the experiment is slightly overfitting, especially when looking at the training loss (not shown in the table).

Table 18: Mean validation loss (lower is better) of the 7-fold cross validation for each epoch for each network trained.

Epoch	cross, 2x2, grey, dropout	cross, 3x3, grey, dropout	cross, 4x4, grey, dropout	cross, 5x5, grey, dropout	cross, 2x2, 5x5, grey, dropout	cross, 2x2, 5x5, RGB, dropout
1	0.8007	0.7966	0.7567	0.7744	0.7649	0.7684
2	**0.7028**	0.8297	0.7131	**0.7017**	0.7746	0.7636
3	0.7964	**0.7456**	**0.7081**	0.7428	0.7933	0.7438
4	0.7376	0.7878	0.7993	0.7697	0.7362	0.8395
5	0.7518	0.7933	0.8082	0.7217	**0.7230**	0.7764
6	0.7363	0.8165	0.8112	0.8352	0.7258	**0.7292**
7	0.7329	0.8362	0.7781	0.7711	0.8173	0.7935
8	0.7423	0.8391	0.7356	0.7770	0.7423	0.8176
9	0.7488	0.9051	0.7616	0.7406	0.7377	0.8211
10	0.8202	0.8993	0.8326	0.7960	0.7591	0.7886

This is probably due to the increased number of filters (70) which allows the network to store more information. In order to still make an informed decisions about which networks perform better, the best performing epoch is used as a guide.

In Table 19 an overview of the best performing epochs is shown. As can be seen the result are very similar, therefore two networks are selected for the next experiment. The *cross, 2x2, grey* architecture for its simplicity and the *cross, 2x2, 5x5, grey* due to its performance in experiment 4.

Table 19: Validation loss (lower is better) for the MVD of the best performing epoch for the six trained networks.

Fold	cross, 2x2, grey, dropout	cross, 3x3, grey, dropout	cross, 4x4, grey, dropout	cross, 5x5, grey, dropout	cross, 2x2, 5x5, grey, dropout	cross, 2x2, 5x5, RGB, dropout
	Epoch 2	Epoch 3	Epoch 3	Epoch 2	Epoch 5	Epoch 6
1	0.6961	0.7920	0.6769	0.6672	0.7461	0.7494
2	0.6870	0.9771	0.7950	0.9870	0.6741	0.7058
3	0.6760	0.6093	0.9354	0.8362	0.7758	0.8360
4	0.6262	0.8386	0.5711	0.6903	0.8863	0.8032
5	0.6992	0.5567	0.4764	0.4421	0.6686	0.5458
6	0.8637	0.6515	0.7521	0.6414	0.5654	0.8697
7	0.6713	0.7943	0.7501	0.6476	0.7442	0.5945
Mean	**0.7028**	**0.7456**	**0.7081**	**0.7017**	**0.7230**	**0.7292**
SD	**0.0750**	**0.1471**	**0.1507**	**0.1707**	**0.1005**	**0.1220**

5.8. Custom loss function: Mean Different Angle

In this experiment the mean different angle is used as a loss function.

5.8.1. Experiment setup

The second loss function tested is mean different angle from formula 7 which computes the angle between the predicted and ground truth vector. The experiment is similar in setup than the previous one with the difference that only the two best performing networks are tested. In order to compare both validation functions the MDA is used as a loss function while the MVD is added as a metric. The trained networks can be seen in Table 20.

Table 20: Two networks trained to test the performance of the mean difference of angles.

Input	Activation	Filter size	# Filters	Colour mode	Dropout	Output
Cross	Hsig	2x2	70	Grey	Yes	Normal
Cross	Hsig	2x2 5x5	70 70	Grey	Yes	Normal

5.8.2. Experiment results

After analysing the result during training the experiment is stopped after 1 fold. This due to the mean different angle resulting in high validation losses.
As can be seen from Table 21 the training is stopped after two epochs by the Keras Earlystopping. When looking at the mean vector distance metric it can also be observed that this is higher than that of the previous experiment. For comparison the difference between the first fold for the *cross, 2x2* network from the previous and current experiment is shown in Table 22.

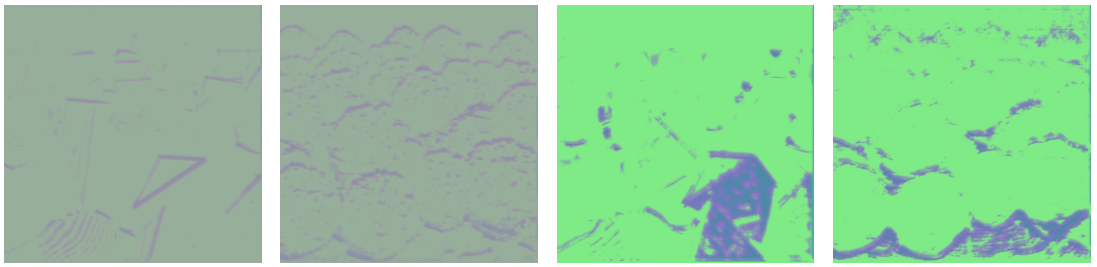
Table 21: Mean difference angles and their corresponding mean vector difference for two trained networks. As can be seen the loss function does not decrease the metric.

