TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

AI-driven analysis and image acquisition of in-vivo neuronal network activity

Authors	Bastian Krämer, Michal Klein, René Romen, Xingying Chen
Mentor(s)	Prof. Dr. Gil Westmeyer, Dr. Tobias Lasser, Anca Stefanoiu
	TUM Chair of Biological Imaging
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Jul 2019

Abstract

Zebrafish larva is optimal for studying the stimulus on neurons because its body, including its, brain is transparent. We want to reconstruct a 3D model with light field microscopy images efficiently. The first step is to find a decent depth estimation given light field images.
To increase the size of training data, we generated simple geometric objects (speres, cones, rhomboids, etc.), simulated them and computed the targets (depth estimations). Multitude of ata augmentation techniques from EPINET were tried out, but they are not suitable since our images are gray-scale and the data doesn't fit into RAM after applying these techniques.
We explored different network architectures, but quickly abandoned the architectures with 3D convolutions, because they had worse results than the ones with 2D convolutions. EPINET had very promising results on the light field benchmarks, but the pre-trained version fails on microscopy data because the baselines of these two images are very different. We had also reduced the number of blocks in subnetworks since our images are have very small spatial dimension, achieving the best mean squared error and bad pixel ratio on the test data. The predictions on real data are worse than Lenslet network and Views Network. All networks can somewhat capture the structure of fisheye, but the biological details are blurred since the training data only contain simple geometric objects. We also phrased the problem as a classification task, implemented the classification versions of several networks, but they do not generalize to the real dataset.
Although the results are far from satisfactory, we still think that a better depth estimation is possible by improving the quality of the training data. One option is to acquire more real data and calculate their depth estimation, but it is rather challenging since there is a high probability to get a blurred picture from the microscopy. A more realistc way is to introduce different grades of overlapping and transparency as well as some noise, making it biologically more realistic. Our networks are already capable of predicting the basic structure of the fisheye, they can definitely be improved if the training data is more heterogeneous.

Contents

1 Introduction

1.1 Problem definition and goals of the project

The zebrafish larva has many very useful properties for research. Neuronal activities can be measured on zebrafish larva since its body and embryos are transparent. This helps to learn a lot about the functioning of the brain and that is why it is so useful for the Westmeyer Laboratory. You can work with an experimental system where you can control the input, each component and the behavioral output really well.

The main goal of this project is to find an algorithm to compute the light field reconstruction based on light field images. In Fig. 1), computing the forward model and the back projection is not so time-consuming while solving the inverse problem takes hours. With a depth estimate and a reconstruction guess from the back projection, a light field reconstruction in real-time should be possible. So the first step is to learn the depth estimate from the light field images. Based on the quality of the depth estimate and remaining time, we can decide whether we proceed with the reconstruction.

In the first part of our paper, we provide you with some theoretical background like terminologies, the data we had and the state of approaches. After that, we explain our approaches and show you the experimental results. In the last part, we summarize and give possible ideas for further improvements.

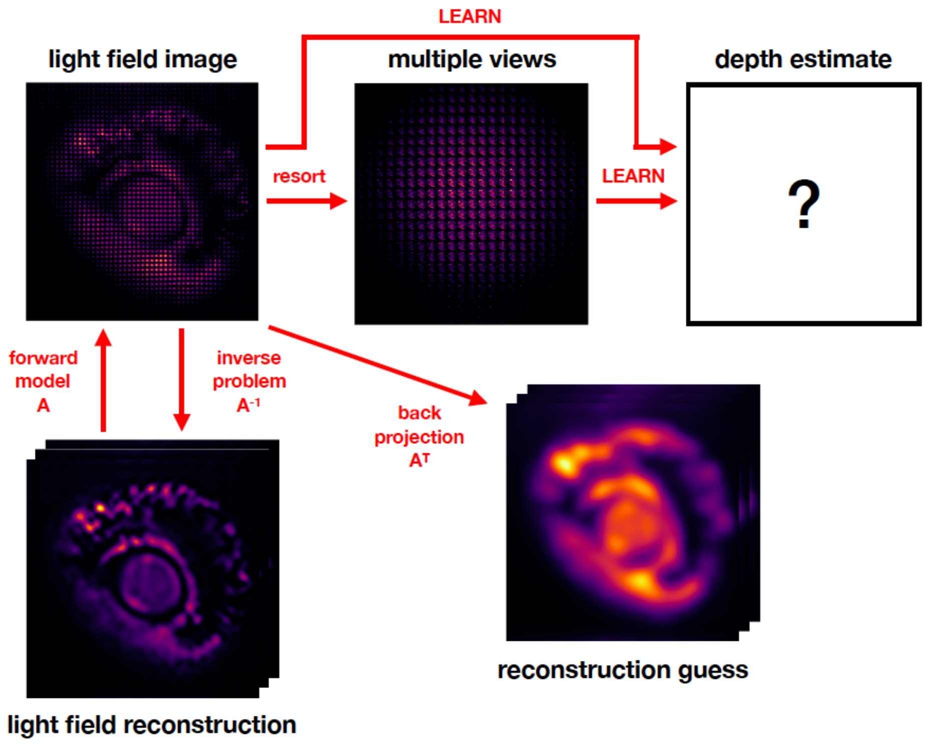

Figure 1: Project overview

2 Theoretical Background

2.1 Light Field Microscopy

Light field microscopy provides a method to acquire scanless volumentric images by placing by putting a microlens array at the native image plane of the microscope [2]. The resulting light field microscope image contains spatial and angular information of the observed object. Different from conventional microscope images, light field microscope images can be used to compute the 3D model and the re-focusing to different heights with higher quality.
Fig. 2 from Broxten et al. explains how light travels through the microlens array and gives an intuition of a light field image. After moving the object from the native image plane, light rays concentrated in the middle change their trajectory and scatter on different part of microlenses.

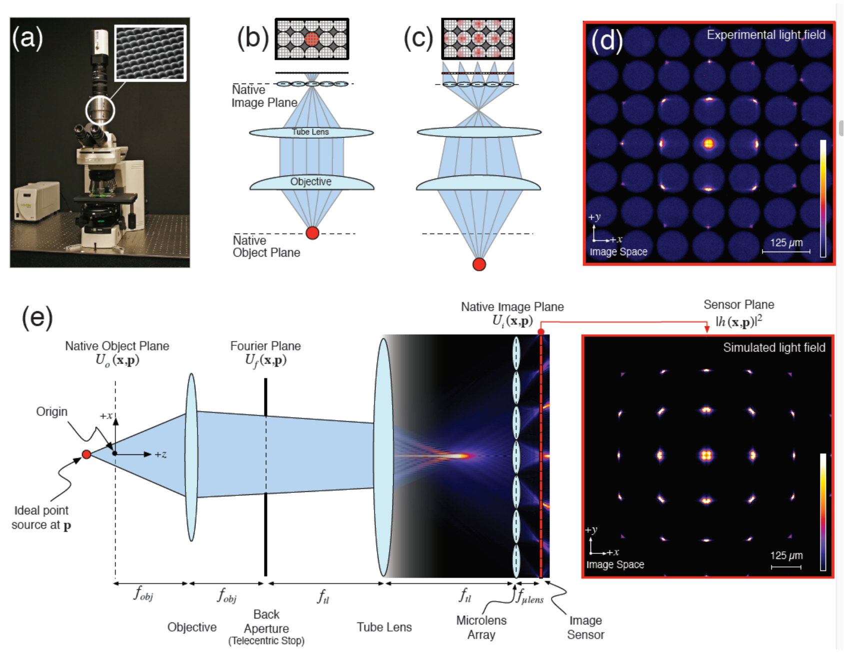

Figure 2: (a) A microscope with a microlens array (b),(c) The red point stands for a point source generating illumination. The red regions on the top shows the intensity of illumination arrived at the sensor plane. (d), (e) A real light field compared to a simulated one [2].

