university of groningen / faculty of science and engineering

Robot Localization and Dreaming using Convolutional Neural Networks and Denoising Autoencoders

Bachelor's Project Thesis

Y. Chong, s2709961, y.chong@student.rug.nl,
C.G. Kuiper, s2721600, c.g.kuiper@student.rug.nl,
Supervisors: A. Shantia, dr. M.A. Wiering

Abstract: Denoising autoencoders have been used with success for indoor localization based on images in a 3D simulated environment. With the recent successes of convolutional neural networks in image recognition tasks, convolutional neural networks are a good candidate for the localization task. In this thesis, the performance of (stacked) denoising autoencoders and a convolutional neural network are compared in a localization problem using color image information. The networks are provided with images labeled with location and orientational coordinates. The performance of the networks are compared by the mean error in location and mean orientation error. Both networks are trained in two 3D simulation environments (a small and large room) and a real environment. The convolutional neural network performed better on all three environments by a large margin. The convolutional neural network attained a mean error of 6cm in location error and 5° in orientation error on the small environment and 8cm in mean location error and 4° in mean orientation error on the large environment. On the real environment the convolutional neural network attained 51cm in mean location error and 27° in mean orientation error.

1 Introduction

Localization is an important aspect of domestic robots, for example autonomous robotic vacuum cleaners, lawnmowers or even drones. While there is already a range of autonomous robots available, an affordable price is needed in order to make those robots more attractive for the general public. Localization is commonly done by mapping the environment by 2D or 3D sensory data and updating the location within the map. The updating method normally consists of spatial alignment of data frames, loop closure detection, and a global fine-tuning step for the alignment of the complete data sequence. While those methods are effective, the hardware that is required is expensive. The cost of a cheap 2D laser range finder is already substantial, with a cost of several hundreds of dollars. This means that a solution to the localization problem is needed, which can make use of affordable hardware.

Research on the localization problem using inexpensive hardware has been done. It has been shown that RGB-D cameras, such as the PrimeSense*, can be used for localization using 3D mapping combined with visual features and shape-based alignment (Henry et al., 2012). Henry et al. combined visual and depth information for view-based loop-closure detection, followed by a fine-tuning step for optimization of the pose.

Further research on RGB-D mapping has been carried out by extending upon the approach of Henry et al. (Endres et al., 2014). Endres et al. extracted visual keypoints from color images and used depth images to localize the keypoints in 3D. The geometric relations between keypoints were used to estimate the motion of the robot. By using an environment measurement model to validate the transformations they improved results in challenging environments.

Other recent studies by Shantia et al. showed a

*Primesense Ltd. Patent No.US7433024 [3]

new approach by using image color information and neural networks for the localization of a robot in 3D simulation (Shantia et al., 2015). The results were promising with a mean location error of 10cm and a mean orientation error of 4°. This research aimed to estimate the location and orientation of a robot based on image color information. A network with a denoising autoencoder (DA) (Vincent et al., 2008) was used to approximate the location and orientation of a robot based on HSV images (Figure 1.1). DAs were used for an initial unsupervised learning step that maps the input images into intermediate representations. Stacking DAs on top of each other can be used to initialize deep neural networks, which is useful for the localization task due to the advantages in generalization (Bengio et al., 2007), (Vincent et al., 2011).

In this paper we reenacted the approach of Shantia et al. by training networks using DAs and SDAs. This is done with semi-supervised learning, which uses the location and orientation values provided by traditional mapping methods. The models were tested in two 3D simulated environments, but also on a real environment which was not done by Shantia et al.

Furthermore we also trained a convolutional neural network (CNN) for the same localization task. Ever since the winning entry on the Imagenet by Krizhevsky et al. in 2012 with their deep convolutional network called AlexNet(Krizhevsky et al., 2012), most image recognition competitions have been dominated by CNNs. Their CNN achieved an error rate as low as 16%, outperforming the previous 25% error classification rate, which was achieved with classical image processing at that time. In addition, since then CNNs have been successfully applied to a wide variety of computer vision tasks such as object detection, segmentation (Long et al., 2015), style transfer (Gatys et al., 2016) and image generation (Goodfellow et al., 2014). Since localization from an image is a computer vision task, CNNs are a perfect candidate to use for this problem.

Our first research question was: How well does localization using a CNN work, compared to localization using a (stacked) denoising autoencoder network? This research question was answered by comparing the performance of the (S)DA with the performance of the CNN.

Next to the localization task we also did something we will refer to as "dreaming". In principle the dreaming task is the opposite of the localization task. Instead of estimating a location and orientation based on an image, an image is generated based on a location and orientation (Figure 1.2). This image represents what the model thinks it should see at the given location and orientation. Our second research question was: is it possible to train a convolutional neural network and a network with a (stacked) denoising autoencoder to dream an expected image from a certain position?

2 Methodology

In this section we will explain the basic denoising autoencoder DA and stacked DAs (SDA) and how they were applied to the localization and dreaming task, followed by an explanation of the basic convolutional neural network (CNN) and how it is applied to the localization and dreaming task. Then the basic outline to go from an image to a location will be shown.

2.1 Denoising Autoencoder

The DA is an adaptation of the regular autoencoder. The difference is that the DA uses a corrupted input image instead of a regular input image. For a DA the input x is corrupted into $\tilde{x}$ depending on a corruption level (equation 2.1).

$$x \to \tilde{x} \sim q_D(\tilde{x}|x) \tag{2.1}$$

This corruption level indicates the chance for a value to be either destroyed or changed. In our adaptation a corrupted value is destroyed, meaning the value is put at 0. This corrupted input $\tilde{x}$ is then mapped to a hidden representation y by multiplying $\tilde{x}$ by the weights W, adding the bias b and putting those results into a sigmoid function sig (equation 2.2).

$$y = f_\theta(\tilde{x}) = sig(W\tilde{x} + b) \tag{2.2}$$

The next step is reverse mapping the hidden representation y back to a reconstruction z of the image (equation 2.3). The weights W' and the bias b' of the reverse mapping are used. The cost is calculated by taking the cross entropy of the original input x and the reconstruction z (equation 2.4).

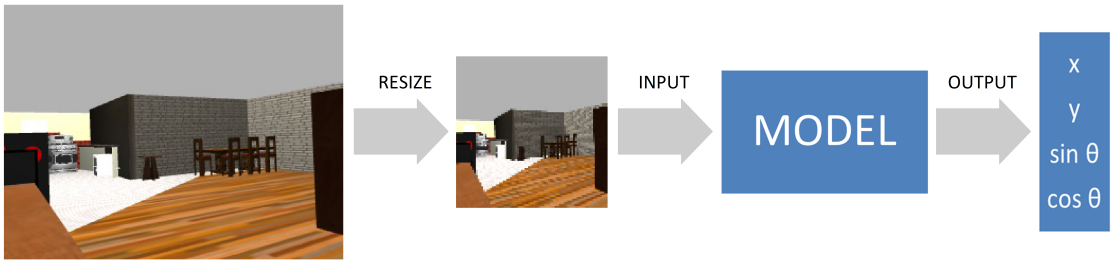

Figure 1.1: The basic structure of the encoder, where x, y are the x and y coordinates and $sin(\theta), cos(\theta)$ the angle of where the picture was taken.

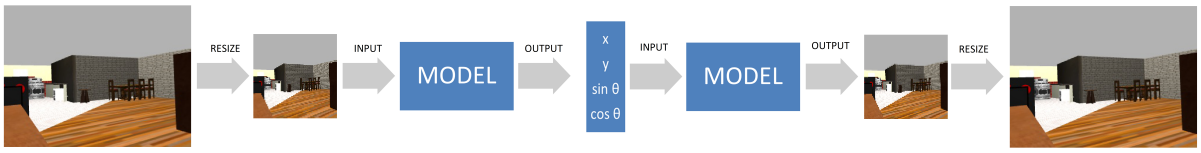

Figure 1.2: The basic structure of the decoder, where x, y are the x and y coordinates and $sin(\theta), cos(\theta)$ the angle.

