An Evaluation of Human Pose Estimation Using a Deep Convolutional Neural Network

SVEN ABELSSON RUNING

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2017:11

An Evaluation of Human Pose Estimation Using a Deep Convolutional Neural Network

SVEN ABELSSON RUNING

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2017

An Evaluation of Human Pose Estimation Using a Deep Convolutional Neural Network
SVEN ABELSSON RUNING

Supervisor: Erik Landolsi, Visionists AB
Examiner: Fredrik Kahl, Department of Signals and Systems

Master's Thesis 2017:11
Department of Signals and Systems

Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Network predictions of the human pose from a set of test images.

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2017

An Evaluation of Human Pose Estimation Using a Deep Convolutional Neural Network

SVEN ABELSSON RUNING
Department of Signals and Systems
Chalmers University of Technology

Abstract

The purpose of this master thesis was to evaluate how a deep convolutional neural network can be used to estimate a human body pose in a 2D image. The evaluation was done by implementing a state of the art network and test its performance on the MPII human pose dataset. Three different ways to preprocess the training data was tested and the network's accuracy went from 60.8% for the first method to 71.2% for the third method, a roughly 17% increase in performance. These results indicate that the choice of preprocessing method have a large effect on the overall performance of the network.
An analysis of the MPII dataset revealed that the distribution of poses is skewed towards poses facing the camera in an upright position. A score was calculated for each pose that measures how much a pose deviates from the norm. Using this pose score it is shown shown that the network performs much worse on poses that are rare in the MPII dataset. This indicates that network has learned how the keypoints are distributed and has overfitted to the MPII dataset. To improve the generalization of the network a more thorough data augmentation should be implemented.

Keywords: pose estimation, CNN, MPII, preprocessing, training data, rare pose, augmentation

Acknowledgements

I would like to thank my superviser at Visionists AB, Erik Landolsi for all the help and support during the thesis work. I'm also grateful to Mattias Johansson and Stefan Karlsson at Visionists for their help and many constructive ideas. Finally I would like to thank my examiner Fredrik Kahl for his feed back and support.

Sven Abelsson Runing, Gothenburg, November 2017

Contents

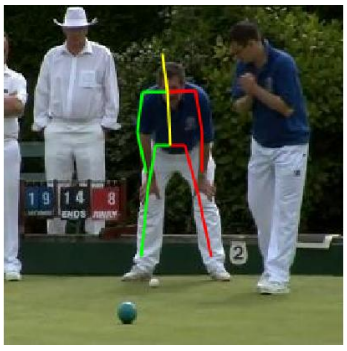
List of Figures

List of Tables

1
Introduction

The task of finding the pose of a human in an image, is a complex and longstanding problem in computer vision. Developments in deep convolutional neural networks (CNNs) have led to major advances in recent years. The goal of this thesis is to evaluate how well a state of the art deep CNN can estimate the pose of a human in a 2D image. Figure 1.1 shows an example of a pose estimation.

(a) An image where you want to estimate the pose.



(b) After cropping the image around a person, the network can make an estimation of the pose.

Figure 1.1: An example of a pose estimation in a 2D image.

The network will be trained on the MPII Human Pose dataset [1], which contains a great variety of annotated poses with different backgrounds and clothing.
There will be extra focus on the effect of different preprocessing techniques of the training data and how well the network can handle unusual poses.
The ability to automatically find the articulated body pose of a human is applicable in a wide range of technologies. Some of the technologies that would benefit from an accurate body pose estimation includes: human-robot interaction, surveillance, assisted living for the elderly. From the estimation of a human body pose in a single image, the step is not far to be able to analyze a sequence of images in a video. The constant increase in image data, driven by easier access to digital cameras in all areas of society, adds to the need of automatically being able to analyze and label images and videos of human activities.

1.1 Aim

The aim of this study is to explore how a deep convolutional neural network (CNN) can be used to estimate the human body pose by implementing a state of the art network in a deep learning framework called caffe [2]. The main questions this thesis will address is:

- How well can the network handle different conditions in backgrounds, occlusion, clothing, body sizes?
- How does the preparation of the training data influence the performance of the network?
- Analyze the MPII dataset by scoring each pose depending on its rarity. Are poses that are more rare in the MPII dataset more difficult to predict?

Furthermore, the network's predictions will be compared to standard benchmarks and there will be a discussion of what optimizations that can be made to improve the network and what the main limitations of the network are.

1.2 Limitations

The position of the human body will be described by the location of a set of keypoints: the wrists, elbows, shoulders, ankles, knees, hips, lower neck, head top, pelvis and thorax. The problem of finding the articulated body pose will be limited to 2D images and the output coordinates will be in 2D image coordinates.
It will be assumed that the rough position of the person to be annotated is known. This is because the network lack ability to decide what person it should estimate the pose of. For the network to perform well, the image need to be preprocessed in such a way that the person to be annotated is roughly in the center of the image.
The network can only handle images of size 256 x 256 pixels, images of other sizes needs to be cropped and/or resized.

2

Background

2.1 Human pose estimation

The goal of the human pose estimation can vary. Work has been done to use a single 2D image to generate a 3D body pose estimation [3], as well as using a 2D depth image to generate the 3D body pose estimation [4]. The focus of this thesis work is to estimate a human body pose from a single 2D image. The body pose is represented by the major joints in the body (Figure 2.1), wrist, elbow, shoulder, ankle, knee, hip, lower neck and top of the head, thorax and pelvis. The task is to estimate the image coordinates of these joints.

Figure 2.1: The keypoints that make up the pose.

According to [5] the most challenging aspects of human pose estimation is: (1) The great variability in human visual appearances (clothing, accessories, hairstyles), (2) variability in lighting conditions, (3) variability in human physic (tall, short, overweight, thin), (4) partial occlusion due to self occlusion or layering of objects in the scene, (5) complexity of the human skeleton (the human body has 230 joints), (6) this leads to a high dimensionality of the pose (244 degrees of freedom), (7) the loss of 3D information from viewing a 2D image. The images in Figure 2.2 are taken from the MPII dataset and illustrate the challenging conditions of human pose estimation.

Figure 2.2: Images from the MPII datase, illustrating the difficulty of human pose estimation.

Attempts to solve human pose estimation before CNNs generally lacked a holistic view of the problem [6] [7] and could not produce results close to those achieved by CNNs. To the best of my knowledge, the first attempt to solve the human pose problem with a convolutional neural network was done in 2013 by two Google employees [8]. They used convolutional and fully connected layers to regress to the x-y coordinates of the body joints. Their results were state of the art at the time and showed that CNNs could get a holistic view of the pose. Since then, state of the art in body pose estimations have involved CNNs.
The network layout used for this thesis work, closely follows the layout in a paper from 2016 [9] that achieved top results on the MPII Human Pose dataset [1]. This type of fully convolutional neural network seems very suitable for pose estimation. This is probably due to a CNN's ability to both handle local image information (detecting parts, elbow, wrist etc) and global image information (connecting the parts).

2.2 An introduction to Convolutional Neural Networks - CNNs

Over the past couple of years, CNNs have made great contributions to human pose estimation and computer vision in general. This section aims to give a brief intro-

duction of how CNNs work and why they are effective for computer vision.
The basic computational unit in a neural network is the artificial neuron, Figure 2.3. The input values, x_1 to x_m, are multiplied with their associated weights, w_0 to w_m, and the result is summed up and passed through an activation function. The The output y of the neuron is calculated as:

$$y = g(\sum_{i=0}^{m} w_i x_i)$$

The artificial neuron is inspired by the neuron found in the brain of animals. The inputs can be seen as the dendrites and the activation function controls when and how strongly the axon (output) should fire to the next neuron. Typically the network has many neurons in a layer and the neurons in the first layer forms the input to the neurons in the second layer and so on.

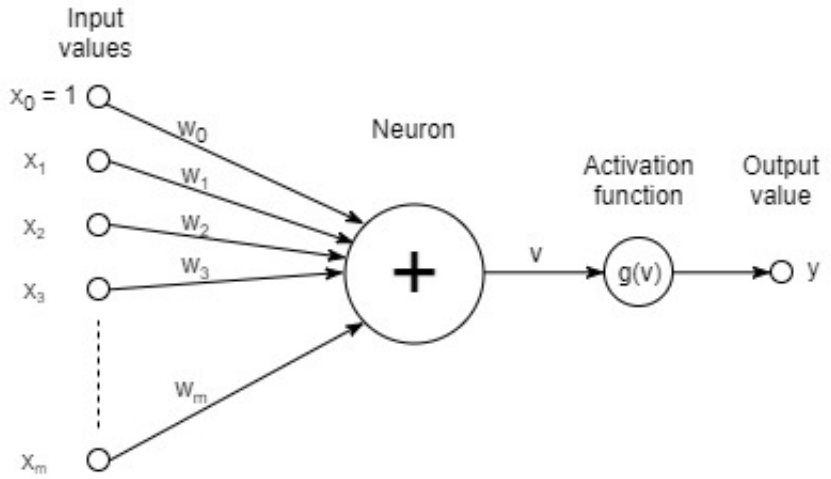


Figure 2.3: An artificial neuron, x_0 is called a bias and is commonly set to 1.

