Hand Segmentation and Hand Pose Estimation

Deep Vision Project

Nils Boehringer, Michael Hartmann

August 1, 2019

Abstract

by Michael Hartmann

With the rise of virtual reality the need for fast and reliable gesture recognition is steadily increasing. Since depth cameras are still a rare sight we introduce an approach for two dimensional hand gesture recognition based solely on regular RGB images.

For this application we implemented a system consisting of two *U-Nets*, with the first net being used for detection and segmentation of hands in the picture and a second net for keypoint detection in the cropped output of the first net.

The segmentation net performed well, with an almost real-time detection and segmentation while achieving an average of around *80%* Intersection over Union on our test set.

Sadly the keypoint detection net was, even after many parameter augmentations, not able to produce any reliable results.

Contents

1 Introduction

by Nils Boehringer

Motivation

The onset of the Internet of things means that there will be more even smaller computers in nearly every daily device. This integration means that the computers have to be small and therefore lack a physical method of inputing commands. A possible solution for this would be to add a small camera that takes a low-res video that can be analyzed for gesture commands.

Our goal of this project is to train two Neural Networks to detect hands and the pose they are in. The output of these nets could then further be used to deduce commands connected to the poses.
As the video has to be analyzed in real-time we aimed for a Network structure that can rapidly segment the hands and estimate a pose for them.

Approaches

Our first approach was to use the architecture proposed in the paper [2] belonging to the Rendered Handpose Dataset. They used two FCN's based on the MobileNet architecture to first predict the center of the palm and then crop the image around this center point with a set padding. The second network then takes this as an input and tries do predict the 21 keypoints. In the project they used 3D information to better predict the keypoints. In our approach we only used the 2D data to better fit our usecase (cheap cameras without depth scanning).
We had to simplify the networks as they required to much computational power and were built for 320×320 images.

As we noticed the bad results of the first approach we searched for more modern approach to the problem. We kept the model of two Neural Networks with the same function but switched out the MobileNet's for UNet's. The first Network (HandSegNet) now tries to segment the image into hand and not-hand. We the cropped the picture around the minimal and maximal coordinates of this segmentation to use the resulting image in the second net (PoseNet). Like in the first approach the PoseNet tries to find the keypoints.

This approach allowed us to get better results for the HandSegNet. It also produced good results that where faster than the outdated MobileNet architecture. Sadly the PoseNet did not manage to estimate the right keypoints. The estimated ones where scatter around the corners of the image.

Data

We used the *Rendered Handpose Dataset* created by the *University of Freiburg* [1] for training and testing our Neural Networks. The images show rendered 3D models of persons in wildly different poses and from many different angles. Each image shows one or two hands which are sometimes partly hidden. The dataset consists of 41258 training and 2728 testing images. Each image contains a RGB image, a depth map multiple segmentation masks (background, person, three classes for each finger and one for each palm) of size 320×320. It also stores information about 21 ordered keypoints in each hand (1 in the palm and 4 for each finger), the visibility of each keypoint and an intrinsic camera matrix.

For our usecase we only used the RGB image, the combined segmentation masks and the keypoints. We did this as we wanted our networks to work with 2D images. The project from the dataset uses the remaining 3D informations to better predict the keypoints in space.
For training we manually altered the images to get better results and archive a faster training time. For the HandSegNet we reduced the image size to 128×128 and for the HandPoseNet we cropped the images to only contain one hand (with a smaller padding to make sure the whole hand is visible), adjusted the keypoint coordinates accordingly and resized the images to 128×128 aswell. We applied a small gaussian blur to the keypoints to limit the loss when the predicted keypoint is atleast in the right range.

We deemed this rather low resolution sufficient, as we wanted to quickly train our Neural Networks and have a fast (in terms of frames per second) live demonstration at the end. Also this would be close to the limited resolution of small low cost cameras built into small devices we aimed for in our use case. The wide differences between the pictures allow us to prevent over-fitting without having to alter the images our self.

Hardware

In order to train the U-Nets, we used an Intel I7-7500U as a CPU with an Nvidia M1000 as Graphics Card on a Lenovo ThinkPad P1. This was sufficient as our U-Net was structured fairly simple.
Both Neural Networks were trained using Nvidia Cuda-Technology.