Here x denotes a pixel from the original input, z denotes a pixel from the reconstruction and k is the index of the pixel. d denotes the set of indices for all pixels. This cost can be used to calculate the error and updates one stochastic gradient descent.

$$z = g'_\theta(y) = sig(W'y + b') \tag{2.3}$$

$$Cost = \sum_{k=1}^{d} [x_k \log z_k + (1 - x_k) \log(1 - z_k)] \tag{2.4}$$

In Figure 2.1 the basic structure can be seen of the DA.
The reason behind using DAs is that they are able to learn general representations, which are robust to partial noise. In a localization task the model needs to be able to deal with noise from small changes in the environment like moving objects, for example a door being open or closed or chairs that are moved. Lighting changes from either the sunlight or indoor lighting can also change what the environment looks like. Images in a localization task can also be very similar to each other,

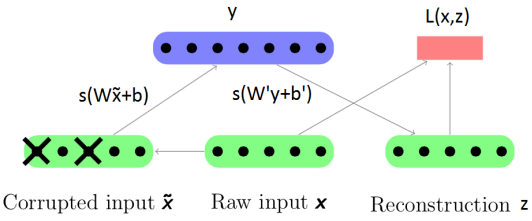

Figure 2.1: The basic structure of a DA. The raw input x is corrupted (equation 2.1). The resulting corrupted input $\tilde{x}$ is mapped to the hidden representation y using equation 2.2. Afterwards a reconstruction is made by reverse mapping y using equation 2.3. Finally a cost is calculated using equation 2.4.

which means that generalization is important for the model in order to learn the environment.

Next to regular DAs we also trained networks using SDAs. An SDA is made by stacking at least two DAs on top of each other. The first layer DA is trained in the same procedure as the regular DA. A DA in layer n+1 will take the hidden represen-

tation of the layer n as the input x instead of the original input image. This x is again corrupted to $\tilde{x}$ (equation 2.1), which is again mapped to a hidden representation by multiplying $\tilde{x}$ by the weights, adding the bias and putting those results into a sigmoid function (equation 2.2). A reconstruction attempt is made to reconstruct the original x, which is the hidden representation y of layer n (equation 2.3). The cross entropy of the reconstruction z and input x is again taken as the cost of the function (equation 2.4).

2.2 Localization Network DA

DAs can be used to initialize the weights in their corresponding layers in a pretraining phase. After the pretraining phase an additional top neural network is trained to associate the encoded result of the images with a location and orientation. This top neural network is connected to the final DA layer and consists of a hidden layer with a sigmoid activation function (the relation between images and odometry is nonlinear) and an output layer with a linear activation function. The output of the top neural network consists of four values: $x, y, cos\theta, sin\theta$. The x and y value denote the estimated location, while the $cos\theta$ and $sin\theta$ denote the estimated orientation. The top neural network is a regular multilayer perceptron which uses the mean squared error for its cost function.

2.3 Dreaming Denoising Autoencoder

The inverse of the DA network for localization has an inherent problem. For the localization network, DAs can be used to initialize the weights by pretraining on the images. For a dreaming network, the inputs are location and orientation values instead of images. Pretraining on images with DAs can be used to denoise the image and bring forward useful features in the image. Pretraining using DAs on location and orientation values makes no sense however, since there is no real noise and features to be found in those four values. The dreaming network is the inverse of the localization network without the pretraining with DAs. This means it is a regular multilayer perceptron.

2.4 Convolutional Neural Network

Most convolutional neural network (CNN) have a typical structure: a number of convolutional layers, with possibly pooling layers in between to reduce dimensionality of the feature maps. After these convolutional layers there are a few fully connected layers. For our implementation we decided to use stride in favour of pooling for dimensionality reduction. The reason to this is very similar as to why Google's DeepMind DQN (Mnih et al., 2013) does not use pooling. What pooling does is giving you translation invariance, something that would make sense in image recognition tasks, but it would not make sense in a localization task where location variance of a feature is very important in determining the robots location.

More recently, Google introduced their GoogleNet CNN (Szegedy et al., 2015), which uses inception modules. The use of inception modules strays from the general approach in which convolutional layers are stacked sequentially. In their network, convolutions do not only happen sequentially, but also in parallel with different kernel sizes. The idea behind using convolutions in parallel is that a smaller kernel will look for finer/smaller features and a bigger kernel will respond to larger features. The produced feature maps from both kernels are then concatenated depthwise to create one feature map. This is computationally and memory wise expensive. Concatenating all the produced feature maps would create an extremely deep feature map, which is what happens in the naive version (Figure 2.2). To address the problems that arise from a large feature map size, the authors used 1×1 convolutions to reduce the depth of feature maps created from the parallel kernel convolutions, think of this as feature pooling (Figure 2.3).

2.5 Dreaming Convolutional Neural Network

To create a decoder to go from a location and orientation to a "dreamt" image, a reversed version of the CNN was used. This means that instead of convolution, deconvolution was used which will upsample the input feature map. The final layer uses a sigmoid activation function, which should keep the output values within a range of $[0, 1]$, which is

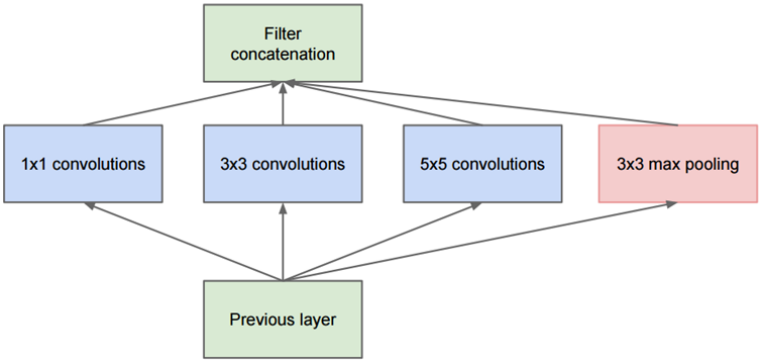

Figure 2.2: Inception module, naive version (Szegedy et al., 2015)

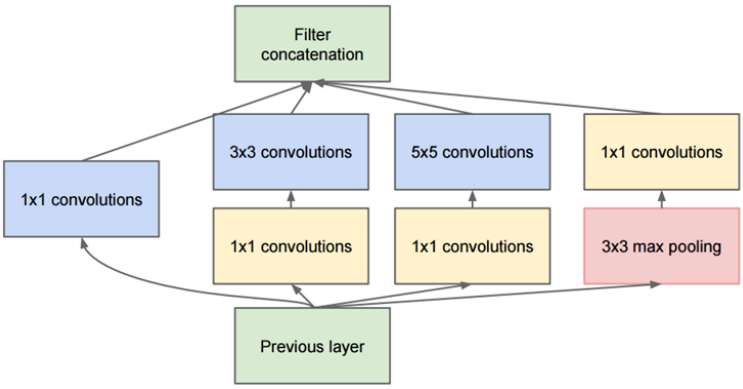

Figure 2.3: Inception module with dimension reductions (Szegedy et al., 2015)

then scaled to $[0, 255]$ for the RGB images.

To reverse an inception module, the first step is to reverse the concatenation, which is done by splitting the feature map, going from one feature map with a 128 channel depth to 2 feature maps with a 64 channel depth. After the split, a deconvolution is applied to the separated feature maps with different kernel sizes. To create a single feature map again the two outputs are summed together, creating a single feature map, with a depth of 64 channels.