CNNs are a group of feed forward, artificial neural networks that are built up by convolutional and max-pooling layers. In a CNN, a neuron has a limited number of inputs from the previous layer (receptive field), x_1 to x_m, controlled by the kernel size. The effective receptive field of neurons grow larger in the deeper layers and gives CNNs the ability to combine local features into global features in the deeper layers. Figure 2.4 shows a simple structure for a CNN that is used for classification. The input to the network is an image and the output is the predicted class of an object.

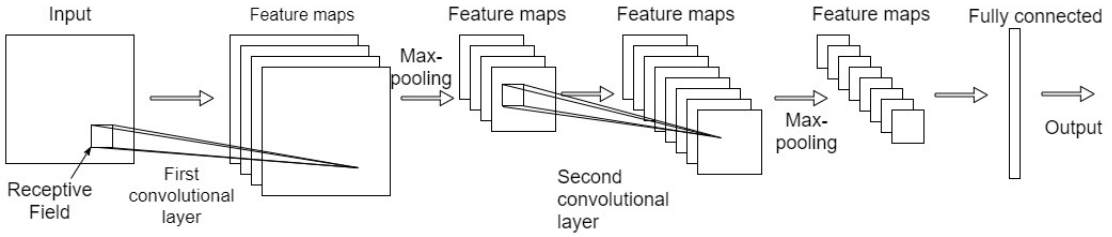


Figure 2.4: A simple CNN architecture with two convolutional layers.

This type of structure is useful when determining *what* is in an image e.g. to see if there is any human keypoints in the image, however it does not give much information of *where* the object is.

2.2.1 Convolutional layer

The main building block of a CNN is a convolutional layer with learnable filters. The layer performs a convolutional operation between the input and the filters and pass on the result to the next layer. The result of the convolution is a 2D plane called a feature map. Normally several learnable filters are used so that the output of the convolutional layer is a 3D volume, $H \times W \times D$ where H and W are the spatial dimensions and D is number of feature maps (numbers of filters used). The network in Figure 2.4 for example has 4 filters in the first layer and 8 filters in the second layer. The spatial dimension W_{out} of the output is determined by the filter kernel size (K), stride (S), zero padding (P) and W_{in} according to the following formula:

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

Almost all convolutional layers in the network implemented in this thesis use a kernel size of 3×3, a zeropadding of 1 and a stride of 1 so that the spatial dimensions are unchanged. A nice illustration of the different types of convolutions can be found at [10].
The design of the convolutional layer is inspired by biology and how the cat's visual cortex functions [11]. Each cell in the visual cortex looks at a small region of the visual field, called a receptive field. Together, the cells cover the entire visual field and each cell behaves like a local filter, detecting features like edges. The first convolutional layers detects local features like edges and textures. Convolutional layers later in the network combine these local features to higher level features [12]. During training, the network updates and learn the filter parameters that extract the most useful features for the task. Letting the CNN learn these filters instead of handcrafting them both saves time and give better results.

2.2.2 Max-pooling layer

The max-pooling operation returns the largest value in a kernel and discards the other values, Figure 2.5 shows the most common type of max-pooling. The max-pooling layer is inserted between convolutional layers to gradually bring down the spatial resolution. This has the effect of decreasing the amount of parameters and thus the amount of computations in the network is decreased [13]. Reducing the amount of parameters in the network also has a regularizing effect and decrease overfitting.

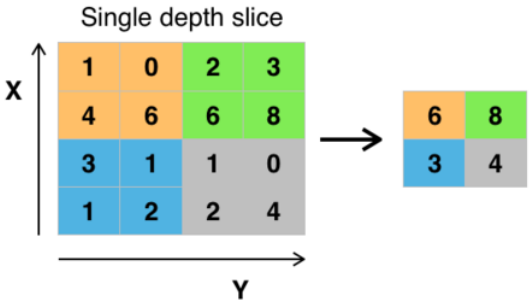


Figure 2.5: Max-pooling operation with kernel size 2x2 and stride 2.

2.2.3 ReLU layer

The ReLU (Rectified Linear Unit) [14] activation function is used after each convolutional layer. The ReLU is a non-linear function: $f(x) = max(0, x)$ and it has been found to speed up training for deep networks [15] compared to the hyperbolic tangent and sigmoid activation functions. The ReLU used in this network is called a Leaky ReLU [16]. It has been found that the Leaky ReLU can converge faster than the regular ReLU. The Leaky ReLU differs from the regular ReLU in that it has a non-zero slope for the negative part of the activation function: $f(x) = 0.1x, x < 0$ and $f(x) = x, x > 0$.

2.2.4 Training the network

During training, an image is input to the CNN and convolutional and max-pooling layers calculate their outputs and feed them to the next layer until finally outputting a prediction, this process is called forward propagation. In supervised learning, the predicted result is compared to a ground truth and a loss function is applied and the closer the prediction is to the ground truth, the smaller the loss is. The network in this thesis uses a euclidean (also called L2) loss function and it has the following definition, where N is the number of outputs, y is the ground truth and $\hat{y}$ is the predicted output:

$$Loss = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i))^2$$

After the loss has been calculated, the back-propagation algorithm [17] is used to find how much each neuron contribute to the loss. It does this by taking the derivative of the loss function with regards to the network parameters. If Θ is the parameter vector of the model, α the learning rate and $J(\Theta)$ the loss, then the updated of the i-th parameter can be written as:

$$\Theta_i = \Theta_i - \alpha\frac{\partial}{\partial\Theta_i}J(\Theta)$$

The error contribution is then propagated backwards through the network and the parameters are updated to minimize the loss. The learning rate is a metaparameter of the model and has to be fine tuned for the specific task. Figure 2.6 shows the basic steps of supervised learning.

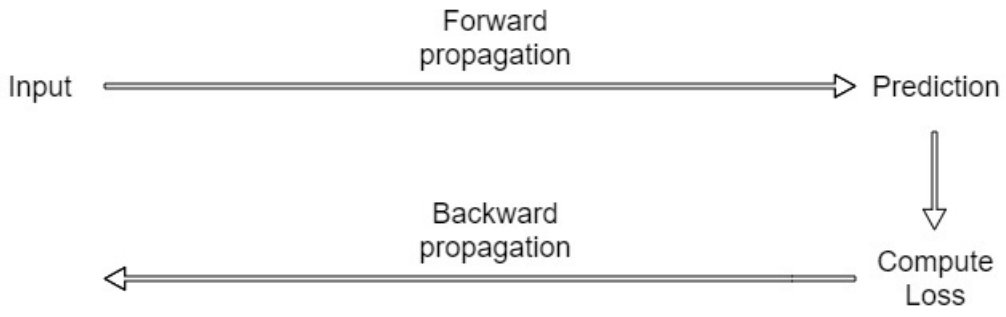


Figure 2.6: One iteration of supervised learning.

To speed up the training, several training examples (called a mini-batch) are typically processed in parallel during one training iteration.

It is common practice to split up the training data into three parts: a training set, a validation set and a test set. The training set is used for training the network and periodically the training is interrupted to let the network do predictions on the validation set. The predictions on the validation set will result in a loss that is plotted to keep track of how the training is progressing. As long as the loss on the validation set is decreasing, the training continues. When the network has seen all training examples in the training set one time, it is called an epoch and Figure 2.7 shows the loss during training when plotted against the number of epochs. The reason the graph is jagged is that the network has been trained with stochastic gradient descent [18].

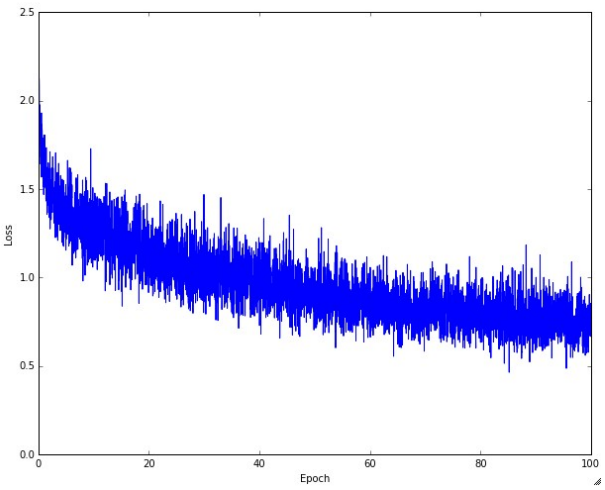


Figure 2.7: The loss plotted as a function of the number of epochs.

When the loss has stopped to decrease, the network is tested on the test set. If the network performs well on the validation set but poorly on the test set, is called that the model has overfitted to the validation set and failed to generalize well.

2.2.4.1 Batch Normalization

Batch normalization is used consistently throughout the network implemented in this thesis. Batch normalization addresses a common problem when training deep neural networks, that parameter changes in the early layers can greatly change the distribution of the input to later layers [19]. To solve this problem and make learning more easy for later layers, batch normalization normalizes all inputs to a layer for a given mini-batch (a set of training examples that are processed in parallel during training). If x is an input to a layer, $B = (x_{1...m})$ is all the inputs of x in a batch and y_i is the batch normalized output, batch normalization is described by the following formulas:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma\hat{x_i} + \beta \equiv BN_{\gamma,\beta}(x_i)$$

The parameters γ and β are introduced to keep the representation power of the network and are learned during training. The mini-batch mean and variance, μ_B and σ_B are only used during training. When using the trained network for inference, the global mean and variance of the entire training set is used. The larger the mini-batch size is, the better batch normalization performs, since the estimations of the mean and variance becomes less noisy. After normalization layers will have inputs with zero mean and a standard deviation of one and this more predictable distribution allows for the network to be trained with a higher learning rate. Batch normalization also has a regularizing effect and makes the network less sensitive to how the network parameters are initialized.