	cross, 2x2, grey, dropout		cross, 2x2, 5x5, grey, dropout	
Epoch	MDA	MVD	MDA	MVD
1	54.7168	1.4158	54.7168	1.4158
2	54.7168	1.4158	54.7168	1.4158

Table 22: Difference between validation loss for two similar networks for the first fold. The first network uses the MDA as a loss function and the MVD as a metric. The second network uses the MVD as a loss function and the MDA as a metric.

	cross, 2x2, grey, dropout		cross, 2x2, 5x5, grey, dropout	
Epoch	MDA (loss)	MVD (metric)	MDA (metric)	MVD (loss)
1	54.7168	1.4158	38.80	0,6941
2	54.7168	1.4158	34.83	0,6961
3			33.22	0,6775
4			32.56	0,6764
5			31.75	0,7082
6			31.14	0,7199
7			30.85	0,7575
8			30.70	0,7126
9			30.49	0,7250
10			30.36	0,7244

Visual inspection
When comparing the results from this and the last experiment visually in Figure 15 the difference become clear as well. Here to current experiment classifies inputs as white images while the former experiment predicts something that could go for a normal map. Based on these outcomes the next and last experiment will be done with the mean vector distance as a loss function.



(a) Scene 7, 2x2, 5x5, mean vector distance as loss function

(b) Scene 8, 2x2, 5x5, mean vector distance as loss function

(c) Scene 7, 2x2, mean vector distance as loss function

(d) Scene 8, 2x2, mean vector distance as loss function

(e) Scene 7, 2x2, 5x5, mean different angles as loss function

(f) Scene 8, 2x2, 5x5, mean different angles as loss function

(g) Scene 7, 2x2, me mean different angles as loss function

(h) Scene 8, 2x2, mean different angles as loss function

Figure 15: Visual comparison of the predictions done by similar networks trained on the MVD and MDA.

5.9. Flipped and rotated dataset

The last experiment in this research is done on an extended dataset to see if this results in a better prediction.

5.9.1. Experiment setup

For this experiment the simplest of the two chosen networks from experiment 7 and 8 is picked. This due to there being no significant difference between the networks used.

For training the dataset has been expanded by utilizing flipping and rotating of the light field as described in section 4.4 since this would increase the dataset size from 4104 to 24624 samples. The configuration of the final network trained can be found in Table 23.

Table 23: The final network trained on an expanded dataset of 24624 samples.

Input	Activation	Filter size	# Filters	Colour mode	Dropout	Output
Cross	Hsig	2x2	70	Grey	Yes	Normal

5.9.2. Experiment results

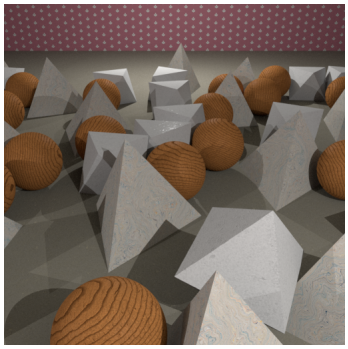
The result of the experiment can be found in Table 24. When comparing the validation loss with the best epoch (mean: 0.7029 , sd: 0.075) of the same network from experiment 7 it can be concluded that there is no significant improvement (P = 0.8083) for using an extended dataset.

Table 24: Validation loss (mean vector distance) of the last epoch. Network is trained on an extended dataset.

	cross, 2x2, grey, dropout
Fold	**MVD**
1	0.7681
2	0.7329
3	0.7260
4	0.6842
5	0.8241
6	0.5515
7	0.7070
Mean	**0.7134**
SD	**0.0846**

Visual inspection

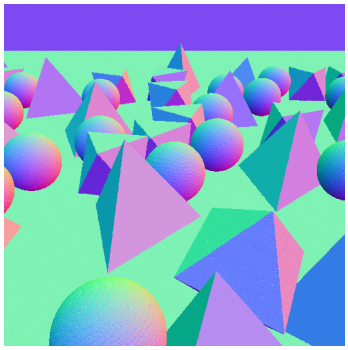
When analysing the result visually in Figure 16 a clear improvement with regards to the outcome of experiment 4 can be seen. However as will be discussed in of the results these predictions are still far off from the ground truth.



(a) Scene 7 centre image.

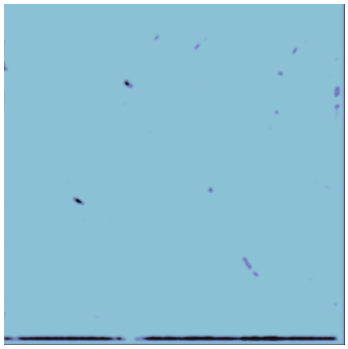
(b) Scene 7 prediction result experiment 4.

(c) Scene 7 prediction result experiment 9.

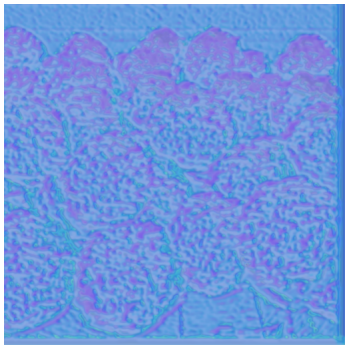


(d) Scene 7 ground truth

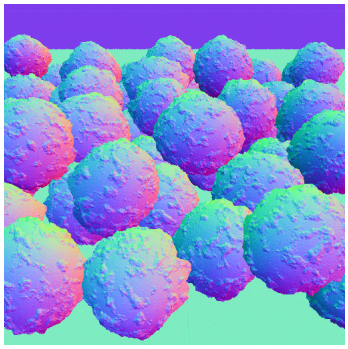
(e) Scene 8 centre image.