2.6 Basic Outline

The images were resized to 36×36 for the DA and to 84×84 for the CNN. Resizing the pictures to small sizes means fine-grained information like small colorful objects are partly lost. The basic structure of the scene in the images is however retained. The reason that higher resolution images are used for the CNN is due to the simple reason that the DA network functions largely as a multilayer perceptron. It is a fully connected network, which means that using larger images result in a huge amount of weights. The convolutional layers in a CNN are not fully connected and thus can have a smaller number of weights with an equally large or larger size input.

3 Experiments

Experiments were done for two 3D simulated environments and one real environment. The experiments in the 3D simulated environments were done using Gazebo (Koenig and Howard, 2004). The first 3D simulated environment is a kitchen room of 5×7 meters from Google's 3D warehouse[†], which we will call the "small room" (Figure 3.1). The second 3D simulated room is 17×9 meters, which we will call

[†]https://3dwarehouse.sketchup.com.

the "large room" (Figure 3.2). The data were gathered by using an automatic process to move the robot with steps of 0.25m through the environment (Figure 3.3). On every location, the robot rotates 360° and captures an image every 1°. The images are labeled by the location and orientation values, which are taken from the current ground truth location. The location values are normalized by dividing them by the respective largest x and y value, such that the values are between -1 and 1. In the small room we gathered a total of 24,000 images and for the large room we gathered 195,000 images. The size of the taken images are 320×240 pixels.

The real environment is a hallway of 10×3 meters. A robot was used with two xtion cameras[‡], one on the front of the robot and one on the back. Next to this the robot has a laser scanner on the front, with a range of 180°. An obstacle avoidance system was used to prevent collisions by using information from the laser scanner and xtion camera on the back. At each location, the position and orientation values were given by Adaptive Monte Carlo Localization (Fox et al., 1999) and gmapping (Grisetti et al., 2007). The data were again gathered by using automatic processes which move the robot through the environment with 0.25m steps. At each location the images were captured by the front xtion camera, which can rotate 180°. The pitch of the camera was set in such a way that it looks forward. Each location was visited twice in order to capture 360° worth of images on each location. The total amount of images is 115,280 for 369 locations and the picture size is 640×480 pixels. The location values are here normalized by shifting them such that the center of the map has the value [0,0], after which the values are divided by the respective largest x and y, such that all values are in the range of -1 to 1. In Figure 3.4 some example images of the real environment are shown.

The datasets from the 3D simulated rooms were both split into a training, validation and testing set. Both datasets were split by using five degree steps in the dataset. The first 5° on a location were put in the training set and the next 5° of images were split into validation and testing. This is done by taking two thirds for testing and one third for the validation set. The dataset from the real environment was split into a training and validation set. The real dataset was split in the same way as in the datasets from the 3D simulated rooms, but in this case the testing part would also be put into the training set. The testing set for the real environment was made by moving the robot continuously through the environment while saving the images. In total the test set consists of 4000 images.

The images can be represented in different color models to both neural networks. To see which color model would work best, both neural networks were trained using the RGB and HSV color space. The color space with the smallest average error was cho-

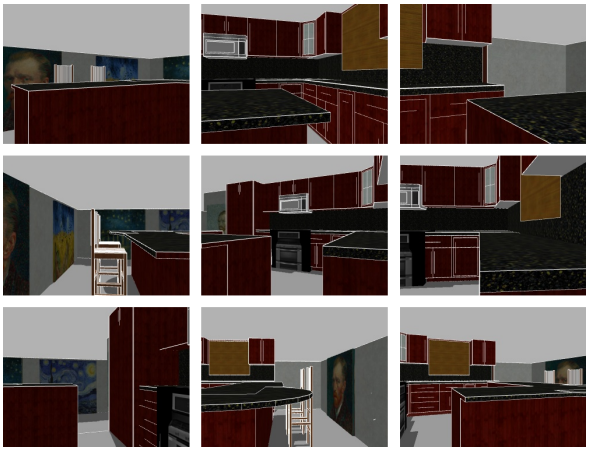

Figure 3.1: Example images from the small room dataset.

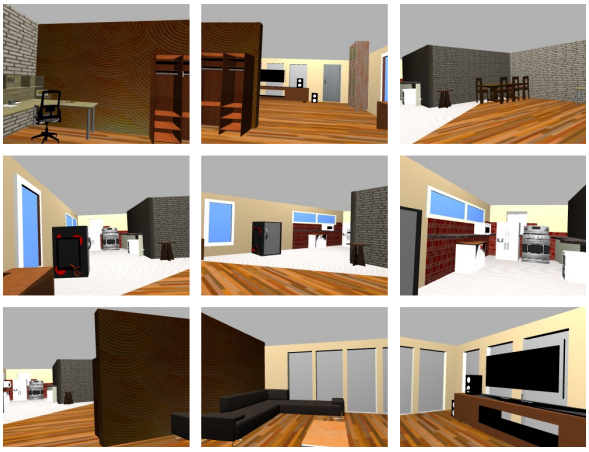

Figure 3.2: Example images from the large room dataset.

[‡]https://github.com/OpenNI/OpenNI/blob/master/README

Figure 3.3: An overview of the map of the large room. Every green cross represents a location where images were gathered. The steps of 0.25m are done in both x and y direction on every possible location.

Figure 3.4: Example images from the real environment.

sen for that neural network. Similarly, there are multiple optimization algorithms which could be used. The Adam optimizer (Kingma and Ba, 2014) and stochastic gradient descent (SGD) were tested for both neural networks and the best optimization algorithm was chosen for each neural network.

3.1 Denoising autoencoder

The pretraining phase was done with a learning rate of 0.05 for 500 epochs on the training set. The location estimating network was trained with the same training sets for 10,000 epochs with a learning rate of 0.1. The location estimating network used two hidden layers, each consisting of 100 neurons. After this training phase is done the whole network is trained once more for 10,000 epochs with a learning rate of 0.01. 19 different structures were trained for the small room, including differences in the amount of DA layers, hidden neurons and corruption levels (Figure 3.5). A single DA was trained as well as two layer SDAs and three layer SDAs. Due to a large amount of training data for the real environment and the big simulation room, only successful models from the small room were tested. For the simulation rooms a validation and test set were used. The validation set is used to validate the model after each 100 training epochs. If the performance of the model on the validation set is an improvement over earlier models, it is chosen as the new best model in order to prevent the model from overfitting. When a better model is found, this model is also tested on the testing set. For the real environment only a training and validation set were used. The testing of the model was done by live testing on the real robot by moving it through the environment (pseudo randomly in a way every part of the map is visited) and recording the error in location.

For the dreaming network (inverse DA network) an MLP was trained for 1000 epochs with a learning rate of 0.0001. The MLP has the structure shown in Figure 3.6.

3.2 Convolutional Neural Network

In total, 3 different CNNs were implemented. The first CNN Model 1 is 7 layers deep, the second CNN Model 2 is 9 layers deep and the final CNN Model 3 is 9 layers deep where some layers are inception modules. All CNNs have a $84 \times 84 \times 3$ RGB image as input and use Adaptive moment estimation (Adam) (Kingma and Ba, 2014) for gradient descent optimization. The CNNs used the default parameters for Adam as proposed by the authors (decay rate $\beta 1 = 0.9$, decay rate $\beta 2 = 0.999$ and $\epsilon = 10^8$) with a learning rate of 0.00001. The error was determined by the mean squared error. The inputs of a convolution layer are zero padded such that the output dimension is the same as the input dimension, when convolving with a stride of 1.