2.2.5 Convolutional layers vs fully connected layers

Compared to neural networks made of fully connected layers, CNNs need much fewer weights and they better exploit the property that pixel values that are close to each other are more correlated than pixel values that are far apart. Neural networks made of fully connected layers tend to not scale well to larger images. For an image of size 32×32 with three color channels, a single fully connected neuron would have $32 \times 32 \times 3 = 3072$ weights. If the size of the images goes up to $256 \times 256 \times 3$, a single fully connected neuron would have 196 608 weights. Normally there would be several neurons in a layer which means that the number of weights goes up fast. The large number of weights demand a lot of memory during training and makes the fully connected network prone to overfitting [13].
In CNNs, the weights of a filter is shared over the image. The sharing of weights and the small filter sizes makes CNNs have much less weights than fully connected layers. In the example of an image of size $256 \times 256 \times 3$, a convolutional layer with a filter size of 3×3 and 128 features would have $3 \times 3 \times 3 \times 128 = 3456$ weights. This is significantly less than the fully connected layer. That the CNNs use less memory makes it possible to stack many layers after each other in so called deep CNNs. This has been proven to be effective when the network needs to learn more abstract and complex tasks. For example deep CNNs have been succesfully used in Alphago [20], that was able to beat the world's best player in the board game Go and in the ImageNet challange [21], [22].

3
Methods

3.1 Before training the network

Before any training could begin it was important to know what type of data the MPII data set contained, decide what data should be included in the training and how to preprocess the images.

3.1.1 The MPII dataset

The training data used to train the network comes from a dataset called MPII Human Pose dataset [1]. The dataset is a state of the art benchmark for evaluating articulated human pose estimations. The images are taken from youtube videos and are greatly varied in regards to human poses, backgrounds, clothing, body size, distance and angle to the annotated individual. The dataset consists of roughly 25k images, containing over 40k people with annotated body joints. The size of the dataset is 12.9GB for the images and 12.5MB for the annotations. The annotations are provided in a MATLAB structure and the information per image is listed bellow.

- Annotated list of images
  - Image name
  - Body annotations for each person in the image
    - x1, y1, x2, y2 - coordinates of head rectangle
    - Scale - person scale w.r.t. 200 px height
    - Object position - rough human position in the image
    - Annotated keypoints - person-centric body joint annotations
      - .x, .y - coordinates of a joint
      - id - joint id (0 - r ankle, 1 - r knee, 2 - r hip, 3 - l hip, 4 - l knee, 5 - l ankle, 6 - pelvis, 7 - thorax, 8 - upper neck, 9 - head top, 10 - r wrist, 11 - r elbow, 12 - r shoulder, 13 - l shoulder, 14 - l elbow, 15 - l wrist)
      - is visible - joint visibility
- List of training/testing image assignment
- Single person - contains the id of sufficiently separated individuals

Figure 3.1 shows the annotated data for an image in the MPII dataset.

Figure 3.1: Annotation in the MPII dataset. **Blue rectangle:** head rectangle, **yellow rectangle:** bounding square, **red circle:** center of the annotated person, **green circles:** keypoints.

A bounding square can be calculated with object position (objpos) and scale but it is frequently too small as illustrated in Figure 3.2a. Frequently ankles are cut of and therefor left out of the training data. To address this problem, the y-coordinate is increased by 15 and the scale is increased by a factor of 1.25. Figure 3.2b shows the adjusted bounding square. With the object position and the scale from the MPII dataset it is trivial to calculate the adjusted bounding square as follows (x,y is the upper left corner):

$$side = scale \times 200 \times 1.25$$

$$x = objpos.x - \frac{side}{2}$$

$$y = objpos.y - \frac{side}{2} + 15$$

(a) MPII bounding square **(b)** Adjusted bounding square

Figure 3.2: Bounding square before and after adjustment.

3.1.2 Training data

Only annotations of sufficiently separated individuals are used and there are roughly 24k such annotations in total. Roughly 4200 of these images were used for validation. Before training could start the annotations of each image in the MPII dataset had to be converted into 16 label-images (one for each annotated joint) of size 64×64. A 2D Gaussian hill (7 pixels in diameter and standard deviation of 1) was placed in each label image at the x-y coordinate of the corresponding joint (Figure 3.3). The labeled images were stacked to create a $16 \times 64 \times 64$ volume of labels for each training image (Figure 3.4). The stack of labeled images was used as ground truth during training.

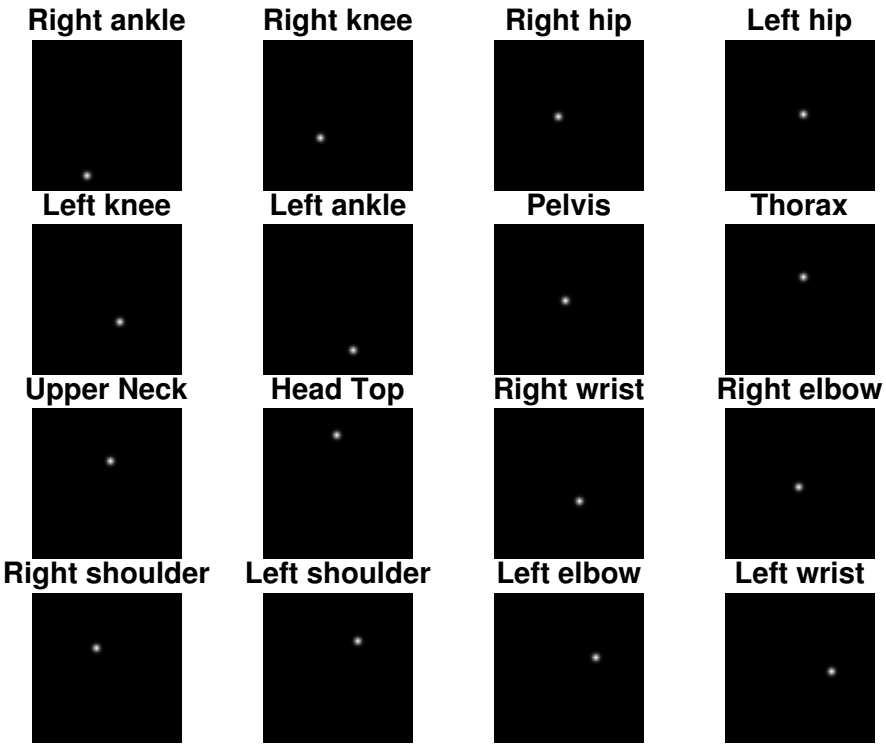


Figure 3.3: The 16 ground truth images for the keypoints in the yellow square in Figure 3.1.

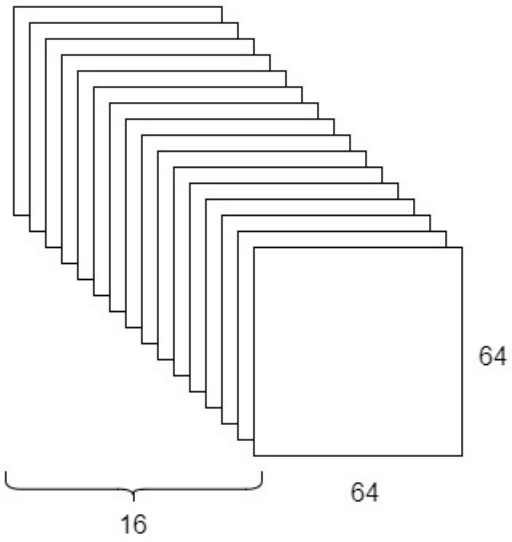


Figure 3.4: Stacked ground truth images.

In the MPII dataset, the ground truth of occluded joints is given and can be included in the training data. However joints that are missing from the image or are severally occluded do not have a ground truth annotation in the MPII dataset. In this case a ground truth of zeros is used as training data.
One data augmentation is used to expand the training data and that is flipping the image around the vertical axis. With data augmentation there is a rough total of 48k training images and 768k label-images. In order to make it easier to handle the large amount of images, a Python [23] script was used to place the images into an LMDB database [24].

3.1.3 Preprocessing of training data

The resolution of the images in the MPII dataset varies and since the CNN takes an images of size 256×256 as input there is a need for both cropping and resizing of the original image before feeding it to the CNN. There are different ways to crop and resize the original image and there is also a choice whether to include occluded keypoints in the training data or not. During the process of trying to improve the performance of the CNN, three different ways of preprocessing were implemented and evaluated. All preprocessing methods use a bounding square to crop the image so that the aspect ratio is preserved. The preprocessing was done using the commercial software package MATLAB [25].

3.1.3.1 First preprocessing method - M1

Figure 3.5 shows the original image of a skier before preprocessing. The bounding square is not allowed to be outside of the original image in the first preprocessing method. If the side of the bounding square is larger than the image, the side is set to min(height,width). This way of cropping the image does not ensure that the

annotated person is in the center of the image as can be seen in Figure 3.6.