(f) Scene 8 prediction result experiment 4.



(g) Scene 8 prediction result experiment 9.



(h) Scene 8 ground truth

Figure 16: Comparison between prediction results from experiment 4, 8 and the ground truth.

Mean of difference in angles

Given both predictions of this last experiments we can also computer the mean in difference of angles for the predicted normal maps of scene 7 and scene 8.

Table 25: Overview of normal maps and their distance from the ground truth.

Normal map	**Mean Different Angle**
Scene 7	
Direct prediction trough machine learning	49°
Scene 8	
Direct prediction trough machine learning	45°

6. Results

6.1. How can machine learning be used to predict normal maps?

Given the experiments done in chapter 5 a network architecture has been defined that can predict normal maps. This network is trained on a large dataset of 24624 light fields based on 6 light field scenes. The mean vector distance is used for optimization. Given the validation loss it can be concluded that on average the distance between prediction and ground truth normals is 0.7134 in normal map space.

An overview of the architecture of the final network can be found in Table 26. A schematic overview can be seen in Figure 17. An overview of the code used for training and the generation of data can be found in appendix 1 and appendix 2. A larger version of the schematic overview of figure 17 can be found in appendix 3.

Table 26: Network used to predict normal maps from light field input images.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 128, 128, 3)	0	
input_2 (InputLayer)	(None, 128, 128, 3)	0	
input_3 (InputLayer)	(None, 128, 128, 3)	0	
input_4 (InputLayer)	(None, 128, 128, 3)	0	
sequential (Sequential)	(None, 128, 128, 70)	100100	input_1[0][0]
sequential_1 (Sequential)	(None, 128, 128, 70)	100100	input_2[0][0]
sequential_2 (Sequential)	(None, 128, 128, 70)	100100	input_3[0][0]
sequential_3 (Sequential	(None, 128, 128, 70)	100100	input_4[0][0]
concatenate (Concatenate)	(None, 128, 128, 280)	0	sequential[1][0] sequential_1[1][0] sequential_2[1][0] sequential_3[1][0]
sequential_4 (Sequential)	(None, 128, 128, 280)	4402160	concatenate[0][0]
sequential_5 (Sequential)	(None, 128, 128, 3)	79313	sequential_4[0][0]
Total params: 4,881,873 Trainable params: 4,876,273 Non-trainable params: 5,600			

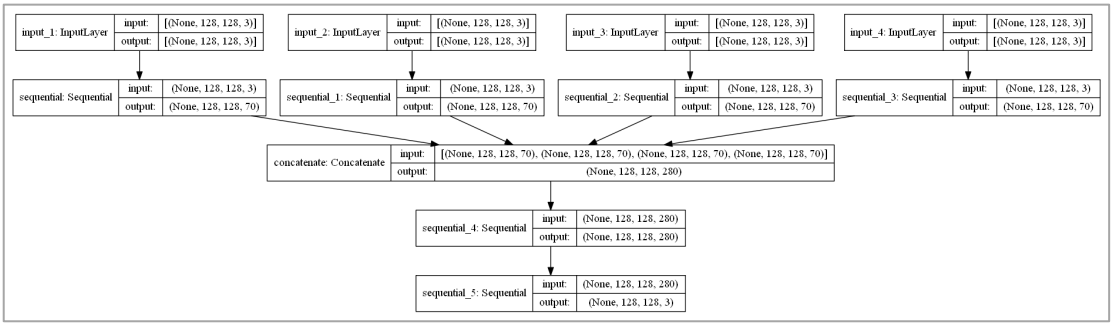


Figure 17: Final network architecture for the prediction of normal maps. See Appendix 3 for a larger rotated version.

6.2. How does a normal map generated with machine learning compare with other depth based normal map generation methods based?

In order to evaluate the trained neural network the predicted normal maps from experiment 5.9 are used for evaluation. A visual comparison with the ground truth can be found in Figure 18.