2 Theoretical Background

2.1 Convolutional Neural Networks

by Michael Hartmann

A Convolutional Neural Network also known as CNN or ConvNet is an artificial Neural Network that finds its most common use in image analysis and classification. Generally a CNN can be used to find patterns in data and classfiy them. This is why CNN's are so powerful for image analysis.

A CNN is an artificial Neural Network but what differentiates it from "normal" artificial Neural Networks is the introduction of *convolutional layers*. Moreover the CNN can have other components like non linear activation functions or fully connected layers.

Convolutional Layers

To understand how a convolutional Network works we first want to discuss convolutional layers. A convolutional layer functions just like a "normal" hidden layer in an artifical neural network in that it receives input and generates output by performing some mathematical transformation on the input. Here the mathematical transformation relies on filters which are convolved over the image. In contrast this results in an output where not every part of the input contributes to every part of the output like in fully connected layers. Convolutional layers consist of many filters or *kernels* applied consecutively. A convolutional layer is then defined by the number of filters, the size of each filter (*kernelsize*) and an optional padding to the input.

Pooling Layers

After some activation layer has been applied to one of the outputs of the convolutional layers, one may use a pooling layer to reduce the volume of the data. The reasoning behind this is that once we have recognized a specific feature, the exact location is not that important for the further process of classification. This reduces computational cost and prevents overfitting. Pooling layers take a filter and stride of a given size and downsample the original input. The method determining how the input is sampled can be specified by different versions of

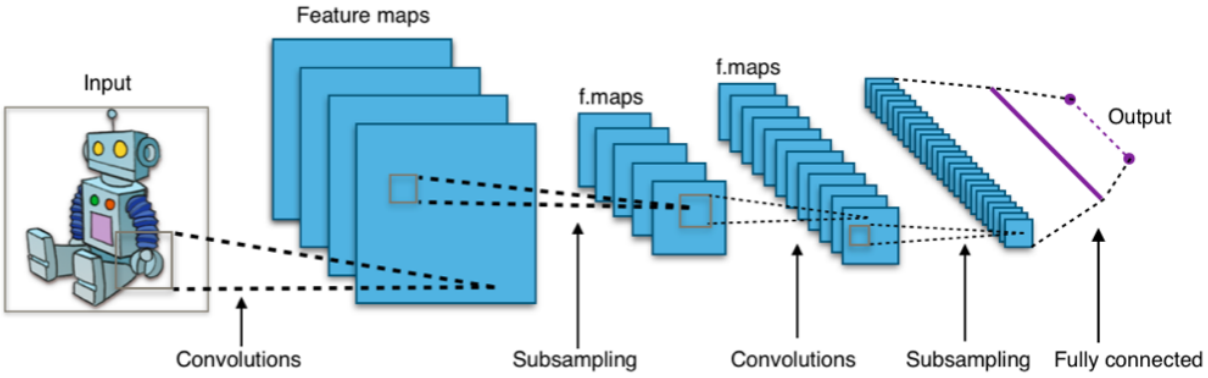

Figure 2.1: **Structure of a typical CNN ??**

the pooling layer.

Fully connected layers

After some subsampling has been applied the final layers in a traditional CNN are fully connected layers. The final output layer then has the output size of the number of classes we want to observe.

2.2 U-Net

by Michael Hartmann

Fully Convolutional Net

The general idea of a fully convolutional Net is to replace the fully connected layers of CNN's with convolutional layers to get spatial information. The pixelwise prediction, produced be the convolutional layer at the end, can then be used to do detection and segmentation rather than just image classification.
One of the main problems of this approach is the need for some rather radical upsampling to get a prediction with the same size of the initial input, which of course has a rather negative impact on the precision.

U-Nets for more precise image segmentation

The U-Net is convolutional network architecture for fast and accurate segmentation of images. It is vastly outperforming conventional FCN's in segmentation tasks.
The reason for the high accuracy of U-Nets is it's unique symmetric architecure.