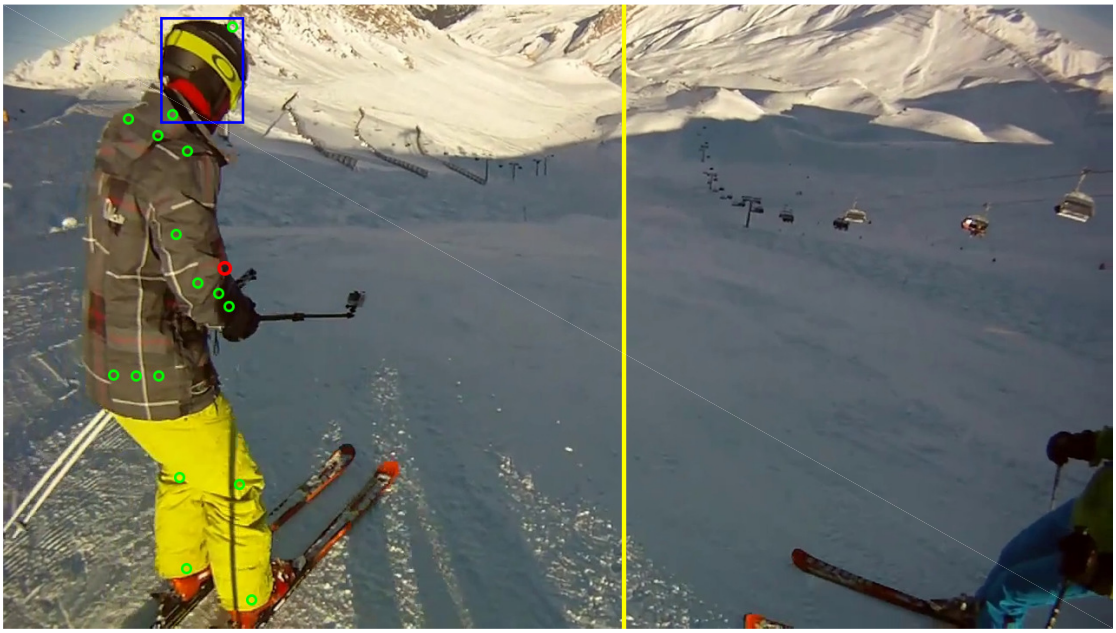


Figure 3.5: An image of a skier before preprocessing.

The first version of the training data only contained ground truth annotations for visible joints, the ground truth for occluded joints were ignored and set to zero. Figure 3.6 shows this type of preprocessing. The reasoning for removing the occluded keypoints was that, keypoints that are not visible can not give the network information about the pose, therefore they can be excluded from the training data. This approach provides the network with less keypoint annotations during training and also prevents the network from learning to predict the location of occluded keypoints.

Figure 3.6: Uncentered training data without occluded keypoints.

3.1.3.2 Second preprocessing method - M2

In the second version of the training data, all keypoints with a ground truth was used for training (Figure 3.7). In the MPII dataset, most keypoints have annotations even if they are not visible as long as they are estimated to be inside the image. Only keypoints that are clearly not in the image or are severely occluded lack annotation. Even if these keypoints are not visible, it was thought that seeing them during training would be beneficial for the network. With the second set of training data, it could be investigated if the network could learn to predict the location keypoints even if they were occluded. The second version of the training data uses the same type of uncentered bounding square and cropping as in the first version.

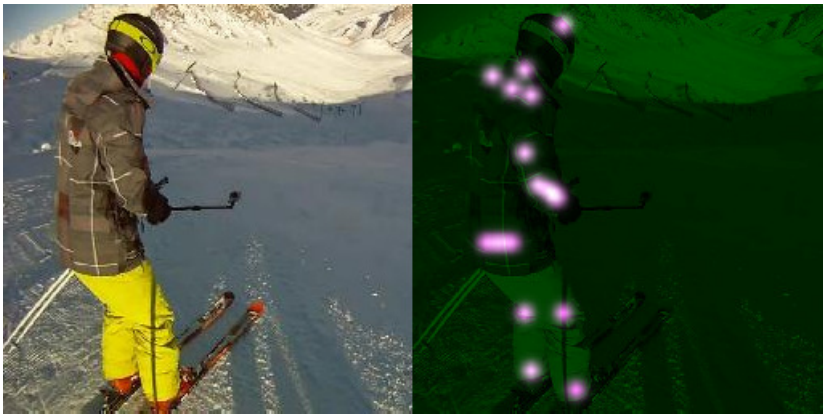


Figure 3.7: Uncentered training data with occluded keypoints included.

3.1.3.3 Third preprocessing method - M3

The third and last version of the training data include occluded keypoints but use a different way to crop the image so that the annotated person is always placed in the center of the image. Figure 3.8 is used to illustrate the third preprocessing method.

Figure 3.8: The boy in the lower left corner is close to the edge.

The second preprocessing method will use the yellow bounding box to crop the image and then resize the image to 256 x 256. When the annotated person is close to the edge like in Figure 3.8, it will result in a stretched image and a changed aspect ratio (Figure 3.9).

Figure 3.9: The second preprocessing method stretches the images and the boy is not in the center.

Zero padding is used to center the annotated person and avoid the problem of stretching the image. In zero padding, zeros are added to the image where the bounding box exceeds the edges of the image. Figure 3.10 shows the result after the third preprocessing method.
Without centering the image around the person to annotate, it would be difficult for the network when there are multiple people relatively close to each other in the image, because it would be unclear who to annotate. The purpose of this change in the training set was to study how much performance could be improved if the network could assume that the center of the person to annotate was always in the center of the image. This represents a simpler problem where the person is already

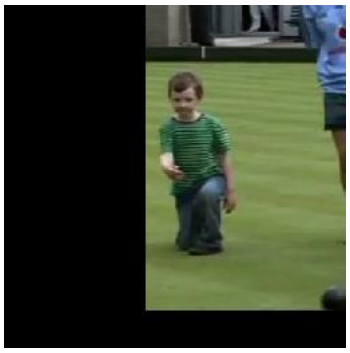


Figure 3.10: Zero padding has been used to center the boy and avoid stretching the image.

located, the remaining task is to estimate the pose.

3.2 Training the network

The network was trained with a layout that closely follows the one used in Stacked Hourglass Networks for Human Pose Estimation [9]. The input is a $256 \times 256 \times 3$ color image and the output is a volume of $64 \times 64 \times 16$, one 64×64 image per keypoint. Figure 3.11 shows an overview of the network layout.

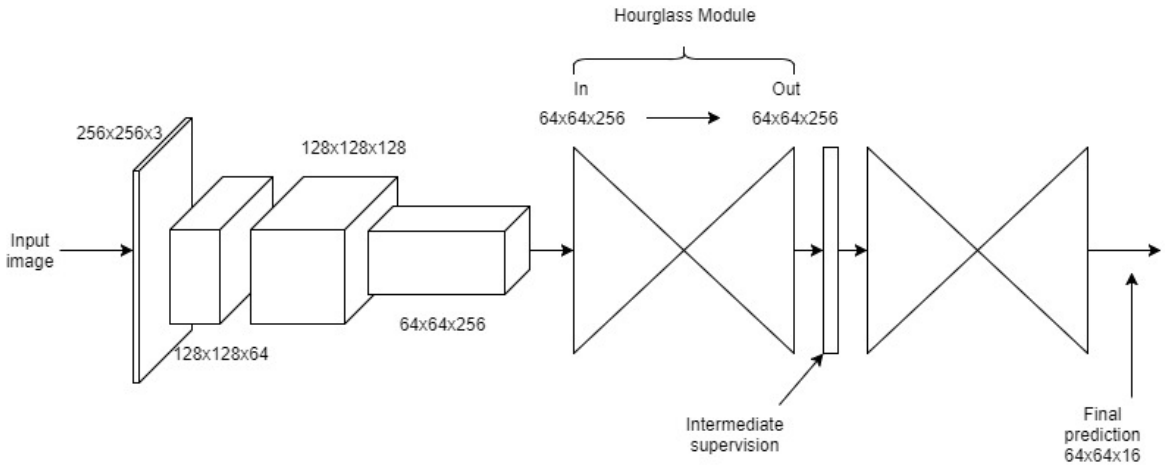


Figure 3.11: The network layout

The main part of the network consists of the hourglass modules which takes an image of size 64×64 and 256 features as input. The first part of the network preprocess the input image so that it can be fed into the first hourglass module. This is done by convolutional layers and max pooling and the details of this first step is shown in Table 3.1.

Layer	Kernel	Stride	Padding	Output
Input image				$256 \times 256 \times 3$
Convolution	7	2	3	$128 \times 128 \times 64$
Convolution	3	1	0	$128 \times 128 \times 128$
Max pooling	2	2	0	$64 \times 64 \times 128$
Convolution	3	1	0	$64 \times 64 \times 256$

Table 3.1: First part of the network that prepares the image for the first hourglass.

3.2.1 Residual unit

The residual units are the main building block of the hourglass network. It combines several of the recent advancements in deep neural networks, made during the last couple of years. Figure 3.12 shows an overview of the residual unit.