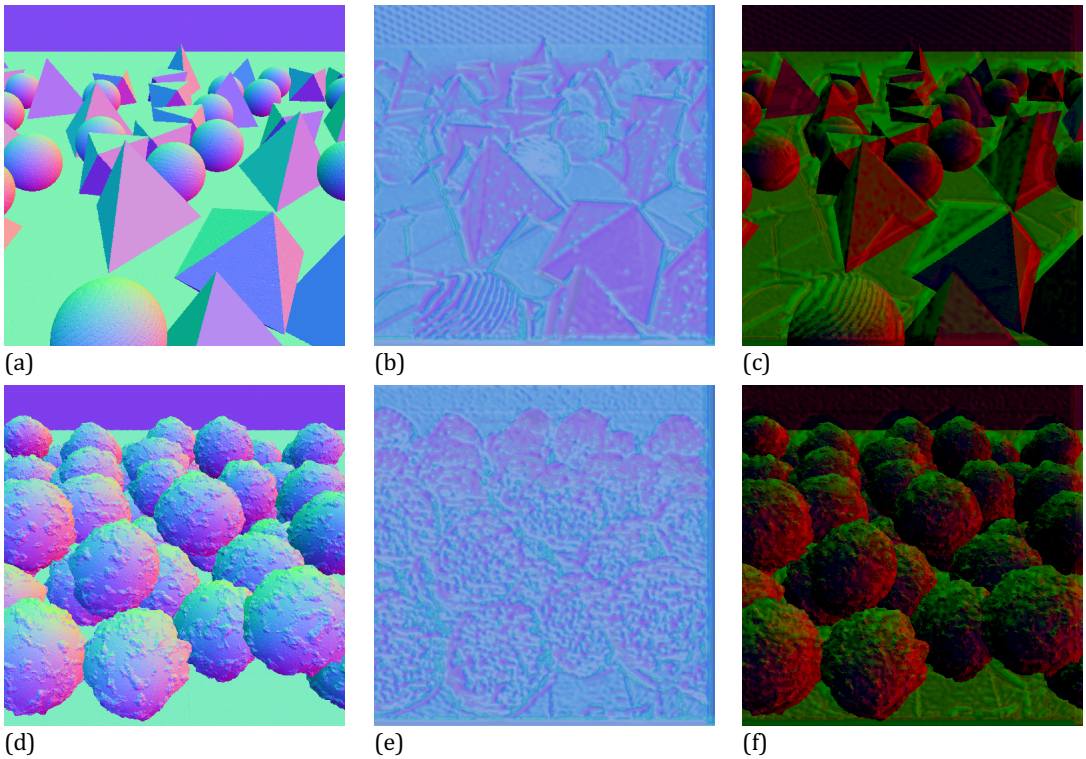


Figure 18: Visual comparison of generated normal maps with their ground truth. (a) Scene 7 normal map ground truth, (b) predicted normal map, (c) absolute differences of the predicted and ground truth normal map. (d) Scene 8 normal map ground truth, (e) predicted normal map, (f) absolute differences of the predicted and ground truth normal map..

As can be seen, there is quite some difference with the ground truths although the general shapes can be recognized. Problem exist in areas of colour transition, at discontinuities and there where the normal are perpendicular to the view vector. Normals pointing at the camera are mostly predict fairly well.

Visual comparison
The results can be compared with the ground truth and normal maps generated in experiment 0. An visual overview of the this comparison can be found in Figure 19.