Model 1 (Figure 3.7) The first four layers convolve with a 4×4 kernel over their input, after

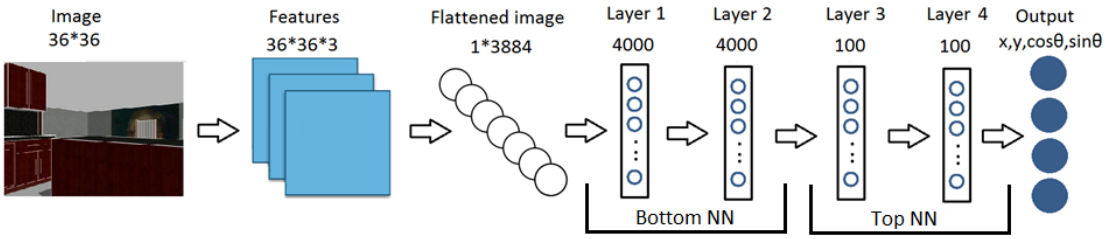

Figure 3.5: Example of a DA network with a two layer SDA. The input images are first resized to 36×36 and then flattened. The first two layers are the SDA (bottom NN) and layer 3 and 4 are the location estimating network (top NN).

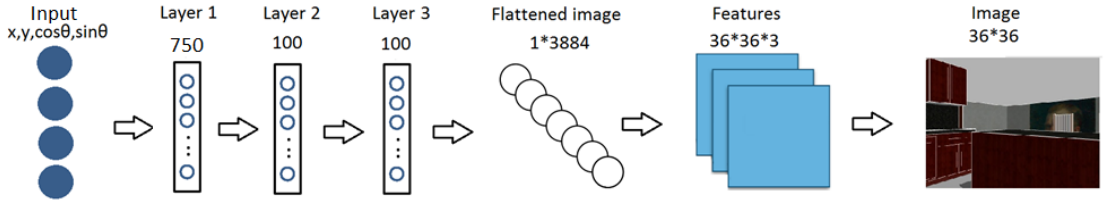

Figure 3.6: MLP used for dreaming. As input the four locational values x, y, cosθ and sinθ are used. The network consists of three layers, the first layer consisting of 750 neurons and the other two layers consist of 100 neurons. The output is a flattened image, which can be unflattened to $36 \times 36 \times 3$ pixel values.

which the output goes through an activation function, which is a rectified linear unit (ReLU) for activation (Jarrett et al., 2009; Nair and Hinton, 2010), which will be used in all layers, except the output layer. The fifth layer convolves with a 3×3 kernel, after which there are 2 fully connected layers, where the last one gives us the final output and uses a linear activation function.

Model 2 (Figure 3.8) Model 2 can be seen as an extended version of model 1, with more convolutions, where some have a stride of 1, so that the final output feature map has the same dimensions as model 1 prior to the fully connected layers.

Model 3 (Figure 3.9) Model 3 is 9 layers deep and uses inception modules. The modules used are the naive version of an inception module, which are slightly altered. The inception module used, only have 2 convolutions in parallel, opposed to the 3 convolutions + 1 pooling in the Inception-v1 (Szegedy et al., 2015) or the even bigger versions in later Inception versions (Szegedy et al., 2016). Since the network has $84 \times 84 \times 3$ input images and is 9 layers deep compared to the 224×224 input images and a 22 layers deep network, there is less concern about the memory and computation time. Two types of inception blocks were used, one where a kernel with 64 filters of size 3×3 is parallel to a kernel of 64 filters with size 5×5. Both have the same stride after which the output are then concatenated. This module will be referred to as inception A (Figure 3.10). The second type (inception B) is where one kernel has 64 filters of 1×1 and the second kernel has 64 filters of 3×3(Figure 3.11).

Model 3 is very similar to model 2, every layer with 4×4 kernel convolutions is replaced by inception A blocks, except the first layer. The layer with 3×3 kernel convolutions is replaced by the inception B block. In addition to that, a dropout (Hinton et al., 2012) was inserted to help making the model more robust. For all the simulated environments there was a dropout of 0.0, for the real environment we use a dropout of 0.5. After the dropout we have 2 fully connected layers, where the last fully

connected layer uses a linear activation function.

3.3 Dreaming Convolutional Neural Network

For the reverse of an inception module, inception A was reversed, which means one deconvolution of 64 filers of size 3×3 parallel to a deconvolution of 64 filters of size 5×5. For the reversed module B we have a deconvolution of 64 filters of 3×3 parallel to a deconvolution of 64 filers of 1×1. The whole dreaming layout for model 3 looks as follows: the 4 input variables are fully connected layers of 512 neurons, this is then connected to a fully connected layer of 15,488 neurons, which is reshape to a 11×11 feature map with a depth of 128 channels. The reshaped feature map is then the input of a reversed inception module B with a stride of 1. After this there are 5 reversed inception module A with a stride of 1, 2, 2, 1, 2 respectively and each parallel kernel has 64 filters. At the end there is a normal deconvolution with a kernel consisting of 3 filters with a size of 4×4 and a stride of 1. As for the optimizer, Adam was used with a learning rate of 0.0001 where the loss was determined with cross entropy.

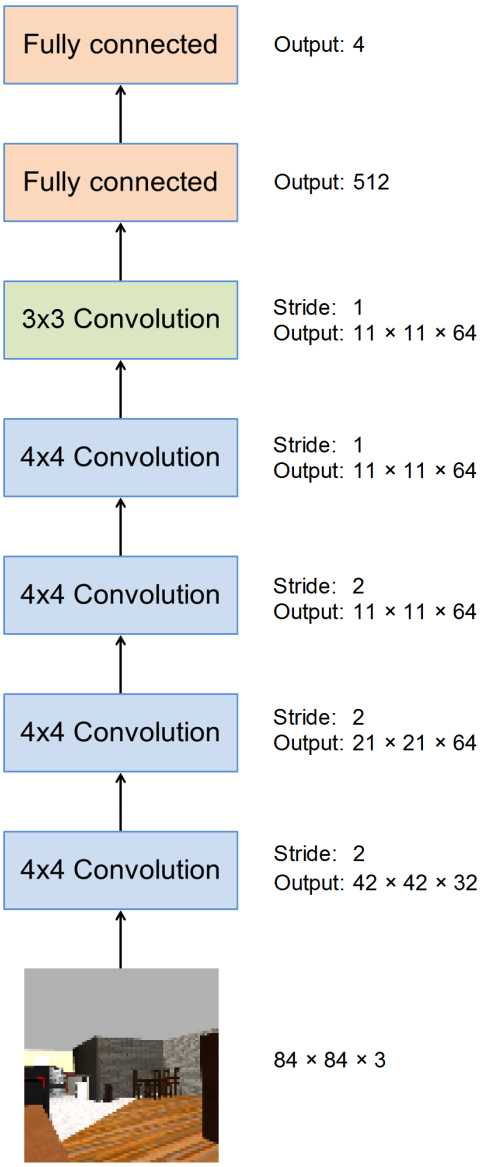

Figure 3.7: Layout of model 1, with 2 different kernel size convolutions and fully connected layers with the final output being the x, y, sinθ and cosθ