2.2 Light Field Microscopy Data

A light field image is a 4D image where each pixel is associated with four coordinates (x, y, s, t). Where (x, y) are interpreted as spatial and (s, t) as angular coordinates. The images in our generated datasets all have a spatial and angular resolution of 43×43 and 23×23 respectively. Therefore, the whole light field image contains $43 \cdot 43 \cdot 23 \cdot 23 = 978121$ pixels. Fig. 3 shows two ways how light field images can be mapped to 2D images. First, by fixing the two angular axes to a fixed value the spatial coordinates form an image which is defined as a view. We receive the views by re-arranging all sub-aperture images where the position of the view is decided by the angular coordinates. Each view captures the whole scene from an angle which is determined by the angular coordinates. Therefore, the views image shows how the whole scene looks from each angle. If we instead fix the spatial coordinates and look at all angular coordinates, the resulting image is capturing one spatial position in the scene from each angle. If we reorder them according to the spatial coordinates into an image, we get the lenslet image. The lenslet image shows how one pixel in the scene looks from each angle for each pixel. As a result, the whole scene can be seen in the lenslet. Lenslet and views image are therefore just a reordering of the light field image. Transforming between the representations is easy and fast. Therefore, our networks can take any of them as input.

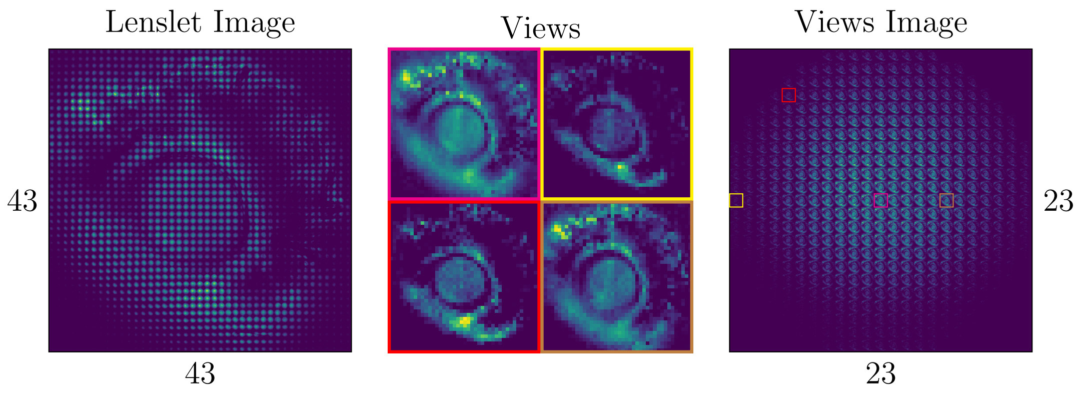

Figure 3: The lenslet image on the left is the photo of the microlens array mounted on top of the microscope. It is therefore a photo of a grid of 43×43 microlenses, where each microlens collects the light of one pixel of the spatial domain from each angle. The views image on the right is constructed from the lenslet image by arranging 23×23 views according to their angular coordinates. The position of the four views in the center is marked in the views image. The magenta view for example has the angular coordinates $(12, 12)$, i.e. it is the central view. A view can be constructed from the lenslet by taking the pixel at the angular coordinates of the view in each lenses image, i.e. take the middle pixel of each of the 43×43 subimages in the lenslet to gain the central view.

2.3 Datasets

For the project, we were provided with three real data points and a realistic light field simulation [2]. The real data consist of volume and light field of a fisheye, an organoid, and spheres. We used the volume to create a depth prediction. Unfortunately, that turned out rather bad since the volume contains a lot of noise. For training of our networks, we had to create a dataset using the simulation. For this, we first extended the simulation by creating phantom images that contain a scene. We used scenes with randomly placed objects like spheres, boxes, and cylinders. When placing them we prevented overlaps of more than 15% and we applied a 3d Gaussian filter. The phantom was then fed to the simulation which calculates the light field. The ground truth depth map can be directly calculated from the phantom. We created light field images of size 989×989 with a depth range of $[-50, 50]$ and depth step of 5. This means the light field distinguishes 21 depths which is close to the 17 to 19 in our real data. The lense spacing was set to 23 pixels. The resulting light fields have a size of $23 \times 23 \times 43 \times 43$, where the first two are the angular dimensions and the second two the spatial dimensions.

2.4 Data augmentation

At the beginning, we did not have many data and generating data takes quite a long time, so we decided to use some data augmentation techniques to get more data. Therefore we implemented the same techniques used in EPINET [14], which are view shifting (on the sub-images - 9x9, 7x7 and 5x5), scaling by $1/N$ ($N = 1, 2, 3, 4$), rotating by 90, 180 and 270 degrees, flipping, color scaling [0.5, 1.2] and gamma adjustment [0.8, 1.2]. With these techniques, we could increase the number of images to as many as we wanted because we selected the values randomly. Unfortunately, it turned out that most of the techniques are not useful for our problem, so in the end, we only used the rotation.

2.5 Deconvolution and artifacts

Deconvolution layers are often used in deep CNN architectures. A well-known problem is they often introduce checkerboard artifacts into the result. Odena et al. [10] show that this problem was found in many research papers. The reasons for these artifacts are usually poor choices for stride and kernel size of layers. To prevent the problem, the stride should divide the kernel size along every dimension. This ensures that all output pixels sum up the same number of activations. But even in this case, there is the possibility that the kernel learns weights in a way that introduce these artifacts. While in theory, the network can learn to prevent them. This will reduce the capacity of the network. Therefore in the paper, they suggest to smooth out the artifacts using convolutions with stride one and a kernel size larger than the artifacts. Deconvolutions can also be replaced by nearest neighbor or bilinear interpolation followed by convolutions.