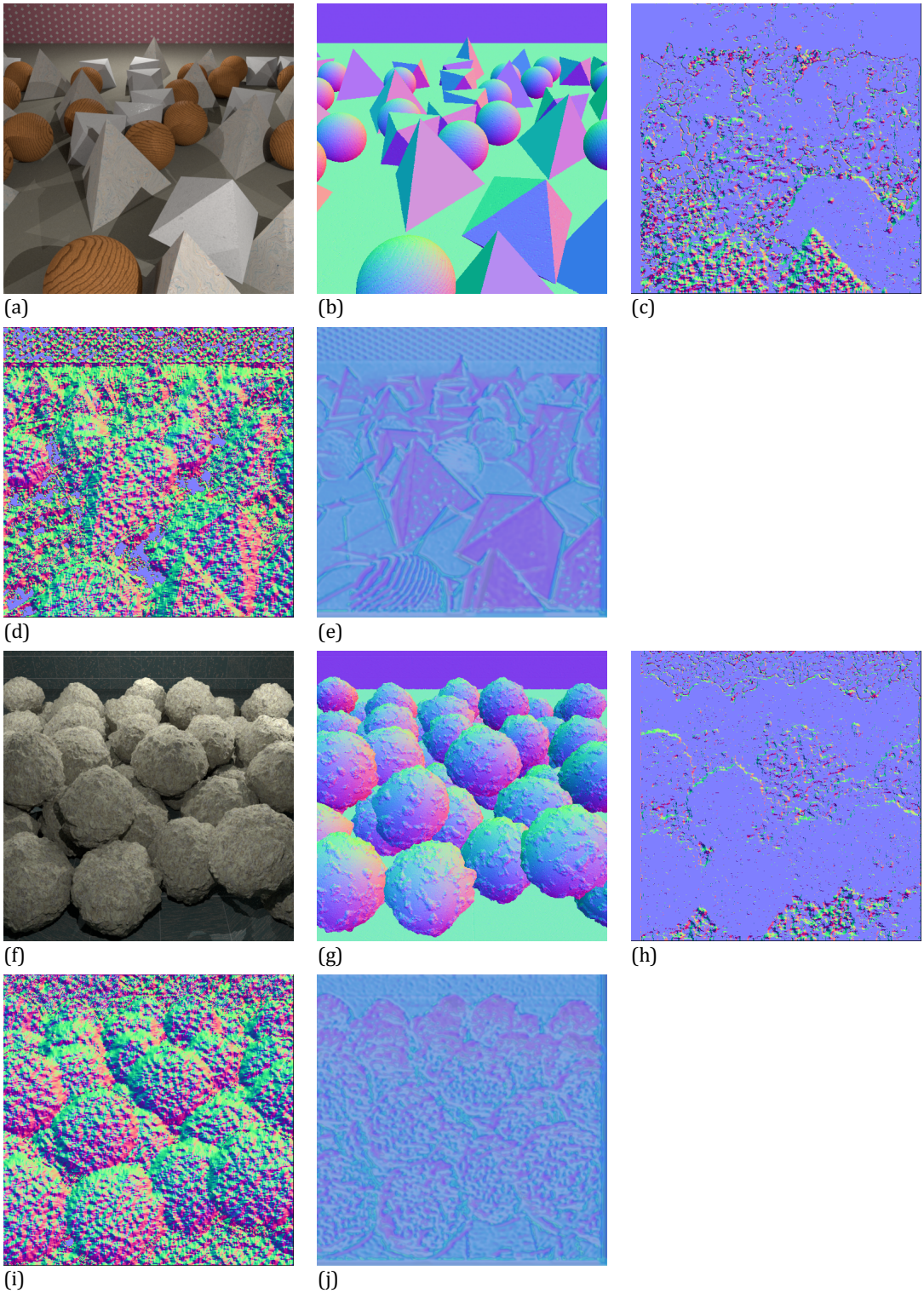


Figure 19: Inputs and normal maps for two light fields scenes. (a) RGB middle input image of the light field, (b) ground truth normal map, (c) normal map generated by Hinterstoisser and naive stereo depth, (d) normal map generated by Hinterstoisser and EPInet (5x5 input), (e) predicted normal map. (f-j) Similar to *a-e* for scene 8.

Mean of difference in angles

Besides a visual comparison the result can also be compared to the baseline normal maps by using the mean of difference in angles. In Table 27 an overview of the MDA for both the baseline and predicted normal maps is given. As can be

seen the prediction done by the neural networks is a good improvement over the baseline methods.

Table 27: Overview of normal maps and their distance from the ground truth.

Normal map	**Mean Different Angle**
Scene 7	
Stereo + Hinterstoisser	60°
EPInet + Hinterstoisser	73°
Direct prediction trough machine learning	49°
Scene 8	
Stereo + Hinterstoisser	52°
EPInet + Hinterstoisser	66°
Direct prediction trough machine learning	45°

7. Conclusion

Given the research results it can be concluded that deep learning can be used for the prediction of normal maps from light field data. Compared with other methods explained in this research, predicting normal maps directly from light field inputs with the trained neural network is a good improvement compared to other state of the art normal map generation methods. In both scenes the neural network performed better: 49° versus 60° & 73° and 45° versus 52° & 66°.

However, as can be seen visually from the predicted results, the normal maps are far from perfect making them difficult to use in real world applications. Problems exist in areas of colour transition, at discontinuities and where the normals are perpendicular to the view vector.

Given these result the next section will address some issues encountered during this research. After the discussion some suggestion for further research will be given.

8. Discussion

Given this research there are a few points that are up for discussion.

1. *Multi-dimensional problem solving*
   As already mentioned in the conclusion, finding a good performing network to accurately predict normal maps based on light field data is a difficult and tedious task. Part of this problem is the variable space in which a solution can be found. Not only does this include different filter sizes, number of filters, loss & activation functions but also input & output configuration; as well as camera and light field camera rig parameters. To solve this multi-dimensional knot the author tried to find an optimum by making smart decision based on experiments and intuition. However it could be that a different strategy would have resulted in finding a better architecture.

2. *Synthetic training data*
   Another point of discussion is the use of synthetic light field data (CGI). Even as Lee and Moloney (2017) & Prakash et al. (2018) shows us that when done right, synthetic data can be used as a substitute, one can wonder if the renders used for this specific research where of sufficient quality. Here one can argue that real world object contain more imperfections and have more visual complex materials, another difference is that artificial images might be devoid of noise naturally captured by real optical camera system. Given these differences closer to real world images might result in a better learnable solution.

   Of course one can counter this argument by saying that this kind of research is impossible without synthetic images; how would one attain ground truth normal map data for multiple light fields? one can wonder if the result where not heavily influenced by the quality of synthetic scenes.

3. *Limitation on computation*
   A big limitation throughout this research is that of computation times. Even though a state-of-the-art GPU is used, memory limitation placed huge restriction on what could be computed in the time available. Given more specialized hardware experiments might be done quicker, resulting in a better performing final network.

9. Recommendation

Given these results, conclusion and discussion the following recommendation can be given for further research on this topic.

1. *Hyper machine learning*
   Given the complexity addressed with regard to the multi-dimensional parameters solving as mentioned in the recommendation, hyper machine learning could be a beneficial testing area for continuing this research.

2. *Precomputation for error reduction*
   One of the areas this research did not explore is how precomputation on the light field images can add to the results of a normal prediction. For instance, optical flow could be used for finding discontinuities or blob detection for finding larger areas of similar colour.

3. *High quality renders and larger datasets*
   As already mentioned in the discussion section; any network used for predicting normal maps would probably benefit from a larger and better rendered dataset. It would be interesting to explore how different light field scenes would influence performance. Here one can think of discontinuities, contrasts, lightness, colour usage, etc.

4. *Camera setup and parameters*
   Another interesting area of research would be that of how light field camera setup would influence performance of a neural network for determining normal maps. This does not only include intrinsic camera parameters but also the number of cameras used within a light field rig.