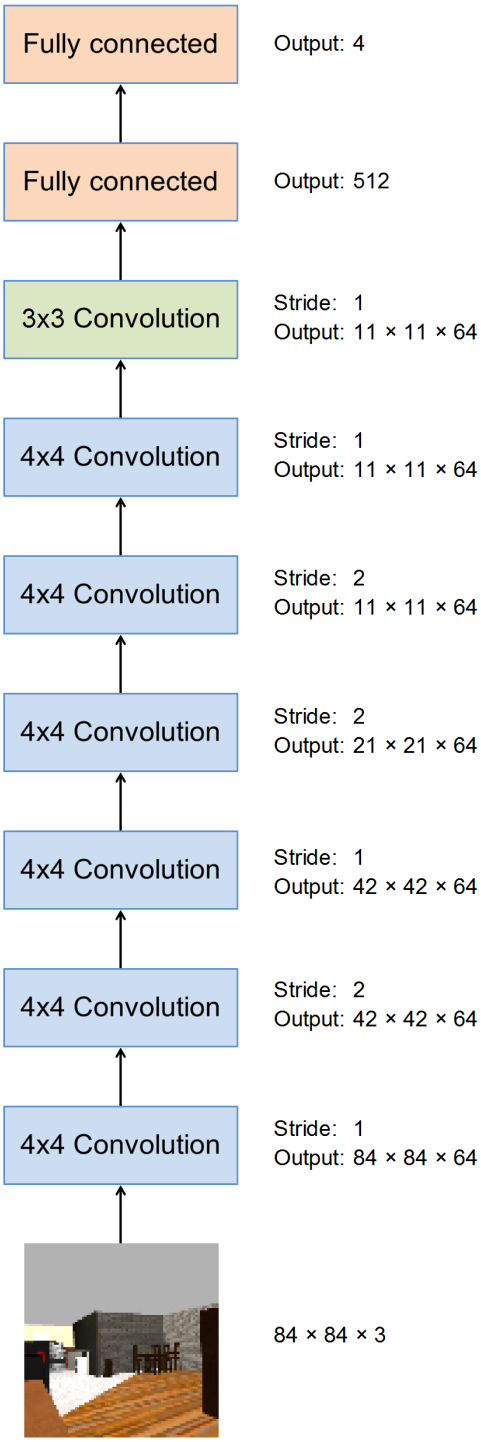

Figure 3.8: Layout of model 2, with 2 different kernel size convolutions and fully connected layers with the final output being the x, y, sinθ and cosθ

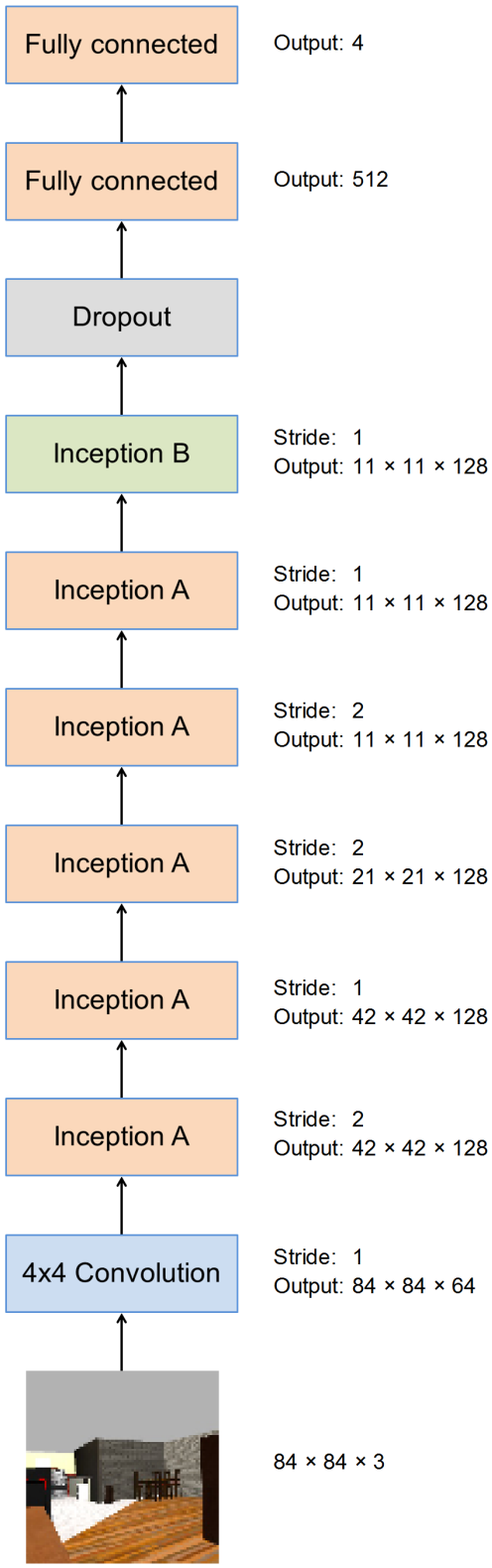

Figure 3.9: Layout of model 3, with normal kernel size convolutions, Inception modules A, B and fully connected layers with the final output being the x, y, sinθ and cosθ

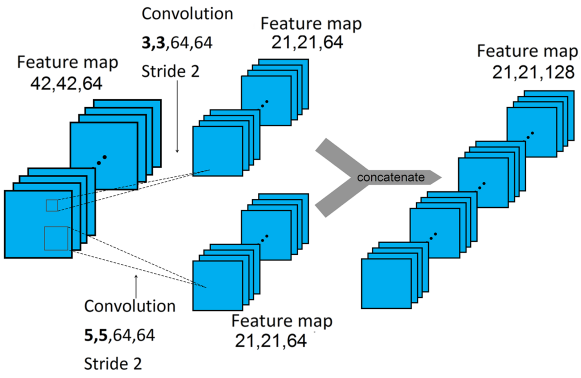

Figure 3.10: Inception module A that is used in our 3rd model, where the input is a 42×42 image and the used stride is 2 in this example

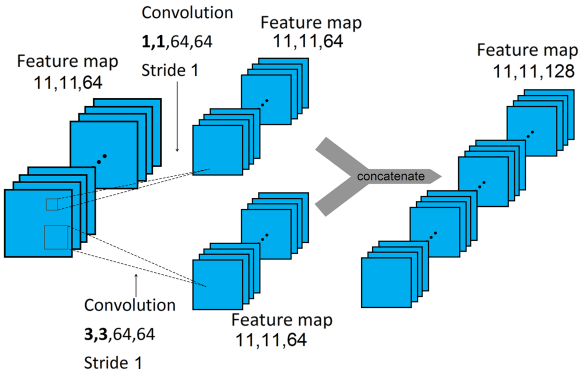

Figure 3.11: Inception module B that is used in our 3rd model, where the input is a 11×11 image and the used stride is 1

4 Results

4.1 Color Space and Optimization Algorithm

The results for training different color spaces and optimizers for the small room are in tables 4.1, 4.2, 4.3 and 4.4. They show that for the DA network the HSV color space and SGD optimizer work best. The difference between the HSV and RGB color space in mean location error is 0.071m and 2.09° in mean orientation error. The difference between the SGD optimizer and Adam optimizer is $0.097m$ in mean location error and 5.63° in mean orientation error. The CNN has better results with the RGB color space and the Adam optimizer. Using the RGB color space results in a mean location error of 0.066m and mean orientation error of 5.0° , while using the HSV color space results in a mean location error of 0.065m and a mean orientation error of 7.7° . It seems that the mean location error is the same for both color spaces, but the mean orientation error is better when the RGB color space is used. The Adam optimizer works a lot better for the CNN, with a difference of 1.452m in mean location error and a difference of 156° in mean orientation error. For further training of the models the HSV color space and SGD optimizer were used for the DA network, while the RGB color space and Adam optimizer were used for the CNN.

4.2 Localization Results

4.2.1 Results Small Room

For the DA models it can be seen in table 4.5 that the most complex model is able to reach the lowest mean location error of $0.182m$ and mean orientation error of 10.71° . The performance between the least complex model and the most complex model is not very large. The amount of neurons in the networks with a single DA layer seems to make a little difference in the performance. The model with the least amount of neurons in the DA layer outperforms the models with more neurons in the DA layer. This indicates that adding more neurons to this layer likely results in overfitting and thus a higher error in the testing set. At a corruption level of 0.2 the network using 4000 neurons improved upon the same network with a corruption level of 0.1 indicating that a higher corruption level results in less overfitting. For the models with less neurons, a higher corruption level increases both the location error and orientation error.

Two layer SDA networks seem to perform better when a high amount of neurons is used in both layers. The network with two layers of 750 neurons performs worse than a network with a single DA layer of 750 neurons. The network using 1500 neurons in the DA layers performs a bit better than the networks using a single DA layer of 1500 neurons. The network with 4000 neurons in both DA layers performs best of the two layer SDA networks. The difference with a network using one DA of 750 neurons is only $0.015m$ and 1.3°, meaning that the added complexity improves the network only a bit.