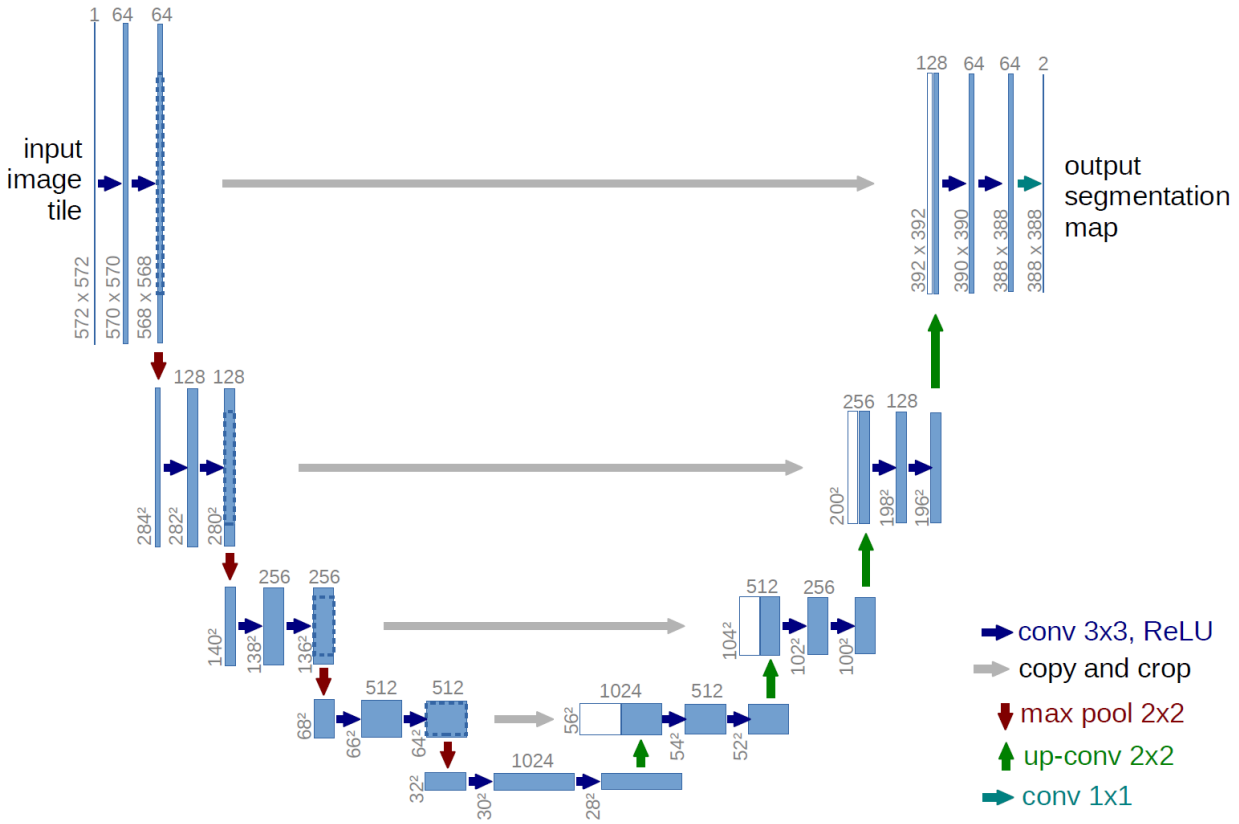

Figure 2.2: **Typical U-Net architecture**

The network consists of a contracting path and an expansive path. The contracting path consists of repeated application of convolutions, each followed by a ReLU and a max pooling operation. During the contraction, therefore the spatial information is reduced while feature information is increased.
The expansive pathway takes the feature and spatial information through a sequence of up-convolutions and concatenates it with the high-resolution features from the contracting path. This results in a high resolution output for precise segmentation.

2.3 HandSegNet

by Michael Hartmann

General

The objective of the *HandSegNet* was to detect and segmentate hands in images.
It uses a customized U-net architecture to make fast and precise segmentation in real-time scenarios possible.
The main goal of this net is to provide an accurately cropped input to the *PoseNet* while additionally providing a segmentation mask.
The Net only knows two classes: Hand and background.

