Facial Landmark Detection and Shape Modeling using Neural Networks

Master's thesis of

Simon Hessner

At the faculty of Computer Science
Institute for Anthropomatics and Robotics

Reviewer:	Prof. Dr.-Ing. Rainer Stiefelhagen
Second reviewer:	Prof. Dr.-Ing. habil. Jürgen Beyerer
Advisor:	Dr.-Ing. Saquib Sarfraz

Duration: 14th January 2019 – 15th August 2019

Computer Vision for Human-Computer Interaction Research Group
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology
Title: Facial Landmark Detection and Shape Modeling using Neural Networks
Author: Simon Hessner

Simon Hessner
uldci@student.kit.edu

Statement of Authorship

I hereby declare that this thesis is my own original work which I created without illegitimate help by others, that I have not used any other sources or resources than the ones indicated and that due acknowledgement is given where reference is made to the work of others.

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 15. August 2019

....

(Simon Hessner)

Abstract

Facial landmarks are distinctive points in human faces that are used for a variety of tasks such as facial expression analysis, lip reading or face recognition. The performance on these tasks depends heavily on the accuracy of the detected facial landmarks. It is challenging to accurately locate facial landmarks even on faces that are partially occluded by glasses, facial hair or other objects. In this work we introduce a new approach to tackle these challenges on unconstrained frontal and semi-frontal face images. The proposed solution is a new deep learning based algorithm that is built on the Stacked Hourglass Network which has proven to be effective for human pose estimation, a task similar to facial landmark detection. The algorithm processes face images by repeatedly down- and upsampling the image and thus analyzes it on multiple scales. The Stacked Hourglass Network is trained using Wing loss and regresses coordinates using a Differentiable Spatial To Numerical Transform. Our algorithm is able to outperform current state-of-the-art solutions on the 300-W and Menpo datasets in terms of the point-to-point normalized error. Additionally, a neural Point Distribution Model is employed as a shape model that refines the predictions made by the Stacked Hourglass Network. By adding the Point Distribution Model, the prediction error on the inner facial landmarks of the challenging test set of 300-W reduces even more. The Point Distribution Model achieves the biggest improvements on the inner landmarks of faces with strong head poses while improving the predictions of landmarks on the outline is more challenging.

Kurzfassung

Facial Landmarks sind markante Punkte in menschlichen Gesichtern, die für die Analyse von Gesichtsausdrücken, Lippenlesen, Gesichtserkennung und eine Vielzahl weiterer Aufgaben verwendet werden. Die Ergebnisqualität bei diesen Aufgaben hängt stark von der Genauigkeit der erfassten Punkte ab. Sie auch auf Gesichtern, die teilweise durch Brillen, Gesichtsbehaarung oder andere Gegenstände verdeckt sind, genau zu lokalisieren, ist schwierig. In dieser Arbeit gehen wir diese Probleme für semi-frontale Gesichter in realen Umgebungen an und stellen einen neuen Ansatz zum Lokalisieren von Facial Landmarks vor. Die vorgeschlagene Lösung ist ein Deep Learning - basierter Algorithmus, der auf dem Stacked Hourglass Network aufbaut. Diese Netzwerkarchitektur hat sich als sehr effektiv für die Bestimmung menschlicher Posen (Human Pose Estimation) in Bildern erwiesen, eine der Erkennung von Facial Landmarks ähnlichen Aufgabe. Bilder von Gesichtern werden von dem Netzwerk durch wiederholtes Herunter- und Hochskalieren auf mehreren Skalen analysiert. Das Stacked Hourglass Network wird mit der Wing loss - Fehlerfunktion trainiert und bestimmt die Koordinaten der Facial Landmarks mithilfe einer Differentiable Spatial To Numerical Transformation. Der in dieser Arbeit vorgestellte Algorithmus ist in der Lage, auf den 300-W- und Menpo-Datensätzen einen geringeren Punkt-zu-Punkt normalisierten Fehler als andere Ansätze auf dem aktuellen Stand der Technik zu erzielen. Zusätzlich wird ein Point Distribution Model als Shape Model eingesetzt, um die Vorhersagen des Stacked Hourglass Networks weiter zu verfeinern. Dadurch erhöht sich vor allem die Erkennungsgenauigkeit für die inneren Facial Landmarks auf schwierigen Bildern von 300-W noch weiter. Das Point Distribution Model verbessert vor allem die Erkennung von Facial Landmarks im Inneren von Gesichtern, die aus großen Winkeln gezeigt werden. Eine Verbesserung der Erkennungsgenauigkeit von Landmarken auf der Gesichtskontur erwies sich hingegen als schwieriger.

Contents

List of Figures

List of Tables

1. Introduction

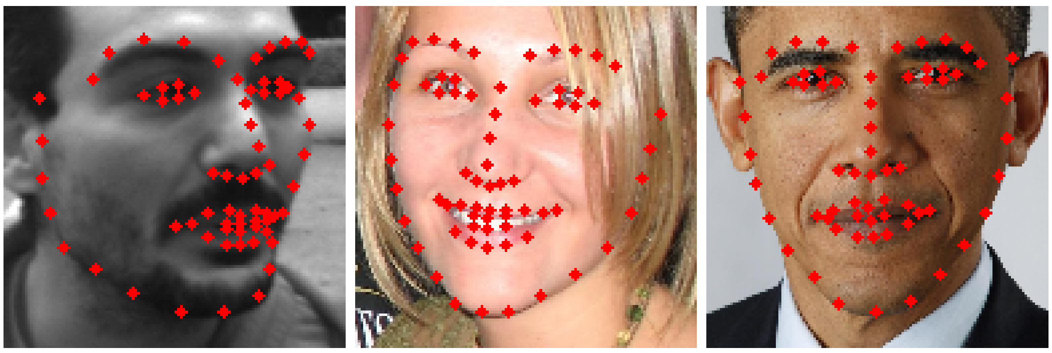


Figure 1.1: Example faces with annotations from the 300-W test set [55, 57].

In order to achieve satisfactory human-computer interaction it is crucial for computers to understand the content of images. Analyzing images of human faces is an active research field with numerous different subtopics. Some focus on face recognition [42, 64, 75] which can be used to unlock a phone or open a door by looking at a camera. Analyzing facial expressions and emotions [23, 70] is another direction that can be useful for medical applications or to detect if a person is stressed or relaxed and muting or enabling notifications over new messages based on this. Furthermore, estimating a person's head pose [46, 52, 58, 59] or gaze direction [61, 62] can be used to understand conversations or to detect whether a car driver is focused or distracted.

Another research direction focuses on detecting facial landmarks which are distinctive points with a semantic meaning that are distributed over a human's face, for example around the eyes, mouth and outline. Figure 1.1 shows three faces with 68 annotated facial landmarks. Facial landmarks serve as input to other tasks such as lip reading [15, 43] to improve speech recognition accuracy, facial expression analysis, emotion detection or face recognition [44, 64].

The problem of facial landmark detection is often called facial alignment or facial landmark localization. It can be formally described as finding the (x, y) coordinates of $l \in \mathbb{N}^+$ semantic points on an image of a human face. Hence, the output of a facial landmark detection algorithm is a vector $p = [x_1, y_1, ..., x_l, y_l] \in \mathbb{R}^{2l}$. There is also 3D facial landmark detection [49, 80], but this work focuses on 2D coordinates.

Various challenges arise when designing a facial landmark detection algorithm. For example, images can be of varying quality due to camera resolution, illumination or distance between the camera and the face. Moreover, faces can be occluded by hats, glasses or facial hair. Furthermore, faces appear differently depending on the facial expression. Finally, the face can be shown in a non-frontal position and there is natural variance due to different ethnicity, gender, age and individual appearance. We focus on semi-frontal faces which include faces viewed from different angles as long as no landmark is occluded by the face itself. Hence, we exclude profile faces from our work. The images are allowed to show faces in uncontrolled settings such as indoor or outdoor, under varying light conditions and with arbitrary backgrounds.

In the early years of computer vision most approaches to analyze images made extensive use of hand-crafted features such as the scale invariant feature transform (SIFT) [40], local binary patterns (LBP) [1] or histogram of oriented gradients (HOG) [20]. These features were used to train classic machine learning algorithms like support vector machines (SVM) [7] or random forests [8]. In 2012, Krizhevsky et al. proposed AlexNet [35], a convolutional neural network that was able to beat the state-of-the-art in image classification by a large margin. Inspired by this success, many researchers in the computer vision domain started to use neural networks and were able to improve their results. The advantage of using (convolutional) neural networks is that no hand-crafted features are needed. Moreover, neural networks are trained to extract hierarchies of complex features by optimizing a loss function using back-propagation. Those features are optimized for the problem at hand, which is the main reason why they outperform hand-crafted features.

In this work we also use neural networks and introduce an approached based on a Stacked Hourglass Network [47] that down- and upsamples the image multiple times to get an understanding of the scene. The Stacked Hourglass architecture is a fully-convolutional neural network architecture that is able to extract complex features from face images. Based on these features heatmaps for each landmark are generated. The heatmaps indicate the likelihood of a landmark at each position in the image. We convert the heatmaps into numerical coordinates using a Differentiable Spatial To Numerical Transform (DSNT) [48]. The network is trained using Wing loss [24] which is insensitive to outliers and converges to a better model than other commonly used loss functions. The predictions from the Stacked Hourglass Network are then further improved by a Point Distribution Model (PDM). The PDM is a neural network based unsupervised generative shape model.

The main contributions of this work are

- The Stacked Hourglass Network [47] is combined with the Differentiable Spatial to Numerical Transform (DSNT) [48] and trained using Wing loss [24].
- Effects of different Hourglass design choices are explored in detail.
- A Point Distribution Model (PDM) is implemented and used as a shape model to fix wrong predictions from the Stacked Hourglass Network. We investigate in which cases the PDM is able to improve predictions the most.
- The whole system is trained and evaluated on the 300-W [55, 56, 57] dataset. To assess the performance on an unrelated dataset, we perform a cross-dataset evaluation on the Menpo [82] dataset.

This thesis is organized as follows: Related work on the field of facial landmark detection is presented in Chapter 2. Both the Stacked Hourglass Network and the PDM are introduced in Chapter 3. Various design choices are evaluated in Chapter 4 for both the Stacked Hourglass Network and the PDM. To conclude, Chapter 5 reviews what has been achieved in this work and outlines which further ideas could be explored in the future.

2. Related Work

In this chapter an overview over relevant publications in the field of facial landmark detection is presented. We first start with classic approaches that allowed for first successes in facial landmark detection in Section 2.1. Similar to other computer vision problems, the rise of deep learning models has enabled more powerful algorithms in the facial landmark domain as well. A selection of deep learning based solutions is presented in Section 2.2. Finally, the baselines we use to compare our models to are presented in Section 2.3.

2.1 Classic approaches for facial landmark detection

Most algorithms for facial landmark detection can be classified as either model-based or regression-based [79]. They differ in how the coordinate prediction is done.

Model-based approaches learn constraints on the arrangement of landmarks relatively to each other. This is useful to locate landmarks in non-rigid objects since the locations of landmarks are often constrained by other landmarks. We present three important model-based algorithms: Active Shape Model (ASM) in Section 2.1.1, Active Appearance Model (AAM) in Section 2.1.2 and Constrained Local Model (CLM) in Section 2.1.3.

Regression-based approaches do not have an implicit shape model. They instead operate directly on the image and regress the coordinates of the landmarks. Cascaded regression approaches run multiple regression models consecutively, each one refining the predictions of its predecessor. An overview over regression-based algorithms as well as cascaded regression solutions is presented in Section 2.1.4.

Some of the most important classic approaches are presented in this section. However, there are many more algorithms for facial landmark detection. Wang et al. [71] give a comprehensive survey on facial landmark detection algorithms that were published before 2014.

2.1.1 Active Shape Models (ASMs)

In 1995, Cootes et al. [17] published ASMs, an algorithm that puts constraints on the the locations of individual landmarks. Possible shape variations are learned from the training set and stored in a Point Distribution Model (PDM). Before learning the variations, all samples in the training set are aligned by rotating, scaling and translating them so that the difference between samples is minimized. The idea is to have all landmarks in all samples approximately at the same position so that the actual shape variations can be estimated without the influence of pose variations. The PDM stores the average locations of each landmark and the main modes of variation, including correlations of shape parameters. The PDM finds a basis of the shape parameters where the shape parameters are uncorrelated, allowing to change each of the parameters and still generating a valid shape. This is done using Principal Component Analysis (PCA). Once learned, the model can be used to generate new shapes that were never seen in training by varying the uncorrelated shape parameters.

The ASM uses the PDM to find the landmarks in an unseen test image. Locating landmarks involves finding shape parameters that correspond to a shape as close as possible to the test image. The original algorithm assumes that landmarks lie on strong edges. An ASM does not model texture but only looks for edges in arbitrary textures (around the current estimate). The initial location estimates are set to be the mean of the training set. Then, iteratively a local region around each landmark is analyzed by looking on the normal through each landmark and finding the strongest nearby edge. This is then used to update the pose and shape parameters to best fit the new landmark locations. Important to note is that the estimates are not updated directly. Rather, they are updated by finding shape parameters that map to locations that are as close as possible to the new estimate. This is important as otherwise the shape constraints could be violated. The whole process is repeated until convergence.

The algorithm can be improved in terms of speed and accuracy by following a coarse-to-fine approach. That means that the algorithm is first executed on a blurred image. The obtained locations are then used as the initial estimate when running the algorithm on a finer resolution. The process is repeated until the finest resolution is reached.

Although the ASM paper used resistors and human hands as example objects, the algorithm was also an important milestone in the history of facial landmark detections since it allowed to model non-rigid shapes [18].

2.1.2 Active Appearance Models (AAMs)

Building on the success of ASMs, Cootes et al. [16] introduce AAMs. The main difference to ASMs is that texture variation around the landmarks is learned in addition to the shape variation. The AAM can be used to synthesize a whole image using the shape and appearance parameters, hence it is a generative model. Landmarks in unseen images are located by finding shape and appearance parameters that minimize the difference between the generated image and the test image.

The shape model is built in the same way as in ASMs. The appearance model learns the appearance variations in shape-free images. A shape-free image is obtained by aligning the original image with the mean image. To generate a synthetic image, first the appearance parameters are used to generate an image that is shape-free. Then the shape parameters are used to warp it to the desired shape using triangulation-based interpolation.

To find the optimal shape and appearance parameters, the error between the original image and the synthesized image is minimized. In comparison, the ASM is not minimizing distances between images but seeks to improve the current estimate by searching around a small local area. However, the AAM is a holistic model.

Similar to ASMs, the authors construct a multi-resolution model on a Gaussian image pyramid. For each level on the pyramid there is a separate model. Working from coarse to fine, the algorithm is both faster and more accurate than using only the finest resolution.

The algorithm is applied to the problem of facial landmark detection in the original AAM paper. According to [18], ASMs are more accurate and faster than AAMs, but the latter are able to better match the texture. According to [21] AAMs suffer from lightning changes and bias towards the mean face.

2.1.3 Constrained local models (CLMs)

Another class of shape models are CLMs, proposed by Cristinacce et al. [19]. CLMs are similar to AAMs as both model shape and appearance variations of deformable objects like faces. CLMs model the appearance around each landmark in local templates, whereas AAMs model the whole object. The templates are normalized to have zero mean and unit variance.

When using the CLM to locate landmarks, the shape and appearance models are used to generate estimated locations and texture templates around each landmark. Then for each landmark the correlation between the template and the actual image at this position is computed. The iterative fitting process maximizes these correlations while respecting shape constraints. CLMs achieve a lower localization error than AAMs when predicting facial landmarks [19].

2.1.4 Regression and cascaded regression

An alternative to shape-based models like ASMs, AAMs or CLMs are regression-based approaches. According to Zadeh et al. [79], cascaded regression has mostly replaced CLMs as they aren't able to model complex appearance variance that is caused by different facial expressions, facial hair or makeup.

An example for non-cascaded regression is the work from Dantone et al. [21]. They use random forests [8] conditioned on the head pose to regress the locations of facial landmarks. There is one random forest for each of five head pose ranges and during inference the predictions of each random forest is weighted by the probability of this specific head pose. Each random forest is only trained on one specific head pose. This allows the random forest to focus on the appearance specific for that head pose. In addition to the actual regressors for the facial landmarks, a regressor for the head pose probability is trained.

Zhu et al. [86] use the cascaded regression pattern to search the so-called shape space that contains all shapes seen during training. In each stage the best fitting shape is picked and then refined in the next stage. This is done by restricting the search space to the most likely candidates. In the last stage the final predictions are computed. Their approach does not require an initial estimate for the location of the landmarks. Moreover, by starting from a coarse scale (considering all possible shapes with all possible poses) and refining the shape space on each cascade level, the algorithm is less prone to local minima. They use different features on each level. The first level makes use of fast but less accurate features while the last stage uses slower and more accurate features. This ensures allows them to run the algorithm in real-time. We will compare our results to this approach and refer to it as **CFSS** (Coarse to Fine Shape Searching) in Chapter 4.

Cao et al. [13] train a cascaded regression model that regresses shapes which are guaranteed to be a linear combination of shapes seen during training. This ensures that the algorithm predicts valid shapes without the need to learn a shape model. The regressors in the cascade are trained to explicitly minimize the prediction error on the training data and all landmarks are jointly predicted. The first regressors in the cascade handle large variations (e.g. head pose) while the final regressors are responsible for fine-grained predictions like closed/open eyes/mouth. The regressors are built on simple features that compare two pixels. This makes the algorithm both fast and invariant against variations caused by illumination.

2.2 Deep learning based facial landmark detection

Early facial landmark detection algorithms, such as the ones presented in the previous section, were implemented using manually engineered features and classical machine learning algorithms like random forests [8] or Support Vector Machines (SVMs) [7]. Since the rise of artificial neural networks (commonly just referred to as *neural networks*) most work in the field of facial landmark detection relies on Convolutional Neural Networks (CNNs), a special kind of neural networks.

Deep neural networks are also extensively used in this work. In the rest of this chapter we first give a short introduction into neural networks in Section 2.2.1 and CNNs in Section 2.2.2 and then present some approaches using neural networks in Section 2.2.3. More deep learning based approaches have been summarized by Bodini et al. [6].

2.2.1 Neural Networks

Neural networks are a powerful class of machine learning algorithms. For some applications like image processing, they do not require manually crafted features but are able to learn suitable features themselves (representation learning). When using classic algorithms, features have to be engineered manually. This enables more accurate models since humans do not have to come up with optimal features for the task at hand. It also allows the use of complex hierarchical features because neural networks are typically organized in multiple layers that compute features on the features from the previous layer [26].

Neural networks can be applied to a variety of different tasks in speech processing [27, 69], natural language processing [25, 77], image processing [10, 11, 29, 35, 44, 46, 47, 63, 76, 79] and more [39]. The main drivers for the success of neural networks are large amounts of available data and enough computational power to train these networks. Since these conditions were fulfilled, it has been possible to push the state of the art results by a large margin for many applications.

A neural network can be seen as a complex non-linear function that maps the input (e.g. an image or audio signal) to a set of output values (e.g. class probabilities or coordinates). The neural network itself is comprised of a sequence of linear transformations followed by non-linear functions. A non-linear activation function is crucial because otherwise, multiple linear functions in a sequence collapse to one single linear function which drastically reduces the variety of functions the network can approximate.

Such a network usually consists of multiple layers, each containing a number of neurons. The neurons in each layer are connected to each neuron of the previous layer. There is a weight on each connection. The first layer is called the input layer and as such has

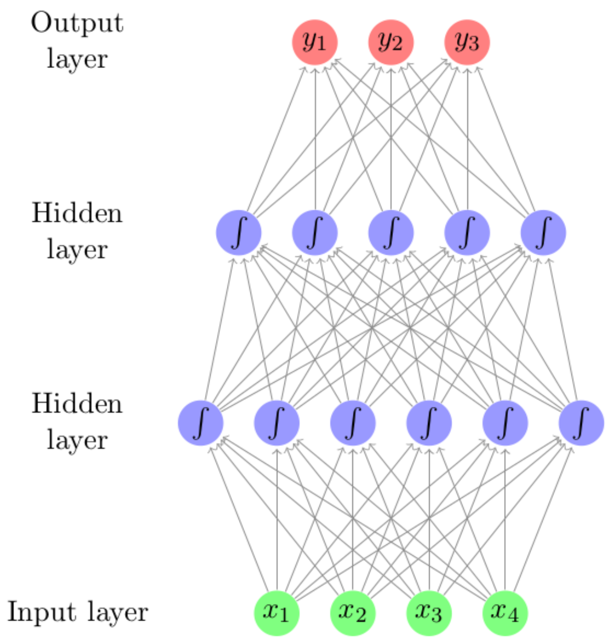


Figure 2.1: Visualization of a feed-forward neural network with two hidden layers from Goldberg et al. [25]

no input connections. Analogously the last layer is the output layer and has no output connections. All layers between the input and output layer are hidden layers and they are fully connected with both the previous layer and the succeeding layer. An example network with two hidden layers is depicted in Figure 2.1. As the data is fed from the input layer to the following layers and never backwards, this kind of network is called feed-forward network. The objective of the training process is to minimize a loss function by learning the network weights. This is possible through back-propagation [54].

The network can be applied to a sample by feeding the sample into the first layer. The values in the next layer are computed by multiplying the values from the previous layer with the connection weight and then computing the sum of these products. The sum is then transformed by the non-linear activation function and the result is the output of the current layer. This process repeats until the last layer is reached.

A more detailed introduction to neural networks can be found in many publications, e.g. the book by Goodfellow et al. [26] or the introduction to natural language processing by Goldberg et al. [25]. The next subsection is focused on a special class of neural networks that is mostly used for image processing.

2.2.2 Convolutional Neural Networks

In standard feed-forward neural networks the input data is an one-dimensional vector and all layers are fully-connected, which means that each neuron is connected to each neuron from the previous layer. However, this is not ideal when the input is an image, because images have a width, height and color channels, so the input is a three-dimensional tensor. Furthermore, when analyzing images, a desired property is to have spatial invariance. This means that an object should be recognized regardless of its position in an image. When using fully-connected layers this is not possible because there is a connection weight for each of the possible pixels. To have spatial invariance in this case, the network must be trained on images where the objects appear in all possible locations. This is an unrealistic requirement because training data is still limited and training is time-consuming.

Convolutional Neural Networks (CNNs) [37] solve these issues by organizing the layers in three dimensions (width, height, channels) and replacing the fully-connected layers by convolutional layers. CNNs are inspired by Time-Delay Neural Networks (TDNNs) [69]. A convolutional layer consists of a set of filter kernels with a relatively small size (e.g. 3×3 px) that operate on all channels of the input layer. The kernel is convolved with the input layer and creates a filter response for each location. The output of the convolutional layer consists of one channel for each of the filters and each channel contains of the filter responses. A channel is often referred to as feature map because the filters that produce it compute features on the input channels. It is important to note that the weights of the convolutions are the independent of the location where the convolution is applied. In other words, the filter kernels share the weight among all locations (parameter sharing). This has the advantage that for each kernel only a relatively small number of weights has to be learned in contrast to a fully-connected network. It also makes the filters invariant to translations. The output of the convolutions is transformed by a non-linear activation function, similar to feed-forward networks.

The size of a filter is also called its receptive field. It describes the area around a pixel that influences the value of the convolutional layer's output. Filters with a small receptive field can only compute local features and are not able to analyze an image at a larger scale. To overcome this issue, many modern CNN architectures include subsampling layers that reduce the spatial extent (width and height) of a layer. The subsampling process is often called pooling. A commonly used pooling operator is the maximum function which returns the largest value in a $n \times n$ px region. However, other functions such as the average function could also be used. By using the maximum, the network is also able to get rid of noise and focus on the most prominent features. A frequently used pooling size is 2×2 px. It transforms an input feature map of size $h \times w \times c$ into an output feature map of size $\frac{h}{2} \times \frac{w}{2} \times c$. The pooling operation does not reduce the number of channels but only the width and height. When a convolution is applied to the pooled output of a convolution, the receptive field grows since more input neurons have an influence on the value at each pixel. Therefore, the features computed in early layers detect local patterns like corners and edges and get more complex after each subsequent layer.

Figure 2.2 shows a simple CNN with two convolutions and two subsampling steps. The fully connected layers in the end convert the features computed by the convolutional part of the network into an output vector. The output values can represent class probabilities or regressed coordinates. A CNN without fully-connected layers is called a Fully Convolutional Neural Network (FCNN) and its output is a multi-dimensional tensor. Part of this work is realized using a FCNN.

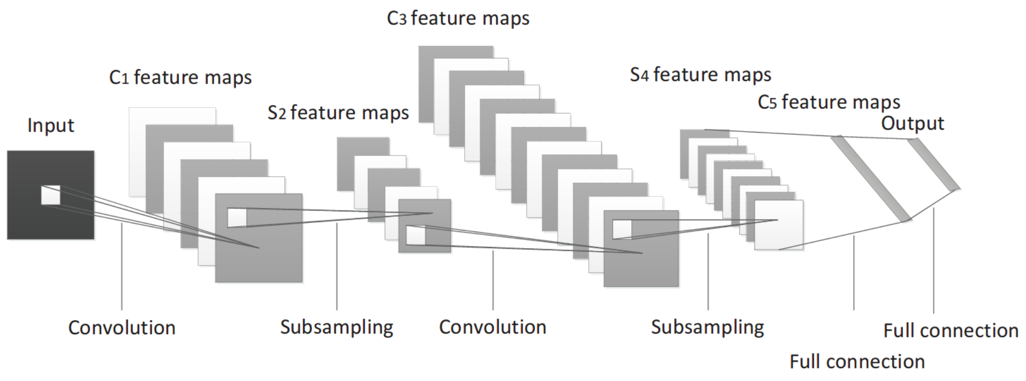


Figure 2.2: Visualization of a Convolutional Neural Network from Liu et al. [39]

2.2.3 State-of-the-art approaches

Since the rise of deep convolutional neural networks the performance of facial landmark detection systems has massively improved. Deep learning based approaches mostly outperform classic solutions. In this section, a brief overview over publications with state-of-the-art results in the field of facial landmark detection is given. Most of the deep learning based algorithms shown here are regression based approaches, except the one by Zadeh et al. [79] which combines a CLM with a neural network as local detector and by Merget et al. [44] which uses an additional shape model on top of the regressed coordinates. The paper titles are written in bold.

Deep Convolutional Network Cascade for Facial Point Detection

Sun et al. [63] present a multi-stage neural network that locates five facial landmarks. This is different to most other presented publications and our work that locate 68 landmarks. The system is one of the first deep learning based facial landmark detection solutions. It regresses the coordinates directly rather than regressing a heatmap. A three-stage cascade of convolutional networks is trained. The first stage produces initial predictions and the next two stages only operate on a window around the current estimate and refine the prediction.

Each stage consists of multiple networks whose predictions are averaged before running the next stage. On the first stage there are three networks. One predicts all five landmarks and operates on the whole image and is already able to produce decent initial predictions. The other network in the first stage analyzes only the top and middle part of the image and locates the eyes and nose. The last one operates only on the bottom and middle part of the image and locates the nose and mouth. After fusing the predictions the next stage employs two neural networks for each landmark that operate on patches with different sizes. As opposed to the first stage the other stages only predict relative shift vectors to the last predictions. These predictions are averaged again before running the next stage. There are two patches with different sizes for each landmark because sometimes predicting landmarks based on small regions is ambiguous. The second and third stages are only allowed to shift the prediction by a small value to avoid drifting. The final prediction is the sum of the first predicted location and the shifting vectors of the last two stages.

Convolutional Experts Constrained Local Model for Facial Landmark Detection

Zadeh et al. [79] state that standard CLMs [19] are not able to model all possible variations of regions around facial landmarks that can be caused by facial expressions, facial hair or makeup. To overcome this problem, they employ a Convolutional Experts Network (CEN) as a local detector for each landmark. Based on heatmaps produced by the CEN a CLM produces the final coordinates. The CLM has a global view while the CEN operates only locally and does not take the location of other landmarks into consideration. Each CEN is optimized to detect one specific landmark in multiple possible variations such as head pose, skin color or facial hair.

The CE-CLM (Convolutional Expert Constrained Local Model) algorithm is model-based and works iteratively. The CENs analyze a $n \times n$ region around the current estimate of a landmark and return a heatmap corresponding to the probability in that region. The region shrinks in each iteration as the model gets more confident. The individual experts vote for a location probability and these votes are combined with non-negative weights to obtain the final prediction of the CEN.