Table 4.1: DA RGB/HSV results over the test set from the small room for a model using one DA layer of 1500 neurons. Both the mean error and the standard deviation (std) are calculated over the test set.

Number of neurons in the DA layer	Color space	Corruption level	Location error (meters)		Orientation error (degrees)	
			mean	std	mean	std
[1500]	RGB	[0.1]	0.286	0.303	14.78	13.92
[1500]	**HSV**	**[0.1]**	**0.215**	**0.215**	**12.69**	**14.19**

Table 4.2: CNN RGB/HSV results, Model 1 trained on the small room

Color space	Location error (meters)		Orientation error (degrees)	
	mean	std	mean	std
RGB	**0.066**	**0.087**	**5.0**	**19.6**
HSV	0.065	0.086	7.7	25.7

Table 4.3: DA ADAM/SGD results from the small room for a model using one DA layer of 1500 neurons

Number of neurons in the DA layer	Optimization	Corruption level	Location error (meters)		Orientation error (degrees)	
			mean	std	mean	std
[1500]	ADAM	[0.1]	0.312	0.350	18.32	22.14
[1500]	**SGD**	**[0.1]**	**0.215**	**0.215**	**12.69**	**14.19**

Table 4.4: CNN, Model 1 trained on the small room with Adam compared to SGD

Optimization	Location error (meters)		Orientation error (degrees)	
	mean	std	mean	std
Adam	**0.066**	**0.087**	**5.0**	**19.6**
SGD	1.518	0.585	161	100

Table 4.5: Results DA for the small room for different model structures and different corruption levels for the DA layers

Number of neurons in each DA layer	Corruption levels for each layer	Location error (meters)		Orientation error (degrees)	
		mean	std	mean	std
[750]	[0.1]	0.208	0.216	12.6	13.6
[750]	[0.2]	0.216	0.221	12.5	13.8
[1500]	[0.1]	0.215	0.215	12.7	14.2
[1500]	[0.2]	0.218	0.213	12.6	14.0
[4000]	[0.1]	0.238	0.238	13.6	15.1
[4000]	[0.2]	0.217	0.207	12.8	13.6
[10000]	[0.1]	0.225	0.223	13.4	13.9
[750,750]	[0.1,0.1]	0.218	0.222	12.4	14.3
[750,750]	[0.2,0.2]	0.218	0.222	12.7	14.3
[1500,1500]	[0.1,0.1]	0.198	0.216	11.5	13.5
[1500,1500]	[0.2,0.2]	0.204	0.221	11.6	13.7
[4000,4000]	[0.1,0.1]	0.198	0.208	11.9	13.5
[4000,4000]	[0.2,0.2]	0.193	0.205	11.3	12.8
[750,750,750]	[0.1,0.1,0.1]	0.234	0.232	13.2	14.4
[750,750,750]	[0.2,0.2,0.2]	0.239	0.239	13.5	14.7
[1500,1500, 1500]	[0.1,0.1,0.1]	0.198	0.212	11.6	13.8
[1500,1500, 1500]	[0.2,0.2,0.2]	0.198	0.214	11.6	13.9
[4000,4000, 4000]	[0.1,0.1,0.1]	0.186	0.205	10.9	12.5
[4000,4000, 4000]	**[0.2,0.2,0.2]**	**0.182**	**0.205**	**10.7**	**12.5**

Table 4.6: Results CNN for the small room. Model 1 is the 7 layers deep, model 2 is 9 layers deep, model 3 is 9 layers deep with inception modules.

Structure	Location error (meters)		Orientation error (degrees)	
	mean	std	mean	std
Model 1	0.066	0.087	5.0	19.6
Model 2	0.069	0.101	6.2	26.0
Model 3	**0.056**	**0.079**	**4.9**	**20.7**

The models with a 3 layer SDA also performs well when a high amount of neurons is used. The model using 4000 neurons in the hidden layers reaches the best performance. When compared to the least complex single layer model, it is only an improvement of $0.026m$ in location error and $1.98°$ in angular error.

The results show that increasing the complexity of the model through means of extra layers or neurons improves the performance minimally. Higher corruption levels on the networks with 4000 neurons in the DA layers seems to counteract overfitting, because it increases the performance slightly.

Results from the CNNs in table 4.6 show that there are some minor differences in Model 1 and Model 2, but that Model 3, which uses inception, has a smaller mean in both the location and angular error, and also a smaller standard deviation. Interesting to note is that the angular standard deviation is relatively high compared to the location error, 4 times higher than the mean compared to the almost equal mean to standard deviation ratio in location error. This could be due to the fact that the image gets reduced to an 11×11 feature image in the last convolution, which makes it hard to get the precise angle of that feature.

4.2.2 Results Large Room

In table 4.7 it can be seen that the results of the large room are a lot worse than the results from the small room, but it still shows the same pattern. Increasing the complexity still has minimal effect on the performance. The most complex model with an SDA with 3 DA layers of 4000 neurons only has $0.039m$ less location error and $3.74°$ less orientation error than a model with only one DA layer of 750 neurons.

In table 4.8, the results of the convolutional neural networks in the large room can be seen. A bigger room is harder to learn for all CNNs, as seen in the error in both location and angle. It can also be seen that the gap between Model 1 and the deeper Model 2 increases, while Model 3 outperforms both other models.

4.2.3 Results Real Environment

In table 4.9 it can be seen that adding complexity and corruption has a greater effect on the results of the DAs for the real dataset. In this case the difference between the least complex model of one 750 DA layer and the most complex model of three 4000 DA layers was 0.114m which is quite a large difference. The two layer SDA with 1500 neurons in the hidden layer and a corruption level of 0.2 improved with $0.155m$ in mean location error and $5.28°$ in mean orientation error upon the one layer network with 1500 neurons in the hidden layer. This shows that adding an extra layer is effective when dealing with real data. Adding higher corruption levels to one DA layer networks seemed to improve networks by $0.060m$ to $0.064m$. Adding more layers and higher corruption levels seems to be more effective on the real data than it was on the 3D simulated data. The best network for the CNN was trained for the real environment, which is Model 3. Only the best model was trained, since training a single model took a lot of time (more than a day). The results (Table 4.10) differs a great amount from the simulated environments, as one would expect. One of these reasons could be that one side of the real environment was looking out on a room which contained a significant amount of noise (scattered objects, people etc.), compared to the remaining sides. Other contributions such as external lighting variations during the validation compared to the circumstances in the data set could also have attributed to the increase in error.

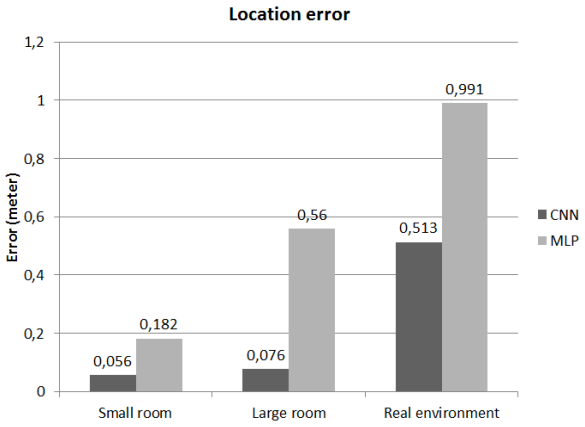

Figure 4.1: This barplot shows the mean location error of the SDA and the CNN for all three datasets

Table 4.7: Results DA for the large room for different model structures and different corruption levels for the DA layers

Number of neurons in each DA layer	Corruption levels for each layer	Location error (meters)		Orientation error (degrees)	
		mean	std	mean	std
[750]	[0.1]	0.60	0.80	32.6	48.5
[750]	[0.2]	0.60	0.86	31.0	48.2
[1500]	[0.1]	0.58	0.83	31.7	50.9
[1500]	[0.2]	0.57	0.83	31.5	50.2
[4000]	[0.1]	0.59	0.84	32.5	50.2
[4000]	[0.2]	0.59	0.85	31.3	50.6
[750,750]	[0.1,0.1]	0.57	0.83	31.4	49.5
[1500,1500]	[0.1,0.1]	0.55	0.82	29.7	50.9
[4000,4000]	[0.1,0.1]	0.57	0.82	29.9	51.2
[750,750,750]	[0.1,0.1,0.1]	0.60	0.87	31.2	48.8
[**4000,4000, 4000**]	[**0.2,0.2,0.2**]	**0.560**	**0.869**	**28.84**	**48.90**

Table 4.8: Results CNN for the large room. Model 1 is the 7 layers deep, model 2 is 9 layers deep, model 3 is 9 layers deep with inception modules.