Figure 2.3: **Segmentation and cropping of test image**

Training

The HandSegNet was trained on 41258 training images for 25 epochs. The pictures were not augmented except for their size. Since 320x320 pixels seemed to be a bit too much for our puny graphics memory we had to reduce it to 128x128.
Other than this we did not apply any transformations. Due to the huge variety of poses and positions in the dataset we figured that applying transformations would not lead to any significant change in accuracy. Training Parameters:

- Epochs: 25
- Batch size: 32
- Learning rate: 0.001
- Loss function: Cross-Entropy-Loss

- Accuracy measurement: Intersection over Union (measured every 100 steps on a small validationSet of images that were neither in the test nor the training set)
- Optimizer: ADAM

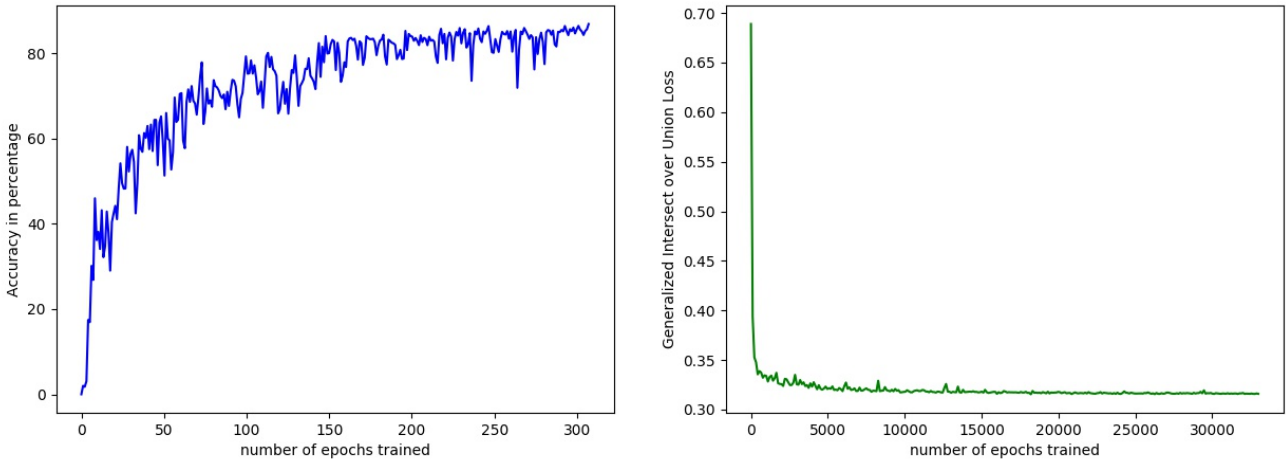

Figure 2.4: **Loss and accuracy during training**

scenarios The training process was originally intended to be 50 epochs long, but due to time constraints we had to manually interrupt after 25 Epochs.
The training took about 14 hours. The state dictionary was saved for future usage.

Testing

We tested the net on the 2700 test images and achieved an average of around *83%* accuracy for the predicted masks.
The accuracy was measured on the masks rather than the bounding boxes since we implemented only one bounding box per image which means that if two hands are present in the image both would share one bounding box no matter how far away they are. This of course would result in a far lower accuracy score even with high accuracy on the predicted masks.