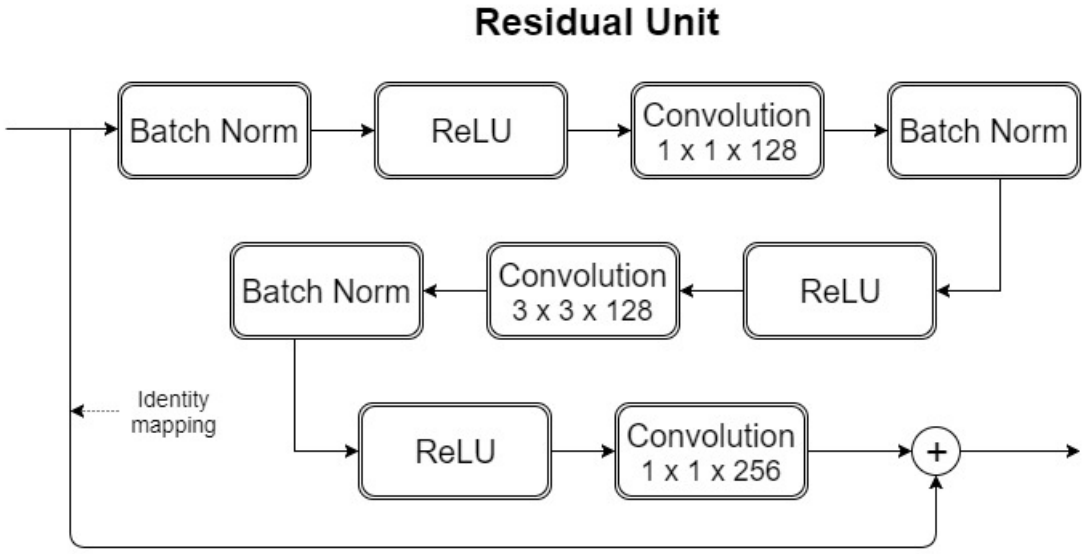


Figure 3.12: The residual unit used throughout the network.

An identity mapping of the residual units indata is elementwise added to the output of the last convolutional layer. This type of skip-layer helps to optimize and train deep neural networks [22]. Batch normalization is performed after each convolution in the residual unit. Batch normalization has been shown to greatly decrease the training time of deep neural networks and have a regularizing effect [19]. The number of features coming in and out of the residual unit is 256 throughout the network. The reason for the 1x1 convolution before and after the 3×3 convolution is to lower the number of weights needed to be saved for the residual unit [21]. A 3x3 convolution with 256 features coming in and out requires $256 \times 9 \times 256 \approx 590000$ weights. A 1x1 convolution to bring down the features, followed by a 3x3 convolution and a 1x1 convolution to get back to 256 features requires: $256 \times 1 \times 128 + 128 \times 9 \times 128 + 128 \times 1 \times 256 \approx 213000$ weights. Using half the number of features reduces the amount of parameters saved in memory by roughly 64% compared to do the

3x3 convolution with full features. Saving memory on the GPU is important as it allows for deeper nets and larger batch sizes which in turn speed up training and the efficiency of batch normalization.

3.2.2 The Hourglass module

The network needs to capture both local features like elbows and wrists as well as global features of how the keypoints are interconnected. The first part of the hourglass module (Figure 3.13) is a series of down sampling units. The spatial resolution is progressively reduced by max pooling, from 64×64 at the start to 4×4 in the middle. This first part is similar to network layouts used for image classification [15], where the main part of the network is a series of convolutional and max pooling layers. The first downsampling units can learn more local features like edges and patterns while the last units learn more global and abstract features [12]. Generally speaking, the downsampling part of the hourglass is concerned with what is in the image.

In the case of human pose estimation it is not enough to know that there is elbows, wrists and other keypoints in the image, the problem also consist of determining their location. The problem is similar to image segmentation and the hourglass modules layout is inspired by network layouts that have successfully been used for image segmentation. The parallel connections in the hourglass module is also found in [26] where they are used to increase the granularity of the predictions. The gradual upsampling used in [27], as well as in the hourglass module, further improves the details of the prediction.

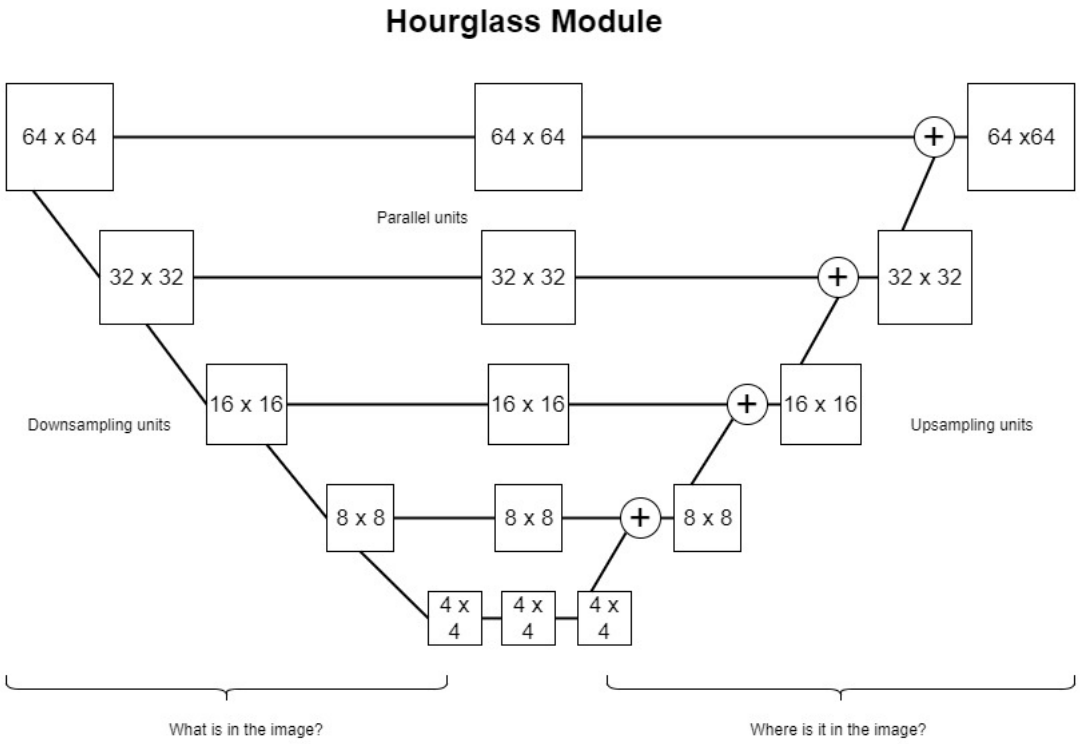


Figure 3.13: The Hourglass Module

The localization of the keypoints takes place in the upsampling units. Before each upsampling unit, the spatial resolution of the previous unit is increased through deconvolution and added element-wise to output of the parallel unit. In the upsampling units, more global features from the low spatial resolution layers are combined with more local features from higher spatial resolution layers. Through a series of such upsampling units the spatial resolution is brought back up to 64×64. The purpose of the parallel connections are to preserve the granularity of the features.

3.2.3 Connection layer

The network consists of two hourglass modules linked with an intermediate supervision. This connection layer can be seen in Figure 3.14. The output of the first hourglass is split in two directions before being element-wise added together again before the second hourglass. One path goes through two residual units and the other one makes predictions for the 16 keypoints and a euclidean loss function is applied. Since all residual units have a skip-layer, the original input to the first hourglass is also part of the input to the second hourglass.

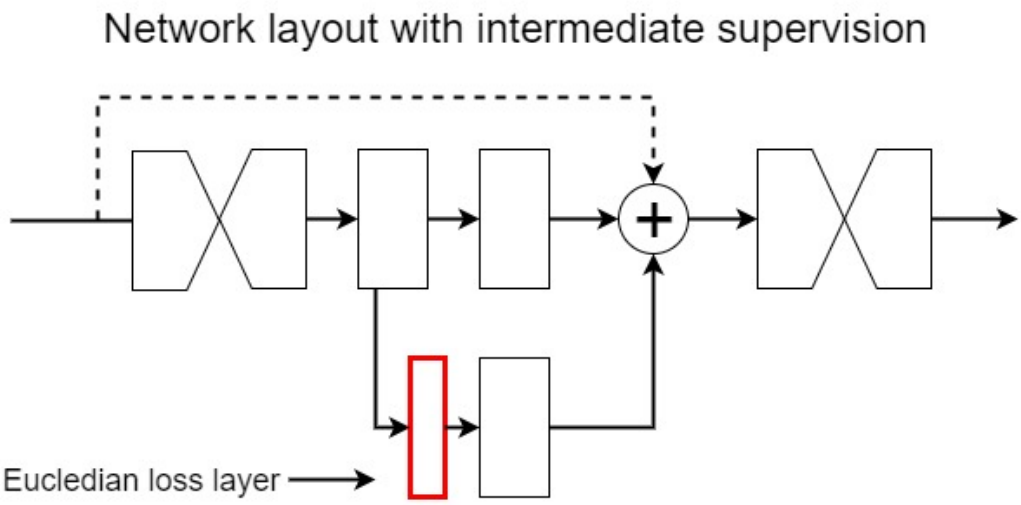
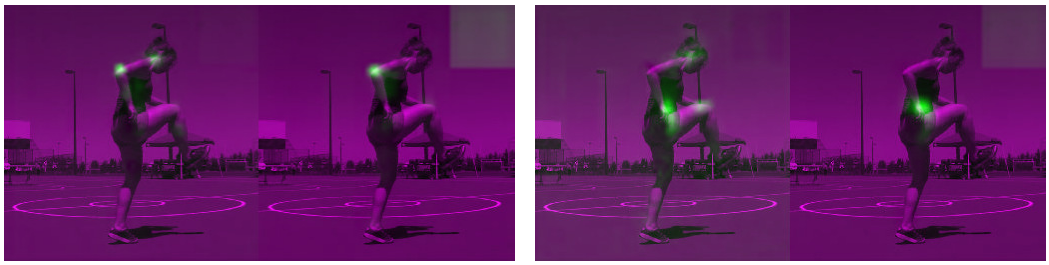


Figure 3.14: The connection layer

Figure 3.15 shows how the predictions in the connection layer and the final prediction. It can be seen that the prediction is gradually refined by each hourglass.