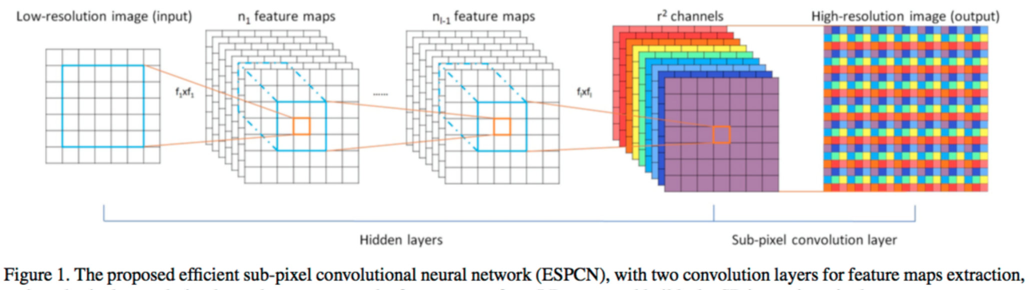

Figure 1. The proposed efficient sub-pixel convolutional neural network (ESPCN), with two convolution layers for feature maps extraction, and a sub-pixel convolution layer that aggregates the feature maps from LR space and builds the SR image in a single step.

Figure 4: Sub-pixel layer defined in [12].

2.6 Hyperparameter optimization

We have implemented three different optimization strategies: grid search, random search, and greedy search. Grid search tries out all possible combinations, leading to combinatorial explosion. Random search samples from a predefined space, e.g. uniform or

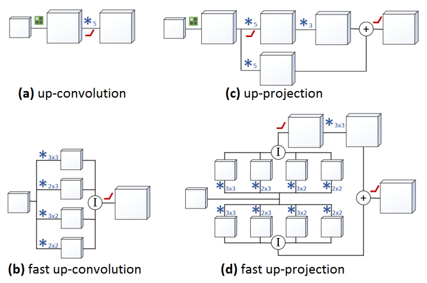

Figure 2. **From up-convolutions to up-projections. (a)** Standard up-convolution. **(b)** The equivalent but faster up-convolution. **(c)** Our novel up-projection block, following residual logic. **(d)** The faster equivalent version of (c)

Figure 5: Different ways of upsampling. Image courtesy of [8].

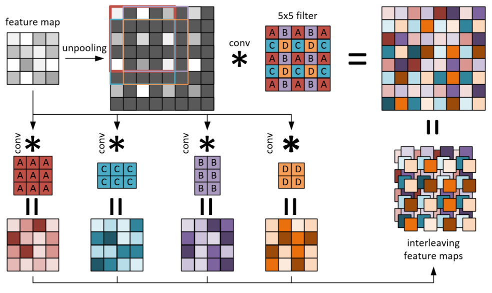

Figure 3. **Faster up-convolutions.** Top row: the common up-convolutional steps: unpooling doubles a feature map's size, filling the holes with zeros, and a 5 × 5 convolution filters this map. Depending on the position of the filter, only certain parts of it (A,B,C,D) are multiplied with non-zero values. This motivates convolving the original feature map with the 4 differently composed filters (bottom part) and interleaving them to obtain the same output, while avoiding zero multiplications. A,B,C,D only mark locations and the actual weight values will differ

Figure 6: Efficient way of doing faster up-convolution. Image courtesy of [8].

log-uniform for numerical values, for a predefined number of steps. Greedy search optimizes parameters in a certain order by selecting the best value at each step and fixing it for the next step. We have tried optimizing different parameters, including but not limited to: optimizer, learning rate, kernel size, number of filters, and pooling type.

2.7 Losses

We found a few quite interesting loss functions which might help optimize our network - especially sharpen the edges of the objects in our prediction since we had many problems with them. We implemented the following three loss functions.

1. VommaNet [9]

$$l_{MAE} = \sum_{i=1}^{N} \frac{|D_i|}{N}; l_{grad} = \sum_{i=1}^{N} \frac{|\nabla_x(D_i)| + |\nabla_y(D_i)|}{N}, \tag{1}$$

$$l_{normal} = 1 - \sum_{i=1}^{N} \frac{cos < \overrightarrow{n}_i^d, \overrightarrow{n}_i^g >}{N}, \tag{2}$$

$$\overrightarrow{n}_i^d = (-\nabla_y d_i, -\nabla_x d_i, 1), \overrightarrow{n}_i^g = (-\nabla_y g_i, -\nabla_x g_i, 1), \tag{3}$$

$$L_{VommaNet} = \lambda_1 l_{MAE} + \lambda_2 l_{grad} + \lambda_3 l_{normal}, \tag{4}$$

$D_i = d_i - g_i$ is the difference between network estimation and its ground truth at the i^{th} pixel. N is the total number of pixels, ∇_x is the spatial gradient in x-axis, ∇_y is the spatial gradient in y-axis. λ_i, $i \in \{1, 2, 3\}$ are weights. The first term l_{MAE} is the mean absolute error. The second term aims to penalize the error around the edges. The last measures and penalizes the accuracy of the normal to the surface of the prediction concerning the ground truth.

2. Loss Blog [1]

$$L_{Blog} = \frac{1}{N}\sum_{i=1}^{N} d_i^2 - \frac{1}{2N^2}(\sum_{i=1}^{N} d_i)^2 + \frac{1}{N}\sum_{i=1}^{N}[(\nabla_x d_i)^2 + (\nabla_y d_i)^2], \tag{5}$$

$d_i = log(Predicted\ depth\ map) - log(Ground\ truth\ depth\ map)$ for pixel i. N is the total number of pixels. The first part is the mean squared error of the logs. The second function credits errors that are in the same direction, so it is more important for the prediction whose mistakes are consistent. The last term penalizes the difference in gradients.

3. Loss Self-Build:

$$l_{grad} = \frac{1}{2N}\sum_{i=1}^{N}(\nabla_x d_i + (\nabla_x d_i \circ \lambda_1 \cdot e_i) - (\nabla_x g_i + \nabla_x g_i \circ \lambda_1 \cdot e_i) \tag{6}$$

$$+ \frac{1}{2N}\sum_{i=1}^{N}(\nabla_y d_i + (\nabla_y g_i \circ \lambda_1 \cdot e_i) - (\nabla_y g_i + \nabla_y g_i \circ \lambda_1 \cdot e_i), \tag{7}$$

$$L_{Self} = \lambda_2 \sum_{i=1}^{N} \frac{(D_i)^2}{N} + \lambda_3 l_{grad}, \tag{8}$$

$\nabla_x d_i$ are the gradients of the network estimation, $\nabla_x g_i$ of the ground truth, e_i is the edge mask at pixel i, $\lambda_i \in \{1, 2, 3\}$ are weights. With Loss Self-Build we wanted to weight the edge mask highly so that the edges become smoother.

Unfortunately, all three loss functions - even if the idea behind them seems very reasonable - did not help at any network because our data for example don't have reflection and texture-less area.

3 Networks

The depth estimation can be treated either as a classfication task or a regression task. In the classification setting, we tried to predict per-pixel probabilities of falling into predefined depth bins. Classification setting can also be seen as a form of semantic segmentation, with the difference being that our classes also have an order. In this setting, we do not make use of this additional information; the number of depth bins was set to 50, since it seemed like a good middle ground between the granularity of the prediction and hardness of the task. As in semantic segmentation, pixel's value is heavily influenced by the values of neighboring pixels. To get a better prediction, a solution based on conditional random fields (CRF) is usually used, either as a post-processing step [7] or even during training. Our idea was to start with the former and see if we can achieve positive results and if so, switch to the latter approach. However, during our project, we have found out that the classification approach didn't yield promising results (as seen below), so we have primarily focused on the regression problem. All of our depth maps are 256x256. All the figures from the classification networks were not post-processed by CRF.