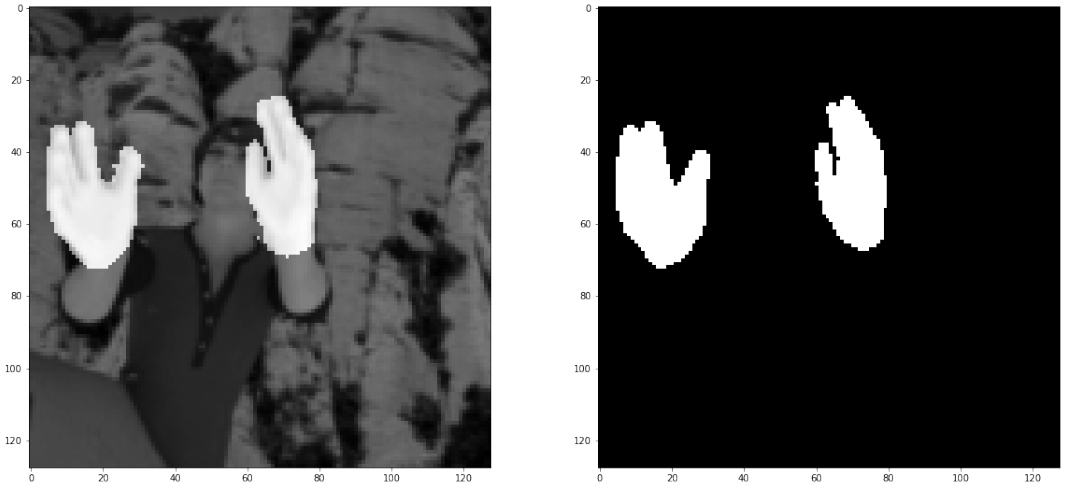

Figure 2.5: **Predicted mask (left) and Ground truth (right) IoU: 86.5%**

Possible Improvements

While the segmentation masks were in general quite good, their accuracy was of course dampened by the comparatively low resolution of the input images. Training and testing it on the original 320x320 pixel images could potentially improve the overall precision by a few points.
Additionally the training parameters were manually optimized to a certain extent but it is quite likely that the performance could be improved further by changing these parameters.
It is difficult to predict whether or not a longer training would have yielded any significant improvement, but it is not unlikely that at least 1 or 2% more would have been possible with more training time.
Furthermore we realized that the accuracy on images of real people was considerably lower than on the rendered test images. This inaccuracy was mainly due to the net not recognising the middle parts of real fingers (joints) as part of the hand. We assume that the reason for this is the smoothness of the Hands in the synthetic dataset, thus a possibly solution would be a mixed dataset of real and rendered hands.
Predicting a single bounding box resulted in some problems as well. Single misclassified pixels could radically influence the predicted boundaries resulting in unsteady and rather shaky predictions. Making two separate predictions, one for each hand, would reduce extreme boundary changes due to the appearance of a second hand in the frame.
Additionally it would help to filter out outliers to achieve a steadier prediction. A first step towards this would be to ignore single pixels with no neighbouring pixels of the same class.
Lastly a minimum size for hands would be a good idea. Not only would this lead to a more stable prediction but also help ensuring that the output is informative for the PoseNet. If the hand is too small in a picture, even a correctly cropped output would be too small for the PoseNet to use.

Figure 2.6: **Difference between rendered hands and real hands as possible performance factor**

2.4 PoseNet

by Nils Boehringer

The PoseNet used the same UNet as the HandSegNet. The only difference is that the net outputs 21 prediction heatmaps with the size 128×128. The heatmaps show the networks certainty of the estimation that there is a keypoint at each coordinate.
For training these heatmaps could then be compared with the blurred keypoint images from the modified test dataset. For evaluation and we used *Intersection over Union*.

Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. This metric is often used for object detection problems to evaluate the performance of Convolutional Neural Network detectors. IoU wasn't the best metric to choose for this usecase as we can't compare two bounding boyes. In our implementation the bounding box around the blurred keypoint was easy. Fitting a bounding box to the computed heatmaps is not that easy as the heatmaps represent the certainty of the network that the keypoint is at the coordinate. So there is more than one point with a high probability that we could use to fit the bounding box.
To get the most likely position of a keypoint for plotting it, we converted each heatmap to a one-hot map and took the coordinate that was left.

As shown in the picture below this approach did not work out and nearly all keypoints are on the corner of the picture.

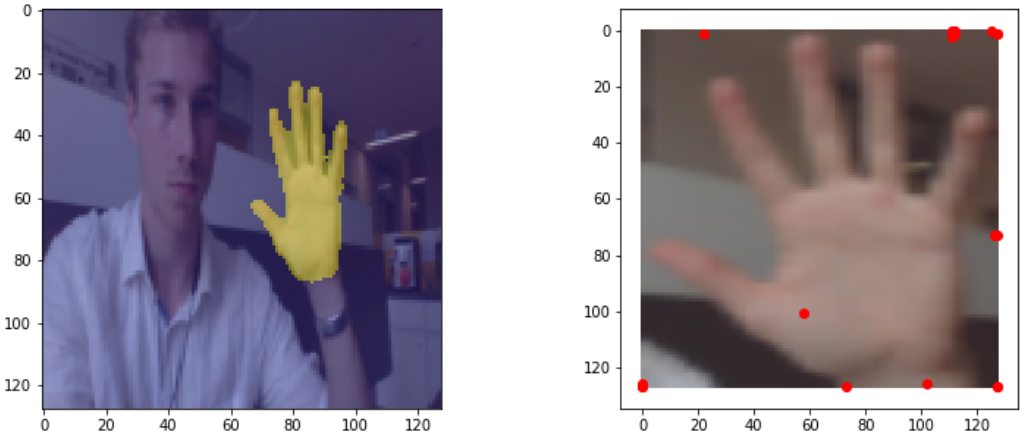

Figure 2.7: **Unet Image Error**

To fix these problems we tried different solutions:

To begin with, we probably used the wrong method to judge the accuracy of the network. IoU works with bounding boxes. While we created a "bounding box" around the heatmap coordinates around the minimal and maximal nonzero points, the same isn't working for the

output of the net. The outputs show the certainty and are therefor not centered around a point. In those majority cases the "bounding box" would be the same size as the image. But this is an issue with measuring the accuracy and it doesn't influence the training of our network.