Structure	Location error (meters)		Orientation error (degrees)	
	mean	std	mean	std
Model 1	0.16	0.19	4.0	26.7
Model 2	0.11	0.13	3.2	22.2
Model 3	**0.08**	**0.07**	**3.9**	**16.4**

Table 4.9: Results DA for the real environment for different model structures and different corruption levels for the DA layers

Number of neurons in each DA layer	Corruption levels for each layer	Location error (meters)		Orientation error (degrees)	
		mean	std	mean	std
[750]	[0.1]	1.17	1.00	35.6	82.3
[750]	[0.2]	1.11	0.97	32.2	75.9
[1500]	[0.1]	1.23	1.03	41.2	91.2
[1500]	[0.2]	1.21	1.04	31.3	77.0
[4000]	[0.1]	1.21	1.05	33.2	75.6
[4000]	[0.2]	1.15	1.03	36.6	86.8
[1500,1500]	[0.1,0.1]	1.08	0.99	36.0	84.7
[1500, 1500]	[0.2,0.2]	1.05	0.89	30.9	76.4
[4000,4000, 4000]	[0.1,0.1,0.1]	1.07	0.98	35.6	85.3
[**4000,4000, 4000**]	[**0.2,0.2,0.2**]	**0.99**	**0.97**	**33.5**	**83.4**

Table 4.10: Result Model 3 on real environment

Optimization	Location error (meters)		Orientation error (degrees)	
	mean	std	mean	std
Model 3	0.51	0.39	27.1	50.4

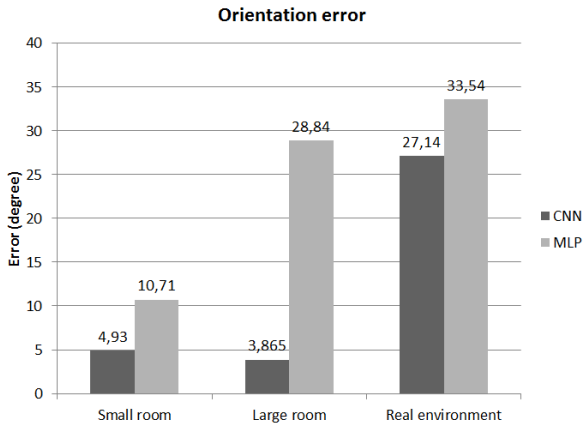

Figure 4.2: This barplot shows the mean orientation error of the SDA and the CNN for all three datasets

4.3 Heatmaps

Heatmaps were made in order to show in which areas of the environment it is hardest for the networks to estimate a location 4.3, 4.4. After the models were trained, heatmaps were made in the same manner as the data gathering was done. The robot went in steps of 0.25m through the environment. On every location the robot turns 360° and takes an image every 1°. The images that the robot captured were fed to the network which gives an estimated location. This location is compared to the ground truth to give an error in location. The mean of all errors on a location are taken and used to determine the color of the location in the heatmap. The minimum mean error of all locations was taken for the green color and the mean plus 2 standard deviations was taken as the red color. This means green indicates the area where the model performs best and red indicates where the model performs the worst. Both the CNN and SDA seem to have most problems close to walls and objects. The CNN however seems to have a lot less problematic areas than the SDA, indicating it works more consistently in general.

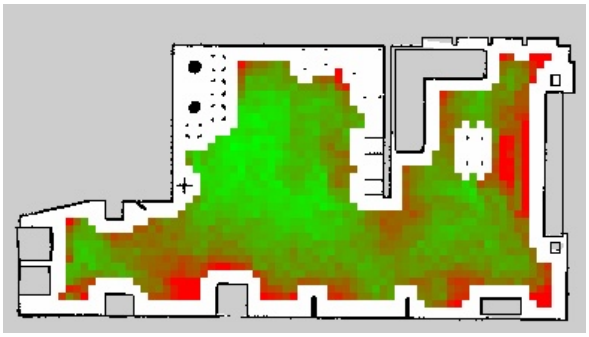

Figure 4.3: This heatmap shows an overview of which areas are hard or easy for the DA network to estimate in the large room. Green indicates good estimation while red means bad estimation. A completely green square indicates an average location error of 0.43m and a completely red square indicates a location error of 1.4m

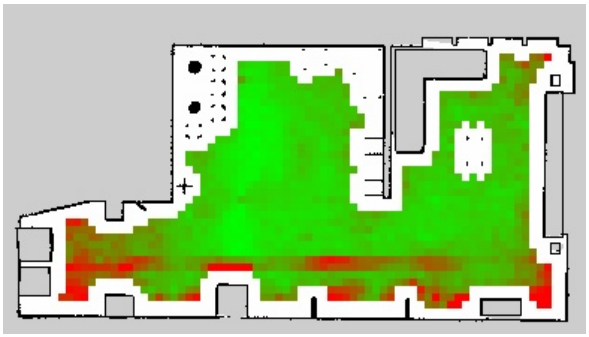

Figure 4.4: This heatmap shows an overview of which areas are hard or easy for the CNN to estimate in the large room. Green indicates good estimation while red means bad estimation. A completely green square indicates an average location error of 0.03m and a completely red square indicates a location error of 0.20m

4.4 Dreaming

The dreaming was done to visualize what the networks "think", but it is hard to judge them in a mathematical way while still giving a clear idea as to what this means. It is for example possible to compare the pixel values of the original image with a generated image, but this does not judge how well the networks generated patterns in the image. We will discuss some of the images generated by both the CNN and SDA for all three environments to sketch an idea of the performance.

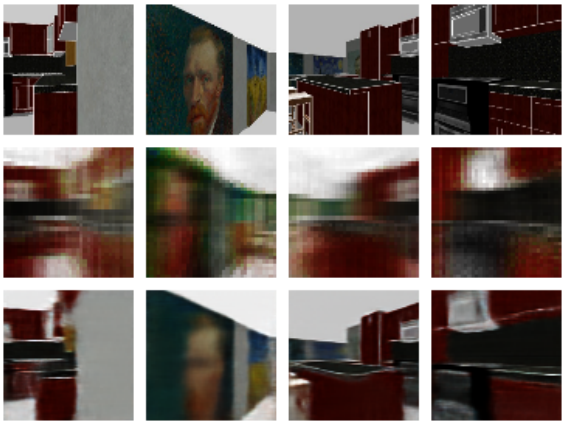

Figure 4.5: Images created from the decoder. The top images are the original images from the small room simulation, below are the images decoded with an MLP, at the very bottom the CNN.

In figures 4.5, 4.6, 4.7 some dreaming images are shown from both networks. For the 3D simulation the dreaming images are very similar to the real images, except that it is blurrier. Even very small details are still shown in the CNN as can be seen in the first image in figure 4.6, where it is even able to recreate the speakers including the small white dots on it. The dreaming images from the MLP are very vague however. They show blurry general features and contours of the environment, but the images are not very precise.

The dreaming images from the real environment are a lot worse for both networks. The CNN is able to create some images which are very similar to the real image however. The second and fourth image in figure 4.7 are good reconstructions of the real image. The first image is a lot harder for the CNN

Figure 4.6: Images created from the decoder. The top images are the original images from the large room simulation, below are the images decoded with an MLP, at the very bottom the CNN.