3.1 Epinet

Epinet [14] proposes an architecture for depth estimation using light field images. To prevent overfitting problem, they use general data augmentation techniques such as scaling and flipping. Moreover, they develop own strategies for light field data, shifting center viewpoints and rotation.
Instead of the whole angular information, only 4 angular directions are fed into the network. They use a multi-stream network, as shown in 7, to process the stacked information of viewpoints. Each subflow in the multistream network consists of three convolutional blocks. The subflows are merged to pass the eight convolutional blocks. The first few blocks are with a Conv-Relu-Conv-BN-Relu structure, whereas the last layer has a Conv-Relu-Conv structure.
Their training data is from 4D Light Field Benchmark [4] with reflection, refraction and textureless regions masked out.

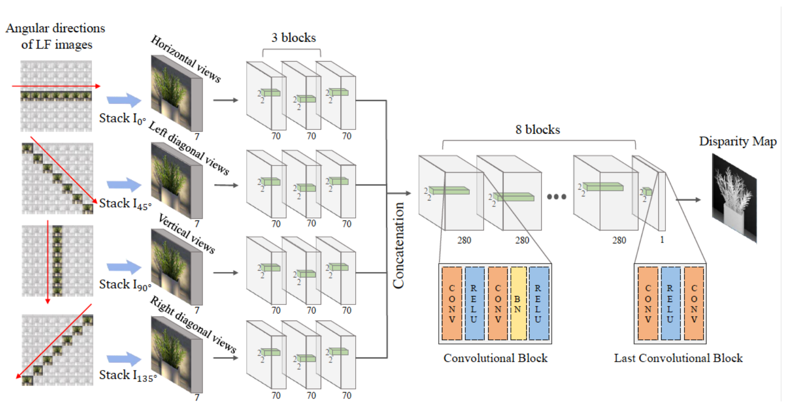

Figure 7: EPINET structure [14]

At first, we implemented EPINET as it was done it the paper. Unfortunately, it turned out that we can not just directly use their structure and parameters as the results look quite bad 8. Then, we downloaded its pre-trained version [13]. We tried the pre-trained EPINET with and without frozen weights, as suggested by general transfer learning techniques [6]. But as shown in the result, their network was not really suitable for our problem 8.

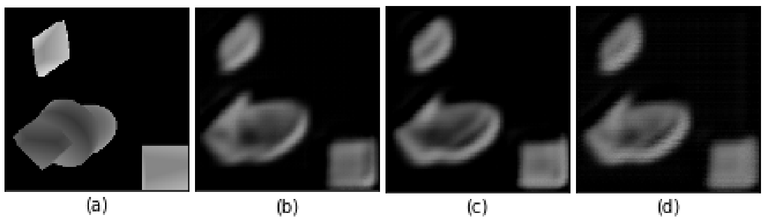

Figure 8: Epinet - ground truth (a), from scratch (b), without frozen weight (c), with frozen weights (d).

We decided to optimize the EPINET structure. In the end, we did not use data augmentation techniques since they were not useful for us. More specifically, EPINET used the view shifting because they could not fit all the sub-aperture images in the memory due to their high spatial dimension. Since our spatial dimension is very low, this was not necessary for us. Furthermore, the networks improve with more number of input views [14]. We did not use the rotations since it did not improve the network in terms of generalization and slowed down the training process. The less angular directions and therefore subnetworks we used, the worse the results were. We decided to use the horizontal, vertical and diagonals views at the end. Each subnetwork in the multistream network consists of three convolutional blocks. The subflows are merged to pass the five convolutional blocks. The first few blocks are with a Conv-Relu-Upsampling-Conv structure, whereas the last layer has a Conv-Upsamping-Conv structure. Parameters can be seen in the Appendix. In the end, the network had 2.4 million parameters. Although the loss converges very nicely, unfortunately, the network does not work well for our problem. It estimates the shape of the objects more or less well but it is still very blurry around those. Furthermore, it does not truly recognize the depth in a good way 9. After two-third of our project time without getting good results albeit a lot of trying, we decided to concentrate on other networks.

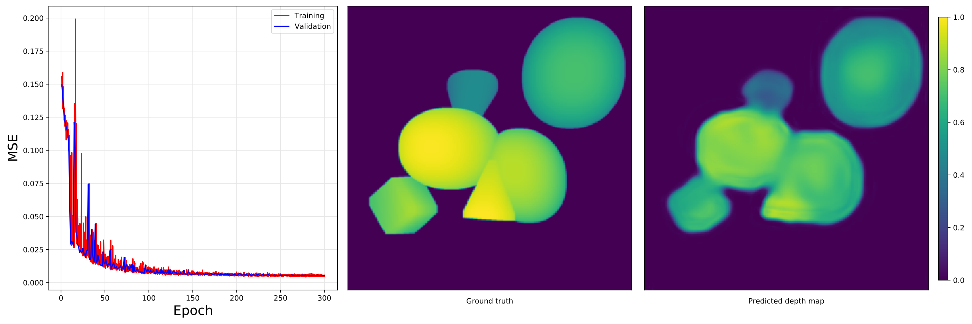

Figure 9: Traning/validation curve and predition on an image from test set.

3.2 3D Convolution

Another approach we used was 3D-Convolution. We built two subnetworks with another angular resolution as a channel. In the subnetworks, we used different number of layers, kernels as well as filters and tried different network architectures. In the last version, each subnetwork contained five 3D convolution layers, then the subnetworks were concatenated, upsampled with deconvolution and in the end smoothed with another 3D convolution layer. The activation function is 'Leaky ReLU', the learning rate is 10^{-4} and error function is the mean absolute error. In general, it turned out that all networks with 3D-Convolution had much worse results than the one with 2D-Convolution (Fig.10). Therefore we had decided to focus our attention on 2D-Convolution.

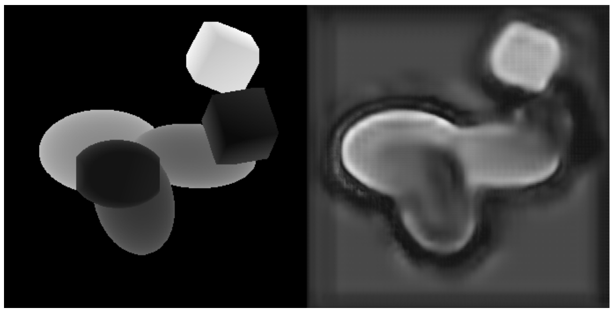

Figure 10: 3D-Convolution, on the left: ground truth, on the right: network estimation.