The CLM penalizes unlikely landmark arrangements and thus causes the model to predict more likely locations in the next iteration. The CLM is based on a PCA shape model that modifies the 3D mean face shape using principal component shape parameters. The so obtained 3D face shape is transformed by a 3D affine transformation (rotation, scale, translation) and then projected to 2D. The new estimate is the one obtained by finding shape and affine parameters that make the difference to the initial predictions small. Then the process repeats and the CEN crops the region around it. The overall objective is to find locations with a high probability for each landmark while at the same time having a low regularization error, which effectively means that the landmarks arrange in a shape that is likely to be a face. During minimization, the optimization process finds affine parameters and shape parameters that represent a face that can be constructed by manipulating the mean face using the shape parameters and afterwards applying the transformation and projection to it. By doing this it is possible to model both different face appearances like facial expressions (shape parameters) and different head poses (affine parameters).

The algorithm can be compared to the classic CLM by Cristinacce et al. [19]. They share the same PDM (based on the definition by Cootes et al. [17]) but the texture modeling is different. Zadeh et al. [79] use a neural network (CEN) to produce a heatmap for each landmark around the current estimate and to move the estimate to a more likely position. However, Cristianacce et al. [19] have a local appearance model that creates templates which are then correlated with the input image to find more accurate estimates. The templates are created using a linear PCA-based appearance model while the CEN is a non-linear neural network. As such it is able to model more complex variations. We refer to the approach by Zadeh et al. [79] as **CE-CLM** in Chapter 4.

Robust Facial Landmark Detection via a Fully-Convolutional Local-Global Context Network

Merget et al. [44] have proposed a fully-convolutional local-global context network that solves an issue of ordinary fully-convolutional neural networks, namely that they are not good at aggregating global context because of their local receptive field. The local-global network introduces global context into the fully-convolutional network rather than using cascades of neural networks or fitting a statistical model. This is done by an implicit kernel convolution that blurs the output of a local-context subnet which is then refined by a global-context subnet using dilated convolutions [78]. The implicit kernel convolution makes the gradients less steep and local minima more shallow so that the gradient flow is improved. It furthermore allows to use dilated convolutions which have a bigger receptive field compared to regular convolutions. The receptive field of dilated convolutions grows exponentially with the network depth while the receptive field of regular convolutions grows only linearly. Hence, dilated convolutions allow for a receptive field of the whole image within only a few layers without losing resolution and thus make pooling unnecessary. This means that global context can be analyzed without the need of down- and upsampling again. Without the implicit kernel convolution dilated convolutions can lead to sampling artifacts, but the implicit kernel convolution acts like a low-pass filter and thus undersampling is avoided [44].

The local-context subnet predicts initial landmark coordinates which are later refined by the global-context subnet. It consists of 15 zero-padded convolutions followed by a 1×1 convolution which produce 68 heatmaps, one for each landmark. These heatmaps are convolved with the implicit kernel and then processed by the global-context network. The implicit kernel is defined for only one channel and applied in the same fashion to all channels. It smooths the heatmaps from the local-context subnet and thus allows for the use of dilated convolutions in the following global-context subnet. The implicit kernel is a mixture of Gaussians and it has half the size of the whole input image.

The global-context subnet consists of 7 zero-padded dilated convolutions with dilation factor 4. The output of this subnet is a heatmap from each landmark. The algorithm is not restricted to tightly cropped images, so multiple faces can occur in one image. Therefore it is not possible to just take the argmax of the heatmaps. To extract the coordinates for each face, a PCA-based PDM is used. As a consequence the algorithm does not require a face detection step and is able to handle multiple faces at different resolutions in one image.

We refer to the algorithm by Merget et al. [44] as **FC-LGCN**. A detail that is not mentioned in their paper but shown in their figures is that they apply an intermediate loss after the local-context network [1]. Similarly to the Stacked Hourglass Network for human pose estimation [47] two branches follow the local-context network. One predicts the intermediate landmark locations that are used for the intermediate loss. The other branch contains the implicit kernel convolution. Both branches are stacked afterwards again and then fed into the global-context network.

Stacked Hourglass Network for Robust Facial Landmark Localization

Yang et al. [76] follow an approach similar to ours. Their basic element is a Stacked Hourglass Network [47] that is used to generate heatmaps for each landmark. However, in contrast to this work they first apply a supervised face transformation [14] in order to get rid of the translation, scale and rotation of the input faces. The normalized face is then used as input to the Stacked Hourglass Network. The advantage of normalizing the faces

[1] As the paper does not mention the second branch but it is shown in the architecture visualization, we asked Daniel Merget via e-mail and he explained the meaning of the two branches.

before processing them in the Stacked Hourglass Network is that the Stacked Hourglass Network needs a lower capacity since it has do deal with less variation.

The supervised face transformation first detects faces by running a region proposal network (RPN) which also detects five facial landmarks named 5L (it remains unclear in the paper which landmarks are detected). The candidates are then warped to a mean face shape and only the candidates that show valid faces after warping are further processed. In the next step they use a third-party pre-trained facial landmark detector [4] to extract 19 facial landmarks. These are then used to remove the effects of rigid transformations. The resulting face is cropped from the original images so that it is centered and normalized. The Stacked Hourglass Network is only trained and evaluated on images that were pre-processed with this pipeline. They use the heatmaps produced by the Stacked Hourglass Network to decide which landmarks are occluded based on a confidence threshold.

The algorithm by Yang et al. [76] is the closest to the work presented in this master's thesis since both use the hourglass structure to produce heatmaps. However, there are fundamental differences. First of all, this work does not require the face to be normalized first. To deal with variations in the dataset we make extensive use of data augmentation instead and use a model with high capacity. Next, to train our model, we do not use the standard L2 loss but a new loss function named Wing loss [24]. Their approach also differs from ours in what training objective is used. They generate ground-truth heatmaps while our algorithm makes use of the Differentiable Spatial to Numerical Transform (DSNT) [48] and this does not require artificially generated heatmaps. Instead, we can directly optimize the final numerical coordinates. Details on our approach are presented in Chapter 3.

Their model is trained on the Menpo dataset [22, 82] and evaluated on different datasets (COFW [12], iBUG [56], indoor-outdoor 300-W [55, 56, 57], Menpo) using the euclidean point-to-point normalized distance. The normalization is done by the bounding box diagonal (left-top and bottom-right point of shape bounding box). For completeness they also report the Inter-Ocular Distance (IOD) normalized error. They only compare themselves to baselines implemented by themselves. We compare our system to the algorithm by Yang et al.[76] on the iBUG [56] dataset because they don't provide numbers for the other datasets that we used in our evaluation. We refer to their algorithm as **YANG-HG** in Chapter 4.

Mnemonic Descent Method: A recurrent process applied for end-to-end face alignment

Trigeorgis et al. [66] apply a recurrent neural network (RNN) combined with a CNN as feature extractor to the problem of facial landmark detection. This approach is different from many deep learning based facial landmark detection systems as it does not only use convolutions but also recurrent layers that are mostly known for speech and audio processing [27, 30]. The approach follows the cascaded regression pattern where predictions are refined iteratively. In contrast to classic cascaded regression algorithms, the regressors can be trained jointly and end to end. The convolutional part of the network extracts features which are then analyzed by the recurrent part that returns the final predictions. By using a RNN the network can use the whole history to base its prediction on.

The algorithm starts by assuming the landmarks belong to the mean face shape. Then, a 30×30 px region around each landmark is extracted and processed by a two-layer CNN to get an abstract representation of the region. Once obtained for all of the landmarks, these representations are fed into the RNN which outputs a direction that points to a new estimate for each landmark. This is repeated three times with the newest estimate and the final estimate is used.

The big advantage of this system compared to classic cascaded regression architectures is that no hand-crafted features are required. Instead, by training the CNN and RNN end-to-end, optimal features for the task at hand are learned via back-propagation.

How far are we from solving the 2D & 3D Face Alignment problem?

Bulat et al. [10] follow a similar approach as Yang et al. [76] by using a Stacked Hourglass Network [47] to produce heatmaps for each landmark. However, they replace the residual module [29] by a hierarchical, parallel and multi-scale block [9] that does not use 1×1 convolutions. Their Face Alignment Network (FAN) consists of four stacked hourglasses. The model is evaluated on about 220,000 face images. The differences to this work include that our work returns numerical 2D coordinates while they regress heatmaps and that they replaced the residual module by a different block.

Super-FAN: Integrated facial landmark localization and super-resolution of real-world low resolution faces in arbitrary poses with GANs

Bulat and Tzimiropoulos [11] use Generative Adversial Networks (GAN) to simultaneously improve face resolution and detect facial landmarks. A sub-network for facial landmark heatmap regression is used within the GAN. The generator converts a low-quality and low-resolution face image into a higher-resolution image and the discriminator decides if it was an original or generated image (only during training). In parallel to the discriminator, the Face Alignment Network (FAN) regresses heatmaps for all facial landmarks. The FAN is the same as in [10] that was shown to perform poorly on low-resolution images. However, when combined with the super-resolution GAN, the accuracy is improved for both the super-resolution and landmark detection tasks.

The heatmap loss compares the output of a FAN operating on the high-resolution image (ground truth) and of the FAN operating on the image produced by the generator. The goal is to have a FAN that generates the same heatmaps on the original and the generated image. Note that the two FANs do not share weights and the high-resolution FAN is pre-trained and not updated during the GAN training.

2.3 Evaluation baselines

Out of the previously described approaches we use **CFSS** from Zhu et al. [86], **YANG-HG** from Yang et al. [76], **CE-CLM** from Zadeh et al. [79] and **FC-LGCN** from Merget et al. [44] to compare our own system to. The other presented systems are not used in the evaluation since they were evaluated on different datasets. We also compare our results to other works:

- **CLNF**: Baltrusaitis et al. [3] propose Constrained Local Neural Fields for robust facial landmark detection. It consists of a probabilistic patch expert which assigns per-landmark alignment probabilities to the input image. CLNF is a variation of classic CLMs.

- **SDM**: Supervised Descent Method (SDM) by Xiong et al. [74] uses cascaded regression to minimize a Non-Linear Least Squares function, e.g. the mapping from a face image to shape coordinates.
- **CFAN**: Zhang et al. [83] use cascaded regression in their Coarse-to-Fine Auto-Encoder Network (CFAN). In each cascade stage the image resolution is increased and a finer prediction is made.
- **DRMF**: Asthana et al. [2] present Robust Discriminative Response Map Fitting (DRMF) with CLMs. For each landmark heatmaps are regressed and from these heatmaps the landmark coordinates are regressed.
- **PO-CR**: Georgios Tzimiropoulos [67] proposes a cascaded regression approach that works especially well on the iBUG dataset [56].
- **TCDCN**: Zhang et al. [85] use multi-task learning to predict facial landmark coordinates, head pose and facial attributes like gender, smile or wearing glasses. Learning multiple tasks helps achieving better performance on the landmark detection task.
- **3DDFA**: Zhu et al. [87] focus on profile faces. They present a 3D dense face alignment (3DDFA) which fits a dense 3D model to the face image which allows them to not only locate the visible but also the hidden landmarks (due to pose).
- **DU-Net**: Tang et al. [65] build on the Stacked Hourglass Network [47] and design a Quantized Densely Connected U-Net. This architecture differs from the original Hourglass architecture because feature maps of the same size are not only connected within each Hourglass but also with all following Hourglasses in the stack. To improve the model size and inference speed, they use quantized weights and gradients.
- **LAB**: Wu et al. [73] choose a novel approach and detect the face boundary first. These boundaries can be interpolated from existing datasets and are independent of the exact number of landmarks, thus allowing to use datasets with different annotation schemes together. Based on the detected boundaries the actual facial landmarks are regressed. Their architecture builds on the Stacked Hourglass Network [47].
- **AWL**: In April 2019, Wang et al. [72] have proposed Adaptive Wing loss which is an extension to the Wing loss function [24] that is designed to optimize heatmap regression networks. They use it to train a facial landmark detection network which is based on the Stacked Hourglass Network [47]. Their solution achieves state-of-the-art results on various datasets.

The evaluation results are presented in Chapter 4.

3. Methods

A detailed overview about the system that is designed and implemented during this thesis is given in this chapter. First, the overall architecture is described on a high level in Section 3.1. Then, assumptions about the input images are made in Section 3.2 and the coordinate system is explained in Section 3.3. The two main components of the system are the Stacked Hourglass Network and the Point Distribution Model. Details about the architecture of the Stacked Hourglass Network can be found in Section 3.4 and the Point Distribution Model is covered in Section 3.5.

3.1 High-level system overview

The algorithm developed in this work is fully based on neural networks and divided into two steps.

First, initial coordinate predictions for each facial landmark are computed using a fully-convolutional neural network that regresses heatmaps for each landmark. This network architecture is known as Stacked Hourglass Network and described in Section 3.4.1.3. It is designed to analyze an image on multiple scales at the same time and to combine features on different scales. This allows the network to learn how the landmarks are typically arranged relatively to each other. The resulting heatmaps are converted into numerical coordinates using the Differentiable Spatial To Numerical Transform (DSNT) as described in Section 3.4.2. The network is trained using Wing loss because it has some desirable properties for the optimization of regression networks that are explained in Section 3.4.3. All components mentioned in this paragraph belong to the first step and are described in Section 3.4.

Since the first step does not include a shape model, it is not guaranteed to predict correct shapes for all face images. For example, it is possible that some landmarks are predicted to be located at positions that are unlikely due to human anatomy. Thus, the second step takes the initial predictions and optimizes them using a neural shape model called Point Distribution Model (PDM). The PDM is trained to encode valid 3D face shapes in a latent vector representation. By using 3D instead of 2D coordinates internally it is able to learn representations that are independent from the head pose or camera position. Different head poses are represented by a 3D affine transformation and a 2D projection. The PDM iteratively finds a latent vector that gets mapped to a shape that is as close as possible to the predictions from the Stacked Hourglass Network. Since it only encodes valid shapes it is expected to converge to a valid face shape. To prevent correctly predicted landmarks from being modified, the first step also returns a confidence measurement for each landmark. This confidence measurement is based on the variance of the probability distribution represented by the heatmaps that the Stacked Hourglass Network produces. Section 3.5.6 contains details about the heatmap variance and confidence calculation. The whole PDM architecture is described in Section 3.5.

To summarize, in the first step the face image is converted into a set of 2D coordinates and confidence measurements which are then fed into the second step to optimize these coordinates. The PDM never operates on the image but only on the coordinates.

To the best of our knowledge, we are the first to combine the Stacked Hourglass Network with a shape model. Various design choices for the system components will be explained for each of the components. The performance of the components and the overall system are evaluated in Chapter 4.

3.2 Input images

The input to the algorithm are 128 $\times$ 128 px RGB images that contain a single cropped face. Larger or smaller images are scaled to 128 $\times$ 128 px. Cropping faces from arbitrary images can be done using a face detection algorithm such as Viola & Jones [68] or MTCNN [84]. Face detection is not part of this work.

It is important to use the same face cropping method during model training and evaluation. For example, training the model on images that show tightly cropped faces and evaluating it on images that contain more of the background would result in worse predictions since the Stacked Hourglass Network learns a different bias because it was trained on faces with a different scaling than it is evaluated on. We empirically confirmed this behaviour by training and evaluating the network on different bounding box sizes. If the goal is to use the system on faces found by arbitrary face detectors, the training data must be augmented accordingly.

Since the bounding boxes provided by the different datasets that are used in this work (Section 4.1) often are too tight and do not comprise all landmarks, we decided to create artificial bounding boxes instead. The bounding boxes are created based on the facial landmark annotations. The smallest rectangle that contains all landmarks is increased by 5% in both width and height and used as the bounding box for the face cropping. This procedure can only be applied when the landmarks are already annotated, e.g. when evaluating the model on benchmark datasets. When the goal is to use the system on arbitrary images without annotations, it either has to be trained using a real face detector or a mapping between the detections of a real face detector and the artificial bounding boxes has to be learned. Alternatively data augmentation as described in the previous paragraph can be used. These are open topics not covered in this thesis that could be further investigated in future works. This work is focused on proving the concept of the Stacked Hourglass Network and the PDM.

The face images are assumed to show only frontal or semi-frontal faces which means that all landmarks are visible in the image. Faces with head poses different from the neutral pose are allowed as long as no landmark is occluded by the face itself. The system could also be used for profile pictures if the Stacked Hourglass Network was adapted to additionally output a visibility indicator for each landmark. The PDM would not have to be adapted as it already can handle occluded landmarks using its internal 3D face model if their confidence is set to zero. Both the Stacked Hourglass Network and the PDM would have to be re-trained on datasets that contain profile faces. However, this is not part of this work as we focus on frontal and semi-frontal images.

3.3 Coordinates

The whole system uses normalized coordinates in the range $[-1, 1]$ instead of real image coordinates in $[0, 127]$. This is because regressing values that are centered around 0 is numerically more stable than regressing only non-negative values, since the network weights are initialized randomly and the expected output in the first epoch is 0. Using positive values could lead to exploding gradients in the first epochs of the training.

Real image coordinates can be converted into normalized coordinates using this formula applied to both the x and y component:

$$normalize(c) = \frac{c}{63.5} - 1 \tag{3.1}$$

and back to image coordinates using

$$unnormalize(c) = (c + 1) \times 63.5 \tag{3.2}$$

The value 63.5 is the image width/length divided by two.

3.4 Stacked Hourglass Network: Computing initial facial landmark predictions

The first step in the facial landmark detection pipeline is described in this section. Based on the input image the (x, y) coordinates for each of the 68 facial landmarks are predicted. The input image is assumed to only contain a cropped face as described in Section 3.2.

The main component of this pipeline step is the Stacked Hourglass Network described in Section 3.4.1. It works directly on the input image and produces a set of heatmaps that represent the likelihood for each of the landmarks to be present at a specific location. Since we are interested in numerical 2D coordinates, these heatmaps have to be transformed into numerical values. This is done using the Differentiable Spatial To Numerical Transform (DSNT) as presented in Section 3.4.2. The Wing loss function described in Section 3.4.3 is used to train the network. A short recap about this pipeline step is given in Section 3.4.4. The whole procedure is evaluated in Chapter 4 in Section 4.5.

3.4.1 Stacked Hourglass Network architecture

Convolutional Neural Networks (CNNs) are well-suited for various image processing tasks [11, 29, 35, 47]. Many common image analysis networks are special CNN architectures. This work focuses on finding and implementing a neural network architecture that is able to produce accurate facial landmark location predictions.

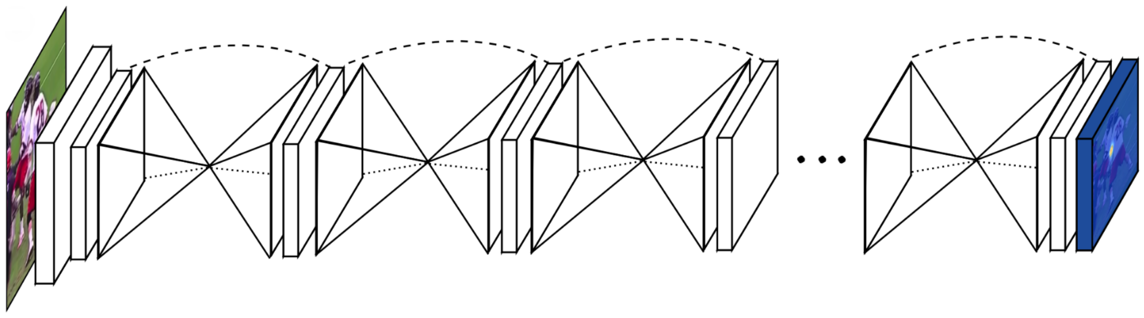


Figure 3.1: Scheme of a Stacked Hourglass Network [47]. The dashed lines mark the limits of a single hourglass. The left image is the input image and the right blue box is the set of heatmaps produced by the Stacked Hourglass Network.

Newell et. al [47] have defined a novel regression-based CNN architecture that is used for human pose estimation. It produces heatmaps for each of the landmarks of a human body, for example the knee, elbow, wrist or the shoulder. The shape of this network architecture reminds of an hourglass and the network is thus called an Hourglass network. We will refer to the Hourglass network as Hourglass (HG) from now on. It is a Fully Convolutional Neural Network (FCNN) that processes the image by downsampling it to a certain resolution (this is the bottleneck of the HG) and then upsampling it again in order to produce the heatmaps. Between layers of the same size there are shortcut connections. This allows the HG to analyze the image on different scales and, at the same time, combine features from different scales. When multiple landmarks whose locations depend on each other have to be located, this is helpful because the HG can use global cues to do local predictions. For the use case of human pose estimation the locations of different joints depend on each other because the human skeleton has only a certain degree of freedom. These relationships can be learned by a HG. The final output is a heatmap for each of the landmarks that indicates the likelihood for each pixel in the input image.

In order to produce even more accurate results, Newell et al. [47] stack multiple HGs and feed the output of one HG into the next HG. A stack of multiple HGs is depicted in Figure 3.1 and called a stacked HG. The idea behind this is to allow the network to first produce an initial prediction which is then refined by the following HGs. Before the first HG, the image is pre-processed as described in Section 3.4.1.3 in order to reduce the image dimensions and the memory usage. The more HGs are stacked, the more refinement can be done. The downside is that the cost also grows, e.g. number of parameters, memory usage, training time, inference time.

Newell et al. [47] use this architecture to estimate human pose from an image. Human pose estimation is related to facial landmark detection as in both tasks the position of each landmark depends on the position of the other landmarks to a certain degree. For example, if the position of the left eye is known, the position of the right eye can already be roughly estimated. Furthermore the rough layout of a human face is fixed: There is no situation where an eye would be at the position of the mouth or the nose and the mouth are confused. However, due to different face shapes, head poses, emotions and occlusions it is not possible to exactly predict landmarks based on only a few other landmarks. In order to produce exact predictions the model must have a general understanding of a human face shape and the typical appearance of landmarks. A HG is able to encode a general model of a face in its internal weights and use this information to produce accurate predictions.

As facial landmark detection is related to human pose estimation, this architecture has been chosen as the basic element for this work. We investigate how well stacked HGs work for facial landmark detection. Results are shown in Section 4.5. The stacked HG architecture and its basic elements are described in detail in this section and the changes that were made to the original architecture are outlined.

3.4.1.1 Residual modules

Newell et al. [47] make extensive use of residual modules [29] which are frequently used in image recognition tasks. These modules simplify the training of deep neural networks as they allow the gradients to flow back through not only one path but through a main path and a shortcut connection. Residual modules are the basic building blocks of HGs.

The general form of a residual module is shown in Figure 3.2. The module input is a 3D tensor with n_{in} channels and the output a 3D tensor with n_{out} channels. Both tensors have the same width and height. In other words, a residual module does not change the spatial dimensions (width and height) of the input but only the number of channels. The residual module consists of two branches, a main branch and a shortcut branch and the output of both branches is combined in the end. Both branches are labeled in green in Figure 3.2.

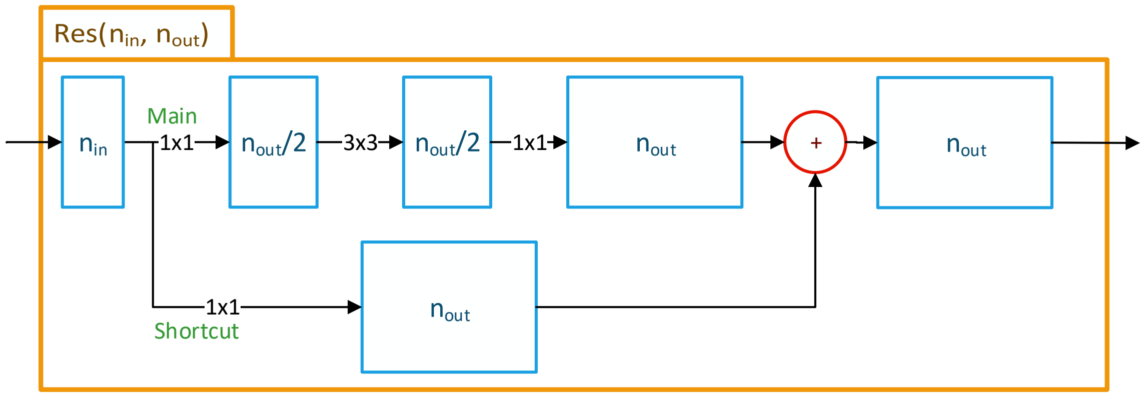


Figure 3.2: The basic building block of a HG is a residual module (Res). Blue boxes represent 3D tensors that have a width, height and a number of channels. The labels within the boxes denote the number of channels. The arrow labels show the kernel size of the convolution that is applied. If $n_{in} = n_{out}$ the 1×1 convolution in the shortcut branch is replaced by the identity function. The red sum symbol represents an element-wise sum operation.

In the main branch, first a 1×1 convolution is applied to reduce or increase the number of features from n_{in} to $n_{out}/2$. Then a padded 3×3 convolution with $n_{out}/2$ output feature maps is performed. The padding ensures that the spatial extent of the input is preserved. In contrast to the previous 1×1 convolution, the 3×3 convolution does not only look at one specific pixel and combine the information of the feature maps at this position, but it also considers a 3×3 px neighborhood around that pixel so that relevant local spatial

patterns in the input feature map can be captured. The last step in the main branch is another 1×1 convolution with n_{out} channels.
The shortcut branch is a 1×1 convolution which maps n_{in} to n_{out} feature maps if $n_{in} \neq n_{out}$ and the identity function otherwise. It branches off the main branch before its first convolution.

Both the main and shortcut branch have n_{out} channels and can thus be combined to the output of the residual module by performing an element-wise sum. Before each convolution in the main branch, batch normalization [31] is performed and the result is activated with $ReLU(x) = max(0, x)$.

This basic building block is used at many places in the (stacked) HG and will be referred to as $Res(n_{in}, n_{out})$ in further figures.

Residual sequences

Residual modules are used in sequences called residual sequences as shown in Figure 3.3. They consist of n_{mod} residual modules which all have the same number of input and output channels ($n_{in} = n_{out}$). Therefore we refer to the number of channels as n_{feat}. Having the same number of input and output channels implies that the shortcut branch in the residual modules is just the identity function.

We refer to residual sequences with n_{mod} modules and n_{feat} features as $ResSeq(n_{mod}, n_{feat})$. Using sequences of two or more residual modules allows the network to generate complex local features by alternatively combining multiple channels of a 1×1 px and a 3×3 px neighborhood.

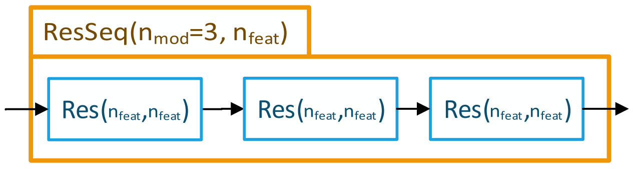


Figure 3.3: A Residual sequence (ResSeq) consists of n_{mod} residual modules with the same input and output dimensions.

3.4.1.2 Hourglass

Based on the residual sequence from Section 3.4.1.1 a HG is be defined. Its architecture is formulated recursively with a certain recursion depth and visualized in Figure 3.4. We will later refer to a HG with the recursion depth *depth* and residual sequences containing n_{mods} residual modules which have n_{feat} features as $HG(n_{feat}, n_{mods}, depth)$.

On each recursion level a HG has two branches which operate on different scales, a big and a small branch. The green labels mark the two branches in Figure 3.4. The big branch operates on the original input resolution. The small branch performs 2D max pooling and therefore operates on a smaller scale where width and height are halved. It processes the downscaled input and then upsamples it using nearest neighbor upsampling. The output of both branches is summed element-wise so that features of different scales are combined.

The input is first downscaled (max pooling) in the small branch before it is processed by a residual sequence $ResSeq(n_{feat}, n_{mods})$. The next step is called the bottleneck and defined

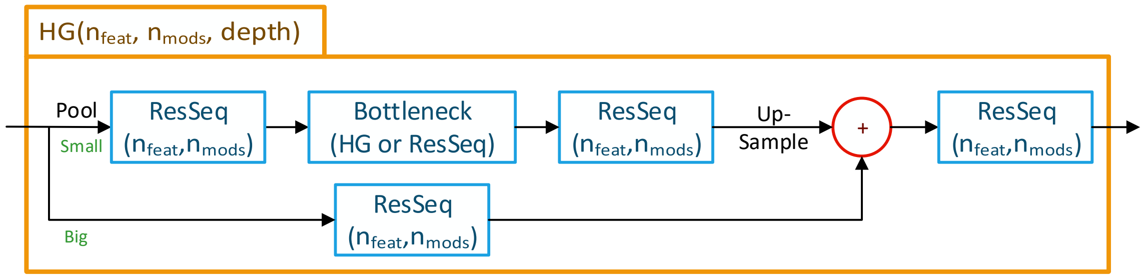


Figure 3.4: Scheme of an Hourglass (HG) before stacking. Blue boxes illustrate sub-networks and the bottleneck is a nested HG unless the maximum recursion level has been reached. The small branch operates on a smaller resolution (after pooling) than the input resolution while the big branch operates on the original input resolution. The last ResSeq is only present on the highest recursion level.

recursively. If the recursion level has reached *depth*, a $ResSeq(n_{feat}, n_{mods})$ is placed in the computation graph. Otherwise, a $HG(n_{feat}, n_{mods}, depth - 1)$ is inserted. This nested HG works on the smaller resolution and will further downsample it if the recursion end has not yet been reached. Note that the downsampling leads to a bigger receptive field in the 3×3 convolutions in the nested HGs. The more downsampling steps, i.e. the higher the *depth* value, the more of the global context can be captured by the convolutions at a specific depth.