5. *Light field computer vision pipeline*
   Research the possibility to train a neural network to predict all the physical properties of a scene. That is: normals, depth, refractive & reflective properties. This could be very interesting for artificial relighting or reconstruction within a computer program.

Literature List

4D Light Field Benchmark. (2019). In.

Gardner, A., Tchou, C., Hawkins, T., & Debevec, P. (2003). Linear light source reflectometry. *ACM Transactions on Graphics (TOG), 22*(3), 749-758.

Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., & Lepetit, V. (2011). *Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes.* Paper presented at the 2011 international conference on computer vision.

Lee, K., & Moloney, D. (2017). *Evaluation of synthetic data for deep learning stereo depth algorithms on embedded platforms.* Paper presented at the 2017 4th International Conference on Systems and Informatics (ICSAI).

Ma, W.-C., Hawkins, T., Peers, P., Chabert, C.-F., Weiss, M., & Debevec, P. (2007). *Rapid acquisition of specular and diffuse normal maps from polarized spherical gradient illumination.* Paper presented at the Proceedings of the 18th Eurographics conference on Rendering Techniques.

Prakash, A., Boochoon, S., Brophy, M., Acuna, D., Cameracci, E., State, G., . . . Birchfield, S. (2018). Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data. *arXiv preprint arXiv:1810.10093*.

Shin, C., Jeon, H.-G., Yoon, Y., So Kweon, I., & Joo Kim, S. (2018). *Epinet: A fully-convolutional neural network using epipolar geometry for depth from light field images.* Paper presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

Weyrich, T., Matusik, W., Pfister, H., Bickel, B., Donner, C., Tu, C., . . . Jensen, H. W. (2006). *Analysis of human faces using a measurement-based skin reflectance model.* Paper presented at the ACM Transactions on Graphics (TOG).

Appendix 1 – Final network code

```
from models.loss_functions.losses import *
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input , Activation
from tensorflow.keras.layers import Conv2D, Reshape
from tensorflow.keras.layers import Dropout,BatchNormalization
from tensorflow.keras.layers import concatenate

# Create layer functions
def get_layer_multistream( filters, input_dimension, convolution_depth ):
            seq = Sequential()

            for i in range( convolution_depth ):
                        seq.add( Conv2D( filters = filters, kernel_size = (2,2), strides = (1,1),
padding = 'same', input_shape = input_dimension ) )
                        seq.add( Activation( 'hard_sigmoid' ) )
                        seq.add( Conv2D( filters = filters, kernel_size = (2,2), strides = (1,1),
padding = 'same' ) )
                        seq.add( BatchNormalization( axis = -1 ) )
                        seq.add( Activation( 'hard_sigmoid' ) )

            return seq

def get_layer_merged( filters, input_dimension, convolution_depth ):
            seq = Sequential()

            for i in range( convolution_depth ):
                        seq.add( Dropout(0.5) )
                        seq.add( Conv2D( filters = filters, kernel_size = (2,2), padding = 'same',
input_shape = input_dimension ) )
                        seq.add( Activation( 'hard_sigmoid' ) )
                        seq.add( Dropout(0.5) )
                        seq.add( Conv2D( filters = filters, kernel_size = (2,2), padding = 'same' )
)
                        seq.add( BatchNormalization( axis = -1 ) )
                        seq.add( Activation( 'hard_sigmoid' ) )

            return seq

def get_layer_last( filters, input_dimension, convolution_depth ):
            seq = Sequential()

            for i in range( convolution_depth ):
                        seq.add( Conv2D( filters = filters, kernel_size = (2,2), padding = 'same',
input_shape = input_dimension ) )
                        seq.add( Activation( 'hard_sigmoid' ) )

            seq.add( Conv2D( filters = 3, kernel_size = (2,2), padding = 'same', input_shape =
input_dimension ) )
            seq.add( Activation( 'hard_sigmoid' ) )

            return seq

def NORnet_cross_grey_n_2_dropout_hsig_hsig_md( fil = 35 ):
            # Model settings
            lfs = 3
            filters = fil
            dimension = ( 128, 128, lfs )

            # Create model layers
            input_1 = Input( dimension )
            input_2 = Input( dimension )
            input_3 = Input( dimension )
            input_4 = Input( dimension )
```

```
            layer_multistream_1 = get_layer_multistream( filters, dimension, 3 )( input_1 )
            layer_multistream_2 = get_layer_multistream( filters, dimension, 3 )( input_2 )
            layer_multistream_3 = get_layer_multistream( filters, dimension, 3 )( input_3 )
            layer_multistream_4 = get_layer_multistream( filters, dimension, 3 )( input_4 )

            layer_concatenated = concatenate( [layer_multistream_1, layer_multistream_2,
layer_multistream_3, layer_multistream_4] )

            layer_merged = get_layer_merged( filters * 4, ( 128, 128, filters * 4 ), 7 )(
layer_concatenated )

            layer_last = get_layer_last( filters, ( 128, 128, filters * 4 ), 1 )( layer_merged )

            model = Model( inputs = [input_1, input_2, input_3, input_4], outputs = [layer_last] )

            # Compile model
            optimizer = RMSprop( lr = 0.1 ** 5 )
            model.compile( loss = mean_distance, optimizer = optimizer, metrics =
['mean_squared_error', mean_different_angle, mean_distance] )

            return model
```