3.3 Lenslet Network

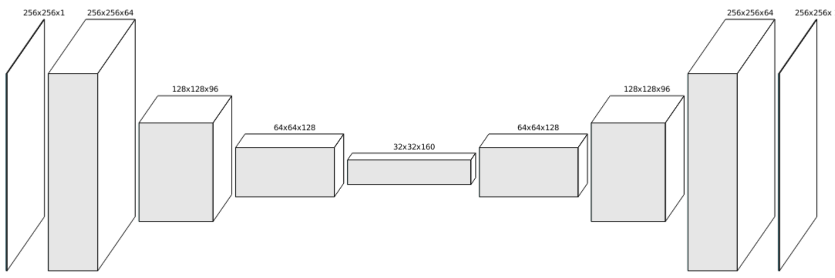

Figure 11: General encoder-decoder CNN style architecture.[1]

[1]This figure has been created by `https://github.com/gwding/draw_convnet`.

This type of network operates on the lenslet image, whose original size 989 × 989 was downsampled to 512 × 512 using nearest-neighbor interpolation. It follows a general encoder-decoder CNN architecture as in Fig. 11, which is a de-facto a standard in similar [5, 16] or different tasks, such as biomedical image segmentation [11]. The encoder-decoder architecture consists of 2 parts: contracting and expansive. The contractive part downsamples the spatial dimensions of the input, while increasing the number of features every layer (usually number of channels of a convolutional block). The expansive part upscales the spatial dimension, reducing the number of features, until the desired dimensions are reached. In most of the encoder-decoder networks, some form of skip connections are present to increase the information/gradient flow between the encoder and the decoder, e.g. channels from the encoder are concatenated to the channels of the decoder with the corresponding spatial dimensions. Other networks [3] try to predict their target at smaller resolutions and concatenate the intermediate prediction to the channels of next upsampling layer. When trying out this approach, we observed no significant difference in terms of the quality of the prediction. Our encoder consists of 4 parts, each followed by a max-pooling layer. The first two parts consist of two *inception-resnet-v2-35x35* blocks each, the second two parts consist of two *inception-resnet-v2-17x17* blocks. Each of these blocks has been scaled in terms of number of channels since the original [15] was too large for our dataset. It shall be noted that we have also experimented with a scaled-down version of the Inception-ResNet-v2, but the objects in the predicted depth map appeared to fused together. This can either arise from the *stem* block, 8 block or the conversion blocks (*35to17* or *17to8*). We have decided not to investigate this matter further, given the shortness of time. The encoder transforms the input into 64 × 64 × 512 features, which are then fed into the decoder.
The decoder consists of 2 upsampling layers, the first layer having 256 channels, the second 128. Furthermore, we also predict the intermediate depth maps (of dimensions 64x64, 128x128) and concatenate these predictions to the channels of the next layer, similarly as in [3].
In the course of the network design, we have tried different upsampling strategies, as mentioned in Section 2.5. For the lenslet network, we observed that in terms of reducing checkerboard artifacts. They have opted for the up-projection as our go-to upsampling block because the fast up-convolution and fast up-projections layers introduce too many parameters, which do not improve our results.

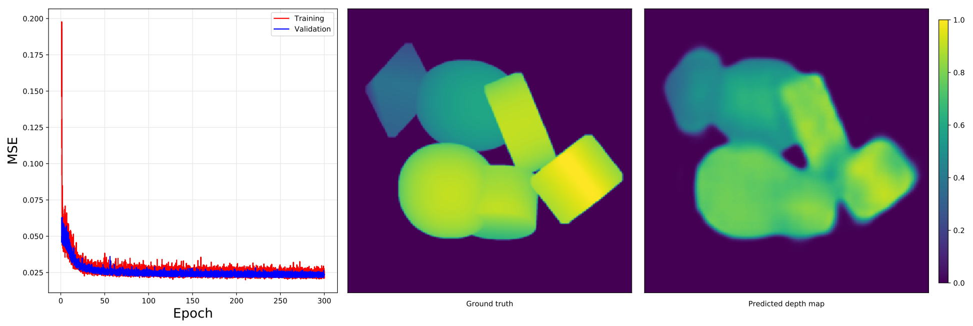

Figure 12: Traning/validation curve and predition on an image from test set.

3.4 Views Network

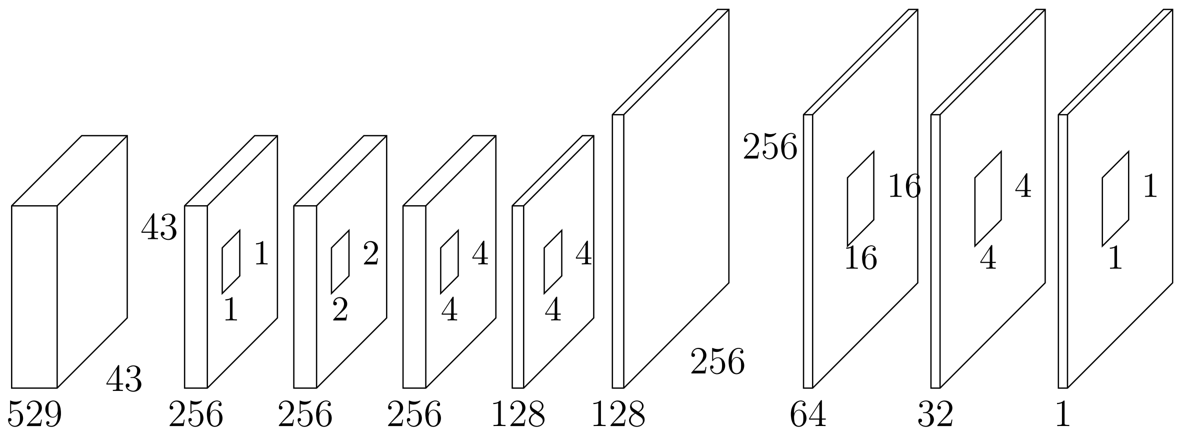

Figure 13: Left to right, Input, 4 convolution layers, upscaling by nearest neighbor interpolation, 3 convolution layers.

The views network receives the full light field image as input. The light field is preprocessed by extracting all views and stack them as image channels. Therefore the input for our network is an image of dimension 43×43 with 23^2 channels, on which we can directly apply 2D convolutions. The network architecture is shown in figure 13. The views stack first goes trough 4 convolutions. Then we upscale the channels by nearest neighbor interpolation. After upscaling a convolution with a large 18×18 kernel smooth out upscaling errors. Finally, 2 more convolutions are used to reduce the channel size to 1 and we receive as output the predicted depth map. The inspiration from the network comes from EPINET (see section3.1). EPINET also uses stacked views as input but instead of using the full light field image only a subset is actually used. In the views network, the whole light field is given as input. Furthermore, the first layer is a 1×1 convolution which allows the network to select the channels that it wants to use, i.e. it can learn which views are the most important ones. EPINET then uses one subnetwork for each input stack, combines the outputs and feeds it to another network. Here we always work on the full light field instead. The upscaling by nearest neighbor interpolation and the following convolution layer were chosen because they introduced no checkerboard artifacts as discussed in section 2.5. Each layer except the last one uses a leaky ReLU activation function. The kernel parameters are initialized with the glorot uniform distribution and the bias with zero. The network has around 4.1 million parameters. The network was trained using the ADAM optimizer with a learning rate of 10^{-6}. As error function, we used the absolute error between the predicted depth map and the ground truth depth map rescaled to 256×256 pixels.

$$l\left(x,y\right) = \sum_{i=1}^{256} \sum_{i=1}^{256} \left| (y)_{i,j} - (f\left(x\right))_{i,j} \right|$$

The loss function is then the average over the batch of size 32.

$$L\left(X,Y\right) = \frac{1}{32} \sum_{i=1}^{32} l\left(X_i, Y_i\right)$$

Since the gradients sometimes exploded we used gradient clipping by norm. In detail, we first grouped the kernel and bias variables of each layer. Then, we applied an L2-norm gradient clipping where we allowed the norm of the gradients to be at most the number of weights in the group.