The next element in the small branch is another residual sequence in both recursion cases. To be able to combine the features from the small and big branch the feature maps in the small branch need be to upsampled. Newell et al. [47] use nearest neighbor upsampling, however more advanced interpolations such as bilinear interpolation could also be used. Since nearest neighbor upsampling already achieves low error rates (shown in Section 4.5), we also used nearest neighbor upsampling and did not compare the effect of different interpolations.

The big branch is a residual sequence of n_{mods} residual modules which have n_{feat} features and works on the pre-pooled input, i.e. on the original resolution. It is used to transport and transform information from the network part before the downsampling to the part after the upsampling step.

The output of both branches is summed up element-wise. Since both branches operate on different scales, the element-wise sum operation is the place where features from different scales are combined.
On the highest recursion level (original resolution) the output of the sum is fed into the last *ResSeq*. It provides the HG with an opportunity to transform its features before the next HG is executed or the final heatmaps are predicted. This block is omitted for nested HGs and the sum is directly used as its output.

Modifications of the original HG definition

The final residual sequence after the sum operation is not used by Newell et al. [47] in the original HG definition. They use a sequence of two 1×1 convolutions instead. A disadvantage of 1×1 convolutions is that they have no access to information of their neighbors but only to information of one specific pixel in all channels. Therefore we replace it by a residual sequence since we believe that the network can benefit from the 3×3 convolutions within the residual sequence. They allow to combine information from a local neighborhood instead of only combining information of all channels at one pixel. Since our implementation worked well we did not compare it to the original definition by Newell et al. [47].

The sum operation could be replaced by concatenating the feature maps of both branches and then applying a 1×1 convolution to reduce the number of channels afterwards. This could improve performance because during training a more complex combination of the two inputs can be learned instead of being forced to sum them. On the other hand this would increase the number of network parameters and the memory usage. As the sum already worked well, we did not explore the effects of this change.

The recursion depth of a HG is fixed to 4 in the original paper [47]. In this work the depth of the HG is a hyper-parameter that was explored using grid search (see Section 4.5.3). The maximal possible depth is defined by the input image dimensions. The images in this work have the size 128×128 px and the pre-processing of the stacked HG (see Section 3.4.1.3) reduces the dimensions to 32×32 px. On each recursion level the width and height are reduced by a factor of 2. Thus, the deepest HG evaluated in this work has 5 levels ($2^5 = 32$) and the bottleneck is only 1×1 px large.

3.4.1.3 Stacked Hourglass Network

A single HG as defined in Section 3.4.1.2 performs only one round of down- and upsampling. Stacking multiple HGs can improve the predictions as the first HG computes initial predictions which are refined by the following HGs. When multiple HGs are stacked, the image is down- and upsampled multiple times, allowing the network to get a fine-grained understanding of the scene.

Newell et al. [47] have proposed a stacked HG, as shown in Figure 3.5 for the case of $n_{hg} = 2$ HGs. $StackedHG(n_{hg}, n_f, n_m, n_o, depth)$ describes a stack of n_{hg} HGs with a certain $depth$ that contain residual sequences with n_m residual modules that have n_f features. The stacked HG produces n_o output feature maps that can be interpreted as heatmaps. Each of the n_o heatmaps corresponds to one landmark and indicates the likelihood of this landmark at each location in the image.

Pre-processing part of the stacked HG

The image is pre-processed by a neural network before running the actual HGs. The first step in the pre-processing network is a 2-strided 7×7 convolution with 64 channels that serves two purposes: It captures local patterns directly in the RGB image and reduces the image width and height by 50%. The image is further processed by a residual module which takes 64 channels and returns 128 channels. To further reduce the width and height of the tensors, a 2D max pooling operation is executed. After this step the width and height are four times smaller than in the original image. This is important as it reduces the memory usage and processing time. Since the image size in this work is 128×128 px, the resolution at this point in the pre-processing pipeline is 32×32 px. Before the first HG is executed, two more residual modules compute n_f features. The original stacked

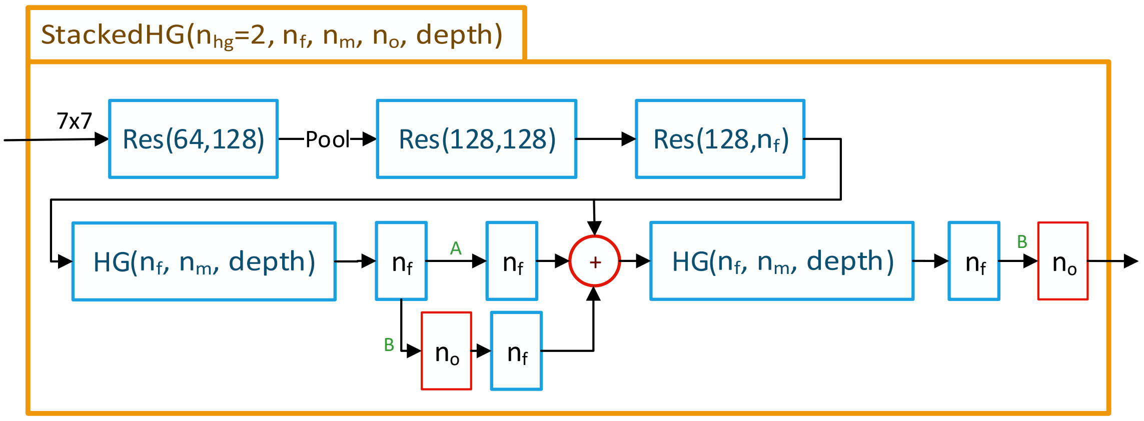


Figure 3.5: A stack of $n_{hg} = 2$ HGs including pre-processing steps. For better readability the number of features is designated as n_f instead of n_{feat} and the number of modules is n_m instead of n_{mods}. n_o is the number of heatmaps that are be produced.

HG architecture uses $n_f = 256$ features throughout the whole network. We make n_f a hyper-parameter and explore the effects of different values on the task of facial landmark detection. Results are presented in Section 4.5.3.3.

After these pre-processing steps, the resolution is 32×32 px. All HGs in the stack work on this resolution, Thus, after four downsampling steps (HG depth 4) the size is only 2×2 px while it is 4×4 px in the original paper (due to 256×256 px input images). The maximum possible depth is 5 as this will reach 1×1 px feature maps in the bottleneck. A way to increase the resolution the HGs work on and maximum possible depth is to replace the initial 7×7 convolution by a 3×3 convolution or completely remove it. However, in this work we kept the original pre-processing pipeline since the models would get too big to fit on a single GPU otherwise and the processing time would increase too much.

Main part of the stacked HG

Depending on the number of HGs n_{hg} in the stack the following steps are repeated:

A HG with n_f features in sequences of n_m residual modules and a specific *depth* is executed. A 1×1 convolution with n_f features processes the output of the HG before branching into two paths which are marked with green A and B labels in Figure 3.5. Branch A is another 1×1 convolution with n_f features and branch B produces n_o heatmaps using a 1×1 convolution (red box in Figure 3.5). If the HG is the last one in the stack, these heatmaps are the final predictions of the stacked HG. Otherwise, the heatmaps can be used to apply a loss function for intermediate supervision. Intermediate supervision means that the error between the predictions and the ground-truth is computed after every HG rather than after only the final HG. Newell et al. [47] found that this can lead to a small accuracy improvement. The intermediate heatmaps are transformed back into the feature space with n_f channels by performing a 1×1 convolution.

The input for the next HG in the stack is the element-wise sum of

- The input of the current HG. In case of the first HG this is the pre-processed image. This can be seen as a skip connection between adjacent HGs. It allows an HG to add its own information to the set of features and also improves the gradient flow.

- The output of branch A
- The output of branch B

To clarify: In the case of the last HG in the stack, branch A is not executed and branch B does not map the heatmaps back into the feature space, but the heatmaps visualized by the red box are used as the final prediction. This means that the red box of the last HG in Figure 3.5 is the final output.

The different HGs in a stacked HG do not share weights. The reason for this is that the input to each HG represents different features that can not be processed in exactly the same way. For example, the input to the first HG is the pre-processed image and thus contains only low-level features. However, the input to the second HG is the output of the first HG (plus the input of the first HG) which means that it is based on multiple representations of the original image on different scales, thus it contains high-level features that require different processing in the subsequent HGs (if there are any in the stack).

Stacked HG overview

Figure 3.6 shows the data flow of the whole architecture with a stack of $n_{hg} = 2$ HGs. The boxes are sized according to the width and height of the tensors they represent. For simplicity the temporary heatmaps (branch B) for the first HG and the skip connections are not shown in this image.

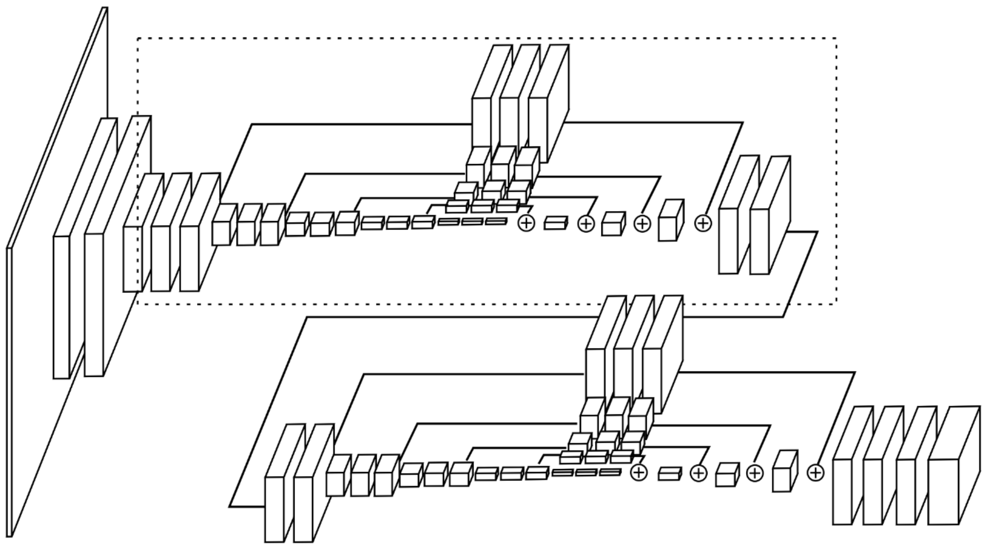


Figure 3.6: Tensor sizes visualized throughout a stack of two HGs. The dashed box shows the limits of a single HG. The four recursion levels can be seen through the differently sized boxes. Image source: `http://pocv16.eecs.berkeley.edu/camera_readys/hourglass.pdf`

The final output of the stacked HG is a set of unnormalized heatmaps. There is exactly one heatmap for each landmark. The higher the value in the heatmap at a specific location, the more likely the landmark is present at this location in the input image. The heatmaps are unnormalized because the sum of the elements in one heatmap is not guaranteed to be exactly 1 and elements can be negative, thus they don't represent a real probability distribution. For applications that require (x, y) coordinates the heatmaps need to be transformed into numeric coordinates. The next section explains the method used in this work.

3.4.2 Differentiable Spatial to Numerical Transform

In order to train and evaluate the model, the heatmaps produced by the stacked HG have to be converted into numerical (x, y) coordinates. In this section we present multiple possible solutions to this problem. The optimal method should have the following properties:

- **Spatial generalization**: Must be able to detect objects in each possible location in the image, even if it was never seen at this location during training.
- **Differentiability**: The operation should be differentiable to allow that the error can be directly computed between the regressed numerical coordinates and the ground-truth coordinates. This is better than computing the error between the regressed heatmap and an artificially generated ground-truth heatmap because it leads to an end-to-end trainable system. In such a system the gradients can be back-propagated through the whole network, from the numerical output to the input image [48].
- **No parameters and hyper-parameters**: Since training data is limited, the network is more likely to overfit when the system has too many parameters. Moreover, each additional hyper-parameter requires tuning to find the optimal value. Thus, an optimal coordinate regression method does not have trainable parameters and does not add hyper-parameters to the model.

Argmax. A simple solution is to look for the highest value in each heatmap and to use the location of this value as the regression result [47, 48]. However, if there are two similarly high values far apart in the heatmap it is not clear if the higher value is the correct one or if the true output should be the second highest or somewhere in between. Another problem is that the precision of this method depends on the resolution of the heatmap [48]. In other words, argmax is not able to return sub-pixel coordinates.

Furthermore, the argmax operation is not differentiable. Hence, to train the model, one must generate a ground-truth heatmap that can be used to compute of the prediction, e.g. using Mean Squared Error (MSE) or more advanced heatmap loss functions such as the Adaptive Wing loss [72]. Newell et al. [47] follow the argmax approach and generate a 2D Gaussian distribution with a standard deviation of 1px around the true location of a landmark. While this approach can achieve good results without adding network parameters, it is not differentiable and thus not well-suited for a end-to-end trainable system.

Method rating:

✓ Spatial generalization
✗ Differentiabilty
✓ No parameters

Fully connected regression. Another possibility is to add a fully connected regression network after the heatmap. While this approach is differentiable and consequently end-to-end-learnable, it has problems generalizing to unseen locations during test time [48]. For example, if a given landmark appears only in the left top corner area of an image during training but it is in the middle of the image during inference, then this solution will not be able to regress the location correctly because the connection between the heatmap and the regressor is fully connected. In other words, there is a specific network weight learned for each location in the heatmap and thus this approach requires more training samples with a wider distribution.

Using a fully connected layer after a set of heatmaps requires many parameters. For example, when 68 landmarks have to be predicted, there are 68 heatmaps with an example size of 32×32 px. A fully connected solution requires a minimum of $32 * 32 = 1024$ parameters for each of the landmarks and $68 * 1024 = 69632$ parameters for all landmarks. This is only the lower bound of the number of parameters when only one fully connected layer is used. In practice, using two or more layers might produce better predictions.

Method rating:

- ✗ Spatial generalization
- ✓ Differentiabilty
- ✗ No parameters

Differentiable Spatial to Numerical Transform. To overcome the aforementioned issues, Nibali et al. [48] propose a Differentiable Spatial To Numerical Transform (DSNT). The DSNT layer has no learnable weights and no hyper-parameters that have to be tuned. Intuitively, they calculate the weighted sum of a static coordinate grid and use the normalized heatmap values as the weights. Figure 3.7 visualizes how the DSNT layer works. There is a static grid for both the x and y component of the coordinate. For a $h \times w$ heatmap, the grids are defined as follows:

$$X_{i,j} = \frac{2j - (w+1)}{w} \tag{3.3}$$

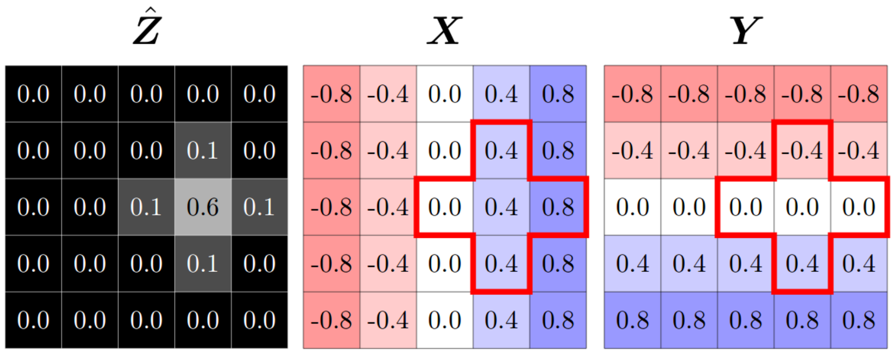
$$Y_{i,j} = \frac{2i - (h+1)}{h} \tag{3.4}$$

for $i = 1...h$, $j = 1...w$. The example in Figure 3.7 shows the case $h = w = 5$.

Before applying the DSNT layer, the unnormalized heatmaps Z from the stacked HG are normalized using the softmax activation function:

$$\hat{Z}_{r,c} = \frac{\exp(Z_{r,c})}{\sum_{i=1}^{h}\sum_{j=1}^{w}\exp(Z_{i,j})} \tag{3.5}$$

In this formula, r and c denote the row and column of the heatmaps and $\hat{Z}$ is the normalized heatmap where all entries are non-negative and sum up to 1.



$$x = \left\langle \hat{\boldsymbol{Z}}, \boldsymbol{X} \right\rangle_F = \left(0.1 \times 0.0 \quad + \quad \begin{matrix} 0.1 \times 0.4 & + \\ 0.6 \times 0.4 & + \\ 0.1 \times 0.4 & \end{matrix} \quad 0.1 \times 0.8 \quad + \right) = 0.4$$

$$y = \left\langle \hat{\boldsymbol{Z}}, \boldsymbol{Y} \right\rangle_F = \left(0.1 \times 0.0 \quad + \quad \begin{matrix} 0.1 \times -0.4 & + \\ 0.6 \times 0.0 & + \\ 0.1 \times 0.4 & \end{matrix} \quad 0.1 \times 0.0 \quad + \right) = 0.0$$

Figure 3.7: The Differentiable Spatial to Numerical Transform (DSNT) [48] applied to a normalized heatmap $\hat{Z}$. The coordinates x (y) are computed as the scalar dot product between the vectorized matrices $\hat{Z}$ and X (Z). Image source: [48]

To produce the output coordinates, each heatmap value is multiplied with the corresponding grid value and the products are summed up. This can be formalized using the Frobenius inner product:

$$DSNT(\hat{Z}) = \left[\left\langle \hat{Z}, X \right\rangle_F, \left\langle \hat{Z}, Y \right\rangle_F \right] \tag{3.6}$$

An example of the Frobenius inner product is shown in Figure 3.7. It is defined as the scalar dot product between two vectorized matrices.

Normalizing ensures that the values in the heatmap represent a probability distribution, so that applying Equation 3.6 is guaranteed to output values in $(-1, 1)$.

$\mu_x = \mathbb{E}[X] = \left\langle \hat{Z}, X \right\rangle_F$ and $\mu_y = \mathbb{E}[Y] = \left\langle \hat{Z}, Y \right\rangle_F$ have a probabilistic interpretation. They are the mean of X and Y with the probability distribution $\hat{Z}$. In contrast, the argmax operation is the mode of X and Y with the probability distribution $\hat{Z}$ [48].

Symmetrically distributed values around the center cancel each other out, so the center is still predicted correctly. The example in Figure 3.7 shows four values of 0.1 around the 0.6 that cancel each other out. The DSNT layer is designed to predict the correct location even when there is evenly distributed noise around the highest value.

Nibali et al. [48] claim that the DSNT layer can improve accuracy in stacked HG models when used instead of the argmax method. We furthermore have compared the prediction results using a fully connected regression network to the results using DSNT in

Section 4.5.4.3 and found that DSNT leads to a significant improvement. This is why we use DSNT to convert the heatmaps into coordinates.

DSNT generalizes spatially and is differentiable, so when training a coordinate regression network (e.g. a stacked HG), the loss can be directly calculated between the ground-truth coordinates and the predicted coordinates, without the need to create artificial heatmaps and to calculate the loss between them.

Method rating:

✓ Spatial generalization
✓ Differentiabilty
✓ No parameters

Combining DSNT with heatmap regularization

The DSNT layer as previously described satisfies all requirements to a optimal regression method. Nibali et al. [48] have shown that results can be further improved by regularizing the heatmaps.

Since the normalized heatmaps can be interpreted as probability distributions, the Jensen-Shannon divergence is a suitable regularization function. It compares two probability distributions and penalizes differences between them. Using the Jensen-Shannon divergence between the normalized heatmaps and a 2D Gaussian around the ground-truth coordinates forces the model to generate heatmaps that are shaped like a Gaussian distribution. Example heatmaps are shown in Figure 3.8. The top row contains heatmaps that are shaped like a 2D Gaussian as intended by using the regularization. The heatmaps in the bottom are wide-spread across the image and thus lead to inaccurate predictions when using DSNT.

The goal is to produce heatmaps with a high probability at the correct location of the respective landmark and a probability close to zero everywhere else, as this will lead to more accurate and robust predictions. This is true for the top row in Figure 3.8. If there were noisy probability values on one side of the correct location, the nature of the DSNT layer would lead to a prediction that is slightly moved to that side (second face in the second row). This is because the probability values are used to calculate the weighted mean of the coordinate grid which is then used as the final prediction. DSNT can only compensate noisy values when they are evenly distributed around the ground-truth location.

Using only the final coordinate loss can not directly avoid this unwanted behaviour. Nevertheless, the HG will still learn to produce heatmaps that have their highest value at the ground-truth location, but there is no guarantee that the values around that location are close to zero. The DSNT layer would also predict the correct location if the heatmap had the same value everywhere but a slightly higher value at the correct location.

Another extreme example is a heatmap that is zero everywhere except two places, where it has high probabilities. In that case DSNT would predict a point between those two values. An example is shown in the third picture in the second row. This means that when only using the end coordinate loss, the model could learn to predict multimodal distributions

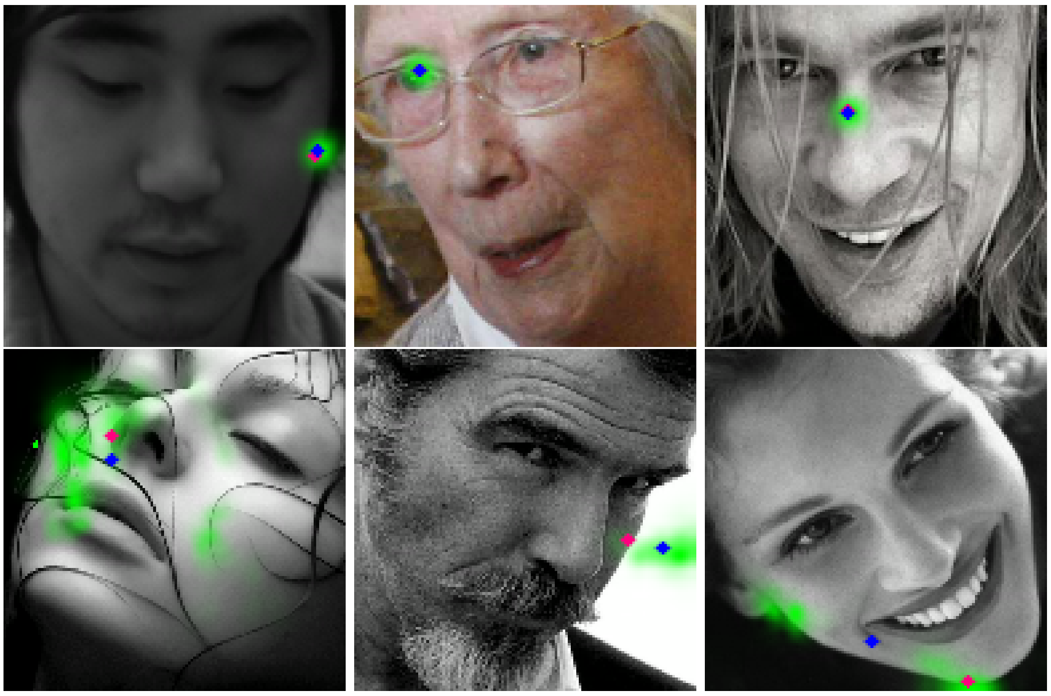


Figure 3.8: Example heatmaps produced by a stacked HG. High heatmap values are visualized with green color. The ground-truth location is marked red, the predicted location blue. Top row: Low heatmap variance and accurate predictions. Bottom row: High heatmap variance and inaccurate predictions.

around the true location and the final prediction would still be correct. However, the heatmaps would not be meaningful in this scenario.

For these reasons, we use both the final coordinate loss and the Jensen-Shannon divergence to train the stacked HG. The Jensen-Shannon divergence introduces the variance of the Gaussian as a new hyper-parameter. We compare using the divergence with different variances to not using it in Section 4.5.4.4.

For the sake of completeness, we note that Luvizon et al. introduce the soft-argmax layer [41] which works similarly to the DSNT layer. Nonetheless, we decided to use the DSNT layer since it is available for PyTorch [51], which we used in this work.

3.4.3 Wing loss function

In order to train a neural network a loss function is required. The network prediction is compared to the ground-truth and the loss is small if the prediction is close to the ground-truth and high if it is further from the optimal value. Loss functions are differentiable and can be minimized using back-propagation. The goal is to have a neural network that predicts the correct value for all samples, in which case the loss would be zero. There is a multitude of possible loss functions and the choice of a suitable function can have a big impact on the model performance. In this section some commonly used loss functions are reviewed and an alternative is shown.

Many facial landmark detection systems are optimized by minimizing the L1, smooth L1 or L2 loss [11, 24, 48, 76]. Feng et al. [24] name various problems when using the L2 loss to train a deep neural network. It is sensitive to outliers as the difference between the ground-truth and prediction is squared. Besides that, predictions that are already close to the ground-truth but not exactly correct have little influence on the gradient since squaring a small value makes it even smaller. The closer the prediction is to the ground-truth the smaller the gradient gets. Consequently it is hard to reach optimal predictions.

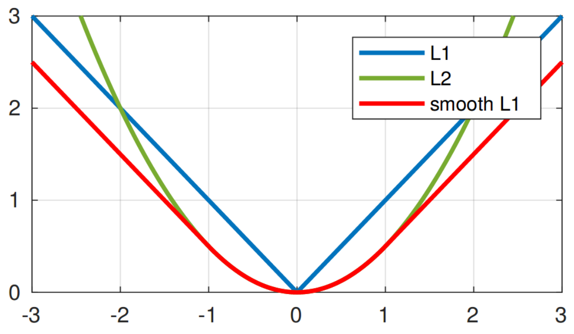


Figure 3.9: Comparison of L1, smooth L1 and L2. The x-axis is the difference between prediction and ground-truth (only one dimension is shown). Image from Feng et al. [24]

Figure 3.9 shows the behaviour of the L1, smooth L1 and L2 loss functions. L1 and smooth L1 are less sensitive to outliers than L2, but they are not optimal either: The step size of the L1 loss is dominated by large errors. The smooth L1 loss behaves like L2 around $x = 0$ and thus is not able to approach the optimal solution quickly.
Feng et al. [24] propose the Wing loss function that mitigates these problems. It is composed of a logarithmic part and a linear part:

$$wing(x) = \begin{cases} w \ln(1 + |x|/\epsilon) & \text{if}|x| < w \\ |x| - (w - w \ln(1 + w/\epsilon)) & \text{else} \end{cases} \tag{3.7}$$

The parameter w defines the width of the logarithmic part and ϵ describes its steepness. Figure 3.10 shows the Wing loss defined in Equation 3.7 for different choices of the hyper-parameters w and ϵ. The logarithmic part has a gradient that grows when x approaches 0. The linear part has a constant gradient and thus prevents outliers from having a big impact. At the same time it ensures that the network can recover from large errors caused by faces with a strong head pose. Using only the logarithmic part would lead to gradients that approach 0 for large errors.

Combining these two functions avoids the downsides of the L1, smooth L1 and L2 loss functions. Outliers have a smaller influence on the gradient than in the L2 loss function. Moreover, since the gradient grows as the error decreases, the network can converge to a state where it produces optimal predictions. In other words, in contrast to the L2 and smooth L1 losses, the Wing loss strengthens the effect of errors within $(-w, w)$. We found that better models can be trained with the Wing loss than with the L1 and L2 losses. Different loss functions and their effect on the facial landmark detection accuracy are compared in Section 4.5.4.1.

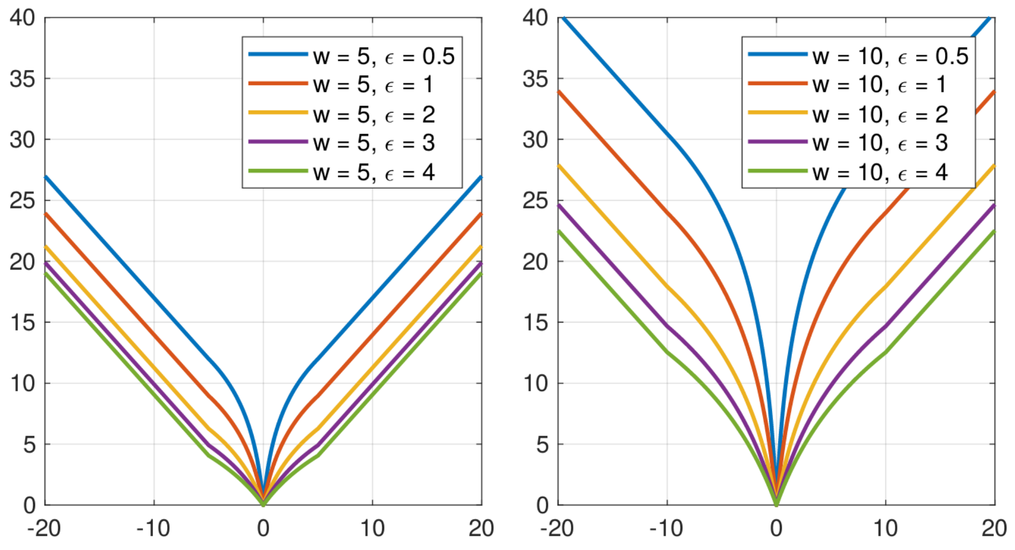


Figure 3.10: Shape of the Wing loss function for different values of w and ϵ. Image from Feng et al. [24]

3.4.4 Recap: Initial predictions

The previous sections covered all components of the first step of the pipeline. The predictions produced by these components can already be used for tasks that use facial landmarks as input features, e.g. facial expression analysis. The stacked HG combined with the DSNT layer and the Wing loss as presented in Section 3.4.1.3 is able to produce accurate state-of-the-art predictions without the need of any post-processing. We compare the results to multiple state-of-the-art systems in Section 4.5.