To fix the issues with the training we first tried to use a different loss function (L2 to L1). But as expected this didn't change the outcome much. The L1 loss minimizes the sum of the absolute differences between the target value Y_i and the estimated values $f(x_i)$, while the L2 loss minimizes the sum of the square of the differences S between the target value Y_i and the estimated values $f(x_i)$.
We tried to increase the size of the gaussian blur in an attempt to make the keypoints "move" into the right direction faster. Another idea was to add the hand segmentation to the heatmap with a smaller opacity. This would lead to a reduced Loss if the keypoints are in the area of the segmentation and therefor closer to the right position.

Sadly we did not find the solution for our problem in time, so we can only make some guesses how to fix the problem. Some of the solutions proposed can be used to improve the accuracy of a functioning PoseNet (like switching the loss function or varying the gaussian blur size).

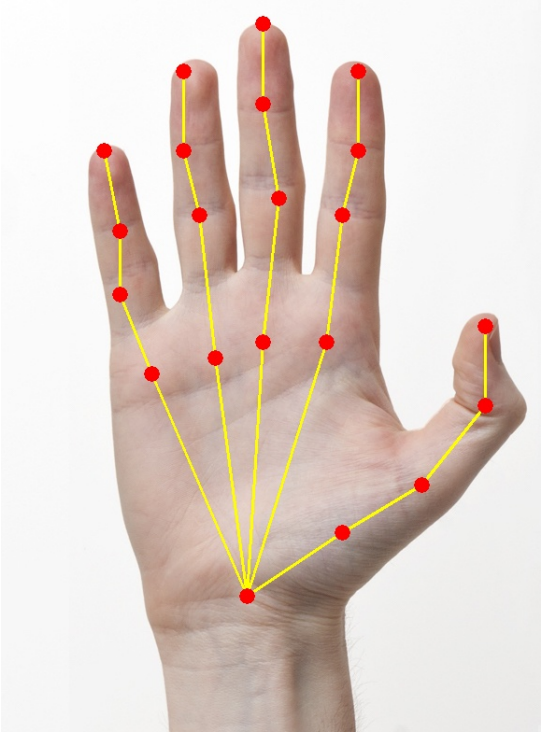

Figure 2.8: **Target Output**

3 Conclusion

3.1 Summary

by Nils Boehringer

For the Deep Vision project during the summer semester of 2019 we tried two different approaches to hand segmentation and handpose estimation. Both approaches were trained with Rendered HandPose Dataset [1]. For training those images got rescaled and cropped to have a better performance. Our general approach was to train two Neural Networks; One for segmenting the hand from the background and cropping the picture to a fitting size and the second one to use this result to estimate the coordinates of 21 keypoints.

Our first approach of implementing a fully Convolutional Network based on the MobileNet architecture did not work out, as it did mostly only recognize the palm of the hand and not the fingers. As we want to use the segmentation to crop the image to the full hand this did not work out for us. The PoseNet did not produce viable results aswell.

After switching to the more modern UNet architecture we noticed better results almost immediatly. After only a few training epochs, the network already had a visibly better segmentation which rapidly got better each epoch. Another upside is the fast computation time the network needs per image. Because of that we managed to get a smooth live-demo to work.
Our PoseNet did not estimate the right keypoints in the hand. The results were not close to the ground truth. We tried different solutions to fix this problem, but we could not get it to work properly in time.

References

1. Rendered Handpose Dataset University of Freiburg
   https://lmb.informatik.uni-freiburg.de/resources/datasets/RenderedHandposeDataset.en.html

2. Learning to Estimate 3D Hand Pose from Single RGB Images
   https://arxiv.org/pdf/1705.01389.pdf

Formalities

Michael Hartmann

- 3329547
- Deep Vision

Nils Boehringer

- 3362777
- Deep Vision