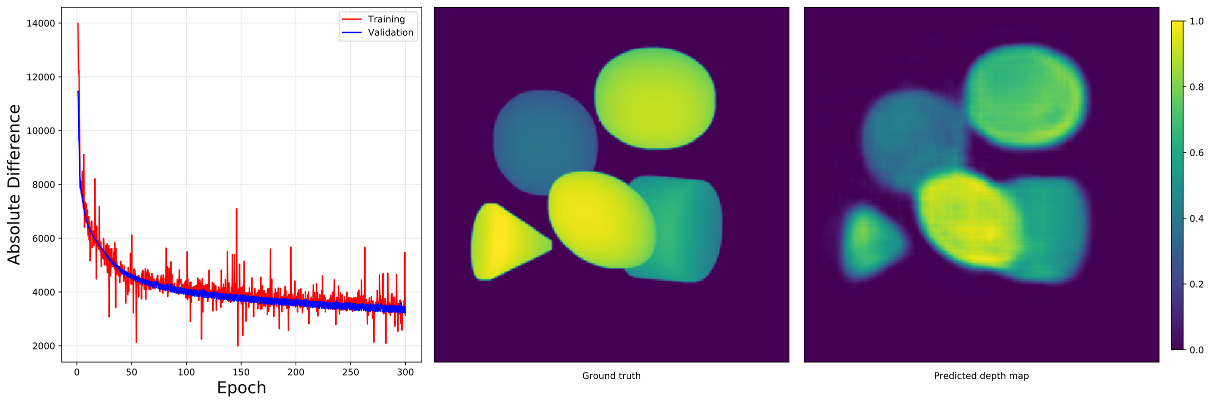

Figure 14: The first figure shows the views network training and validation loss over 300 epochs. The images on the right show the ground truth and predicted depth map for one example from the test set.

3.5 Classification

3.5.1 Lenslet Classification

Since we did not get the results we wanted, we started to implement classification networks. First, we built the 'Lenslet classification' network. In the beginning, we used the same architecture as at the 'Lenslet' network used. The training result looked promising. The data was predicted very well and the loss looked promising. However, the prediction on the real data was not good at all. Especially for the fisheye, the classification network only predicted zeros and ones.
Then we decided to implement fast upconvolution as upsampling technique [8]. The network were too big, so we had to downsize it. The final network architecture is as follows: 2× 35x35 inception block and a max pooling layer, followed by 2× 17x17 inception block and yet another max pooling layer. Then we upsampled the features 2×, followed by 4 convolution blocks and 3 fully connected layers. Parameters can be seen in the Appendix. The final network has around five million parameters. The training data and the loss looked even more promising. But testing the network on the real data had the same problems we had before, even if the MSE and bad pixel ratio increased a little bit (see Fig. 15).

3.5.2 Inception Classification

Another classification network we implemented is inception classification which consists of 3 different types of inception blocks and fast convolutional upsampling blocks. The architecture is (2x) *inception-resnet-v2-35x35* – (2x) *inception-resnet-v2-17x17* – (2x) inception-*resnet-v2-8x8* – (4x) fast convolutional layers – resize layer with nearest neighbor inter-

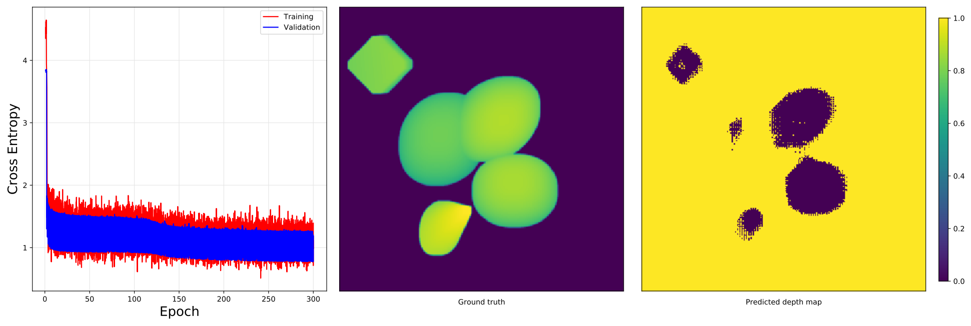

Figure 15: Traning/validation curve and predition on an image from test set.

polation to 256 × 256. The training process resembles the training process of the lenslet classification network. It first learns to predict the central part of one object, then refines the edges. Similar to the lenslet classification, although the predicted depth on the synthetic training data keeps improving, the network fails to give a meaningful depth prediction on the real fisheye data. We also compared the network with and without any fast convolutional layers. Without any fast convolutional blocks, even the prediction on the training data looks quite low resolution since we choose nearest neighbor as the resize strategy. With 4 fast convolutional blocks, the geometry of the fisheye appears and then quickly disappears due to overfitting on the training data. Therefore we have ultimately decided to abandon this network.

3.5.3 Views Classification

Since the Views Network has quite promising results, we tried out its classification version to see whether the depth prediction would be better than other classification networks. We also tried different numbers of fast upconvolutional layers. The more upconvolutional layer it has, more complicate structure it can predict on the real data. But with 3 upconvolutional layers, the prediction quality on the real image decays earlier than the networks with fewer upconvolutional layers. The training on the synthetic data cannot be transferred to the real images.

3.6 Autoencoder

Figure 16: Reconstruction of an EPI. Left: original. Right: reconstruction.

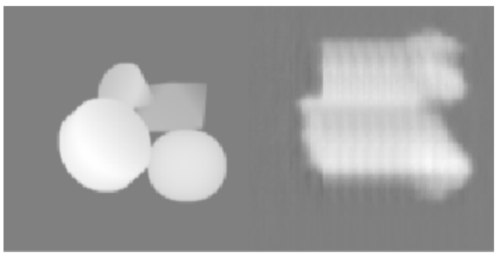

Figure 17: Depth map prediction using 43 × 32 × 66 input. Left: ground truth. Right: predicted depth map.