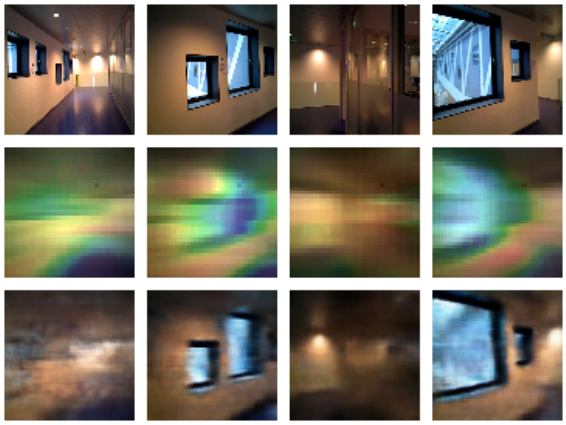

Figure 4.7: Images created from the decoder. The top images are the original images from the real environment, below are the images decoded with an MLP, at the very bottom the CNN.

to recreate and only shows a few similarities with the original. The SDA is even worse and shows only very general color patterns.

Looking at all the images decoded by both the MLP and CNN, we see that the CNN creates more recognizable and detailed images compared to the MLP.

5 Conclusions

We implemented a stacked denoising autoencoder (SDA) and convolutional neural network (CNN) for a localization task using color image information. Different structures were made for both neural networks, which were trained and tested on three different datasets. Two of the datasets were made in simulated rooms and one dataset was made in a real environment. The datasets consisted of images labeled with location and orientational coordinates. The performance of the two neural networks were compared, based on the error in location (meters) and the error in orientation (degrees). In addition, an SDA and CNN were made to do the opposite of the localization task, which we called "dreaming". The purpose of the dreaming networks was to go from locational and orientational coordinates to an image.

For the CNN the inception model had the best results on the localization task in all three environments, although the other two models perform almost as well on the small room. The SDA network with three DA layers of 4000 neurons had the best performance overall for all datasets, but for the 3D simulated environments a model using only one DA layer of 750 neurons performed almost as well. This indicates that adding complexity adds only minimal improved performance when dealing with the simulation environments. For the real environment increasing the complexity of the SDA network results in a lot more improvement. The real environment is more complex than the 3D simulated environment, so more complex models have more to gain over less complex models. This also indicates that the 3D simulations rooms are not a good representation of the real world.

For the small room the SDA is able to reach a mean location error of $0.182m$ and a mean orientation error is $10.71°$. The CNN reached a mean location error of $0.056m$ and a mean orientation error of $4.93°$. For the large room the SDA is able to reach a mean location error of $0.560m$ and an orientation error of $28.84°$. The CNN was able to reach an error of $0.076m$ and an orientation error of $3.87°$. In the real environment the SDA is able to reach a mean location error of $0.99m$ and an orientation error of $33.54°$. The CNN reached a mean location error of $0.51m$ and an orientation error of $27.14°$.

This leads us to answer the first research question: How well does localization using a CNN work, compared to localization using a (stacked) denoising autoencoder? Localization with a CNN works better than localization with an SDA in both 3D simulated environments and real environments. The performance of the CNN stays almost equal in smaller and larger environments while the SDA performs a lot worse on larger environments. The fact that the CNN is a lot deeper than the MLP means it can learn a better representation of the data. Not only is the CNN deeper, but CNNs are also very suited for learning features from images compared to the MLP, due to the kernel convolution.

An MLP and CNN were made to generate images based on location and orientation values for each of the three environments. For the 3D simulated environments the CNN generated images which looked like a slightly blurred version of the original images, even including some small details. The MLP is able to generate images showing very general patterns and contours matching the original images, but it is blurry and not precise. For the real environment the CNN is able to generate some images which resembles the original images very well, but also generates images which only show very small resemblance to the original image. The MLP is only able to generate images with very general color patterns and contours. Our second research question was: is it possible to train a convolutional neural network and a denoising autoencoder to dream an expected image from a certain position? This question is a lot harder to answer, since it is hard to judge the generated images in a scientific way. This judging was done with cross entropy or mean squared error for the training, because this guides the overall image generation, but it does not necessarily judge the details we would have liked it to prioritize. Looking back at figure 4.7, the MLP is able to generate very blurry images that roughly match the colors in the original picture. Using the mean squared error of the pixels would result in a reasonable error, but this does not mean it generates patterns very well. The generated images show that the CNN is able to dream the expected images in a 3D simulation, since they very closely resemble the original images. On the real environment the CNN is able to generate some images which closely resemble the original, but also generates images which are unlike

the original. The reason for this is that the data from the real environment contain a lot more noise due to blurring from the moving camera, but also changes in lighting from outside (there were several windows which the sun shone through) and the indoor lighting. Another reason is that the real environment contains a lot of small items scattered on one side of the hallway, which makes it a lot harder to generalize. In the 3D simulation environments the MLP is able to generate images with the general features and contours of the original image. In the real environment it is only able to generate very general color patterns, which resemble the original images only slightly. The CNN is able to decode more recognizable and detailed images when compared to the MLP and thus we conclude that the CNN is also better suited for the dreaming task.

6 Future work

Even though we obtained a relatively good result from the simulated environments, the SDA and especially the CNN did considerably worse in the real environment. A bigger dataset with a wider variety of environmental changes for the same scene (chairs, people and other movable objects, different lighting situations) might help to generalize the model. With a more diverse dataset of the environment, we could also make a Conditional Generative adversarial network (Mirza and Osindero, 2014) for generating images of the environment. It could learn the change in scene as the hidden mapping of the noise input for the Generator and Discriminator which are also conditioned by the additional location and orientation information.

Some changes in the model itself could also improve the generalization and optimization, such as batch normalization or different model structures.

Another possible way to improve this approach is by using depth information next to the RGB information. Especially for indoor environments depth information can be useful, since the measured depths will always be relatively small.

7 Task Division

Cornel Kuiper: implementation and testing of Stacked Denoising Autoencoders.
Cornel Kuiper: all sections regarding SDA and dreaming SDA (MLP) (section 2.1-2.3, 3.1)
Yiebo Chong: implementation and testing of convolutional neural nets and dreaming/decoder CNN.
Yiebo Chong: all sections regarding (dreaming) CNN (section 2.4, 2.5, 3.2, 3.3)
Cornel Kuiper & Yiebo Chong: implementation of dreaming MLP.
Cornel Kuiper & Yiebo Chong: implementation and execution of data gathering in small room, large room and real environment.
Cornel Kuiper & Yiebo Chong: implementation and execution of heatmaps
Cornel Kuiper & Yiebo Chong: Abstract, Introduction, Experiments, Results, Conclusion (section 1, 2.6, 3.0, 4-6)

References

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems.*, (19):153–160.

Endres, F., Hess, J., Sturm, J., Cremers, D., and Burgard, W. (2014). 3-D mapping with an RGB-D camera. *IEEE Transactions On Robotics*, 30(1):177–187.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 343–349, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA*, pages 2414–2423.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial networks. *CoRR*, abs/1406.2661.

Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions On Robotics*, 23(1):34–46.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2012). RGB-D mapping: Using kinect-style depth cameras for dense 3D modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision*, pages 2146–2153.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Koenig, N. P. and Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *IROS*, pages 2149–2154. IEEE.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA*, pages 3431–3440.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *CoRR*, abs/1411.1784.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proc. International Conference on Machine Learning*, pages 807–814.

Shantia, A., Timmers, R., Schomaker, L., and Wiering, M. (2015). Indoor localization by denoising autoencoders and semi-supervised learning in 3d simulated environment. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Computer Vision and Pattern Recognition (CVPR)*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P. A. (2011). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research : JMLR.*, 11(2):3371–3408.