(a) Right elbow **(b)** Right wrist

Figure 3.15: Intermediate and final predictions

3.2.4 Training details

The training was done on a PC with the Ubuntu operating system and a 12 GB NVIDIA TitanX graphics card. The framework chosen for the training is called Caffe [2]. The network consisted of two stacked hourglasses and Table 3.2 shows the metaparameters used during training.

Optimization algorithm	RMSProp [28]
Learning rate	2.5e-4
Momentum	0.9
Momentum2	0.999
Weight decay	5e-5
Mini-batch size	6

Table 3.2: Parameters used for training

When the loss stopped decreasing, the learning rate was lowered by a factor of 5 and training continued for one more day. Total training time was about two days. The Euclidean loss function was used for the supervision between hourglasses and on the final prediction of the network.

3.3 After training the network

The MATLAB interface matcaffe [29] was used for testing and evaluating the network after training. Matcaffe can load the weights and layout files from caffe and run forward propagation on test images to make predictions.

3.3.1 Visual evaluation

One way to visualize the predictions of the network is to overlay the predicted heatmaps over the original image. This gives an indication of what type of joints are difficult and what type of mistakes the network is prone to do and was especially helpful during the early phase of testing. Figure 3.16 shows a test image used to make the predictions.

Figure 3.16: Image used to make predictions.

In Figure 3.17 it is possible to see how sure the network is in its prediction of each keypoint. The predictions of the ankles shows that the network is confused as to the location of these keypoints and this is due to occlusion. The strength of the predictions of occluded keypoints are low compared to visible keypoints.

Figure 3.17: Predictions of all 16 keypoints with a relative color scale where more green means increased likelihood of keypoint.

An easier way to visualizing the network predictions is to only show the maximum value of each keypoint and connect them to a skeleton, Figure 3.16 shows an example. A threshold of 0.1 is used to discard predictions that are of too poor quality.

Figure 3.18: The predictions of the knees and ankles are not over the threshold and are not included.

3.3.2 Evaluation using PCKh

In order to evaluate if an experiment had improved the performance of the network, some type of evaluation metric is needed. Preferably a single number that can be

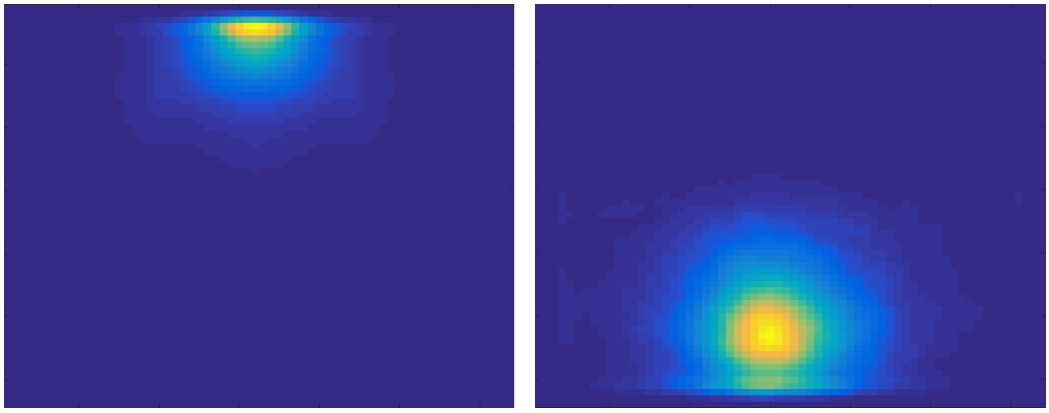
improved during the experimentation. For evaluation on the MPII human pose data set the PCK (probability of correct keypoint) [30] metric is frequently used. PCKh@0.5 was chosen as the metric to improve during training experiments. It measures the percentage of keypoint predictions that is at maximum distance of 50% of the head size from the ground truth. The MPII provides a MATLAB toolkit for calculating the PCK value for a variety of distances.

3.3.3 Evaluating performance on difficult poses

In order to evaluate the performance on different poses, a metric called pose score was created. The intention of the pose score was to measure how much a pose differs from the most likely pose in the dataset. In order to calculate the most likely pose all ground truth images for each keypoint in the dataset were summed up to form distributions of the keypoints:

$$distribution_i = \sum_{j=1}^{N} im_{i,j} \tag{3.1}$$

where $im_{i,j}$ is the j-th ground truth image of the i-th keypoint and N is the number of ground truth images in the training data. Figure 3.19 shows the distribution of the head and right ankle. The color scale is relative where more yellow color means that the keypoints occur in that area more frequently. It can be seen from these distributions that the head and right ankle frequently appear in a rather small part of the image and very rarely appear in large parts of the image.



(a) Head **(b)** Right ankle

Figure 3.19: Distribution of the head and right ankle in the MPII dataset.

All keypoints are concentrated in a similar way as the head and right ankle and Figure 3.20 shows the most likely location of each keypoint. The maximum locations are symmetric because mirroring around the vertical axis is used for augmenting the training data. This means that the most likely pose in the MPII dataset is a person standing up facing the camera with the arms nest to their body.

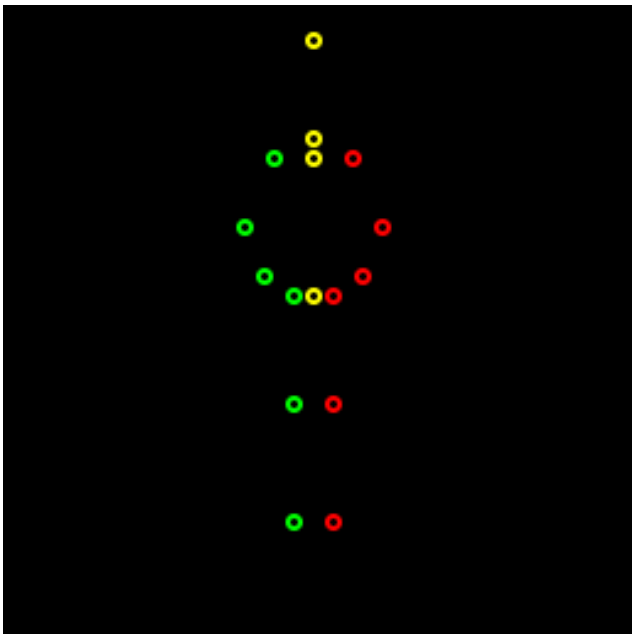


Figure 3.20: Most likely location of each keypoint.

The pose score tries to measure how much a pose deviates from the most likely pose in Figure 3.20. It is calculated by summing up the squared distances between the keypoints in the pose and the keypoints in the most likely pose:

$$posescore_i = \frac{1}{M} \sum_{j=1}^{M} |kp_{i,j} - kpmax_j|^2 \tag{3.2}$$

where M is the number of keypoints in the pose, $kp_{i,j}$ is the location of the j-th keypoint in the i-th pose and $kpmax_j$ is the location of the maximum of the j-th keypoint in Figure 3.20. Figure 3.21 illustrates how the distances are calculated for a pose.

Figure 3.21: The lines are the distances between the most likely location of the keypoints and the location in the current pose. These distances are squared and summed up when calculating the pose score.

4
Results

As can be seen by the sample predictions in Figure 4.1, the network can handle a variety of backgrounds, poses and clothing. Exactly how much changes in background and clothing affects the prediction accuracy is difficult to quantify since the MPII dataset lack information about these parameters for each image. However when observing the predictions, it seems that the network is rather invariant to changes in these conditions. The network generally does few mistakes when estimating a pose that is not occluded and that does not deviate too much from the norm.

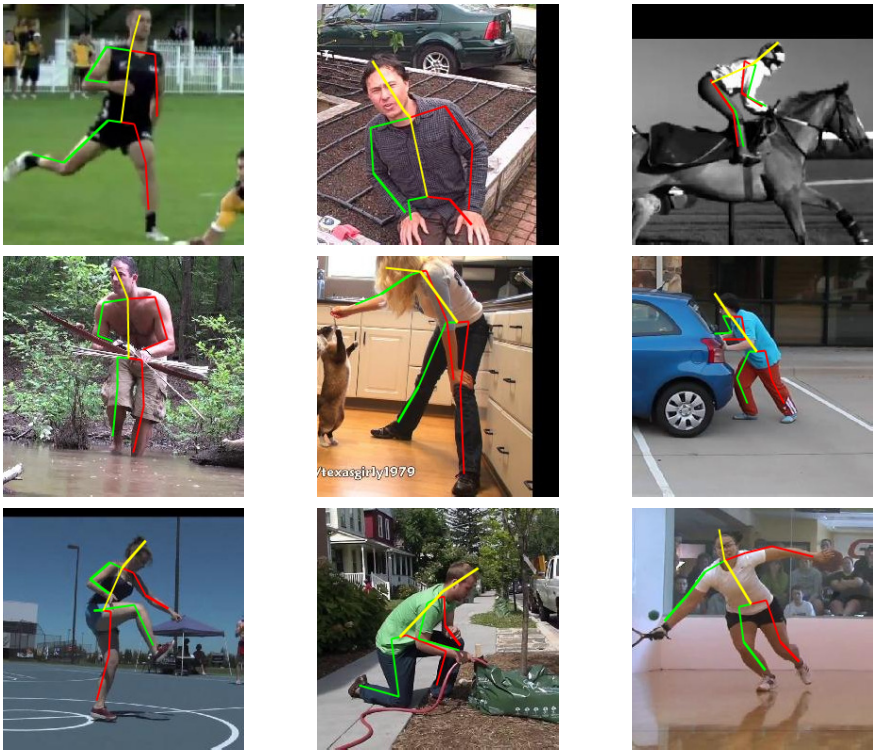


Figure 4.1: Predictions on the MPII test set.