However, even though the stacked HG already learns an implicit shape model (the spatial relationship between landmarks) by extracting features on multiple scales, strong head poses can still lead to wrong predictions. Predictions on images with strong head poses often have a higher error than images with frontal faces. The next section will present another network that is trained to improve the initial predictions. It is executed on top of the stacked HG and is intended to help fixing imprecise predictions caused by strong head poses or occlusion.

3.5 Point Distribution Model: Refining predictions

The second part of the system is the Point Distribution Model (PDM). The PDM is a shape model that is used to post-process the shapes predicted by the stacked HG in a way that ensures spatial consistency among all facial landmarks. Irregular face shapes should be transformed into more likely shapes which means that wrong predictions should be fixed. The PDM is completely independent from the landmark appearance in the face image as it operates purely in the coordinate space and does not use color information from the original image. Instead, it relies on initial coordinate predictions (e.g. by the stacked HG) together with a reliable confidence measurement in order to determine which points to improve and which to keep unchanged.

The PDM is an unsupervised generative shape model that learns the arrangement of facial landmarks in various possible head poses, appearance of different humans, etc. It hence learns how faces in various scenarios are shaped and how the individual landmarks are correlated. This knowledge is used to fix wrong predictions of other models, e.g. the stacked HG.

In Section 3.5.1 the relationship between the PDM and other shape models is outlined. The overall concept of the PDM is explained in Section 3.5.2. The optimal size of the internal representation is discussed in Section 3.5.3. Section 3.5.4 covers the training of the PDM and Section 3.5.5 shows how it is used to reconstruct shapes while fixing wrong predictions. Details on the confidence measurement are given in Section 3.5.6. Finally, two methods to optimize speed and accuracy are presented in Section 3.5.7 and Section 3.5.8.

3.5.1 Relation to other shape models

The PDM developed in this work differs from the PDM used Active Shape Models (ASMs) [16] (see Section 2.1.1) in the way shapes are represented. The PDM in ASMs uses Principal Component Analysis (PCA) to embed face shapes into a vector space that captures the natural variations of faces in its dimensions. The PDM in this work builds its shape representations using a neural network that defines a mapping from a latent vector space and face shapes. From now on, the term PDM always refers to the model designed in this work.

The PDM is an implementation of the Variational Auto Decoder (VAD) [81] which can be seen as an encoder-less Variational Auto Encoder (VAE) [34]. Although the VAE can be used to encode 2D face shapes into a vector space, there are two downsides. First, there is no natural way which allows to encode pose information and actual face shape information separately. VAE encodes the 2D shape into a vector and can decode it from this vector again. However, the VAE has no information about the angle the underlying 3D shape is shown from. Hence, it will encode the same face into different vectors when it is shown from different directions. This means that both the variation in the face itself (such as facial expressions and natural variations due to different persons) and the pose (the angle and scale the face is shown from) are encoded into one vector. Second, if some of the points are self-occluded, there is no way to ignore these points and encode the face into a vector that is similar to the vector for the same shape without self-occlusions. In other words, the VAE is not designed to be used with partial data. This is especially problematic if the VAE needs to encode profile faces where many points are occluded by the face itself. While the second problem is not relevant for this work (since only semi-frontal faces are used), the first one can cause issues. To summarize, the VAE can not separate shape variations from pose variations and it is not able to work well with shapes that have self-occlusions. Both problems are caused because the the shapes are modeled in 2D.

The VAD does not have an encoder. It only consists of a decoder that maps a latent vector to arbitrary data like 2D or 3D shapes. Hence, the latent vector is the encoded version of the data. In order to obtain the latent vector for a new data sample, a way to invert the decoder is required. The VAD achieves this by starting from a random latent vector which is iteratively optimized until it gets decoded to the input data. In other words, the latent vector is obtained by finding a vector that has the lowest reconstruction error. The PDM is a special version of the VAD that is able to model 3D shapes that are observed in 2D. This is the case for all images that show real-world objects, e.g. faces. A face is a 3D object but an image only shows the 2D projection that depends on the pose. Variations in the 3D shapes and pose variations are modeled separately. Zadeh et al. [79] follow a similar approach in their Constrained Local Model (CLM) variant, but instead of latent representations, they rely on principal components. The following sections provide a detailed explanation of how the PDM works.

3.5.2 Overall concept

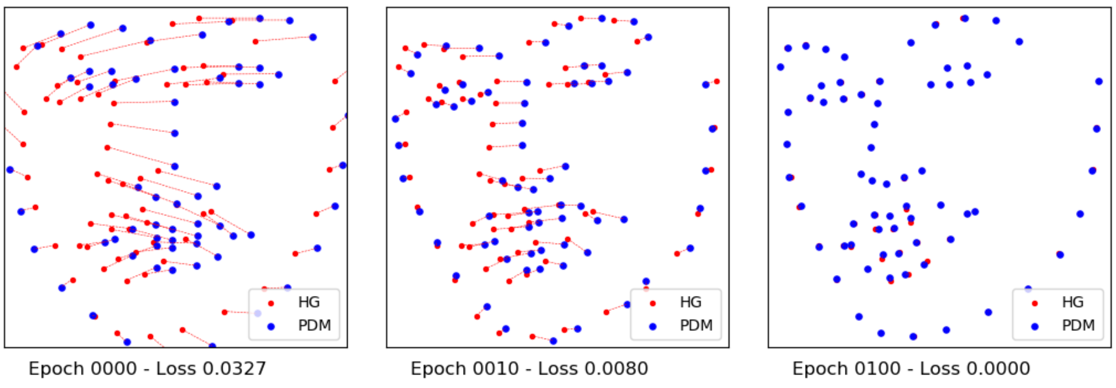
As described in the previous section, the PDM internally models shapes in 3D although only 2D projections of these shapes are used to train and use the PDM. In this work these 2D projections are the 2D faces that the stacked HG detects in an image.

The PDM uses VAD to embed 3D face shapes into a latent vector space. This multidimensional vector space contains abstract encodings of 3D shapes. A latent vector in this space corresponds to a specific shape that can be reconstructed from its vector by applying a (non-) linear function. Possible shape variations correspond to directions in this vector space. For example, in the case of faces there are some dimensions that encode variations caused by facial expressions while others encode gender or the shape of the face outline. To avoid that pose information is also encoded in the latent vectors, the PDM builds on the VAD and applies a 3D transformation (rotation, scale, translation) to the 3D shape that was decoded from the latent vector. This 3D transformation corresponds to the head pose or camera angle. This means that the shapes encoded in the latent vectors correspond to a neutral pose and are transformed to the actual pose using the 3D transformation. Since only 2D shapes are used in this work, the transformed 3D shape is projected to 2D. Thanks to the fact that the latent vector is independent from the transformation parameters, the PDM can effectively model face variations separately from pose variations. This is not only useful for profile faces but also for semi-frontal faces that have in-plane and out-of-plane rotations.

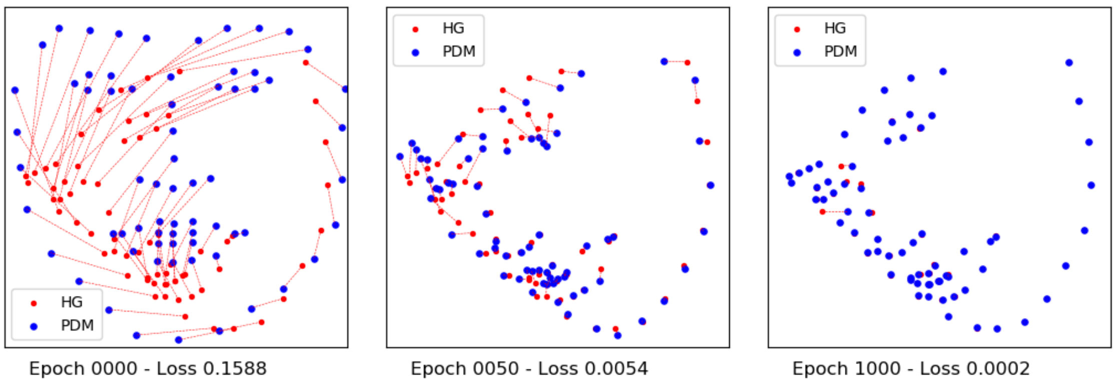
At the core of the PDM there is the decoder which is a fully-connected feed-forward neural network. The architecture of the decoder network can be arbitrarily deep. The only constraints are that the first layer has the same size as the latent vectors and that the output layer has $3N_{lm}$ neurons (in this work $N_{lm} = 49$ or $N_{lm} = 68$ facial landmarks). It decodes a latent vector into a set of 3D coordinates. These 3D coordinates are then transformed using a 3D affine transformation (scale, translation and rotation) and projected to 2D. All steps from the decoder to the projection are fully differentiable. The training procedure of the decoder is explained in Section 3.5.4. In order to encode a 2D face shape, both a latent vector and transformation parameters need to be inferred.

Once trained, the PDM can be used to improve the predictions by the stacked HG. For this purpose, the PDM infers a latent vector and a transformation that correspond to a 2D shape as similar as possible to the stacked HG prediction. Since the PDM is trained on valid face shapes, the produced shape will also be a face. If the hourglass prediction already was a valid face, the PDM will find a latent vector and transformation parameters that fully reconstruct this face. If some of the landmarks were predicted wrong but most others were correct, the reconstruction will match the correct landmarks but differ in the incorrect ones. That way, a wrong shape can be fixed. In order to prevent the PDM from changing the locations of correctly predicted landmarks, a confidence measurement is needed. A point that should remain unchanged should have a high confidence while points that are most likely wrong and should be moved to a better location should have a confidence close to zero. The confidence measurement is reliable if it is inversely proportional to the prediction error. Details on the inference process are given in Section 3.5.5. When the PDM converged, the reconstructed shape is used as the final prediction of the system.

To better understand the process, Figure 3.11a shows the prediction from a stacked HG and the reconstructed shape from the PDM after 0, 10 and 100 epochs. The reason why the blue points after 0 epochs (left image) already show a face rather than random noise is that the PDM has learned a bias that corresponds to the mean face of the training data. In the first epoch, the two shapes have a high distance (high reconstruction error). This distance is minimized iteratively and after 10 epochs (middle image) the reconstructed face is better aligned with the stacked HG prediction. After 100 epochs (right image) the reconstructed shape matches the input shape almost perfectly. This means that the PDM has inferred a latent vector and transformation parameters that encode the shape. The prediction by the stacked HG was already a valid face shape, thus there is no reconstruction



(a) Reconstruction of a correct stacked HG prediction after 0/10/100 epochs



(b) Correction of a stacked HG prediction with some wrong landmarks after 0/50/1000 epochs

Figure 3.11: Fixing a prediction using the PDM inference. The prediction from the stacked HG is drawn red and the blue points belong to the shape that the PDM reconstructs. The red lines visualize the distance between the same landmarks in both shapes.

error left in the end. In many cases there will be a difference between the two faces, but the reconstructed face will always be a valid face that is as close as possible to the input face. An example for this is shown in Figure 3.11b. In this example, most landmarks are reconstructed exactly, but some landmarks around the left eye still have a reconstruction error after 1000 epochs. This reconstruction error is caused by the stacked HG prediction that was not correct for these landmarks. The PDM could successfully fix these landmarks.

The PDM can be used on top of arbitrary models that output numerical coordinates and a confidence measurement. It is independent from the stacked HG and also from the shapes it is trained on. Besides face shapes it could also model human poses or other general shapes that are represented by landmark coordinates. For this purpose it would have to be trained on the shapes to model. In this work the PDM is used on top of the stacked HG to model facial landmarks.

3.5.3 Latent vector size

The size of the latent vector plays an important role. If it is too small, not all variations can be encoded and the reconstructed shapes lack in details. If it is too large, the PDM can store the original coordinates in the latent vector without having to learn possible shape variations and their correlations. If shapes with n 2D landmarks should be modelled, the dimensionality of the latent vector should be at least smaller than $3n$ because the decoder maps the vector to 3D coordinates. If it is larger than or equal to $3n$, the x, y, z components of the 3D landmarks can be stored in the vector and neither the correlations between different landmarks nor the pose variations are modelled. However, since the 3D coordinates are only used internally and only the 2D coordinates are used to compute the reconstruction error, the dimensionality should even be smaller than $2n$. Otherwise the 2D coordinates could be directly encoded in the vector and the transformation parameters could be set to the identity transformation. This would mean that the latent vectors encode both the shape and its pose in the vector. Since many landmarks are correlated with each other, the necessary number of dimensions in the latent vector space should be much smaller than $2n$. By using a small number of dimensions, the PDM is forced to learn a general representation of a human face and to be able to efficiently encode possible variations (emotions, head shape, etc). The optimal size of the latent vector is determined in Section 4.6.2.

3.5.4 Training: Learning a shape model

When training the PDM, the objective is to reconstruct the original 2D shapes from the latent vectors. For this purpose, optimal decoder weights have to be learned. These weights define the function the decoder uses to map the latent vectors to the 3D coordinates. The latent vector space is implicitly defined during training because the PDM finds a latent vector (and a transformation) for each training face that is mapped to a 3D shape which is then transformed to the original 2D face. It is not necessary to define the vector space or the decoder first. Instead, they are initialized randomly and learned jointly. By doing this, the PDM learns which variations can occur in faces and their correlations. These correlations are implicitly stored in the weights of the decoder network that is trained by minimizing the reconstruction error between the decoded and original shape. In total, three things have to be learned jointly during training: The decoder weights, latent vectors for the training samples and transformation parameters for the training samples.

An important detail is that no 3D data is required to train the PDM because 3D is only used internally. When using enough training samples with different poses, the PDM will learn to encode the shape variations for the mean pose in the latent vector space and to encode the pose variations in the transformation parameters. Internally a 3D shape model is learned. Depending on the quality of the training samples, this 3D model is more or less accurate.

The PDM is trained on samples that contain a set of 2D coordinates, e.g. facial landmarks. The training procedure is illustrated in Algorithm 1. For a better understanding of the data flow, the most important steps of the training and inference are shown in Figure 3.12.

Algorithm 1: PDM training algorithm, implementation of VAD training [81]

```
Input: 2D coordinates $X = [x_1, ..., x_n]$ for each landmark of $n$ training samples with
       $x_i \in \mathbb{R}^{N_{lm} \times 2}$, $N_{lm}$ is the number of landmarks for each sample
Result: Trained PDM decoder weights $W$,
        latent vectors $Z = [z_1, ..., z_n]$,
        affine transformation parameters $T = [t_1, ..., t_n]$,
        reconstructed 2D coordinates $C^{2D} = [c_1^{2D}, ..., c_n^{2D}]$ with $c_i^{2D} \in \mathbb{R}^{N_{lm} \times 2}$
1  Initialize weights $W$ of PDM decoder network randomly;
2  Initialize latent vectors $z_i$ randomly;
3  Initialize affine transformation parameters $t_i$ as the identity transformation;
4  while maximum training epochs not reached do
5      shuffle samples;
6      foreach mini-batch do
7          foreach sample $i$ in mini-batch do
8              Decode $z_i$ using PDM weights $W$ into set of 3D coordinates $c_i^{3D}$;
9              Transform $c_i^{3D}$ using sample-specific transformation $t_i$ into $c_i^{3D'}$;
10             Project $c_i^{3D'}$ into 2D space by dropping the $z$-component and store in $c_i^{2D}$;
11         end
12         Compute reconstruction error $err$ between $x_i$ and $c_i^{2D}$ for all samples $i$ in the
            mini-batch;
13         Back-propagate $err$ and update $W$, $Z$ and $T$
14     end
15 end
```

In the beginning of the training, a random latent vector $z_i \in \mathbb{R}^d$ is initialized for each sample i. The affine transformation parameters t_i are initialized to represent the identity transformation (no rotation, no scaling and no translation).

Once the variables are initialized, the iterative process begins. The latent vectors are decoded to a set of 3D coordinates c_i^{3D} for each sample by applying a simple feed-forward fully-connected neural network (the decoder). The input layer takes the latent vector and the output layer returns N_{lm} 3D coordinates. Afterwards, the 3D coordinates are transformed by the sample-specific affine transformation t_i. Since we are only interested in 2D coordinates, the z-dimension is dropped to project the landmarks from 3D to 2D.

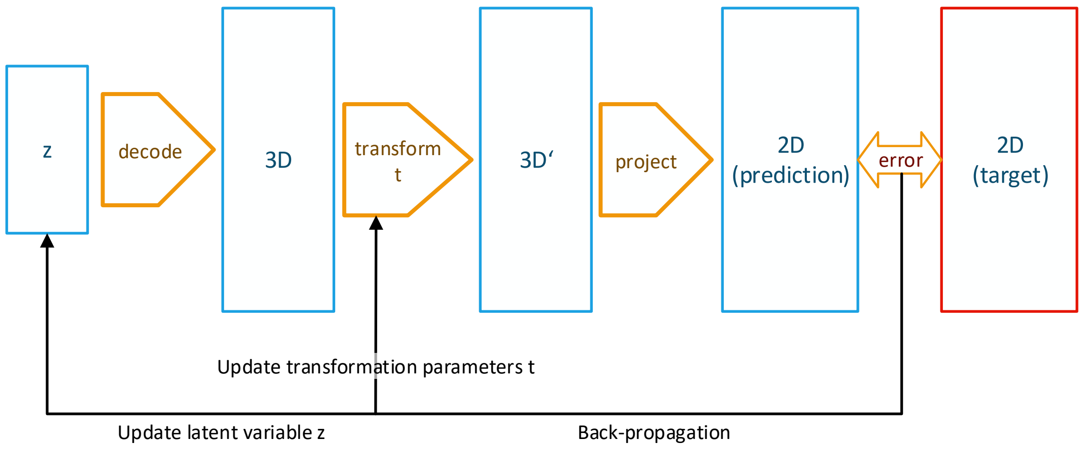


Figure 3.12: PDM inference to reconstruct the target shape: The latent z is decoded into a set of 3D coordinates which are then transformed using the transformation parameters t and projected to 2D by dropping the z-component of the 3D (x, y, z) coordinate. The error between the reconstructed 2D coordinates and the target is back-propagated to both the latent and the transformation parameters. During PDM training, the error is also back-propagated to the PDM decoder weights. The process repeats for a given number of iterations.

After the first iteration the resulting points are random since the latent vectors, the decoder network and the transformation parameters were initialized randomly. The goal is to find a latent vector z_i and an affine transformation t_i for each sample that are mapped to the 2D input shape x_i (the points predicted by the stacked HG). In addition to that a decoder needs to be learned. Thus, the loss between the PDM output shape c_i^{2D} and the input shape x_i is computed and back-propagated to both the decoder network weights W and the latent vector z_i, as well as the affine parameters t_i. Due to the beneficial properties of the Wing loss (Section 3.4.3) we employ it here as well: $err = WingLoss(c_i^{2D}, x_i)$. In the second iteration, the updated latent vectors and affine parameters are used by the updated PDM to produce new 2D coordinates which are more accurate. The process repeats until the maximum of iterations is reached.

During training, the PDM jointly learns optimal decoder weights W and both latent vectors Z and transformation parameters T that lead to a minimal reconstruction error on the whole training set. In order to do so it needs to learn a general 3D model of a human face and which variations are possible (e.g. smile or head shape). The model will learn to encode a frontal 3D face in the latent vectors (including variations like facial expressions or head shape) and to handle different head poses through the transformation parameters.

This model can then be used to generate new faces shapes that were never seen during training but can be generated by inferring a latent vector that gets decoded to that new face. The inference process is outlined in the next section.

3.5.5 Inference: Improving predictions

The inference is similar to the training process. The main difference is that the network weights are not initialized randomly but have already been trained. Thus, the mapping between the latent vectors and the shapes has already been learned and can now be used to infer latent vectors and transformation parameters for unseen face shapes. In the following we will refer to the combination of a latent vector z and the transformation parameters t as shape parameters.

In order to fix wrong predictions from the stacked HG, the PDM infers the shape parameters for these predictions. As the PDM has learned to map latent vectors to face shapes, it will produce a shape c_i^{2D} that is aligned as best as possible with the HG prediction x_i. The procedure is shown in Algorithm 2 and visualized in Figure 3.12.

Algorithm 2: PDM test algorithm, implementation of VAD inference [81]

```
Input: 2D coordinates $X = [x_1, ..., x_n]$ for each landmark of $n$ test samples with
       $x_i \in \mathbb{R}^{N_{lm} \times 2}$, $N_{lm}$ is the number of landmarks for each sample,
       Confidences $\pi = [\pi_1, ..., \pi_n]$ with $\pi_i \in \mathbb{R}^{N_{lm} \times 2}$
Result: latent vectors $Z = [z_1, ..., z_n]$,
        affine transformation parameters $T = [t_1, ..., t_n]$,
        reconstructed 2D coordinates $C^{2D} = [c_1^{2D}, ..., c_n^{2D}]$ with $c_i^{2D} \in \mathbb{R}^{N_{lm} \times 2}$
1  Initialize latent vectors $z_i$ randomly ;
2  Initialize affine transformation parameters $t_i$ as the identity transformation;
3  while maximum test epochs not reached do
4      foreach mini-batch do
5          foreach sample $i$ in mini-batch do
6              Decode $z_i$ using PDM weights $W$ into set of 3D coordinates $c_i^{3D}$;
7              Transform $c_i^{3D}$ using sample-specific transformation $t_i$ into $c_i^{3D'}$;
8              Project $c_i^{3D'}$ into 2D space by dropping the $z$-component, store in $c_i^{2D}$;
9          end
10         Compute reconstruction error err between $x_i$ and $c_i^{2D}$ for all samples $i$ in the
             mini-batch, weighted element-wise by the confidence $\pi_i$ (Equation 3.8);
11         Back-propagate err and update $Z$ and $T$;
12     end
13 end
```

To prevent the PDM from changing points that have already been predicted correctly, a confidence measurement π is used. For each of the N_{lm} landmarks of sample i there is a non-negative confidence value for both the x- and y-component stored in a confidence vector $\pi_i \in \mathbb{R}^{N_{lm} \times 2}$. The PDM is less likely to change the x or y component of a landmark if its confidence is high. Therefore the model that predicts the initial landmark locations must also predict a confidence measurement. Using a separate confidence for both the x and y direction has the advantage that predictions where one coordinate component is most likely correct while the other is unclear can be modelled. This is especially useful for landmarks that have an ambiguous appearance, e.g. the landmarks on the outline. In most scenarios the x component can be predicted easily because the outline landmarks are at edges with skin color on one side and background on the other side (for landmarks on the bottom of the skin the y component can be predicted more easily). However, the exact landmark location can be along the face outline. In this case the confidence in the direction orthogonal to the face border would be high and low in the other direction. A method to define the confidence is shown in Section 3.5.6. The confidence is only used during inference because the points that are used to train the PDM are the ground-truth, so they are all correct and their confidence is 1. Thus is can be omitted during training.

Suppose the stacked HG predicted a set of landmarks where all except one landmark have a high confidence. The PDM should be punished for moving these landmarks to other locations and as a consequence engaged to move the landmark with the low confidence value. This can be achieved by a loss function that has a specific weight for each of the landmarks for both the x and y direction. In Equation 3.8, both the reconstructed shape c_i^{2D} and stacked HG prediction x_i are element-wise multiplied ($\odot$) with the confidence π_i. Hence, the loss gets weighted by the HG confidence measurement.

$$loss(c_i^{2D}, x_i, \pi_i) = WingLoss(\pi_i \odot c_i^{2D}, \pi_i \odot x_i) \tag{3.8}$$

Assume the stacked HG is not sure about the position of a landmark $x_{i,j}$. In this case it returns a low confidence ($\pi_{i,j,x}$,$\pi_{i,j,y}$) for this landmark. In the other case the stacked HG is certain about its prediction and returns a high confidence value. Since the loss is weighted by the confidence, it will be high if the PDM reconstruction of a confident point is far away from its original position x_i, so the PDM gets penalized. At the same time it gets less penalized for a far reconstruction of a landmark with a low confidence. The lower the confidence of a landmark, the more the reconstruction is allowed to differ from the original position. This allows the PDM to move unconfident points to a more likely position while keeping the confident points. By doing so it fixes wrong predictions from the stacked HG.

When equation 3.8 is minimized using back-propagation, the latent vectors z_i and transformation parameters t_i are updated so that they represent a 2D shape that is closer to the confident points from the HG prediction x_i. The unconfident points will be moved to a new location that is more likely to correspond to a valid face. Note that points are not moved actively because that could lead to the generation of an invalid face shape. In other words, it is not possible to only change the location of one landmark without moving the others. Instead, if one point needs to be moved the error for this point is greater than zero. By back-propagating to the latent vectors z_i, the PDM moves this point in the desired direction. Important to note here is that this will also cause other points to move because the landmarks are correlated to each other. Therefore, moving a point makes the whole set of points more likely to be a valid face shape. The design of the PDM ensures that reconstructed shapes are valid faces when the process converged, i.e. when the error approaches zero or when it plateaus.

3.5.6 Confidence estimation

The variance of the heatmap of a landmark is a good indicator for how confident the stacked HG is about its prediction. If the heatmap has a low variance, the prediction is more likely to be correct than if it has a high variance. The more likely a prediction is to be correct, the less likely the PDM should be to modify it and therefore, the confidence value should be high. The heatmap variance in the x-direction $H_{var}[c_x]$ is defined as follows.

$$H_{var}[c_x] = \mathbb{E}[(c_x - \mathbb{E}[c_x])^2] = \left\langle \hat{Z}, (X - \mu_x) \cdot (X - \mu_x) \right\rangle_F \tag{3.9}$$

The $\cdot$ operator denotes a matrix multiplication. With X being a matrix (defined in Equation 3.3) and μ_x a scalar value, $X - \mu_x$ is a matrix produced by subtracting μ_x from every element in X. The heatmap variance in the y-direction $H_{var}[c_y]$ can be defined analogously.

$\mu_x = \mathbb{E}[c_x]$ is the predicted x-coordinate from the DSNT layer that was produced based on probabilities from the heatmap $\hat{Z}$ as explained in Section 3.4.2. c_x is the random variable for the x-coordinate. Equation 3.9 is the expectation of $(c_x - \mathbb{E}[c_x])^2$ with the probability distribution $\hat{Z}$. It can be interpreted as the expected squared distance between the predicted location c_x and any other location that has a probability greater than zero. $H_{var}[c_x]$ is 0 when the normalized heatmap $\hat{Z}$ is 0 in each x location except $x = \mu_x$ (where it is 1). In this case the probability is concentrated at one pixel and thus the variance is 0. The variance is higher than 0 when there are probabilities greater than 0 in positions different than μ_x. The variance grows with the distance between μ_x and locations with high probability values because the expectation of the square of the difference between the mean μ_x and all possible locations c_x is computed. Hence, when the heatmap is 0 everywhere except a small region, the variance is lower than when the heatmap has positive values spread far apart. Thus, the heatmap variance is suited to define a confidence measurement π_x (π_y is defined analogously):

$$\pi_x(H_{var}[c_x]) = \frac{1}{a * H_{var}[c_x] + b} \tag{3.10}$$

a and b are hyper-parameters that control the range of the possible confidence values. b should be positive to avoid dividing by zero when $H_{var}[c_x]$ is zero. The confidence measurement is low when the variance is high and vice versa. Using this confidence definition does not add trainable parameters to the model, but two hyper-parameters a and b. They are the same for π_x and π_y and all landmarks. The reason is to minimize the number of hyper-parameters that have to be set, although better results may be possible with individual values for each landmark.

3.5.7 Initializer for the latent vectors

The PDM inference procedure as presented in Section 3.5.5 initializes the latent vectors z_i randomly. The transformation parameters t_i are initialized to represent the identity transformation (no scale, rotation or translation). Due to the random initialization, the PDM needs many iterations to converge. This process can be sped up by training a small feed-forward neural network that maps 2D coordinates to a latent vector and a transformation parameter vector. This network is called the initializer. These vectors can then be used to initialize the PDM inference process rather than doing the default initialization.