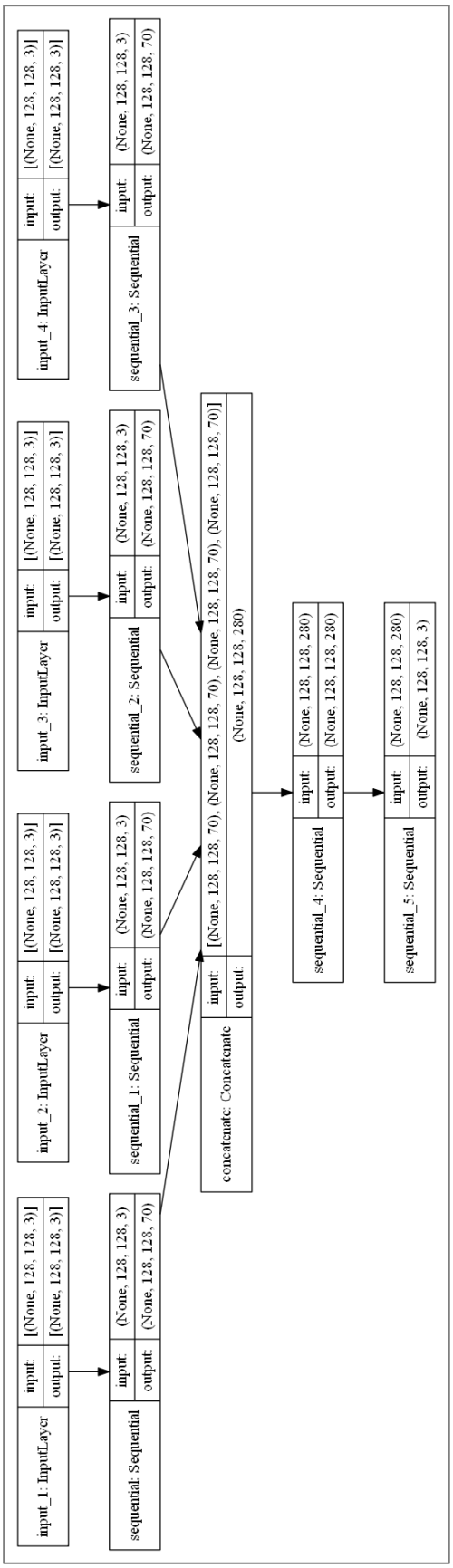
Appendix 2 – Data generator

```
import sys
import numpy as np
from tensorflow.keras.utils import Sequence
from matplotlib import pyplot as plt
from matplotlib.colors import hsv_to_rgb
from matplotlib.colors import rgb_to_hsv

class Data_Generator_Cross_Grey_N_Zadjusted_PNG( Sequence ):
	def __init__( self, folders, batch_size ):
		self.folders = folders
		self.batch_size = batch_size
		self.rgb_adjustements = [
			[ '_gamma_adjustment', 0.6 ],
			[ '_gamma_adjustment', 0.7 ],
			[ '_gamma_adjustment', 0.8 ],
			[ '_gamma_adjustment', 0.9 ],
			[ '_gamma_adjustment', 1.0 ],
			[ '_gamma_adjustment', 1.1 ],
			[ '_gamma_adjustment', 1.2 ],
			[ '_gamma_adjustment', 1.3 ],
			[ '_gamma_adjustment', 1.4 ],
			[ '_saturation_adjustment', -0.2 ],
			[ '_saturation_adjustment', -0.1 ],
			[ '_saturation_adjustment', 0.0 ],
			[ '_saturation_adjustment', 0.1 ],
			[ '_saturation_adjustment', 0.2 ],
			[ '_value_adjustment', -0.2 ],
			[ '_value_adjustment', -0.1 ],
			[ '_value_adjustment', 0.0 ],
			[ '_value_adjustment', 0.1 ],
			[ '_value_adjustment', 0.2 ],
		]

	def __len__( self ):
		return int( np.ceil( ( len( self.folders ) * len( self.rgb_adjustements ) ) 
/ self.batch_size ) )

	def __getitem__( self, index ):

		# Set right start and end index adjusting for RGB adjustements methods
		start = ( index % len( self.folders ) ) * self.batch_size
		end = ( ( index % len( self.folders ) ) + 1 ) * self.batch_size
		folders = self.folders[start:end]

		# Retrieve the right adjustement method and its corresponidng value
		method_counter = int( index / len( self.folders ) )
		method = self.rgb_adjustements[method_counter][0]
		value = self.rgb_adjustements[method_counter][1]

		# Retrieve inputs and output
		inputs, output = self._load_light_field( folders, method, value )

		return inputs, output

	def _load_light_field( self, folders, method, value ):
		input_1 = np.empty( ( self.batch_size, 128, 128, 3 ), np.float32 )
		input_2 = np.empty( ( self.batch_size, 128, 128, 3 ), np.float32 )
		input_3 = np.empty( ( self.batch_size, 128, 128, 3 ), np.float32 )
		input_4 = np.empty( ( self.batch_size, 128, 128, 3 ), np.float32 )
		output = np.empty( ( self.batch_size, 128, 128, 3 ), np.float32 )

		for i, folder in enumerate( folders ):

			cam030_file = folder + 'input_Cam030.png'
			cam031_file = folder + 'input_Cam031.png'
```

```
                                cam032_file = folder + 'input_Cam032.png'
                                cam039_file = folder + 'input_Cam039.png'
                                cam040_file = folder + 'input_Cam040.png'
                                cam041_file = folder + 'input_Cam041.png'
                                cam048_file = folder + 'input_Cam048.png'
                                cam049_file = folder + 'input_Cam049.png'
                                cam050_file = folder + 'input_Cam050.png'

                                cam030 = getattr( self, method, None)( self._load_png_rgb(
cam030_file ), value )
                                cam031 = getattr( self, method, None)( self._load_png_rgb(
cam031_file ), value )
                                cam032 = getattr( self, method, None)( self._load_png_rgb(
cam032_file ), value )
                                cam039 = getattr( self, method, None)( self._load_png_rgb(
cam039_file ), value )
                                cam040 = getattr( self, method, None)( self._load_png_rgb(
cam040_file ), value )
                                cam041 = getattr( self, method, None)( self._load_png_rgb(
cam041_file ), value )
                                cam048 = getattr( self, method, None)( self._load_png_rgb(
cam048_file ), value )
                                cam049 = getattr( self, method, None)( self._load_png_rgb(
cam049_file ), value )