Lastly, we have tried an entirely different approach. Since EPIs are highly structured, we thought we can capture this structure using a lower-dimensional representation z. We have implemented 2 different autoencoders: convolutional and fully-connected. As our loss, we define $l_{AE}(x, y, \hat{x}, \hat{y}) = \lambda * MSE_{reconstruction}(x, \hat{x}) + (1 - \lambda) * MSE_{prediction}(y, \hat{y})$ and taken $\lambda = 0.6$. For the $MSE_{prediction}$, we downscaled the target depth map to 43×43 and for a specific EPI taken a row corresponding to the fixed spatial dimension as our target y. Size of the latent vector z was 66 and 64 for convolutional and fully-connected AE, respectively. For our training data, we have generated all horizontal EPIs (all 0's removed), yielding ~ 250000 samples. The AEs were trained for only five epochs, the best one (convolutional) achieving $MSE_{reconstruction} = 2.18 \times 10^{-3}$, $MSE_{prediction} = 6.39 \times 10^{-6}$. We then stitched the latent vectors for each sample into $23 \times 43 \times 66$ shaped array and fed them to smaller version of Views network to predict 128×128 depth map. Unfortunately, this approach didn't really work, since the AE didn't manage to fully capture the structure of the EPIs, as seen in Fig. 16.

4 Comparsions

Based on Table 1, it would seem like the EPINET is the winner, both in terms of MSE and BPR on the test set. However, upon further inspection, it can be seen in 18 that its prediction on real data is not as good as Views or Lenslet network. This further reinforces our belief that our generated data is too far from the biological ground truth, which is the ultimate goal of our task. Even on the spheres image, which is not biologically plausible, EPINET does not seem to work very well.
Lenslet network achieves second to best numerical results, but similarly to EPINET, it does not generalize well on the test data (albeit subjectively better than EPINET). Possible reasons for the poor test performance might be the large number of parameters or the upsampling transformation.
The Views network achieves arguably the best results after 300 epochs, despite being the simplest architecture. Furthermore, we noticed that this network achieves better results after prolonged training time. We trained the network for 120 hours. The performance on the test set steadily increased, while the performance on the real data got worse after a certain point. During the end of training the prediction on the fish eye was similar to the epinet prediction.
We can also safely conclude, that Lenslet (cls) network, after 300 epochs, does not manage to produce any meaningful result, which positively correlates with results in Tab. 1. As expected it performs better on the test set, since it comes from the same distribution as the training set. However, on the real test data, it completely fails, as seen in Fig. 18. We omit other two classification networks because Lenslet (cls) is already representative. Our best network, in terms of the real test data, is the Views network. It achieves reasonably good performance in terms of MSE and BPR, compared to Lenslet network.

5 Conclusions

Simulating data to train a network is better than training on light field photography data because of following two observations. First, while inputting our real data into EPINET

		Bad Pixel Ratio					
Network	MSE	$\delta = 0.05$	$\delta = 0.1$	$\delta = 0.15$	$\delta = 0.2$	$\delta = 0.25$	$\delta = 0.3$
Epinet	0.0049958	0.18671	0.08947	0.04304	0.02457	0.01540	0.01078
Views	0.0122427	0.26309	0.18338	0.12022	0.07845	0.05044	0.03348
Lenslet	0.00595414	0.21105	0.11301	0.05586	0.03027	0.01930	0.01290
Lenslet (cls)	26.9925	0.99803	0.99610	0.99521	0.99436	0.99392	0.99370

Table 1: Mean squared error and average Bad Pixel Ratio for test set after 300 epochs.

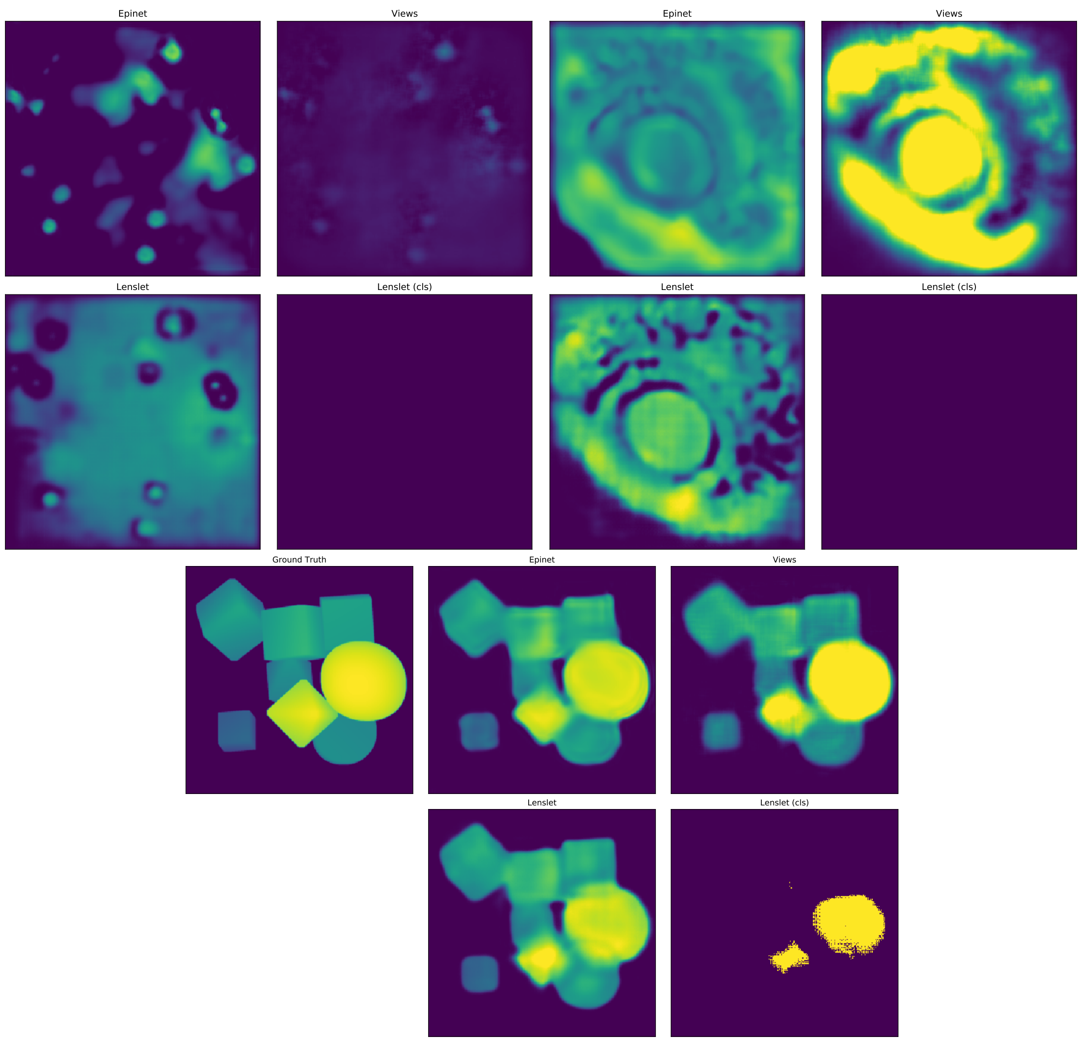

Figure 18: Comparison of predictions of different networks for spheres (a), fish eye (b) and a random image from test set (c).