The initializer is trained after the PDM training has finished. The training samples for the initializer are the 2D coordinates of the facial landmarks in the PDM training dataset and the ground-truth are the latent vectors and transformation parameters that the PDM learned for the training samples during training. In other words, the initializer can approximate the iterative process of the PDM in one forward pass, but it can't replace it. There are still some iterative updates needed in order to have the PDM converge. The effectiveness of the initializer is evaluated in Section 4.6.4.

3.5.8 Learning rate scheduling

During inference the latent vectors and transformation parameters are updated using an optimizer that requires a learning rate. Rather than using the same learning rate for all iterations it can be useful to use a learning rate scheduler. Using learning rate scheduling allows to start with a higher learning rate that is decreased on a plateau of the reconstruction error. This can result in faster convergence and more accurate reconstructions.

The PDM inference can process multiple samples in one batch at the same time to speed up the inference of many samples. Samples are independent from each other since no network weights are updated as opposed to the training process. The inference goal is to minimize the reconstruction loss between x_i and c_i^{2D} (Equation 3.8). Using a constant learning rate for each sample could cause ending up in a plateau. Therefore, the learning rate is reduced when a plateau is reached. As samples are independent from each other, it is not possible to use a single optimizer and scheduler for all samples. Doing so would cause the learning rate to be reduced for all samples simultaneously whenever one sample reaches a plateau. Instead, each sample must have its own optimizer and its own scheduler to be truly independent.

In addition to individual schedulers there must be a separate optimizer for each sample. This is necessary because some optimizers like Adam [33] use momentum. Since samples are processed in batches, not each sample should be updated in each iteration. When processing a batch, only the samples in this batch should get updated and the other batches should remain unchanged until they are processed. However, even if there is no gradient for samples outside the batch, they still receive updates due to momentum. The only way to avoid this is to have one single optimizer for each sample that only operates on this sample.

4. Evaluation

The system proposed in Chapter 3 is evaluated in this chapter. We first describe the datasets that were used to train and evaluate the models in Section 4.1. Next, a definition of the error metric is given in Section 4.2. The evaluation results are reported for samples in different categories. These categories are introduced in Section 4.3. Details about how the models were trained are outlined in Section 4.4. In Section 4.5 different architecture options for the stacked Hourglass (HG) are evaluated and the performance is compared to state-of-the-art algorithms. In Section 4.6 we analyze in which cases the Point Distribution Model (PDM) is able to reduce the prediction error by the stacked HG. We furthermore compare the whole pipeline consisting of a stacked HG and a PDM in that section. To complete the evaluation, qualitative results are shown in Section 4.7.

4.1 Datasets

There are multiple facial landmark datasets that are annotated with different sets of facial landmarks [5, 28, 36, 52, 55, 56, 57]. These datasets differ in the number of landmarks, constrained or in the wild settings, indoor or outdoor scenes, degree of head pose (frontal, semi-frontal or profile faces), and occlusion. For example, Multi-PIE [28] contains only images in controlled indoor settings where pictures of many persons with different facial expressions were taken in different camera angles. Since we want to have an algorithm that is robust and works in uncontrolled settings, we use another dataset.

A common dataset used to train and evaluate facial landmark detection models is 300-W [55, 56, 57]. It is a combination of four different facial landmark datasets: Annotated Faces in the Wild (AFW) [52], iBUG [56] LFPW [5] and Helen [36]. Since all four datasets use a different annotation scheme, the authors of 300-W have unified them using a scheme consisting of 68 landmarks [22, 28] and published it as a new dataset. This annotation scheme was originally defined for the Multi-PIE dataset [28] and is visualized in Figure 4.1. All images used to train and evaluate the models in this work are annotated using this scheme. 300-W contains uncontrolled images of faces in the wild: people of different ages and gender, in indoor and outdoor environments, under varying illuminations, in presence of occlusions, under different poses and from cameras of different quality. All faces in 300-W are frontal or semi-frontal. The latter means that a face is shown from a non-neutral angle but all landmarks are still visible. Since frontal faces are also semi-frontal faces, we only use the term semi-frontal from now on. Profile faces, which have invisible landmarks due to head pose or camera angle, are not included in 300-W.

Note that the original 300-W dataset also contains the XM2VTS dataset [45] that consists of face images in controlled settings. Many publications [53, 79, 86] that train and

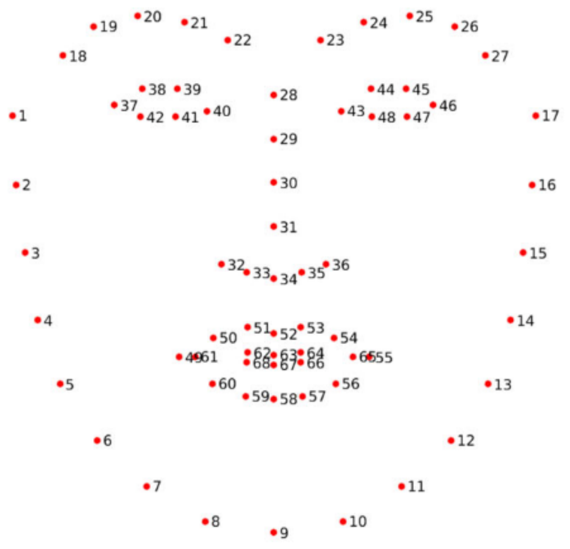


Figure 4.1: The Multi-PIE [28] annotation scheme used in this work consists of 68 landmarks (with outline) and 49 landmarks (without outline: 1-17 and 61,65 removed). Image from [22]

evaluate on 300-W do not include the XM2VTS part for two reasons: Images in controlled settings are easy to predict and the 300-W authors do not provide the images but only the annotations for XM2VTS. Therefore including the XM2VTS dataset requires to download the images manually, but some of them are not available anymore. For these reasons we ignore the XM2VTS part as well.

Consistent with many other works [24, 44, 53, 66, 79, 86], we use the Helen training set (2000 images), LFPW training set (811 images) and the whole AFW dataset (337 images) from 300-W to train the model (3148 images in total). The test is a combination of the full iBUG part (135 images) and the test partitions of LFPW (224 images) and Helen (330 images). In the test set, the iBUG part is considered more difficult and thus, we will evaluate our models in two categories: *easy* (LFPW + Helen) and *difficult* (iBUG). This distinction is also done in various other papers [6, 24, 44, 79]. We do not use the original annotations from these datasets but the unified ones from 300-W.

To demonstrate that our models can be used to produce accurate predictions even on a completely different dataset, we conduct a cross-dataset evaluation on the Menpo dataset [22, 82]. This means that we train the model on the training part of 300-W (described above) and evaluate it on the Menpo dataset in addition to the test part of 300-W. In contrast to 300-W, Menpo includes both profile and semi-frontal faces and is annotated with the same scheme as 300-W (Figure 4.1). Since the models are trained on 300-W (no profile faces included), we only use the semi-frontal faces from Menpo to do the evaluation and ignore the profile faces. Menpo is split into a train set and a test set. However, annotations are only provided for the train set. Therefore we ignore the test set and evaluate our model on the semi-frontal faces in the train set (6679 images), which is consistent with other works [44, 79]. To make it clear: Our models are trained on the 300-W train set in all cases and evaluated on the 300-W test set and the Menpo train set separately. The best results for our implementations of the stacked HG are presented in Section 4.5.5 and for the HG-PDM pipeline in Section 4.6.6.

In addition to the face images and landmark annotations, the 300-W dataset also contains bounding boxes for the faces in each image. However, since we want to train the model on 300-W and evaluate it on both 300-W and Menpo, we need consistent bounding boxes for all datasets. Another problem with the provided bounding boxes is that they often are too tight and do not contain all 68 landmarks. Thus, we use artificial bounding boxes as described in Section 3.2.

4.2 Metrics

In order to evaluate and compare facial landmark detection systems, an error measurement is required. A common metric used for this purpose is the point-to-point normalized Root Mean Squared Error (RMSE) [60]. There are multiple possible normalization strategies and a frequently used one is the Inter-Ocular Distance (IOD) normalized RMSE E_{IOD} [44, 60, 79]:

$$E_{IOD}(x^p, y^p, x^g, y^g) = \frac{\sum_{i=1}^{N} \|(x_i^p, y_i^p) - (x_i^g, y_i^g)\|_2}{N * d_{outer}} = \frac{\sum_{i=1}^{N} \sqrt{(x_i^p - x_i^g)^2 + (y_i^p - y_i^g)^2}}{N * d_{outer}} \tag{4.1}$$

where $x^p, y^p, x^g, y^g \in \mathbb{R}^N$ contain the coordinate components of the N landmarks of the prediction (x^p, y^p) and ground-truth (x^g, y^g). The IOD d_{outer} is the distance between the outer eye corners shown in Figure 4.2. The error E_{IOD} is the average distance between predicted and ground-truth locations, normalized by the IOD.

The motivation behind normalizing the error by the IOD is that images of different sizes and different face sizes can be compared to each other. However, normalizing by the IOD is not possible for profile pictures where only one eye is visible because the IOD is undefined in this case. If profile faces should also be evaluated, alternative normalization factors can be used instead of d_{outer} in Equation 4.1. Some works use the mean of the width and height of the face bounding box [52, 79], others use the face diagonal [76, 79, 82] or the square root of the bounding box area [10]. The bounding box is always the smallest axis-aligned rectangle that includes all ground-truth landmarks.
We use two different normalization factors for the evaluation datasets presented in Section 4.1.

300-W [55, 56, 57]: The IOD normalized RMSE is used for the 300-W evaluation. This is possible since only semi-frontal faces are used in this work. The main reason for this normalization strategy is that our results are compared to other papers that use this metric for this dataset.

Menpo [82]: As stated in Section 4.1 we remove the profile pictures from Menpo and only evaluate on the semi-frontal faces. Thus, the IOD normalization could be used for Menpo as well. However, we use the mean of width and height of the bounding box as a normalization factor because we compare our results to other works [44, 79] that use this normalization for Menpo. Many authors use this normalization factor because Menpo includes both frontal and semi-frontal images, even if only the semi-frontal images are used.

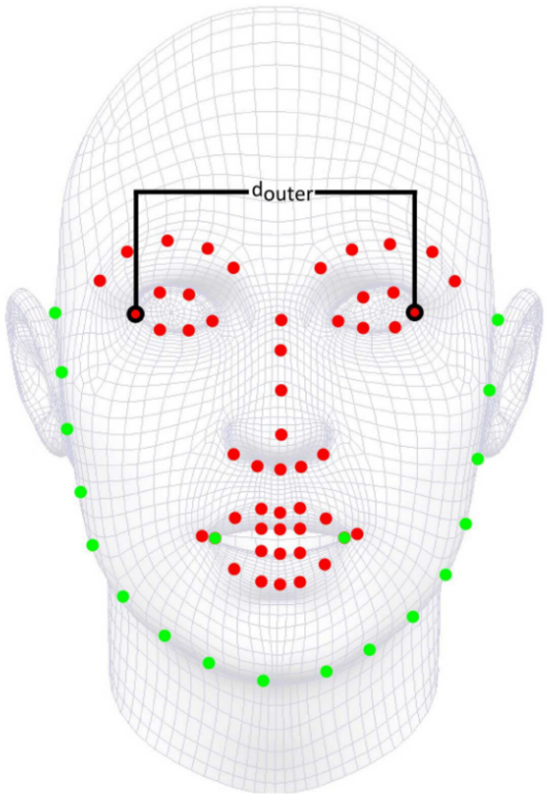


Figure 4.2: Different landmark types shown in the Multi-PIE [28] annotation scheme. Red points: 49 landmarks (without outline), green points: 19 landmarks (outline + inner mouth corner). Green + red points: 68 landmarks (with outline). d_{outer} is the IOD used for normalized error computation. Image from [60]

4.3 Evaluation categories

Some applications might require only a subset of the 68 landmarks (e.g. lip reading or eye tracking). It is possible to train models on only the landmarks that are required by the those applications. To demonstrate this, we train separate models for 49 landmarks (without outline) and 68 landmarks (with outline). The landmarks on the outline are generally more challenging than inside the face because the appearance is more ambiguous. Moreover, the labels are less accurate [44]. See Figure 4.2 for a visualization of the with/without outline cases. The 49 landmarks (without outline) are a subset of the 68 landmarks (with outline). The 49 landmarks for the without outline case are drawn in green and the 19 landmarks on the outline and the inner mouth corner are drawn in red. The green and red landmarks combined form the full set of 68 landmarks, denoted as the with outline case. We trained separate models for 68 landmarks (with outline) and 49 landmarks (without outline) because we want to find out if a model that is trained only on 49 landmarks can achieve lower errors on 49 landmarks than a model that is trained on 68 landmarks but only evaluated on 49 landmarks.

For the 300-W test set (consisting of the *easy* and *difficult* sets described in Section 4.1), there are four possible combinations that are evaluated separately. We report the IOD normalized RMSE for these four categories, following common literature [44, 79, 86].

- **easy (49)**: easy samples from 300-W with 49 landmarks (without outline)
- **easy (68)**: easy samples from 300-W with 68 landmarks (with outline)
- **difficult (49)**: difficult samples from 300-W with 49 landmarks (without outline)
- **difficult (68)**: difficult samples from 300-W with 68 landmarks (with outline)

In order to be able to analyze the models in depth, we store the best model of these four categories and the best model for the averages of the following combinations of these categories during training. These averages are defined as follows:

- **easy average**: The average error of *easy (49)* and *easy (68)*
- **difficult average**: The average error of *difficult (49)* and *difficult (68)*
- **49 average**: The average error of *easy (49)* and *difficult (49)*
- **68 average**: The average error of *easy (68)* and *difficult (68)*
- **overall average**: Avg. error of *easy (49)*, *easy (68)*, *difficult (49)* and *difficult (68)*

Storing these models separately allows to determine the best model for arbitrary scenarios. For example, it is possible that a model has the lowest error on *easy (49)* in a different epoch than on *easy (68)*. If we want to know how the model performs on easy samples in general it is not possible to just average the errors of both categories (*easy (49)* and *easy (68)*) because most likely the error is higher for the one category when it is the lowest for the other category. For this reason we compute the aforementioned averages after each epoch and save the model if the average has improved compared to the current best one. This is useful in case the best model for a subset of all samples, e.g. the easy samples, is of interest. Due to the inference complexity of the PDM it is not possible to evaluate the performance in each epoch. Thus, we evaluate the PDM only every 500 epochs and the stacked HG every epoch. In Section 4.5.5 and Section 4.6.6 the errors for all nine categories (the four categories and their averages) are reported for the stacked HG and the PDM.

Since there is no distinction between *easy* and *difficult* samples in Menpo, we only evaluate the 49 and 68 landmark categories separately on Menpo.

4.4 Training details

All models were implemented using PyTorch 1.0 [51] and Python 3.6 [50]. The Adam optimizer [33] was used to minimize the losses. The models were trained on various GPUs: NVIDIA GeForce 1080 Ti, NVIDIA Tesla K40c, NVIDIA Tesla V100 and NVIDIA GeForce GTX TITAN X.

To be able to efficiently train and evaluate a large number of models, a training framework was implemented and used for all experiments. It includes a config generator that processes a list of hyper-parameters and all possible values for each hyper-parameter. It generates a config file for each of the hyper-parameter combinations. A so-called model trainer reads a folder with these config files and distributes it on an arbitrary number of GPUs. For each GPU a process is started that obtains the configs from a queue, trains the model and returns the results to the trainer which writes them to a result file.

In order to get reproducible results, the random seed of the used libraries was set 0 in all experiments.

The code written during this thesis is available at `https://github.com/simonhessner/masters-thesis-final`. It includes scripts to train and evaluate both the stacked HG and the PDM. In addition to that there are scripts to visualize the predictions and heatmaps produced by the stacked HG.

4.5 Stacked Hourglass Network

The stacked HG as presented in Section 3.4 is evaluated in this section. First we describe the methods used to augment the training samples in Section 4.5.1. Next all hyper-parameters for the stacked HG are listed in Section 4.5.2. The hyper-parameters related to the HG architecture are analyzed in Section 4.5.3. Section 4.5.4 gives an analysis of hyper-parameters that determine how the network is trained and how coordinates are regressed. Our best models are compared to state-of-the-art methods in Section 4.5.5. Since a lot of effort was invested in experiments with Spatial Transformer Networks (STNs) which did not succeed, we outline reasons for the failure in Section 4.5.6.

4.5.1 Data augmentation

The input to the stacked HG are cropped images of human faces as described in Section 3.4. The images are were augmented using the following methods:

- **Random horizontal flip**: With a probability of 50% the image is flipped horizontally. Since there are fixed indices for each landmark, the annotations are also flipped.
- **Color jitter transform**: The brightness, contrast and saturation are randomly changed by up to 40% and the hue up to 10%.
- **Random rotation**: Images are randomly rotated by a certain maximum degree. To find out how much rotation works best, this was made a hyper-parameter that was explored during grid-search. The results are shown in Section 4.5.4.5.

Data augmentation was only performed during training. The samples were augmented individually in each epoch rather than augmenting them only once and using the same version in each epoch. Data augmentation was only used for the stacked HG and not for the PDM.

4.5.2 Hyper-parameter overview

In Section 3.4.1 multiple design choices were mentioned. The hyper-parameters for the stacked HG are explained in the following. The section where the effects of the parameters are studied are shown in brackets unless they were only analyzed empirically.

Network architecture (Section 4.5.3)

- **n_hgs** (Section 4.5.3.1): Number of HGs to be stacked
- **n_res_modules** (Section 4.5.3.2): Number of residual modules in a residual sequence
- **n_features** (Section 4.5.3.3): Number of features in a residual module
- **hg_depth** (Section 4.5.3.4): Number of downsampling steps in a single HG
- **n_lm** (Section 4.5.3.5): Number of facial landmarks to predict: 49 (without outline) or 68 (with outline), see Figure 4.2 for a visualization. When set to 49, the stacked HG is trained to predict only 49 landmarks. This is different from training the model on 68 landmarks and just using the 49 landmarks during evaluation.

The architecture parameters are the same for each HG in the stack. However, it could be further investigated if individual parameter choices for each HG can help to improve the accuracy. In addition to the hyper-parameters listed above we also investigate the effect of the model complexity in terms of trainable parameters in Section 4.5.3.6.

Training and regression (Section 4.5.4)

- **loss_function** (Section 4.5.4.1): L1, L2 or Wing loss [24] with different parameterizations
- **normalize_loss** (Section 4.5.4.2): Whether to normalize the loss to always have a magnitude of 1
- **regression** (Section 4.5.4.3): Method to convert heatmaps into numerical coordinates: Fully-connected or Differentiable Spatial To Numerical Transform (DSNT) [48]
- **heatmap_sigma** (Section 4.5.4.4): Standard deviation for Gaussians used in Jensen-Shannon divergence for heatmap regularization (DSNT only)
- **augment_rotation** (Section 4.5.4.5): Maximal angle to randomly rotate the faces during training (can be disabled by setting to zero)
- **predict_distances** (Section 4.5.4.6): Whether to add another loss that analyzes the pair-wise distances between the predicted landmarks. If yes, weight for this loss is specified.
- **intermediate_loss**: Whether to apply the loss after each HG in the stack. If set to false, only the last HG output is fed into the loss function. This hyper-parameter is discussed together with the loss function in Section 4.5.4.1.
- **n_epoch**: Maximum number of epochs to train a model
- **batch_size**: Number of samples in each mini-batch
- **lr**: Initial learning rate for the Adam optimizer [33]
- **lr_scheduler_patience**: Number of epochs to wait on a plateau before reducing the learning rate
- **lr_decay_factor**: Factor to decrease the learning rate on a plateau

To find out which hyper-parameters have which impact, an extensive grid search was performed.

Since there are many hyper-parameters to explore, the grid search was split into multiple grid searches. It is unfeasible to train a model for all combinations of hyper-parameters when resources (number of GPUs and time) are limited. The first grid search was focused on the architecture of the stacked HG. The results are shown in Section 4.5.2.

However, in order to get good results, suitable values for the other hyper-parameters had to be found first. For this, some empirical experiments were conducted to find good values for the learning rate, patience and decay factor of the learning rate scheduler, number of epochs and batch size. We found that a good initial learning rate value is 0.001. The learning rate was reduced by a factor of 0.5 when the validation error did not change significantly for more than 15 epochs (reduce on plateau). To speed up training, an early stopping criterion was used and the training was stopped when the validation error did not decrease by at least 2.5% within 31 epochs (so that the learning rate scheduler was able to decrease the learning rate at least two times before stopping training). For cases where the loss oscillated or did not reach a plateau, the maximum number of epochs was set to 200. The training set was split into mini-batches of 32 samples that were randomly shuffled in each epoch. These values were used for the entire evaluation of the stacked HG.

Since empirical experiments have shown the power of the DSNT layer, it was used throughout the first grid search. The Gaussian standard deviation for the Jensen-Shannon divergence was set to 1.0 and the maximum rotation degree for the augmentation was set to 0°. As these values worked well in general, we fixed these in all experiments unless stated otherwise. These and other hyper-parameters were further investigated in later grid searches and the results are shown in Section 4.5.4.

Note that due to a misunderstanding the first grid search was done using a train set that slightly differs from the original 300-W train set. The difference is that the AFW [52] part was replaced by a fraction of Multi-PIE [28]. This mistake only affects the training set. The test set was the correct 300-W test set in all experiments. The mistake leads to slightly worse results compared to using the correct 300-W train set because Multi-PIE contains only images in controlled settings while AFW consists of in-the-wild images. However, the general trends are still valid for the correct dataset. As soon as we noticed this mistake, we fixed the dataset and used the correct one for all further experiments, i.e. all experiments in Section 4.5.4.

4.5.3 Effect of architecture hyper-parameters

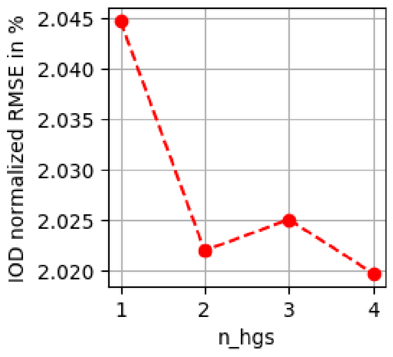
In this section we analyze the effects of HG-specific hyper-parameters on the prediction performance on the 300-W test set. To analyze the impact of one specific hyper-parameter we group the grid search results by all possible values for this parameter and determine the lowest IOD normalized RMSE within each group.
The HG architecture parameters were tested in all combinations of these parameters:

- **n_hgs**: 1, 2, 3, 4 (Section 4.5.3.1)
- **n_res_modules**: 1, 2, 3, 4 (Section 4.5.3.2)
- **n_features**: 64, 128, 256, 512 (Section 4.5.3.3)
- **hg_depth**: 1, 2, 3, 4, 5 (Section 4.5.3.4)
- **n_lm**: 49, 68 (Section 4.5.3.5)

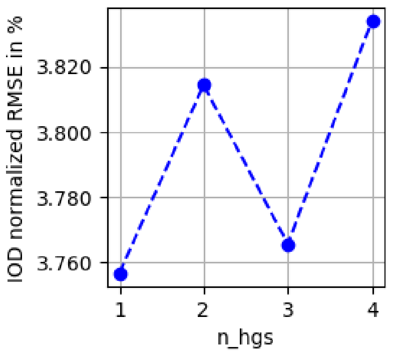
In total 640 models [1] have been trained and compared. The values reported in the next subsections are the median of the IOD normalized RMSE of all samples in each category. The median is more robust to outliers than the mean, thus we use it. For better readability the value is multiplied by 100 as it is done in other works [44, 79]. It can be interpreted as the average error relative to the IOD in percent.

Our best models presented in Section 4.5.5 sometimes were trained with parameters that were not the best ones from this first grid search. The reason for that is that many hyper-parameters influence each other. Since the first grid search only varied the architecture hyper-parameters while the second grid search varied the other hyper-parameters it is possible that the best values determined here turned out to be not the best ones in different settings. Nevertheless, the results presented in this section still help to get a general understanding of the rough effects of these hyper-parameters.

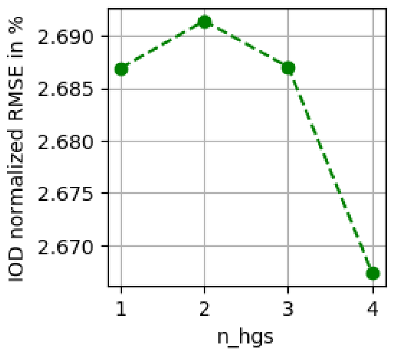
4.5.3.1 Number of stacked HGs



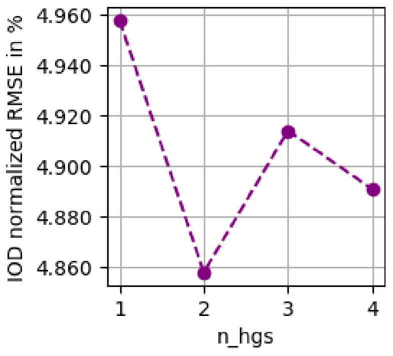
(a) Category: easy (49)



(b) Category: difficult (49)



(c) Category: easy (68)



(d) Category: difficult (68)

Figure 4.3: Lowest error for each category depending on the number of stacked HGs. Due to different error ranges in each category and small variations between different parameter choices there is a separate plot for each category to better visualize the trends.

The results of the grid-search were grouped by the number of stacked HGs. In each group the best performing model was chosen for each of the categories *easy (49)*, *difficult (49)*,

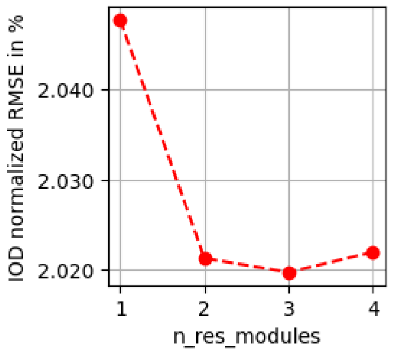
[1]Product of number of options for each parameter: $4 * 4 * 4 * 5 * 2 = 640$

easy (68) and *difficult (68)* as described in Section 4.3.

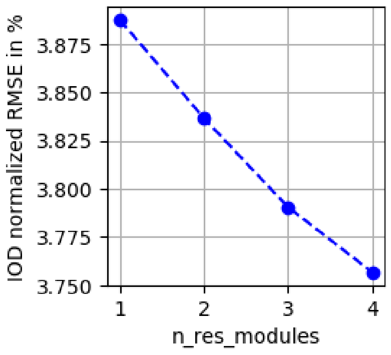
The lowest error that was achieved using a certain number of stacked HGs is shown in Figure 4.3 for each category. Interestingly stacking more HGs helps especially in the *easy* categories, for both 49 and 68 landmarks. Here, four stacked HGs had the lowest IOD normalized RMSE. For the *difficult* categories, a stack of one HG worked best for 49 landmarks and two for 68 landmarks. However, it should be noted that the differences in all four categories are very small, especially for the *easy* categories.

4.5.3.2 Length of residual sequence

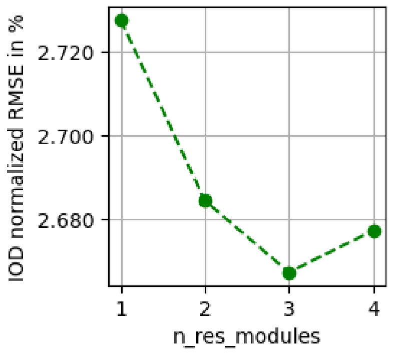
The results were grouped by the number of residual modules in a residual sequence. In each group the best performing model was chosen. Figure 4.4 shows that only one residual module in a residual sequence performs worst in all four categories. The best length of residual sequences is three in all categories except *difficult (49)*. In that case four residual modules in a residual sequence had the lowest error. The fact that the error for the easy categories is higher when using 4 residual modules compared to only using 3 suggests that the model overfits when too many residual modules are used.



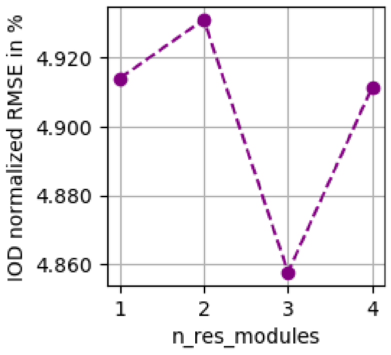
(a) Category: easy (49)



(b) Category: difficult (49)



(c) Category: easy (68)

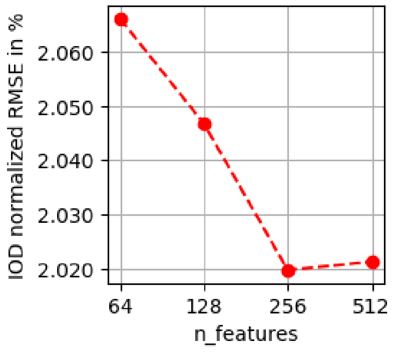


(d) Category: difficult (68)

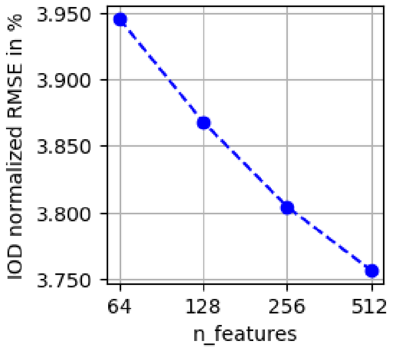
Figure 4.4: Lowest error for each category depending on the number of residual modules in a residual sequence.