4.1 Comparing preprocessing methods

The preprocessing methods were compared on three different sets of the ground truth: on only visible keypoints, on only occluded keypoints and finally on both

visible and occluded keypoints. As can be seen from Table 4.1 the visible keypoints are easier to predict since M3 has a 75.9% accuracy for visible keypoints vs 71.2% for all keypoints. The inclusion of occluded keypoints in the training data does not improve the performance on visible keypoints since M1 performs slightly better on the visible keypoints. M3 has a significantly higher accuracy than M1 and M2, this indicates that centering the annotated person has a large effect on performance.

Training data	Prediction accuracy
M1 - Uncentered, only visible keypoints	67.9%
M2 - Uncentered with occluded keypoints	66.9 %
M3 - Centered with occluded keypoints	75.9%

Table 4.1: Accuracy on only visible keypoints.

The effect of including occluded keypoints can be seen in Table 4.2. For occluded keypoints the network does not get any visual clues to determine the location of the keypoint and the accuracy is much worse compared to the visible keypoints. When visually examining the predictions of the occluded keypoints it seems that the network has learned how keypoints usually make up a pose and use this to predict the occluded keypoints. M2 performs much better than M1 and this is to be expected since the network trained on M1 lacked occluded keypoints to train on. The fact that M1 still has 26.8% accuracy on occluded keypoints is rather fascinating.

Training data	Prediction accuracy
M1 - Uncentered, only visible keypoints	26.8%
M2 - Uncentered with occluded keypoints	41.4 %
M3 - Centered with occluded keypoints	48.7%

Table 4.2: Accuracy on only occluded keypoints.

Figure 4.2 shows the prediction made by the final network (trained on the M3 training data) on an occluded pose. The left side of the body is not visible but the network does a rather good job (except for the left ankle) estimating where the occluded keypoints are.

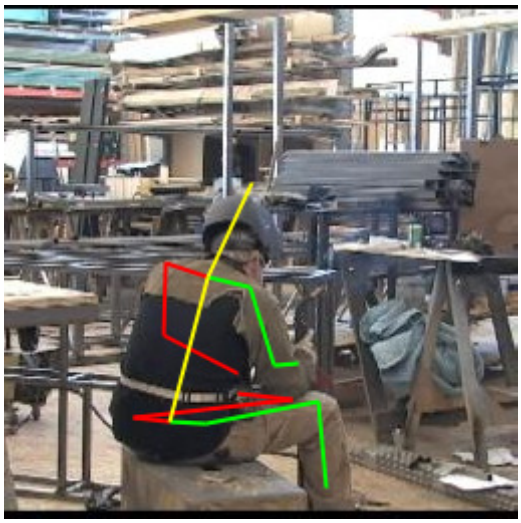


Figure 4.2: An example of prediction of occluded keypoints.

Table 4.3 shows the results when all keypoints are included. The reason for the better performance of M2 compared to M1 is due to M2's better accuracy on occluded keypoints. Since centering the annotated person improves the accuracy of both visible and occluded keypoints it is no surprise that M3 is performing best when all keypoints are included. By gradually improving the preprocessing method, the accuracy was increased by roughly 17%. This improvement in performance was only due to better preprocessing since the network layout and metaparameters were unchanged while testing different preprocessing methods.

Training data	Prediction accuracy
M1 - Uncentered, only visible keypoints	60.8%
M2 - Uncentered with occluded keypoints	65%
M3 - Centered with occluded keypoints	71.2%

Table 4.3: Accuracy on ground truth including all keypoints.

4.2 General performance of M3

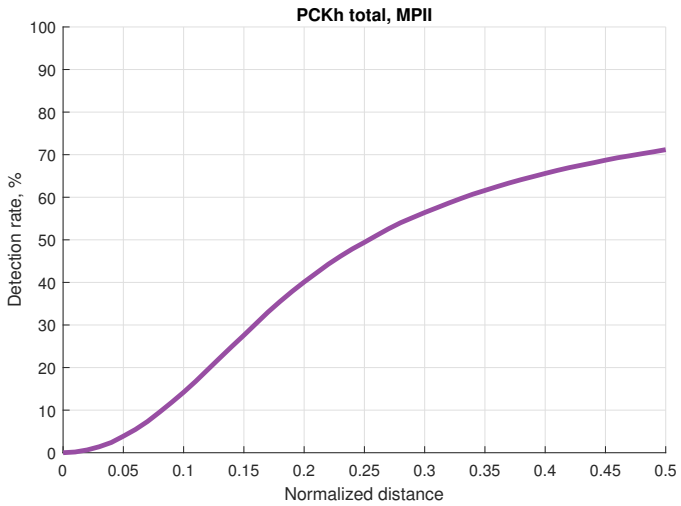


Figure 4.3: Predictions accuracy as a function of the normalized distance for M3.

The third preprocessing method where the annotations of all keypoints are included and the person has been centered showed the best results with a PCKh@0.5 of 71.2%. Figure 4.3 shows how the accuracy varies with the normalized distance. In order to make a comparison, A. Newell et al's. Stacked Hourglass network [9] was trained with just two hourglasses and only mirroring as augmentation. With these settings the Stacked Hourglass network had a prediction accuracy on the validation set of 78%. The reason for the difference in results is not clear, it could depend on differences in implementation of the hourglass modules, metaparameters during training or in the way Torch7 [31] and Caffe [2] implement layers.

Table 4.4 shows the accuracy for each keypoint. The head in this table consists of two keypoints, the head top and the upper neck. The reason the head has the highest accuracy could be that it is always visible in the image and that the head has several features to detect like eyes, nose, mouth and the round shape. The location of the head in the MPII dataset to the upper center part of the image is also fairly consistent. The wrists and ankles on the other side are the most difficult keypoints to predict accurately, probably because they are more frequently occluded, their locations are more spread out in the MPII dataset and they do not have as many features as the head has.

	Head	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Upper Body	Total
Accuracy:	87.6	80.3	70.1	60.7	69.3	65.6	60.8	70.4	71.2

Table 4.4: Accuracy of each keypoint.

4.3 Difficult poses

During visual examination of the predictions of the network it was noted that when the person is facing the camera, with the body fully visible and in a common pose as in Figure 4.4(a), the network had a much higher accuracy than on average. These type of images are common in the MPII dataset, so there is a relatively large amount of training data. When examining images of more uncommon poses like in Figure 4.4 the network made much worse predictions.

(a) Easy poses **(b)** Difficult poses

Figure 4.4: Illustration of easy and difficult poses

In order to examine how the network performs on easy and difficult poses, the pose score metric (Section: 3.3.3) was calculated for each annotated person in the validation set. The accuracy of the predictions on the 20% most difficult poses was compared to the predictions of the 20% most easy poses. The results can be seen in Table 4.5

Type of pose	Prediction accuracy
20% most easy poses	81.2%
All poses	71.2%
20% most difficult poses	59.7%

Table 4.5: Comparison of predictions on easy and difficult poses.

The results in Table 4.5 indicate that the network has learned the distributions of the keypoints and rely rather heavily on the location in the image to predict the keypoints. Figure 4.5 is as an example of a pose that the network has difficulty predicting. Even though the legs and arms are clearly visible the network fails to correctly include them in the pose. It is probably due to the unusual location of the keypoints, the knees are over the hips and the ankles are over the knees and this is a very uncommon setup in the MPII dataset.

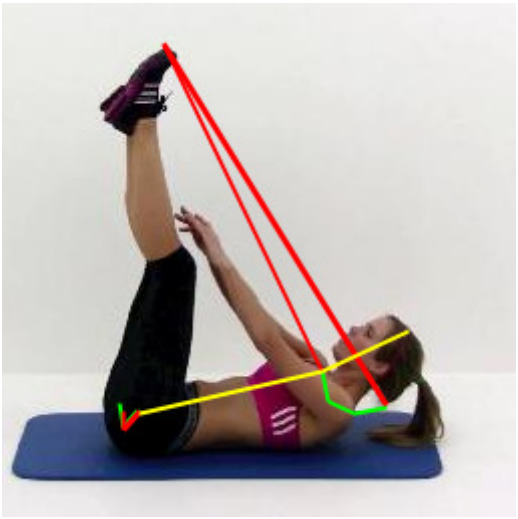


Figure 4.5: Keypoint predictions on a difficult pose.