trained on light field photography data, the result was either unusable or the network saw "double". A possible reason for this is that the baseline of light field microscopy images have narrow baseline compared to light field photography images. Secondly, even small changes in the configuration of generating light field images seems to change the data a lot, making the generalization difficult. We can use the settings exactly as needed while we generate data. The remaining difficulty is to make the scene realistic enough.
While our generated datasets did not enable our networks to generalize to real data, they still outperformed networks trained on light field photography data. This is because the simulation does create realistic light field microscopy images. The problem lies in the objects chosen in the simulation. First of all, we used geometric objects that are not biologically plausible. Despite the fact that they were randomly rotated and placed in the scene, the resulting images looked very similar. Furthermore, our objects were all solid and not transparent and the scene contained no noise. The generated data was too different from the real data that was in our possesion.
There are several approches which are less helpful for the progress. The first one is the data augmentation from EPINET. The techniques for color images did not carry over well to gray scale. Flipping, scaling and rotating could still have been used but ultimately were not used, since the data we had generated already consumed almost our whole RAM. The second one is the research on loss functions. MSE as an error function already suffies to improve the results and the problems other loss functions solve do not match with out simulated data. The third one is the classification version of the problem. Nevertheless, it is just a trial which turned out to be unviable.
Deep learning for depth prediction on light field images is possible, as shown by [14] and [9]. So we modified EPINET to handle light field microscopy images and trained it successfully on our synthetic dataset. The Views and Lenslet networks also achieve similar results on the test set.
While working with light field (LF) microscopy images, we noticed some parallels and differences compare to LF photography images. Both of them benefit from a bigger size of the input data. Therefore we choose to give to our networks as many views as possible as input. Different from LF photograph images, LF microscopy has a much smaller spatial resolution (43×43 in our case) so that the whole light field can be given as input to the network easily. Another difference is the size of depth prediction. EPINET and other networks predict depth map whose resolution is equal to the spatial resolution of the light field. For LF microscopy images, the resolution is too small. Therefore, we aimed for a depth map resolution that is close to the one of the camera used to take the LF microscopy image, in our case 989×989. Due to computational restrictions, we used a even smaller resolution of 256×256, but it is still much larger than the spatial resolution of the light field.
The small spatial resolution were also a hindrance while using networks based on general image processing architectures. They are not able to take advantage of the structure in the data. In LF photography, this is not a problem because the spatial dimension is sufficiently large. Networks which process general images are not able to take advantage of the structure in the data because of small spacial resolution. This is why networks which works good with light field photography images fail to produce reasonable results with light field microscopy images. The small spatial resolution makes upscaling a necessity. Upscaling was a problem at the beginning because it introduced artifacts into the predictions. But

this problem was solved with strategies demonstrated by Odena et al. [10] (described in section 2.5).

6 Future Work

We did not attempt our ambitious goal of predicting the light field reconstruction using a reconstruction guess and the depth estimate. The reason was the time constraint for our project and the lacking performance of the real data samples. Yet we still think a decent estimation can be achieved through further improvement on the training dataset. We suggest creating a more realistic dataset with configurations similar or equal to the real images. For the synthetic data we generated, we can add biologically plausible geometries which are simple for the computation the target depth estimation by hand, such as tubes and rings. Also, we can predefine how they scatter in the space. For example, a sparse distribution of small objects is close to the real data of spheres. The generated data should have a high heterogeneity so that it can generalize to the real data. Ideally, a real dataset with a ground truth depth map would be needed for the training, but often the light field images are given without the depth estimation. Alternatively, a simulation for biologically plausible scenes could help to complete the otherwise very realistic data generation pipeline.

Bibliography

References

[1] Omar Barakat. *Depth estimation with deep Neural networks part 1.* 2018. URL: `https://medium.com/@omarbarakat1995/depth-estimation-with-deep-neural-networks-part-1-5fa6d2237d0d` (visited on 07/20/2019).

[2] Michael Broxton et al. "Wave optics theory and 3-D deconvolution for the light field microscope". In: *Optics express* 21.21 (2013), pp. 25418–25439.

[3] Alexey Dosovitskiy et al. "Flownet: Learning optical flow with convolutional networks". In: *Proceedings of the IEEE international conference on computer vision.* 2015, pp. 2758–2766.

[4] Katrin Honauer et al. "A dataset and evaluation methodology for depth estimation on 4d light fields". In: *Asian Conference on Computer Vision.* Springer. 2016, pp. 19–34.

[5] Po-Han Huang et al. "DeepMVS: Learning Multi-View Stereopsis". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2018.

[6] Andrej Karpathy. *CS231n Convolutional Neural Networks for Visual Recognition.* 2019. URL: `https://cs231n.github.io/transfer-learning/` (visited on 07/20/2019).

[7] Andrej Karpathy. *Image Segmentation with Tensorflow using CNNs and Conditional Random Fields.* 2019. URL: `https://warmspringwinds.github.io/tensorflow/tf-slim/2016/12/18/image-segmentation-with-tensorflow-using-cnns-and-conditional-random-fields/` (visited on 07/26/2019).

[8] Iro Laina et al. "Deeper depth prediction with fully convolutional residual networks". In: *2016 Fourth international conference on 3D vision (3DV).* IEEE. 2016, pp. 239–248.

[9] Haoxin Ma et al. "VommaNet: an End-to-End Network for Disparity Estimation from Reflective and Texture-less Light Field Images". In: *arXiv preprint arXiv:1811.07124* (2018).

[10] Augustus Odena, Vincent Dumoulin, and Chris Olah. "Deconvolution and Checkerboard Artifacts". In: *Distill* (2016). DOI: `11.23915/distill.00003`. URL: `http://distill.pub/2016/deconv-checkerboard`.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: (2015). Ed. by Nassir Navab et al., pp. 234–241.

[12] Wenzhe Shi et al. "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 1874–1883.

[13] Changha Shin. *Epinet: A fully-convolutional neural network using epipolar geometry for depth from light field images.* 2018. URL: `https://github.com/chshin10/epinet` (visited on 07/20/2019).

[14] Changha Shin et al. “Epinet: A fully-convolutional neural network using epipolar geometry for depth from light field images”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 4748–4757.

[15] Christian Szegedy et al. “Inception-v4, inception-resnet and the impact of residual connections on learning”. In: *Thirty-First AAAI Conference on Artificial Intelligence.* 2017.

[16] Jure Zbontar, Yann LeCun, et al. “Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches.” In: *Journal of Machine Learning Research* 17.1-32 (2016), p. 2.

Appendix

Hyperparameter	Epinet	Views	Lenlset	Lenslet (cls)
Epochs	300	300	300	300
Loss	MSE	AE	MSE	Cross Entropy
Batch size	32	32	4	4
Optimizer	RMSProp	Adam	Adam	Adam
Learning rate	10^{-4}	10^{-6}	$5 * 10^{-6}$	$5 * 10^{-6}$
Learning rate decay	None	None	Exp(0.96)	Exp(0.96)
Upsampling	NN	NN	up-projection	up-projection
Pool type	None	None	max	max
Pool size	None	None	2x2	2x2
Activation	ReLU	LeakyReLU	LeakyReLU	LeakyReLU
Regularization	None	None	L_2	L_2

Table 2: Hyperparameters of our networks.