4.5.3.3 Number of features

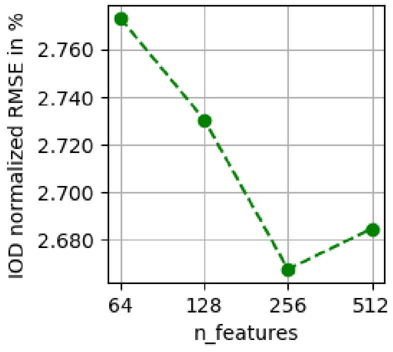
The results were grouped by the number of features in a residual module and in each group the best performing model was chosen.



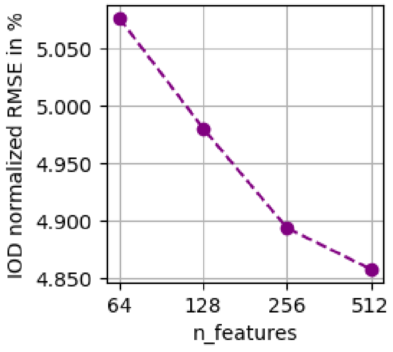
(a) Category: easy (49)



(b) Category: difficult (49)



(c) Category: easy (68)



(d) Category: difficult (68)

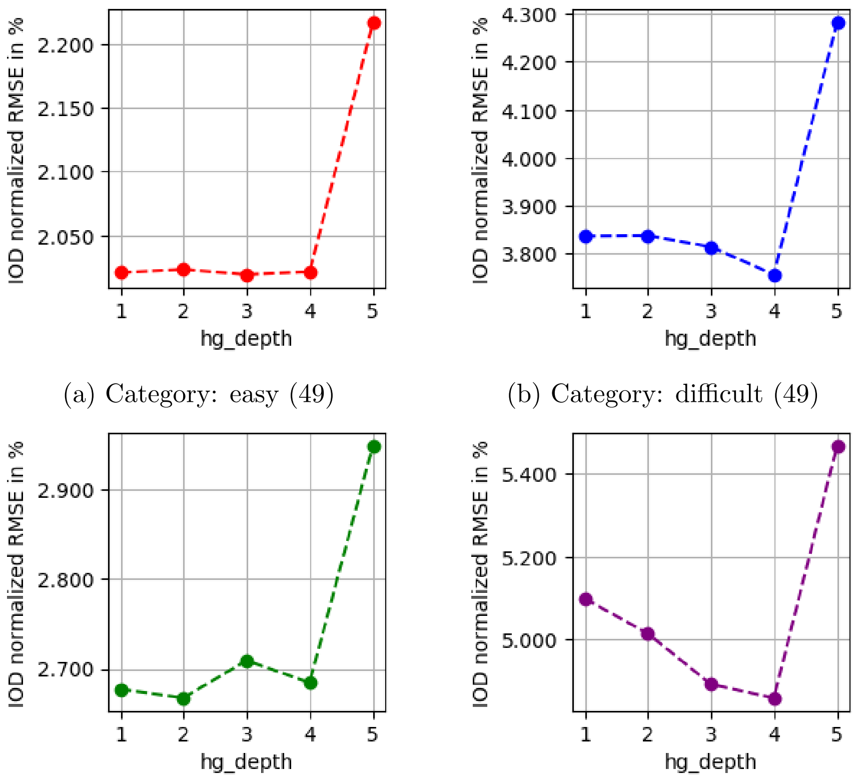
Figure 4.5: Lowest error for each category depending on the number of features in a residual module.

An interesting trend can be seen in Figure 4.5: The error for the *easy* samples is the lowest when using 256 features in residual modules for both 49 and 68 landmarks. This supports the assumption made in Section 4.5.3.2 where we found that three residual modules work better than four for *easy* samples, namely that the model can overfit when the model gets too complex.

When looking at the *difficult* samples, the more features are used, the lower the error gets. This is most likely due to the higher appearance variance in *difficult* images. It is possible that using more than 512 features could improve the error rate even more, but in that case the models get too large to be trained on a single GPU and the improvement would probably be relatively small. Since 256 and 512 features seem to be the best number of features, these two values will be used for various further experiments.

4.5.3.4 Number of downsampling steps

The results of the grid-search were grouped by the depth of a single HG, i.e. the number of downsampling steps. In each group the best performing model was chosen.



(a) Category: easy (49)

(b) Category: difficult (49)

(c) Category: easy (68)

(d) Category: difficult (68)

Figure 4.6: Lowest error in each category depending the on the depth of a HG.

While the original stacked HG implementation [47] uses a fixed HG depth of 4, we investigated how the error changes depending on the depth. The results are shown in Figure 4.6.

All four categories have in common that the error is by far the highest when the network has a depth of 5 which means that the image is downsampled 5 times, resulting in a bottleneck of only 1×1 px. A possible explanation is that the network is using the bottleneck to pass important information from the downsampling part to the upsampling part of the network. Having only 1×1 px is possibly not enough to store all relevant details. However, the shortcut connection could also be used for this task. It is possible that the shortcut connection has not enough capacity and hence the network has to use the bottleneck as well. A depth of 4 has a much lower error than a depth of 5 in all categories. This depth results in a bottleneck of 2×2 px, so it can can store information with some degree of spatial consistency (top-left, top-right, bottom-left, bottom-right). Section 4.5.3.6 contains more plots that clearly show that a depth of 5 performs much worse in all categories.

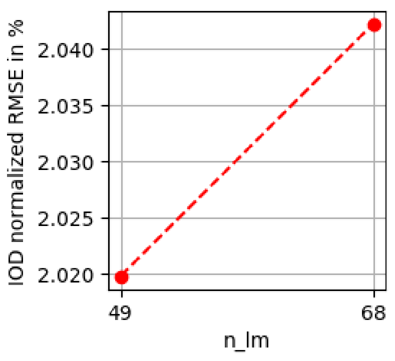
For both 49 and 68 landmarks even a HG that downsamples only one time performs already well for the *easy* categories (plots *a* and *c*). For 49 landmarks, the error is the lowest with three downsampling steps, for 68 with two. However, the difference between the errors with one, two, three or four downsampling steps is very small. This means that for applications that require a low memory footprint or fast processing times, a lower depth should be chosen for these categories.

For *difficult* samples the situation is different. A depth of four has the lowest error for both the 49 and 68 landmark category. An intuitive explanation is that the network does not need to analyze the image on a large scale when processing *easy* images as the faces are mostly frontal and landmarks can be predicted with a smaller receptive field. *Difficult* samples, however, often have strong head poses and require a deeper analysis.

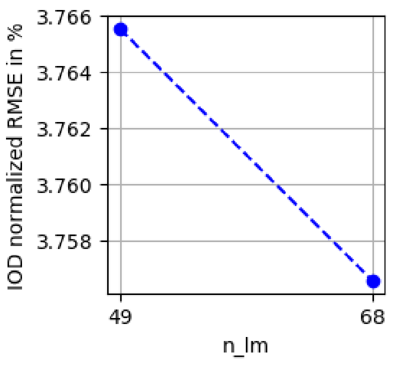
4.5.3.5 Separate models for 49 and 68 landmarks

As already mentioned in Section 4.3, there are applications that require only a subset of the 68 facial landmarks. We want to see if a model that is trained on only 49 landmarks achieves a lower error on these 49 landmarks than a model that was trained on 68 landmarks. The results of the grid-search were grouped by the number of landmarks. In each group the best performing model was chosen.

Figure 4.7 reveals that training a separate model for 49 landmarks helps for easy samples, although the performance gain is minimal. For difficult samples models trained on 68 landmarks have a lower error also when evaluated on 49 landmarks. Again, the difference here is minimal, but an intuitive explanation is that the model is able to learn better features when it is forced to not only focus on the 49 landmarks inside the face but also on the outline. This relates to a common strategy in many machine learning applications called multi-task learning [11, 38, 85]. A model is trained to solve multiple tasks at one time and each task benefits from the features that are learned for other tasks. Here, predicting the outline can be seen as an auxiliary task that helps accurately predicting the inner landmarks in difficult samples.



(a) Category: easy (49)



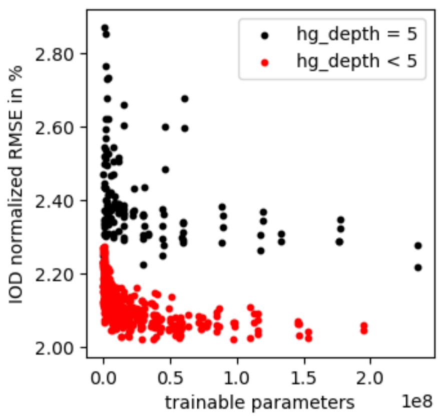
(b) Category: difficult (49)

Figure 4.7: The error without outline (49 landmarks) for models that were trained on 49 or 68 landmarks.

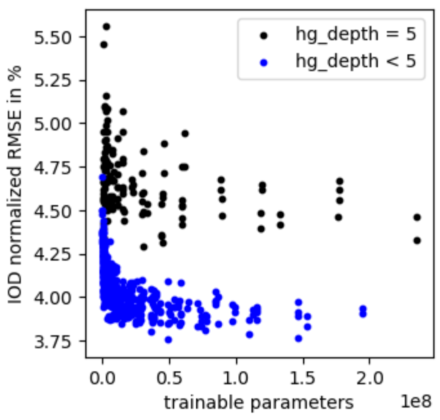
4.5.3.6 Number of network parameters

This section investigates which effect the network capacity, i.e. the number of trainable parameters, has on the localization error. The more parameters a model has, the more memory it requires during training and inference. Moreover, the inference time also depends on the number of parameters. Therefore, for applications that require a low memory footprint and/or fast processing times, one should use the smallest model that has a small enough error.

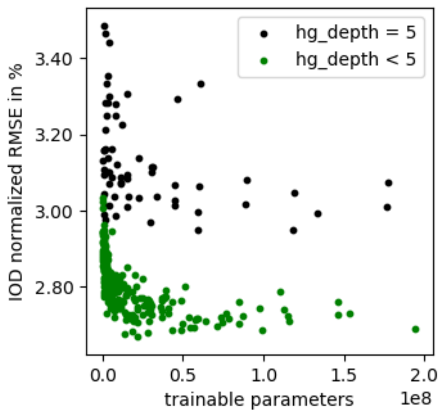
The previous sections have shown trends for the error rate depending on different HG architecture parameters. However, these sections only examined the effect of one single parameter at a time. By looking at the number of network parameters, we can see the effects of increasing the network capacity regardless of one specific parameter. For example, a stack of only one HG can still have a high capacity when the number of features or residual modules is high.



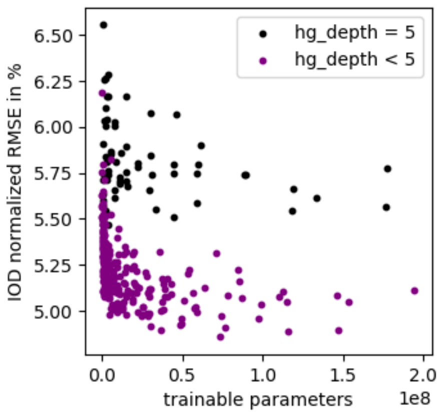
(a) Category: easy (49)



(b) Category: difficult (49)



(c) Category: easy (68)



(d) Category: difficult (68)

Figure 4.8: In (a)-(d) there is one point for each of the 640 models. The figures show the relationship between the number of trainable network parameters and the error. The x-axis has a scale factor of $1e8$. The red, blue, green and pink points show models with $hg\_depth < 5$ and the black points show models with $hg\_depth = 5$.

Figure 4.8 shows the relationship between the number of parameters and the error rate. In contrast to the previous sections, the plots show all 640 models at the same time rather than only the best model for one parameter choice. One can see that networks with a depth of 5 generally perform much worse than others, regardless of the category. This observation is consistent with Section 4.5.3.4 where the effect of the HG depth was studied.

This confirms that downsampling until the feature maps are only 1×1 px small hurts the performance. Therefore we restrict all following experiments to a maximum depth of 4.

Another observation is the trend that more trainable parameters lead to a lower error. However, not in every category the model with the most parameters has the lowest error, see for example Figure 4.8b. A possible explanation is that models tend to overfit when there is too much capacity.

Nevertheless, the trend shown in Figure 4.8 explains the low variance between different architecture choices shown in Section 4.5.3.1, 4.5.3.2, 4.5.3.3, 4.5.3.4 and 4.5.3.5. For example, picking a lower number of features in a residual module can be compensated to a certain degree by using more residual modules.

4.5.4 Effect of training and regression hyper-parameters

Besides the architecture of the stacked HG there are other factors that can have a big impact on the prediction accuracy. A detailed evaluation of many hyper-parameters that define how the network is trained and coordinates are regressed is given in this section.

Section 4.5.4.1 contains a comparison of different loss functions. Whether or not loss normalization can improve the results is examined in Section 4.5.4.2. Methods to convert heatmaps into numerical coordinates are compared in Section 4.5.4.3. In Section 4.5.4.4 the effect of heatmap regularization is investigated. Next, Section 4.5.4.5 covers the effect of using rotation augmentation during training. Finally, an additional loss on inter-landmark distances is evaluated in Section 4.5.4.6.

4.5.4.1 Loss function

In this section, we examine the effect of the loss function that is used to train the stacked HG. The goal is to validate if the Wing loss (Section 3.4.3) helps training more accurate models compared to the L1 and L2 losses.

After the initial grid search (Section 4.5.3) we analyzed the parameters of the three best models for each category and picked the smallest set of parameter choices that covered at least one model in each top 3. Due to resource and time restrictions it was not possible to train all combinations of the parameters of the best models. The chosen parameters are:

- **n_lm**: 49, 68
- **n_hgs**: 3, 4
- **n_res_modules**: 3
- **n_features**: 256, 512
- **hg_depth**: 2, 4

These parameters are combined with the loss function hyper-parameters:

- **loss_function**: L1, L2, wing_5 (Wing loss with $w = 5, \epsilon = 0.5$), wing_10 (Wing loss with $w = 10, \epsilon = 1.0$)
- **normalize_loss**: True, False (see Section 4.5.4.2 for an explanation and evaluation)

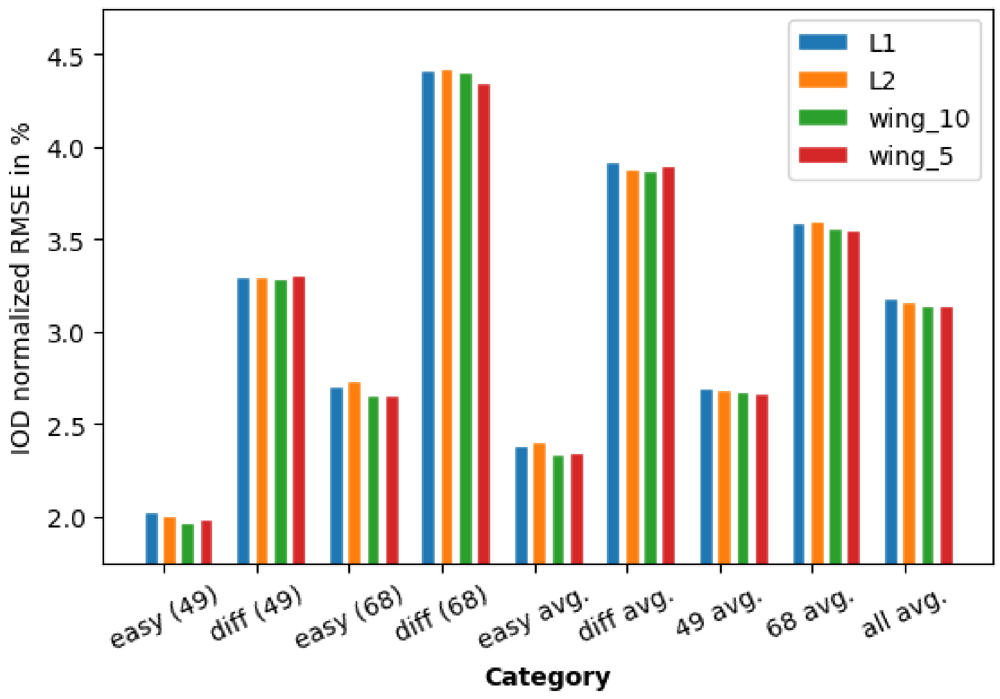


Figure 4.9: The lowest IOD normalized RMSE for each category and each loss function. Wing loss has the lowest error rates in all categories.

There are 128 possible parameter combinations. We decided to use wing_5 and wing_10 because they correspond to a function with a smooth transition between the logarithmic and linear part, see Figure 3.10.

The lowest IOD normalized RMSE for each of the loss functions is shown for each category in Figure 4.9. For a complete analysis we provide the results not only for the main categories *easy (49)*, *difficult (49)*, *easy (68)* and *difficult (68)*, but also the average error for both *easy* and *difficult*, *49* and *68* and the overall average error.

Wing loss wing_10 achieves the lowest error in *easy (49)*, *difficult (49)*, *easy (68)*, *easy*, *difficult* and in the overall average. Wing loss wing_5 achieves the lowest error in *difficult (68)*, *average 49* and *average 68*. Neither L1 nor L2 have the lowest error in one category.

Therefore we conclude that using the Wing loss is beneficial when training a stacked HG for facial landmark detection. The difference between the two tested Wing loss parameterizations is small in most categories. However, since wing_10 is the best loss function in 6 categories and wing_5 only in 3 categories, we use wing_10 for the following experiments.

Intermediate supervision

We also conducted an experiment to examine the effect of using intermediate supervision. To this end, we applied the same loss function to all heatmaps that the HGs in the stacked HG predict instead of only to heatmaps of the last HG. Intermediate supervision did neither help nor hurt the prediction accuracy significantly. This suggests that the architecture of the stacked HG already has a structure that enables efficient gradient flow. Various shortcut branches within each HG (Section 3.4.1.2 and between HGs (Section 3.4.1.3) allow gradients to flow from the final prediction back to the image without the vanishing gradient problem, making intermediate supervision obsolete.

4.5.4.2 Loss normalization

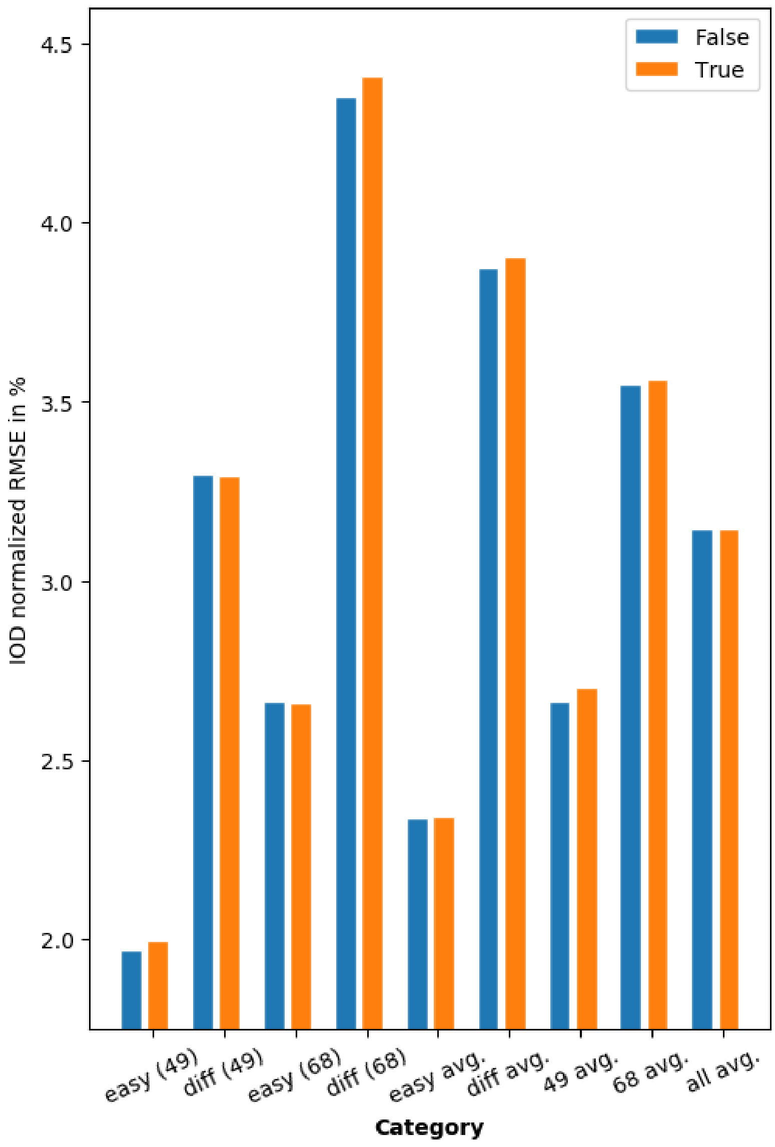


Figure 4.10: The blue bars show the error rates for each category without loss normalization and the orange bars with loss normalization. In many categories, normalizing the loss hurts the performance.

When the training progresses and the predictions get more accurate, the gradients decrease. The consequence is that the network weights receive smaller updates in later epochs. To avoid small gradients, we implement loss normalization to ensure that the loss is always 1. This is done by dividing the loss by its scalar value: $loss' = \frac{loss}{value(loss)}$. In the case of PyTorch [51] the loss is divided by its detached version because otherwise the gradient would be 0.

We investigate whether or not loss normalization can help achieving lower error rates. The grid search performed to determine the best loss function already includes the *normalize_loss* parameter as shown in Section 4.5.4.1. The results in this section are obtained by grouping the previous grid search output by the loss normalization value rather than the loss function.
The lowest error rates for all categories are shown in Figure 4.10. We found that loss normalization does not help to get significantly more accurate predictions and in some categories even leads to worse models. Therefore we disable loss normalization for all following experiments.

4.5.4.3 Regression method

In this section regression via DSNT is compared to a fully-connected regressor as introduced in Section 3.4.2. We have trained and evaluated 8 models for all combinations of these hyper-parameter:

- **n_lm**: 49,68
- **n_hgs**: 3
- **n_res_modules**: 3
- **hg_depth**: 2,4
- **regression**: DSNT, linear

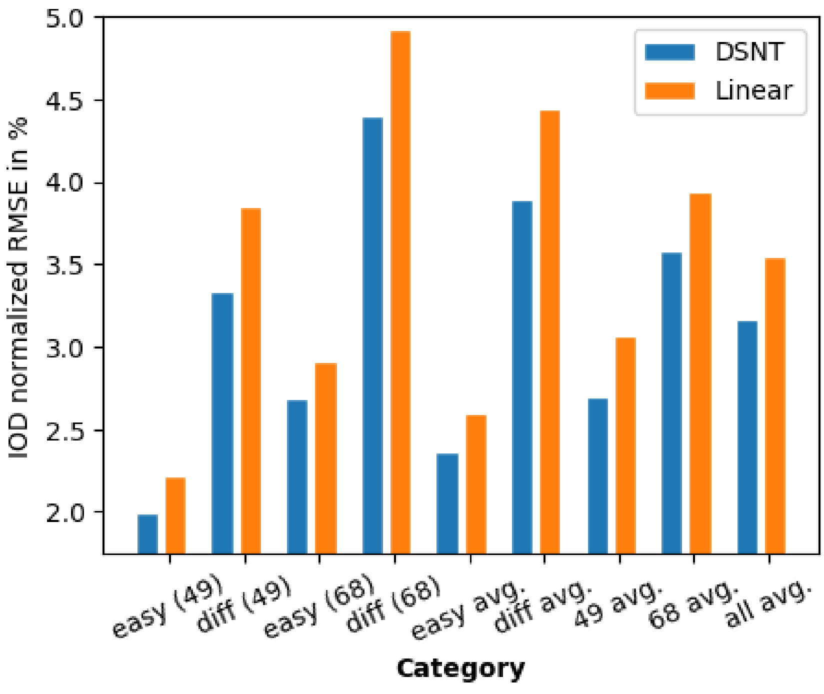
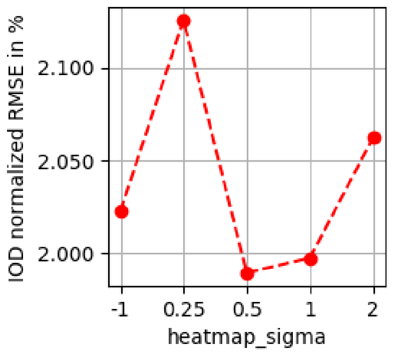


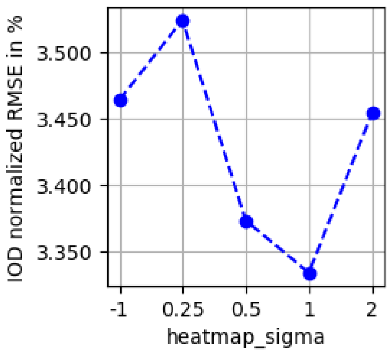
Figure 4.11: Error achieved by DSNT and linear regression for each category.

All models were regularized using the Jensen-Shannon divergence with a standard deviation of 1.0 (other values are evaluated in the next section). The performance difference between using fully-connected regression and the DSNT layer is shown in Figure 4.11. The DSNT layer clearly outperforms the linear regression in all four categories and also in all averages. Therefore, DSNT is used for all further experiments.

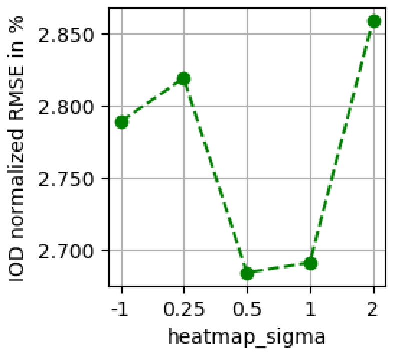
4.5.4.4 Heatmap regularization using Jensen-Shannon divergence



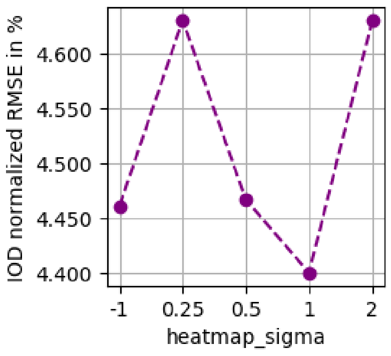
(a) Category: easy (49)



(b) Category: difficult (49)



(c) Category: easy (68)



(d) Category: difficult (68)

Figure 4.12: The best error for each variance value for the Gaussian used in the Jensen-Shannon heatmap regularization. -1 means that no regularization was used. A variance of 0.5 works best for the easy samples (a,c) and 1 for the difficult samples (b,d).

Heatmap regularization using the Jensen-Shannon divergence is introduced in Section 3.4.2. So far all experiments were done using Jensen-Shannon regularization with a Gaussian variance $\sigma = 1.0$. Now we compare the model performance using different variances. To analyze the general benefit of the regularization, we also train models where it is disabled.

These hyper-parameters were tested in all combinations (20 models in total):

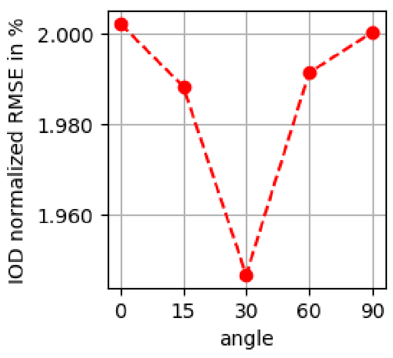
- **n_lm**: 49,68
- **n_hgs**: 3
- **n_res_modules**: 3
- **hg_depth**: 2,4

- **heatmap_sigma**: -1 (disabled), 0.25, 0.5, 1.0, 2.0

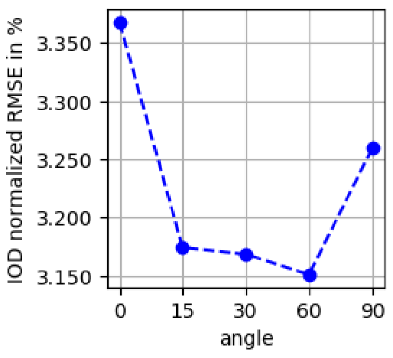
Both the errors of models trained without regularization and the ones with different variance values are shown in Figure 4.12. The easy samples are best predicted by models that were trained with $\sigma = 0.5$ For these categories, $\sigma = 1.0$ also works well with just a very small drop in accuracy. At the same time, $\sigma = 1.0$ is the optimal value for the difficult categories. In comparison, disabling the heatmap regularization ($\sigma = -1$) leads to higher error rates in all categories.