This shows the importance of varied and well balanced training data. Even though the poses in the MPII data set are taken from a great variety of activities, the distribution is still rather skewed toward a pose that is standing up facing the camera (Section 3.3.3). The network seems to have overfitted to the distribution of poses and when a pose deviates too much from the norm it has difficulty predicting the pose. To reduce the problem of overfitting a more thorough data augmentation could have been used. The only data augmentation used was mirroring around the vertical axis.

5
Discussion

The final version of the network had a lower PCKh@0.5 than the Hour Glass network with the same depth and type of augmentation. Apart from the reasons mentioned in Chapter 4, the size of the minibatch greatly affected performance (especially for lower sizes). Batchnormalization works better on large minibatch sizes but due to memory constraints in Caffe, the memory available for the model was about 1/8 compared to if it was trained in Torch7. This meant there was a trade of between a deep network (more hourglasses, using more memory) or larger minibatch sizes. After experimenting with different depths and minibatch sizes it was found that a depth of two hourglasses and a minibatch size of 6 was the best trade off. The Hour Glass network used 8 hourglasses and a minibatch size of 6, this resulted in a PCKh@0.5 of 90.9% which was state of the art at the time.
To the best of my knowledge, the analysis of the MPII dataset used in this report is a new approach. Previous methods have not been focus on how common poses are in the dataset or how much they deviate from the norm. The previous analyses of the poses in the MPII dataset have been focused on dividing the poses into different body pose and view point clusters. This is done to evaluate how the different state of the art networks are doing on different types of poses and viewpoints. The results are consistent with the findings in this report that poses with a high pose score are more difficult to estimate.
The performance of the network could be improved by a more rigorous data augmentation. One way to do this is to rotate the image by an random angle and zoom by a random factor before being feeding the image to the network during training. This would result in a more spread out distribution of the keypoints in the training data. It would also reduce the problem of the network overfitting to the distribution of poses in the MPII dataset. The problem of overfitting could also be addressed by increasing the training data, preferable with more diversified poses. Another improvement would be to have more computer memory available to train a deeper model with more hourglasses. The PCKh@0.5 of the Hour Glass network goes from 78% to 90.9% when increasing the number of hourglasses from 2 to 8.
To improve the performance further on more rare poses, the training data could be increased with more diversified poses.
The network needs to have the person to annotate in the center of the image to work well and this information is provided in the MPII dataset. In a future application where the location of the person to annotate is unknown a R-CNN (a method that uses a CNN to find region proposals [32], [33]) network could be used to find the person in the image. The proposed region could then be fed to the pose estimation network to find the exact pose.

6
Conclusion

Changes in the background, clothing and body size have a small impact on the network's ability to estimate the pose. Occlusion is affecting the predictions negatively with 48.7% accuracy for occluded keypoints and 75.9% accuracy for visible keypoints.
By changing the preprocessing method it was possible to increase the accuracy from 60.8% to 71.2%, a roughly 17% increase in performance without changing the model or metaparameters of the network. This shows that the choice of preprocessing method can greatly affect the overall performance of a network.
Poses that are uncommon in the MPII dataset is challenging for the network. The accuracy on the easiest 20% of poses is roughly 36% higher than the accuracy on the most difficult 20% of poses. The network seems to have learned the skewed distribution of poses in the MPII dataset and have a bias towards the most likely position of a keypoint. To improve performance on rare poses further data augmentation could be implemented and more training data on a varied set of poses could be collected. The performance would also benefit from a deeper model with more hourglass modules.
For the pose estimation to work well, the person should be centered in the image, not be much occluded and have a pose that does not deviates too much from the norm in the MPII dataset. In applications where the center of the person to annotate is unknown, a R-CNN could be used to find a regional proposal that could be further analyzed by the pose estimation network.

Bibliography

[1] Mykhaylo Andriluka et al. "2D Human Pose Estimation: New Benchmark and State of the Art Analysis," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014. Available: http://human-pose.mpi-inf.mpg.de/ [Apr. 16, 2017] 1, 4, 11

[2] Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor, "Caffe: Convolutional Architecture for Fast Feature Embedding", arXiv preprint arXiv:1408.5093, 2014 2, 22, 32

[3] Hashim Yasin et al. "A Dual-Source Approach for 3D Pose Estimation from a Single Image." *CVPR*, 2016. Available: https://arxiv.org/abs/1509.06720 [Apr. 15, 2017] 3

[4] Jamie Shotton et al. "Efficient Human Pose Estimation from Single Depth Images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 12, dec, 2013. Available: http://ieeexplore.ieee.org/document/6341759/ [Apr. 15, 2017] 3

[5] Katsushi Ikeuchi. *Computer Vision a Reference Guide*. New York: Springer Science+Business Media, 2014, pp. 362-70. 3

[6] P. F. Felzenszwalb and D. P. Huttenlocher. "Pictorial structures for object recognition." International Journal of Computer Vision, 61(1):55–79, 2005. Available: http://www.cs.cornell.edu/~dph/papers/pict-struct-ijcv.pdf [Apr. 15, 2017] 4

[7] D. Ramanan. "Learning to parse images of articulated bodies." In NIPS, 2006. Available: https://papers.nips.cc/paper/2976-learning-to-parse-images-of-articulated-bodies [Apr. 15, 2017] 4

[8] Alexander Toshev, Christian Szegedy. "DeepPose: Human Pose Estimation via Deep Neural Networks," in IEEE Conference on Computer Vision and Pattern Recognition, 2014. Available: https://arxiv.org/abs/1312.4659 [Apr. 15, 2017] 4

[9] Alejandro Newell et al. "Stacked Hourglass Networks for Human Pose Estimation," Computer Vision and Pattern Recognition, 2016. Available: https://arxiv.org/abs/1603.06937 [Apr. 16, 2017] 4, 18, 32

[10] "Convolution arithmetic", Available: https://github.com/vdumoulin/conv_arithmetic 6

[11] Hubel, D. and Wiesel, T. (1968). "Receptive fields and functional architecture of monkey striate cortex." Journal of Physiology (London), 195, 215–243. 6

[12] Zeiler, Matthew D; Fergus, Rob., "Visualizing and Understanding Convolutional Networks", CVPR, 2013. 6, 20

[13] Andrej Karpathy, "CS231n: Convolutional Neural Networks for Visual Recognition", Department of Comupter Science, Stanford, 2017, Available: `http://cs231n.github.io/convolutional-networks/` 6, 9

[14] V. Nair and G. E. Hinton. "Rectified linear units improve restricted boltzmann machines." In Proc. 27th International Conference on Machine Learning, 2010. 7

[15] Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E., "ImageNet Classification with Deep Convolutional Neural Networks", Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097-1105. 7, 20

[16] Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li, "Empirical Evaluation of Rectified Activations in Convolutional Network", CVPR, 2015, Available: `https://arxiv.org/abs/1505.00853` 7

[17] Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J., "Learning representations by back-propagating errors", Nature, 1986, doi: 10.1038/323533a0 7

[18] Bottou L. (2010) "Large-Scale Machine Learning with Stochastic Gradient Descent." In: Lechevallier Y., Saporta G. (eds) Proceedings of COMPSTAT'2010. Physica-Verlag HD, Available: http://leon.bottou.org/publications/pdf/compstat-2010.pdf 8

[19] Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", CoRR, abs/1502.03167, 2015, Available: `http://arxiv.org/abs/1502.03167` [Nov. 6 2017] 8, 19

[20] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D. (2016). "Mastering the game of Go with deep neural networks and tree search." Nature, 529(7587), pp.484-489. 9

[21] Christian Szegedy et al. "Going Deeper with Convolutions", CoRR, abs/1409.4842, 2014, Available: `http://arxiv.org/abs/1409.4842` 9, 19

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, "Deep Residual Learning for Image Recognition", CoRR, abs/1512.03385, 2015, Available: `http://arxiv.org/abs/1512.03385` [Nov. 6 2017] 9, 19

[23] Python Software Foundation. Python Language Reference, version 2.7. Available at `http://www.python.org` 14

[24] Lightning Memory-Mapped Database. *Wikipedia.* Available: `https://en.wikipedia.org/wiki/Lightning_Memory-Mapped_Database` [Apr. 17, 2017] 14

[25] MATLAB Release 2016b, The MathWorks, Inc., Natick, Massachusetts, United States. 14

[26] Evan Shelhamer, Jonathan Long and Trevor Darrell, "Fully Convolutional Networks for Semantic Segmentation",CoRR, abs/1605.06211, 2016, Available: `http://arxiv.org/abs/1605.06211` 20

[27] Hyeonwoo Noh, Seunghoon Hong and Bohyung Han, "Learning Deconvolution Network for Semantic Segmentation", CoRR, abs/1505.04366, 2015, Available: http://arxiv.org/abs/1505.04366 20

[28] Tieleman, T., Hinton, G.: "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning (2012) 22

[29] http://caffe.berkeleyvision.org/tutorial/interfaces.html 23

[30] Y. Yang and D. Ramanan. "Articulated human detection with flexible mixtures of parts." PAMI'13. 25

[31] Collobert, R., Kavukcuoglu, K., Farabet, C.: "Torch7: A matlab-like environment for machine learning." In: BigLearn, NIPS Workshop. (2011) 32

[32] Ross B. Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", CoRR, abs/1311.2524, 2013, Available: http://arxiv.org/abs/1311.2524 35

[33] Kaiming He et al. "Mask R-CNN", CoRR, abs/1703.06870, 2017, Available: http://arxiv.org/abs/1703.06870 35