                                cam050 = getattr( self, method, None)( self._load_png_rgb(
cam050_file ), value )

                                cam030 = self._rgb_2_grey( cam030 )
                                cam031 = self._rgb_2_grey( cam031 )
                                cam032 = self._rgb_2_grey( cam032 )
                                cam039 = self._rgb_2_grey( cam039 )
                                cam040 = self._rgb_2_grey( cam040 )
                                cam041 = self._rgb_2_grey( cam041 )
                                cam048 = self._rgb_2_grey( cam048 )
                                cam049 = self._rgb_2_grey( cam049 )
                                cam050 = self._rgb_2_grey( cam050 )

                                input_1[i,:,:,0] = cam039
                                input_1[i,:,:,1] = cam040
                                input_1[i,:,:,2] = cam041

                                input_2[i,:,:,0] = cam030
                                input_2[i,:,:,1] = cam040
                                input_2[i,:,:,2] = cam050

                                input_3[i,:,:,0] = cam031
                                input_3[i,:,:,1] = cam040
                                input_3[i,:,:,2] = cam049

                                input_4[i,:,:,0] = cam032
                                input_4[i,:,:,1] = cam040
                                input_4[i,:,:,2] = cam048

                                output[i,:,:,0:3] = self._load_normal_map( folder +
'input_Cam040_normal.png' )

                        return [input_1, input_2, input_3, input_4], [output]


                def _rgb_2_grey( self, rgb ):
                        r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
                        gray = 0.2989 * r + 0.5870 * g + 0.1140 * b

                        return gray

                def _load_png_rgb( self, file, gamma = 2.2 ):
                        img = plt.imread( file )
```

```
            return img

        def _load_normal_map( self, file ):
            n = plt.imread( file )

            return n

        def _gamma_adjustment( self, rgb, gamma ):
            # Adjust gamma value of rgb input
            rgb = rgb.astype( complex ) ** ( 1 / gamma )
            rgb = np.real( rgb )
            rgb = np.clip( rgb, 0, 1 )

            return rgb

        def _saturation_adjustment( self, rgb, saturation ):
            # Adjust saturation of rgb input
            hsv = hsv_to_rgb( rgb )
            hsv[:,:,1] += saturation
            hsv = np.clip( hsv, 0, 1 )
            rgb = hsv_to_rgb( hsv )

            return rgb

        def _value_adjustment( self, rgb, value ):
            # Adjust saturation of rgb input
            hsv = hsv_to_rgb( rgb )
            hsv[:,:,2] += value
            hsv = np.clip( hsv, 0, 1 )
            rgb = hsv_to_rgb( hsv )

            return rgb
```

Appendix 3


input_1: InputLayer
input: [(None, 128, 128, 3)]
output: [(None, 128, 128, 3)]
input_2: InputLayer
input: [(None, 128, 128, 3)]
output: [(None, 128, 128, 3)]
input_3: InputLayer
input: [(None, 128, 128, 3)]
output: [(None, 128, 128, 3)]
input_4: InputLayer
input: [(None, 128, 128, 3)]
output: [(None, 128, 128, 3)]
sequential: Sequential
input: (None, 128, 128, 3)
output: (None, 128, 128, 70)
sequential_1: Sequential
input: (None, 128, 128, 3)
output: (None, 128, 128, 70)
sequential_2: Sequential
input: (None, 128, 128, 3)
output: (None, 128, 128, 70)
sequential_3: Sequential
input: (None, 128, 128, 3)
output: (None, 128, 128, 70)
concatenate: Concatenate
input: [(None, 128, 128, 70), (None, 128, 128, 70), (None, 128, 128, 70), (None, 128, 128, 70)]
output: (None, 128, 128, 280)
sequential_4: Sequential
input: (None, 128, 128, 280)
output: (None, 128, 128, 280)
sequential_5: Sequential
input: (None, 128, 128, 280)
output: (None, 128, 128, 3)