The conclusion from this experiment is that heatmap regularization helps and $\sigma = 1.0$ is a good value. This value used for all previous experiments and will be used for all further experiments as well.

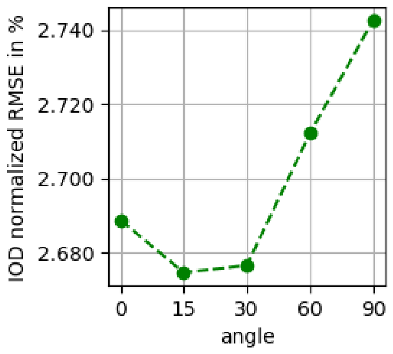
4.5.4.5 Rotation augmentation



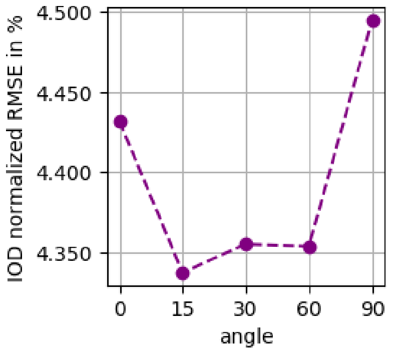
(a) Category: easy (49)



(b) Category: difficult (49)



(c) Category: easy (68)



(d) Category: difficult (68)

Figure 4.13: Error rates for each category depending on maximum angle used for rotation augmentation.

All experiments so far were done without using random rotations as data augmentation. In this experiment we test if rotation augmentation can improve the error rates and in which categories. We run a grid search using these parameters, totaling 40 combinations:

- **augment_rotation** α: 0, 15, 30, 60, 90 degrees
- **n_lm**: 49,68

- **n_hgs**: 3,4
- **n_res_modules**: 3
- **n_features**: 512
- **regression**: DSNT
- **loss_function**: wing_10

The *augment_rotation* parameter specifies the maximal magnitude of the angle α that is used to randomly rotate the faces either left or right in each epoch. The specific angle is randomly computed independently for each sample in each epoch to ensure a large variety of rotations.

Figure 4.13 shows that random rotations with a maximum angle $\alpha = 90°$ are too much to train a model with low error rates. Rotating a face by such a large angle flips it on one side so that the eyes are in one vertical instead of horizontal line. If the original face was already slightly tilted (due to head pose), it can be rotated so far that the landmark labels get wrong. This is because the annotation scheme has fixed indices for example for the left eye. If the left eye becomes the right eye because of the random rotation, the indices would also have to be switched correctly. For this purpose, the rotation of the face in the original image would have to be determined. However, since faces that are rotated by such a large angle do not occur in the test sets, we did not further experiment with this.

In all four categories, $\alpha = 30°$ corresponds to models with low errors. In the 68 landmark categories $\alpha = 15°$ even works slightly better and the *difficult (49)* category benefits from $\alpha = 60°$. All categories have in common that using $\alpha = 15°$ or $\alpha = 30°$ leads to lower errors than using $\alpha = 0°$ or $\alpha = 90°$. Since $\alpha = 30°$ is a good choice for all categories, we choose this value for the following experiments.

4.5.4.6 Loss on inter-landmark distances

In all previous experiments the loss between the predicted location and the ground-truth location was computed for each landmark independently. The spatial relationship between the different landmarks was not directly included in the loss.

In this section, we explore the effects of an additional loss on the distance between all pairs of landmarks. It is based on the predicted locations of the landmarks p_i, p_j and the ground-truth locations g_i, g_j. We compute the distance between all pairs of landmarks $i, j \in \{1...n_{lm}\}$: $d^p_{ij} = p_i - p_j$ and $d^g_{ij} = g_i - g_j$. d^p and d^g are vectors that contain the signed x and y distance between two landmarks. We use the L1 loss between these distance vectors:

$$E_{dist}(p, g) = \sum_{i=1}^{n_{lm}} \sum_{j=1}^{n_{lm}} |d^p_{ij} - d^g_{ij}| \tag{4.2}$$

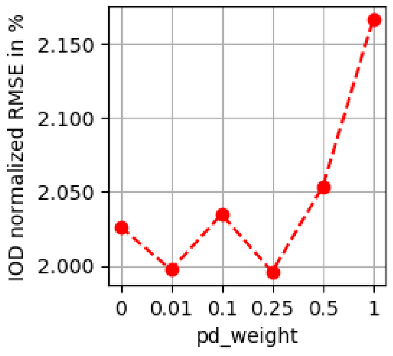
This loss is added to the normal loss E_{coord} that was used in the previous experiments and minimized through back-propagation. The intuition behind this loss is that there are some constraints on the possible locations of landmarks depending on the location of other landmarks. When the predictions violate these constraints, this error can be directly used to back-propagate and update the model so that they are respected in further predictions.

As E_{dist} uses L1 loss instead of Wing loss and the input to the loss function are distances rather than points, the magnitude if this error is expected to be different from the actual prediction error. Therefore a new hyper-parameter λ_{dist} is introduced and the final error is computed as follows:

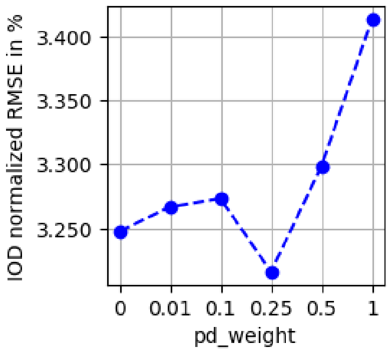
$$E = E_{coord} + \lambda_{dist} * E_{dist} \tag{4.3}$$

This hyper-parameter was subject to a grid search with the following parameters:

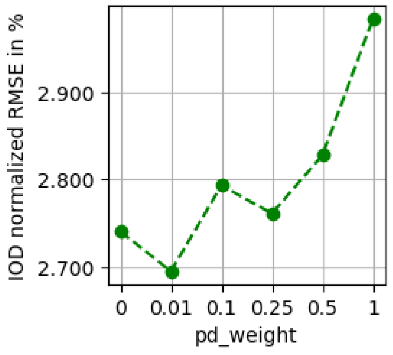
- **n_lm**: 68
- **n_hgs**: 3
- **n_res_modules**: 3
- **n_features**: 512
- **loss_function**: wing_10
- **normalize_loss**: False
- **pd_weight** λ_{dist}: 0, 0.01, 0.1, 0.25, 0.5, 1.0



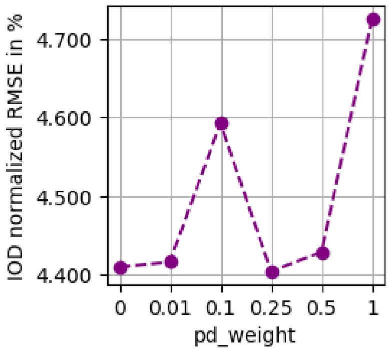
(a) Category: easy (49)



(b) Category: difficult (49)



(c) Category: easy (68)



(d) Category: difficult (68)

Figure 4.14: The lowest error for each weight λ_{dist} (pd_weight) used by the distance loss for each category.

The influence of the distance loss on the final prediction error is depicted in Figure 4.14. The λ_{dist} values are shown as *pd_weight* in the plots. $\lambda_{dist} = 0$ means that the distance loss is not used.

In all categories, using the distance loss lowers the error a little. While $\lambda_{dist} = 0.25$ is the best choice for *easy (49)*, *difficult (49)* and *difficult (68)*, using this weight for *easy (68)* is worse than not using the distance loss at all. $\lambda_{dist} = 0.01$ is the best choice for *easy (68)* and works well for *easy (49)* and *difficult (68)*, however it is worse than not using the loss for *difficult (49)*. If only 49 landmarks are required by the application, $\lambda_{dist} = 0.25$ is the best pick and if 68 landmarks are desired, $\lambda_{dist} = 0.01$ the best option.

Because it is not clear which value to choose, the error for the model that had the lowest average error on all four categories is plotted in Figure 4.15. $\lambda_{dist} = 0.01$ is the optimal if the average error for all categories should be as low as possible.

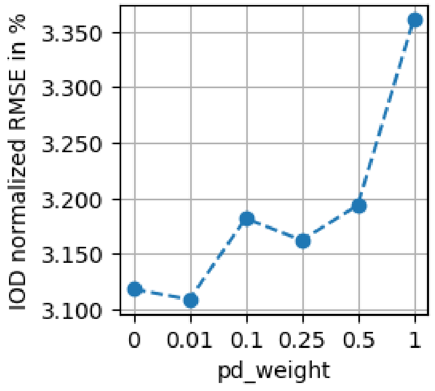


Figure 4.15: The lowest error for each weight λ_{dist} (pd_weight) used by the distance loss is shown for the average of all four categories.

4.5.5 Comparison with state-of-the-art methods

Now that most hyper-parameters for the stacked HG have been analyzed, we compare its performance to current state-of-the-art solutions on two datasets: The 300-W [55, 56, 57] test set and the Menpo [82] train set. The datasets are presented in Section 4.1. For 300-W we use the IOD normalized RMSE as error metric and for Menpo the RMSE normalized by the bounding box size (mean of the width and height of the bounding box). Both metrics are described in Section 4.2.

An introduction to the baseline methods used to compare our method to can be found in Section 2.3. The median errors of the baselines and our models are listed in Table 4.1. The median of the errors is used instead of the mean because the mean is more sensitive to outliers. The lowest error of the baselines in each category is written cursively. The ticks in the table mean that the error is lower than all shown baselines in this column. The lowest value in each column is printed bold.

The results of our models are split into four sections. The first section contains the errors of the models that performed best in one of the categories. The following sections report the errors of models that were the best in the average categories as described in Section 4.3.

Dataset / Method	Helen [36]+LFPW [5]		iBUG [56]		Menpo [82]	
	easy (49)	easy (68)	diff (49)	diff (68)	49	68
CLNF [3] ★ ◆	2.51	3.47	4.93	6.37	2.10	2.66
SDM [74] ★ ◆	3.31	-	10.73	-	2.54	-
CFAN [83] ★ ◆	-	-	6.99	8.38	2.34	2.87
DRMF [2] ★ ◆	4.22	4.97	8.64	10.36	3.44	-
CFSS [86] ★ ◆	2.46	3.20	4.49	5.97	1.90	2.32
TCDCN [85] ★ ◆	3.32	4.11	5.56	6.87	2.81	3.32
3DDFA [87] ★ ◆	5.17	7.27	8.34	12.31	3.59	4.51
LAB [73]	-	2.98	-	5.19	-	-
DU-Net [65]	-	2.82	-	5.07	-	-
YANG-HG [76]	-	-	-	4.90	-	-
PO-CR [67] ★ ◆	2.67	-	*3.33*	-	2.03	-
AWL [72]	-	*2.72*	-	*4.52*	-	-
CE-CLM [79]	2.30	3.15	3.86	5.31	*1.74*	2.23
FC-LGCN [44]	*2.21*	2.86	4.18	5.29	1.79	*2.14*
A_1: *easy 49*	**1.95 ✔**	-	3.35	-	**1.63 ✔**	-
A_2: *easy 68*	2.03 ✔	**2.66 ✔**	3.50	4.53	1.66 ✔	1.95 ✔
A_3: *difficult 49*	2.05 ✔	-	**3.15 ✔**	-	1.69 ✔	-
A_4: *difficult 68*	2.05 ✔	2.74	3.38	**4.34 ✔**	1.66 ✔	**1.94 ✔**
B_1: *average easy*	2.01 ✔	**2.66 ✔**	3.37	4.59	1.67 ✔	1.95 ✔
B_2: *average difficult*	2.08 ✔	2.80	3.20 ✔	4.35 ✔	1.69 ✔	2.03 ✔
C_1: *average 49*	2.02 ✔	-	3.16 ✔	-	1.66 ✔	-
C_2: *average 68*	2.05 ✔	2.74	3.38	**4.34 ✔**	1.66 ✔	**1.94 ✔**
D: *average all*	2.00 ✔	2.72 (✔)	3.19 ✔	4.41 ✔	1.64 ✔	1.99 ✔

Table 4.1: Comparison of our models to state-of-the-art baselines. The error value for Helen [36], LFPW [5] and iBUG [56] is the median IOD normalized RMSE. The error value for Menpo [82] is the median of the bounding box size normalized RMSE. All values are multiplied by 100 for better readability. Bold numbers are the lowest in one category. ✔ means that the value is lower than the baselines. (✔) means that it is as low as the best baseline. ★: Numbers taken from Zadeh et al. [79]. ◆: Numbers taken from Merget et al. [44]. The other numbers are taken from the original papers.

In section A we picked the best model for each of the four categories on 300-W: easy with 49 landmarks (A_1) easy with 68 landmarks (A_2), difficult with 49 landmarks (A_3) and difficult with 68 landmarks (A_4). Rather than only showing the performance of each model in the category where it has the lowest error, we also report its errors for the other categories. This reveals that a model that has the lowest error in one category has a higher error in the other categories when compared to the best ones in these categories. For example, model A_1 has an error of 1.95 on *easy (49)* but an error of 3.35 on *difficult (49)* while model A_3 has only an error of 3.15 on *difficult (49)* but 2.05 on *easy (49)*. In scenarios where it is known beforehand if a sample is easy or difficult this would be valid as one could just use the respective model. Yet this is an unrealistic scenario unless it is guaranteed that only near-frontal images without strong occlusions are used in the case of model A_1 or only images with stronger head poses or occlusions in A_3. Although it is unrealistic to pick the best suited model based on the difficulty of a sample, we show the best average models on both easy (B_1) and difficult samples (B_2) for completeness.

In many applications, a model that performs well on both *easy* and *difficult* samples for 49 or 68 landmarks is desired. Therefore, Section C shows the model that has the lowest average error on *easy* and *difficult* samples for 49 landmarks (C_1) or 68 landmarks (C_2). The assumption here is that a model can be picked based on what the detected facial landmarks will be used for, i.e. if the application requires 49 or 68 landmarks. This is more reasonable since for most applications this is known beforehand. The best *average 49* model is better than all baselines for 49 landmarks. The best *average 68* model is better than all landmarks on *difficult (68)* and *easy (49)* but slightly worse on *easy (68)* and *difficult (49)*.

We furthermore investigated which model has the lowest average error on all four categories. This is shown in Section D. This model achieves an error that is lower or equal to the best baseline in all categories. Thus, our stacked HG D achieves state-of-the-art results on the 300-W dataset.

Cross-dataset evaluation on Menpo

A good indicator for the generalization capabilities of a model is the prediction accuracy on a dataset that was never seen during training, neither to compute the gradients, to update the learning rate or to perform early stopping. More importantly, the performance on this dataset must also not be used to select the best model among a few candidates. For these reasons we also report the errors on Menpo [82] in Table 4.1. Menpo was never used during training and the models listed in the table were not selected based on the Menpo errors. The models in the table were selected based on the error on the 300-W test set, so this can be seen as the validation set while Menpo is the real test set. Note that the error values reported for Menpo are the RMSE normalized by the mean of width and height of the bounding box.

All nine models beat the state-of-the-art baselines on Menpo for both 49 and 68 landmarks. The best predictions for 49 landmarks were obtained using A_1, which is also the best model for *easy (49)*. 68 landmarks on Menpo were best located with both A_4 (best in *difficult (68)*) and C_2 (best average for 68 landmarks). However, all of our models have errors which range from 1.53 to 1.69 for 49 landmarks and 1.94 and 2.03 for difficult landmarks and therefore are all good options.

General observations for all datasets

The errors for 49 landmarks are lower than the errors for 68 landmarks in *easy*, *difficult* and *Menpo* for all of our models and all baselines. This is not due to the fact that more landmarks are used to compute the *68 landmarks* error, because the error is normalized by the number of landmarks and thus represents the average error. This indicates that predicting landmarks on the outline is generally more challenging than inner face landmarks. A reason for that is the ambiguity of the landmark appearance on the outline. Inner landmarks like the corner of the eye or the mouth are well defined because they lie in areas with an unique texture. However, landmarks on the outline can not be located this accurately because multiple points on the outline have a very similar appearance. Thus, both human annotators and landmark predicting models make bigger mistakes for landmarks on the outline than compared to the inner face.

Hyper-parameters of the best stacked HGs

The hyper-parameters that were used to train the models from Table 4.1 are listed in Table 4.2. All hyper-parameters that are not listed are the same for all the models: DSNT for the coordinate regression, Gaussian $\sigma = 1.0$ for the heatmap regularization, Wing loss (wing_10) as the loss function, no loss normalization, no distance loss and no intermediate loss. The batch size is 32 for all models, the optimizer is Adam [33] with learning rate 0.001 and a learning rate scheduler with patience 15 and a decay factor of 0.5.

Model	n_lm	n_hgs	n_res_mod.	n_feat.	hg_depth	rot. α	epochs
A_1: *easy 49*	49	3	3	512	2	30°	118
A_2: *easy 68*	68	3	3	512	2	0°	95
A_3: *difficult 49*	49	3	3	512	2	60°	162
A_4: *difficult 68*	68	4	3	512	4	15°	104
B_1: *average easy*	68	4	3	512	2	0°	83
B_2: *average difficult*	68	4	3	512	4	60°	157
C_1: *average 49*	49	3	3	512	2	60°	161
C_2: *average 68*	68	4	3	512	4	15°	104
D: *average all*	68	4	3	512	4	60°	179

Table 4.2: Hyper-parameter values used to train the models in Table 4.1. The epoch with the lowest error is shown in the last column.

It can be observed that the best model on the *difficult (49)* category (A_3) was trained on 49 landmarks although a finding of Section 4.5.3.5 was that training a model on 49 instead of 68 landmarks does only help with easy samples but hurts with difficult samples. However, A_3 was also trained with 60°random rotations while all models in Section 4.5.3.5 were trained without rotations. This demonstrates the complex interplay between different hyper-parameters. Training on 49 landmarks becomes beneficial for difficult samples when using rotation augmentation while it increases the error when the augmentation is not used.

The same observation can also be made for other hyper-parameters, e.g. the number of features. All of our best models use 512 features while we found that 256 is a better choice for *easy* images in section Section 4.5.3.3. The combination with other hyper-parameters makes 512 a better choice for our best models.

4.5.6 Unsuccessful experiments with Spatial Transformer Networks

During the implementation of the stacked HG we investigated if Spatial Transformer Networks (STNs) [32] can help to improve the accuracy. STNs can be used as part of a Convolutional Neural Network (CNN) to detect if the input image was rotated, scaled or translated. The STN regresses a set of affine parameters that can be used to revert the rotation, scaling or translation before the main network (e.g. a (stacked) HG) analyzes the image. By doing this the main network does not have to learn features for transformed images but only for normalized images, i.e. faces that have no rotation, were not scaled and not translated. This results in less network parameters and thus to faster training and inference. It can also improve accuracy for some tasks [32]. A STN can be seen as a pre-processing step that transforms objects in an image to a neutral pose. The STN is trained jointly with the main network.

We conducted a few experiments to find out if STNs can also help in the task of facial landmark detection. During training we augmented the samples with a random rotation augmentation between −45 and 45 degrees. The first try was to have a single STN before the first HG. Unfortunately the STN always learned to predict the identity transformation (no scale, translation and rotation) after a few epochs, regardless of the complexity of the STN, causing the HG to handle transformations itself. This lead to the assumption that the HG is learning faster than the STN and that the STN thus learns to do nothing. To help the STN learning faster the learning rate was adjusted. However, setting the STN learning rate higher or lower than the HG learning rate did not change the behaviour.

The next assumption was that the STN is not able to detect if a human face was transformed since the STN is only a small CNN with a small receptive field. Therefore the next experiment used a separate HG before the STN and a stack of HGs after the STN. The first HG should be able to deliver useful information to the STN which could then revert the transformation and the stack of HGs could be trained on neutral images. As the STN again only learned the identity transformation, it seemed that the STN is simply not needed because the HG is already powerful enough to handle transformations.

Hence, the final architecture implemented in this master's thesis does not include a STN, but still is trained using rotation augmentation to allow the HG to handle rotated faces.

4.6 Point Distribution Model

In this section the PDM as presented in Section 3.5 is evaluated. The PDM is trained on the landmark annotations of the training split of 300-W [55, 56, 57]. The images are ignored and only the landmark coordinates are used since the PDM does not depend on appearance information. The final performance of the pipeline consisting of stacked HG and PDM is reported on both the 300-W test set and the Menpo [82] training set as introduced in Section 4.1.

A list of all relevant hyper-parameters for both the PDM training and inference is presented in Section 4.6.1. The optimal latent vector size is discussed in Section 4.6.2. Various decoder architectures are compared in Section 4.6.3. Different initialization strategies for the latent vectors and transformation parameters are assessed in Section 4.6.4. The best performing stacked HGs are compared to the best performing HG-PDM pipelines in Section 4.6.5. Finally, the best combinations of stacked HG and PDM with the lowest evaluation errors are compared to state-of-the-art methods in Section 4.6.6, similarly like presented for the stacked HG in Section 4.5.5.

4.6.1 Hyper-parameter overview

Compared to the stacked HG, the PDM has relatively few hyper-parameters that are directly related to the architecture:

Decoder architecture

- **n_lm**: 49 (without outline) or 68 (with outline), see Figure 4.2 for a visualization
- **decoder_layers**: The architecture of the decoder network. The size of the first layer determines the size of the latent vector.

The input to the decoder network is the latent vector. Thus, there is no separate hyper-parameter that specifies the size of the latent vector. Instead, the number of neurons in the first layer of the decoder network determines the number of elements in the latent vector. The last layer of the decoder has $n_{lm} * 3$ neurons because it outputs the 3D coordinates of all landmarks. The minimal decoder consists of just the input layer and the output layer that maps the latent vector to the 3D coordinates. However, to be able to model complex non-linear mappings, deeper decoder networks can be used. Different decoder architectures are compared in 4.6.3. The choices for *n_lm* and the decoder architecture that were used for the models with the lowest errors are listed in Section 4.6.5. The rest of the hyper-parameters configures the training and inference process.

Hyper-parameters used in both training and inference

- **lr_shape**: Learning rate for the latent vectors z and the transformation parameters t
- **loss_function**: Loss function for PDM training and inference

Wing loss [24] (Section 3.4.3) is used with the same parameters as to train the stacked HG. The learning rate for the shape parameters (latent vector and transformation parameter) was subject to a grid search. The value that was used for the best models is reported in Section 4.6.5.

Training hyper-parameters

- **epochs_train**: Number of epochs to train the PDM decoder
- **lr_net**: Learning rate for the decoder network weights
- **batch_size**: Number of samples per mini-batch

We empirically found a batch size of 64 to work well. The optimizer used for the PDM decoder is Adam [33] with a 0.001 as learning rate. Which number of epochs worked well in our experiments is shown in Section 4.6.5.

Inference hyper-parameters

- **epochs_inference**: Number of epochs for the PDM inference
- **confidence_parameters [a,b]**: Hyper-parameters for the conversion between heatmap variance and confidence (see Section 3.5.6)
- **variance_threshold** Only apply the PDM to samples that have at least one landmark with a higher heatmap variance than the threshold. If the threshold is set to 0.0, the PDM is applied to all samples.

- **use_initializer**: If set to true, an initializer network determines starting values for the latent vectors z and transformation parameters t. Otherwise the latent vectors are initialized randomly and the transformation parameters are set to the identity transformation.
- **lr_scheduler**: Whether to use a sample-specific learning rate scheduler to reduce the learning rate (lr_shape) on a plateau of the reconstruction error. If yes, both the patience and decay factor are specified.

The values for these hyper-parameters are reported for the best models in Section 4.6.5.

During inference, the batch size is set to the highest possible value that fits on one GPU. This is possible because all samples are independent and can hence be processed in parallel. Since the PDM inference runs iteratively this saves time when processing a large number of samples. As the PDM decoder network is small, the whole test data in this work set fits on the GPU in one batch.

4.6.2 Latent vector size

As discussed in Section 3.5.2, using a too large number of dimensions for the latent vectors allows the network to learn a mapping that directly stores the landmarks in the vector without learning their correlations. In this case the PDM is able to find a latent representation that gets decoded to the original shape almost perfectly, i.e. the reconstruction error will be very small. In this section we analyze how the number of dimensions relates to the reconstruction error. The input shape can be interpreted as the ground-truth and the reconstructed shape as the prediction which allows to use the same metric that was used for the stacked HG: IOD normalized RMSE.

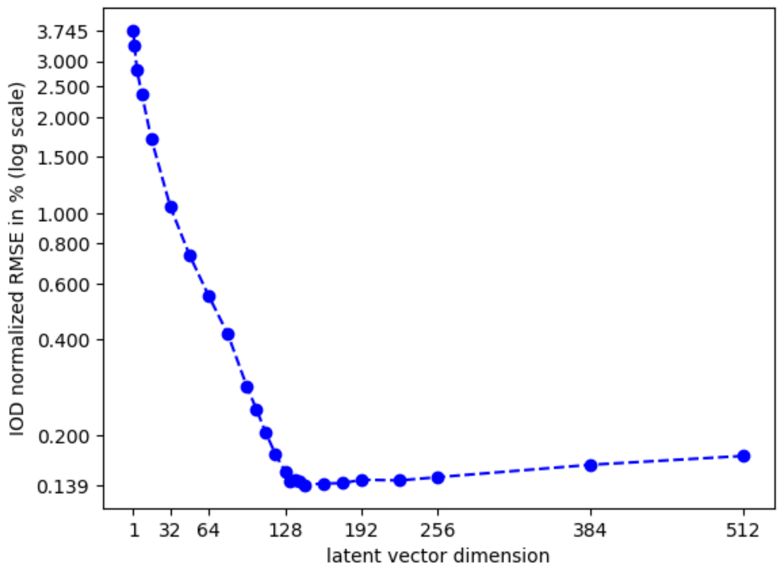


Figure 4.16: Reconstruction error on the test set for 68 landmarks depending on latent vector dimensionality. To better see the trends, the y axis is log_{10}-scaled.

Figure 4.16 shows the reconstruction error of a PDM that was trained with a certain latent vector size. The decoders trained for this experiment all consist of only two layers: The input layer (which size is varied in this experiment) and the output layer that outputs 68 3D coordinates. The reconstruction error is determined by inferring latent vectors and transformation parameters for the 300-W test samples (*easy* and *difficult* combined), decoding and projecting them to 2D. The IOD normalized RMSE between the original test samples and their reconstructed shapes is the reconstruction error.

It can be observed that the error reduces with the increasing vector size until 144 dimensions are reached. At this point the reconstruction error is about 0.139 which means that the mean error among samples is only 0.139% of the IOD. In other words, the vector can encode the input shape almost perfectly. With further increasing vector sizes the error slowly grows again. This is most likely due to overfitting of the decoder. The error for 136 dimensions is 0.144 and only slightly higher than for 144 dimensions. In theory the expectation would be that the error is the smallest for 136 dimensions because that is enough to encode 68 2D coordinates without a loss. However, this might require longer training.

For the use case of fixing wrong predictions it is not desired to be able to encode the shapes without a loss as this does not force the PDM to learn a model of the human face and its variations. Therefore the latent vector dimension should be smaller than 136 dimensions for the case of 68 landmarks and 98 dimensions for the case of 49 landmarks.

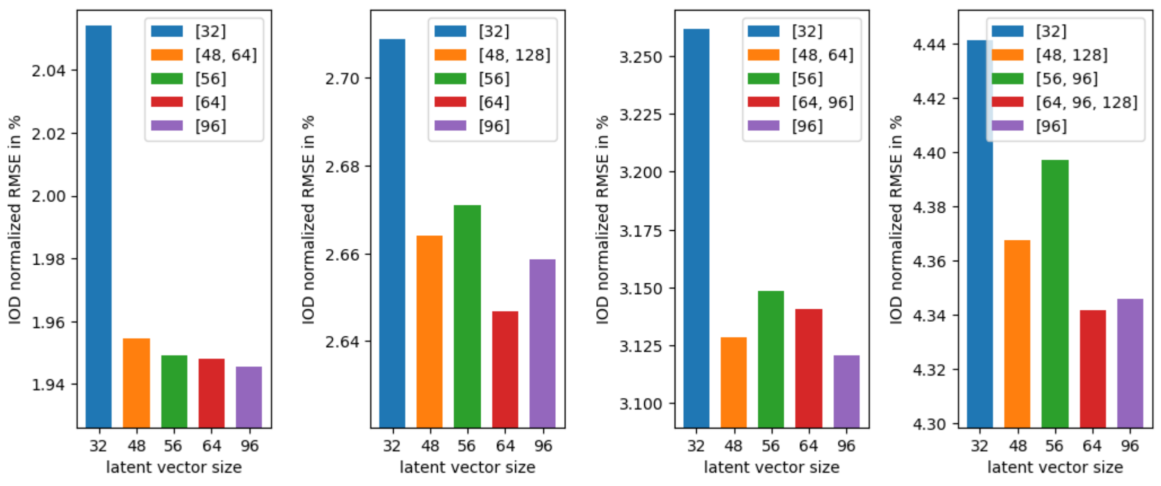
4.6.3 Decoder architecture

In the previous experiment only decoders with two layers were compared. This effectively means that these decoders learned a linear mapping between the latent vector and the 3D coordinates. To model complex variations of a human face it might be helpful to use more layers which allow to learn a non-linear mapping. In this section many different PDM decoder architectures are compared. In contrast to the previous section, we focus on the error rate the PDM achieves when it is run on the output of a stacked HG. The error that was achieved by a PDM with a particular decoder architecture is shown in Figure 4.17.

As it is difficult to visualize the errors of different decoder architectures with different numbers of layers we grouped the results by the latent vector size (which is the size of the first layer) and report the lowest error for each group. The labels in the plots indicate which exact architecture achieved the lowest error. The label *[48,64]* means that the decoder had an input layer with 48 neurons followed by a hidden layer with 64 neurons and an output layer which has $3 * 68 = 204$ elements (68 3D coordinates). All decoders used in this experiment were trained on 68 landmarks.

A latent vector with 32 dimensions had the highest error for all four categories. Thus, 32 dimensions are too small to model the complex variations seen in facial landmarks. For *easy (68)* 64 neurons followed by the output layer performed best. For both *easy (49)* and *difficult (49)* the architecture *[96]* worked best. For *difficult (68)* layers with 64, 96 and 128 neurons followed by the output layer lead to the lowest error. However, the architecture *[96]* is almost as good as *[64,96,128]*. Consequently, for all four categories an architecture with two layers works well. This indicates that it is sufficient to model the dependencies between the different landmarks linearly.

Unfortunately, it is not possible to fully explore the space of all possible decoder architectures. Arbitrarily deep and wide architectures are possible and is not clear if different



(a) easy (49) (b) easy (68) (c) difficult (49) (d) difficult (68)

Figure 4.17: The lowest IOD normalized RMSE for different decoder architectures grouped by the latent vector size (number of neurons in the first layer).

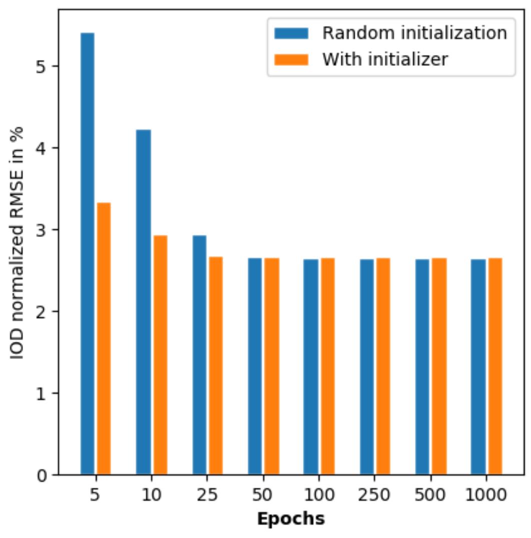
architectures than we tested would work better. For the following experiments, the best performing decoders from this experiment are further analyzed.

4.6.4 Pre-initialization of latent vector

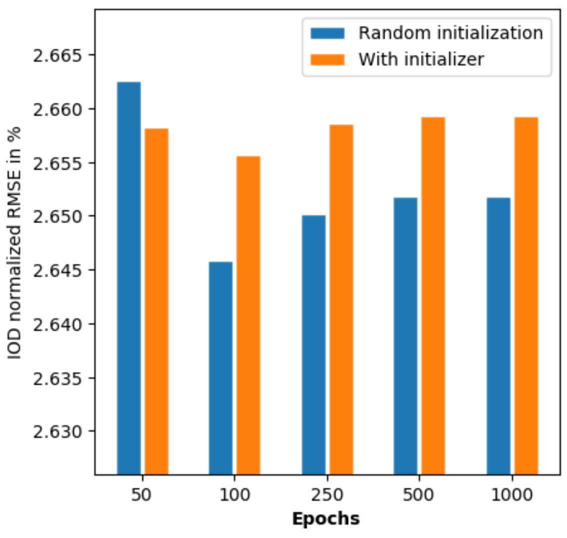
In this section we examine the effectiveness of initializing the latent vector by an initializer as described in Section 3.5.7. For this purpose many different PDM architectures were trained (see Section 4.6.3) and in a grid search the best PDMs were run on the predictions of the stacked HGs that achieved state-of-the-art results (see Section 4.5.5). For each pair different optimizers, schedulers and number of inference epochs were evaluated. Each combination was run both with randomly initialized latent vectors and using the initializer. In total over 11,000 combinations were tested and the results were grouped by the number of test epochs and the type of initialization. For each group the lowest IOD normalized RMSE was determined. This error is not the reconstruction error between the output of the PDM and stacked HG but the error between the ground-truth and the shape that the PDM reconstructed. We show the results for the *easy (68)* and *difficult (68)* categories in Figure 4.18. The 49 landmark categories are omitted because what is described in the following holds for them as well.

There are two plots for each category in Figure 4.18 to be able to observe the error behaviour. The left plots show the errors for PDMs that were run for 5-1000 epochs and the right plots only show the error rates for the range between 50 and 1000 epochs. Yet both the left and right plots are based on the same data. The top row shows the results for *easy (68)* and the bottom row for *difficult (68)*.

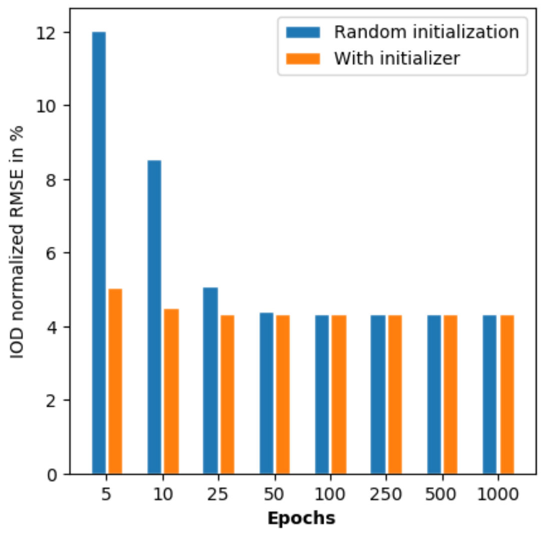
In both categories the errors are very high when the latent vectors and transformation parameters are initialized randomly and the PDM is run for only five epochs. Five epochs are not enough to find latent vectors and transformations that can reconstruct the predictions from the stacked HG. When using the initializer, the error is much smaller even when



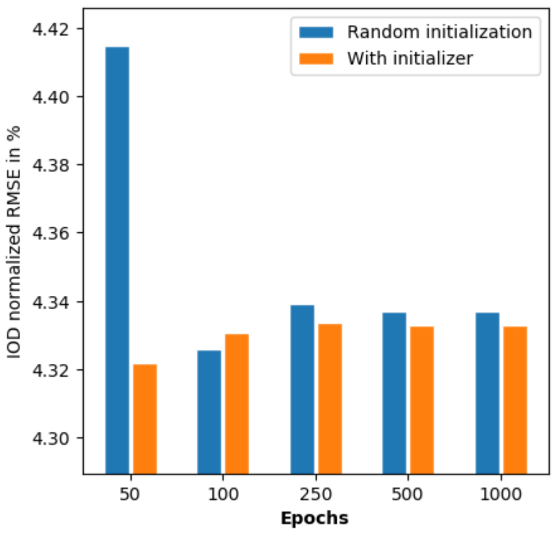
(a) easy (68): 5 - 1000 epochs



(b) easy (68): 50 - 1000 epochs



(c) difficult (68): 5 - 1000 epochs



(d) difficult (68): 50 - 1000 epochs

Figure 4.18: Error depending on the number of PDM inference epochs for both random initialization and using an initializer. Both figures in a row show the error for same category. The right plot is a zoomed version of the left plot that focuses on epoch numbers greater than 50 to better visualize the differences.

only five epochs are used since the inference procedure starts off a better position and is able to faster converge to a more accurate reconstruction. Nevertheless, five epochs are not enough to produce small errors in none of the two categories. The general trend that can be observed from the left plots is that the error decreases as the number of inference epochs increases in both initialization cases. Moreover, the difference between the errors of both initialization methods shrinks with a growing number of inference epochs.

To better see the exact behaviour of errors with a higher number of epochs the right plots are shown. Although the gap between the two initialization methods is small with a number of epochs greater than 100, there is a difference in the error rates. When an initializer is used, the error of *easy (68)* is slightly larger than with random initialization. For *difficult (68)* the initializer leads to a lower error than random initialization. From these observations it can be concluded that the initializer is generally beneficial when fast predictions are desired and the number of epochs is 50 or less. If 100 or more epochs are used, using the initializer leads to worse results on *easy* samples and improves the predictions on *difficult* samples.

Moreover, a general trend can be seen: If too many epochs are used, the error starts to increase again. The reason lies in the nature of the iterative inference process which minimizes the reconstruction loss using back-propagation as described in Section 3.5.5. The reconstruction loss is weighted by the confidence measurement. As long as there are reconstruction errors for landmarks with high confidence, the inference process will be dominated by these until they are reconstructed (almost) exactly. The loss for these landmarks will then be very small and the landmarks with a lower confidence but a higher reconstruction error will dominate the loss. However, since their confidence is small, their predicted locations are likely to be wrong, but the PDM will find a shape that better matches them. In turn, the error for the high confidence landmarks will increase until these dominate the overall error again. Therefore it is crucial to pick a suitable number of test epochs. Reducing the learning rate of a sample when a plateau is reached can also be helpful. An interesting extension would be to implement early stopping that stops the inference as soon as the reconstruction error of high-confidence landmarks falls below a threshold.

4.6.5 Error comparison between Stacked Hourglass Network with and without Point Distribution Model

In this section the best stacked HGs are compared to the best pipelines consisting of a stacked HG and a PDM. Before the error values are presented, the models that lead to these are introduced. The stacked HGs are the ones shown in Section 4.5.5. The landmarks predicted by these HGs were used as the input of the PDMs listed in Table 4.3. The inference hyper-parameters that were used to produce the results for this section are outlined at the end of this section.

PDM	architecture	n_{lm}	$epochs_{train}$
PDM_1	[64, 147]	49	3000
PDM_2	[96, 204]	68	3000
PDM_3	[64, 204]	68	1500
PDM_4	[96, 204]	68	2000

Table 4.3: The PDMs that were used on top of the stacked HGs and have lead to the best predictions. *[a,b]* represents a two-layer network with a and b neurons.

Although we have also evaluated PDMs with deeper architectures, the best results were obtained with two-layer decoder networks. The size of the first layer is the latent dimension and the size of the output layer is $3n_{lm}$. In the following we will refer to a pipeline consisting of a HG and a PDM as HG-PDM. For example, A_2 - PDM_4 is the combination of HG A_2 and PDM PDM_4.

Lowest errors of stacked HGs with and without a PDM

In Table 4.4 the errors on 300-W [55, 56, 57] are shown for each of the categories that were introduced in Section 4.3. The errors of the stacked HG are denoted as *HG* in the table and are the same as in Section 4.5.5. Next to the lowest *HG* errors we report the lowest error that a HG-PDM pipeline could achieve. Analogously to the comparison between stacked HG and state-of-the-art methods in Section 4.5.5, we picked pipelines with the lowest error for each of the four main categories *easy (49)*, *easy (68)*, *difficult (49)*, *difficult (68)* as well as for the averages in *easy*, *difficult*, *49* and *68* landmarks and overall average.

Dataset	Helen [36] + LFPW [5]				iBUG [56]			
Category	**easy (49)**		**easy (68)**		**diff. (49)**		**diff. (68)**	
HG | HG-PDM | Best in	HG	PDM	HG	PDM	HG	PDM	HG	PDM
A_1 | A_1 - PDM_1: *easy (49)*	1.95	$\underline{\mathbf{1.94}}$	-	-	3.35	$\underline{3.33}$	-	-
A_2 | A_2 - PDM_4: *easy (68)*	$\underline{2.03}$	2.04	2.66	$\underline{\mathbf{2.65}}$	3.50	$\underline{3.44}$	$\underline{4.53}$	4.54
A_3 | D - PDM_4: *diff. (49)*	2.05	$\underline{2.00}$	-	2.72	3.15	$\underline{\mathbf{3.11}}$	-	4.41
A_4 | B_2 - PDM_2: *diff. (68)*	$\underline{2.05}$	2.08	$\underline{2.74}$	2.80	3.38	$\underline{3.24}$	4.34	$\underline{\mathbf{4.30}}$
B_1 | B_1 - PDM_3: *avg. easy*	2.01	$\underline{2.00}$	2.66	2.66	$\underline{3.37}$	3.43	$\underline{4.59}$	4.60
B_2 | B_2 - PDM_4: *avg. diff.*	2.08	2.08	2.80	2.80	3.20	$\underline{3.18}$	4.35	$\underline{4.33}$
C_1 | D - PDM_4: *avg. 49*	2.02	$\underline{2.00}$	-	2.72	3.16	$\underline{\mathbf{3.11}}$	-	4.41
C_2 | C_2 - PDM_4: *avg. 68*	2.05	2.05	2.74	2.74	3.38	$\underline{3.36}$	4.34	4.34
D | D - PDM_4: *avg. all*	2.00	2.00	2.72	2.72	3.19	$\underline{\mathbf{3.11}}$	4.41	4.41

Table 4.4: Comparison of the lowest IOD normalized RMSE of HGs with and without a PDM. Results are reported for each category of the 300-W [55, 56, 57] test set. In each category the error of a HG is shown in the *HG* column. The *PDM* columns show the values of a HG-PDM pipeline. Each row shows the performances of the HG and PDM with the lowest error in one (average) category. For example, the models shown in the first row had the lowest errors in *easy (49)* while the ones in the last row had the lowest overall average error. In each category the lower error is underlined. The lowest error in each category is written in bold type.

These models have also been evaluated on Menpo [82]. We do not report the Menpo results in a separate table because the difference between the stacked HG error and the PDM error is less than 0.01 for all categories. The PDM did neither improve nor worsen the predictions from the stacked HG. An explanation for this could be that the stacked HG was already able to predict the samples for Menpo accurately.

Best HG-PDM in a single category

When considering only the performance in one category (first four rows in the table), the HG-PDM pipeline achieves lower errors than a stacked HG alone. For *easy 49* (A_1 - PDM_1) and *easy 68* (A_2 - PDM_4) the HG-PDM error is 0.01 lower than the stacked HG error. For *difficult (49)* and *difficult (68)* the difference is 0.04.

The HG-PDM with the lowest error in *easy 49* is also better than the best stacked HG in *difficult (49)*. There are no values for the categories with 68 landmarks because the models were trained on only 49 landmarks. A_2 - PDM_4 has the lowest error in *easy (68)* and is also better than the HG in *difficult 49* but is worse than the HG in *easy (49)* and *difficult (68)*. *D*-PDM_4 is better than the best HG in the 49 landmark categories. Note that the HG used to compare to *D*-PDM_4 in the *difficult (49)* row is not D but A_3. This is because A_3 had a lower error than D when not combined with a PDM. However, the combination *D*-PDM_4 is better than A_3-PDM_4. The HGs used in the *difficult (68)* row also differ. B_2-PDM_2 is better than the best HG in the difficult categories but worse in the easy ones.

Best HG-PDM in the average categories

The HG-PDM with the lowest average error in the *easy* categories is only able to improve the results from the HG in *easy (49)* by a small margin. In *easy (68)* there is no difference and in *difficult (49)* and *difficult (68)* the results get worse when using the PDM. An explanation is that it is hard to improve predictions for easy samples because the stacked HG already has a low prediction error. Achieving a slightly lower error in *easy (49)* comes at the cost of having worse predictions on the *difficult* samples. The situation for B_2 - PDM_4 is different. It improves the errors in both *difficult (49)* and *difficult (68)* without leading to worse predictions on *easy* samples.

D-PDM_4 beats the corresponding HG C_1 in both *easy (49)* and *difficult (49)* and therefore improves the *average 49* error. C_2-PDM_4 has the lowest *average 68* error but does not improve the individual 68 landmark categories. Surprisingly it improves *difficult (49)* instead. For the other categories there is no difference.

Finally, the HG-PDM with the lowest average error on all four categories (*D*-PDM_4) does neither improve or worsen the HG error in *easy (49)*, *easy (68)* and *difficult (68)*. However, in *difficult (49)* there is a significant improvement from 3.19 to 3.11.

Conclusion on the effectiveness of the PDM

The first thing to notice is that the HG used in a HG-PDM pipeline that achieved the lowest error in one category is not necessarily the same HG that had the lowest error without a PDM in this category. Examples are the rows *difficult (49)*, *difficult (68)* and *average 49* in Table 4.4. A reason for this could be that the heatmaps of the HG in the pipeline have different variances that lead to more accurate confidence measurements.

The biggest performance gain was accomplished in *difficult (49)*. The HG-PDM pipeline had a lower error in all cases except *average easy* where it increased the error by 0.06. In the case of *difficult (49)* the error decreased by even 0.14 and in *average all* by 0.08. This is a remarkable improvement compared to using only the stacked HG alone. Interestingly these observations are not transferable to the *difficult (68)* category. The differences are much smaller in this category. The biggest improvement was achieved in the case of *difficult (68)* where the error decreased by 0.04. In *average difficult* there also was a small improvement, but in all other cases the PDM did not help.

In the *easy (49)* category the HG-PDM combination dropped the error by 0.01 in the case of the best *easy (49)* model, by 0.05 in the *difficult (49)* case, by 0.01 in *average easy* and *average 49*. There is no improvement in *easy (49)* for any case that includes 68 landmarks. The performance in these cases either got worse (*easy (68)* and *difficult (68)*) or the was no difference. For the *easy (68)* category the same can be observed even stronger: Only the best *easy (68)* HG-PDM combination could improve the error in this category. For all other models there is no error difference, except for *difficult (68)* which worsened the error from 2.74 to 2.80.

Two main conclusions can be made based on this information. First, the *difficult* categories benefit more than the *easy* categories when the PDM is used. This can be explained by the low errors that the stacked HG has on these samples. It is hard to improve the predictions even more and it is also possible that there is a natural limit in the theoretical error that can be achieved because of small errors in the annotations, especially on the face outline. Second, the 49 landmark categories received a bigger improvement by using the PDM than the 68 landmark categories. This suggests that the PDM fails at learning a mapping between the latent vectors and the coordinates on the face outline. One possible explanation for this could also be imprecise annotations of the outline which cause the PDM to learn the variations in the annotations rather than the actual variations in the shape of the face outline. Another explanation could be the outline prediction accuracy of the stacked HG. If the confidence of too many landmarks on the outline is low, the PDM has no clue of the general face shape. This is because it is not possible to guess the face outline from the inner of the face (for example, the same inner facial landmarks can belong to a person with a round or slim face). The PDM will then reconstruct an outline that is based on predictions with low confidence and will therefore be erroneous. To confirm this, we have computed the average variances of the heatmaps of each landmark and the ones on the outline are higher than the inner landmarks on average. Hence, the corresponding confidences are lower and the inference is dominated by the inner landmarks.

Inference hyper-parameters of the best PDMs

HG-PDM	init.	*epochs*	*thresh*	lr_{shape}	sch_{pat}	sch_{fac}	$conf_a$	$conf_b$
A_1 - PDM_1	initializer	1000	0.0	0.025	5	0.75	1.0	0.1
A_2 - PDM_4	randomly	100	0.0	0.025	10	0.25	0.1	0.1
D - PDM_4	initializer	50	2.0	0.025	5	0.75	0.1	0.1
B_2 - PDM_2	randomly	100	3.0	0.025	10	0.5	1.0	0.1
B_1 - PDM_3	randomly	250	0.0	0.025	10	0.25	1.0	0.1
B_2 - PDM_4	randomly	1000	2.0	0.025	10	0.25	0.1	0.1
D - PDM_4	initializer	50	2.0	0.025	5	0.75	0.1	0.1
C_2 - PDM_4	randomly	250	2.0	0.1	25	0.1	1.0	0.1
D - PDM_4	initializer	50	2.0	0.025	5	0.75	0.1	0.1

Table 4.5: For each HG - PDM combination that was used in the PDM evaluation, the table shows the inference hyper-parameters. *thresh* is the variance threshold and lr_{shape} the learning rate for the shape parameters. sch_{pat} and sch_{fac} are the patience and decay factor of the learning rate scheduler for lr_{shape}. $conf_a$ and $conf_b$ are parameters that are used to compute the confidence from the heatmap variance.

After a PDM has been trained, there are still some hyper-parameters that influence the inference. These hyper-parameters were introduced in Section 4.6.1. Table 4.5 shows the parameters that were used to produce the results in this section and Section 4.6.6.

Unfortunately it is not easy to draw conclusions for the optimal values of the inference parameters. Small changes in the values lead to big differences in the final prediction. If the error of the final prediction is described as a function of the hyper-parameters, this function is highly non-convex and there is a complex interplay between the hyper-parameters. It is therefore very difficult to find optimal values. This could be an explanation why the value for the number of test epochs is so different between the models.

4.6.6 Comparison with state-of-the-art methods

Similar to Section 4.5.5 where the stacked HG was compared to state-of-the-art methods, this section provides a comparison for the whole system, namely the stacked HG combined with the PDM.

Table 4.6 shows the errors on the test set of 300-W [55, 56, 57] and the train set of Menpo [82] of our models and a selection of the baselines presented in Section 2.3. In the table we list only the best baselines that were listed in Section 4.5.5 to make the table more clear. If a model is better than one of these baselines, than it is also better than all other baselines. We report our results for nine models: The best per category, the best average models for *easy*, *difficult*, *49* and *68* landmarks and the overall best model. These models are described in detail in Section 4.6.5.

Dataset / Method	Helen [36]+LFPW [5]		iBUG [56]		Menpo [82]	
	easy (49)	easy (68)	diff (49)	diff (68)	49	68
PO-CR [67] ★ ◆	2.67	-	*3.33*	-	2.03	-
AWL [72]	-	*2.72*	-	*4.52*	2.12	
CE-CLM [79]	2.30	3.15	3.86	5.31	*1.74*	2.23
FC-LGCN [44]	*2.21*	2.86	4.18	5.29	1.79	*2.14*
A_1-PDM_1: *easy 49*	**1.94** ✔	-	3.33 (✔)	-	**1.64** ✔	-
A_2-PDM_4: *easy 68*	2.04 ✔	**2.65** ✔	3.44	4.54	1.66 ✔	**1.95** ✔
D-PDM_4: *diff. 49*	2.00 ✔	2.72 (✔)	**3.11** ✔	4.41 ✔	1.65 ✔	2.00 ✔
B_2-PDM_2: *diff. 68*	2.08 ✔	2.80	3.24 ✔	**4.30** ✔	1.69 ✔	2.03 ✔
B_1-PDM_3: *avg. easy*	2.00 ✔	2.66 ✔	3.43	4.60	1.67 ✔	**1.95** ✔
B_2-PDM_4: *avg. diff.*	2.08 ✔	2.80	3.18 ✔	4.33 ✔	1.69 ✔	2.03 ✔
D-PDM_4: *avg. 49*	2.00 ✔	2.72 (✔)	**3.11** ✔	4.41 ✔	1.65 ✔	1.99 ✔
C_2-PDM_4: *avg. 68*	2.05 ✔	2.74	3.36	4.34 ✔	1.66 ✔	1.95 ✔
D-PDM_4: *avg. all*	2.00 ✔	2.72 (✔)	**3.11** ✔	4.41 ✔	1.65 ✔	1.99 ✔

Table 4.6: Comparison of our models to state-of-the-art baselines. The models are denoted as *HG-PDM*. The error values are the median IOD normalized RMSE, multiplied by 100 for better readability. Bold numbers are the lowest in one category. ✔ means that the error is lower than the baselines. ★: Numbers taken from Zadeh et al. [79]. ◆: Numbers taken from Merget et al. [44]. The other numbers are taken from the original papers.

All our HG-PDM models achieve lower errors than the state-of-the-art methods in *easy (49)*, *Menpo (49)* and *Menpo (68)*. To focus on the relevant information, we only discuss the models *average 49*, *average 68* and *average all* in the following because in practice it is unlikely to know beforehand if the system will be applied to easy or difficult images.

The best *average (49)* HG-PDM is as good as **AWL** [72] on *easy (68)* and better than all baselines in all other categories. The best *average (68)* HG-PDM is better than all baselines in all categories except *easy (68)* and *difficult (49)*, where it is slightly worse. The overall best HG-PDM has an error that is lower than all baselines in all categories except *easy (68)*, where it is as good as **AWL**. The biggest error difference between our models and the baselines can be observed in the *easy (49)* and *difficult (49)* categories.

4.7 Qualitative results

After the system has been evaluated using a lot of numbers, we finally show some landmark predictions that were created by the system developed in this work in Figure 4.19.

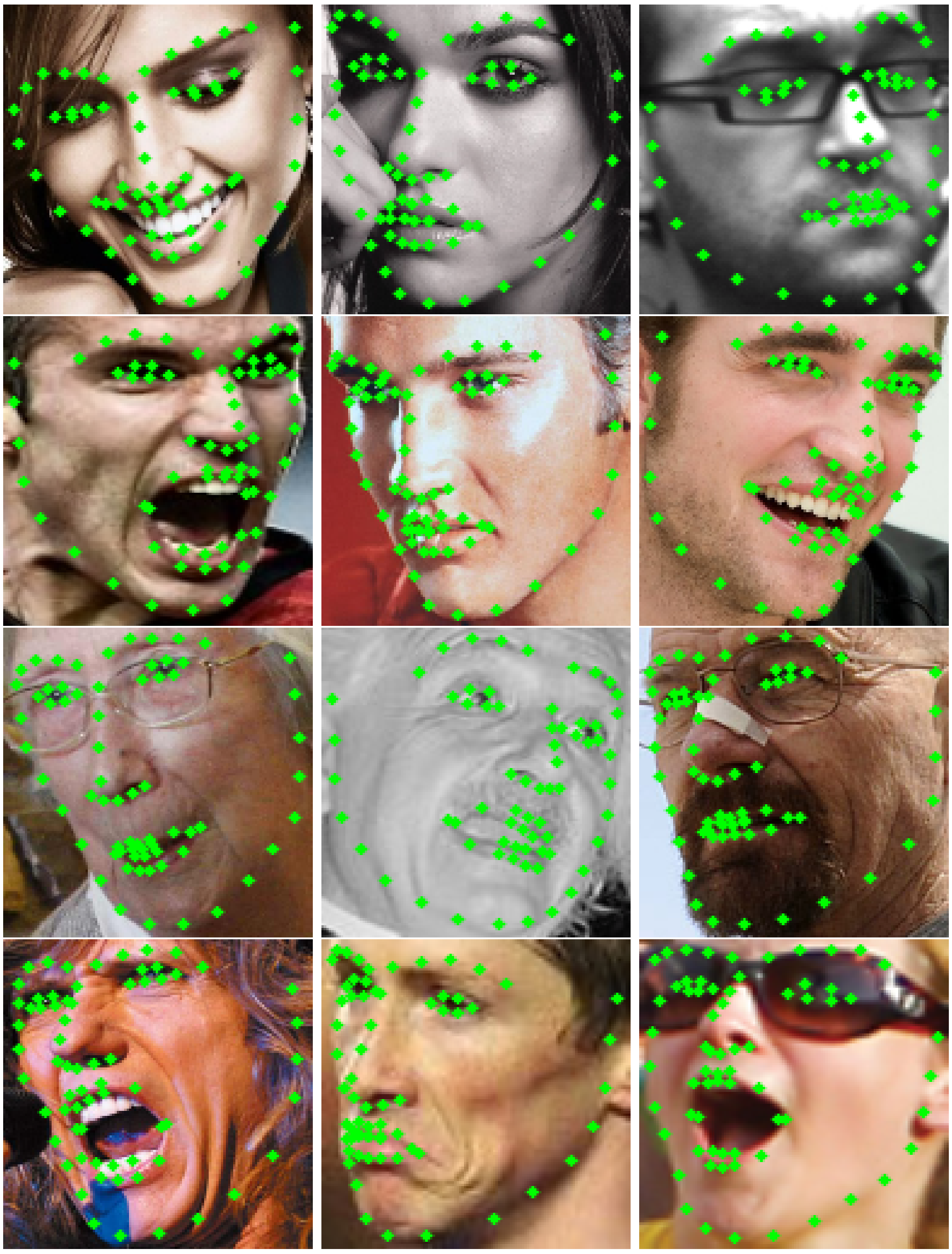


Figure 4.19: Predictions on the difficult 300-W test set [55, 56, 57]

5. Conclusion

5.1 Summary

This work aimed at improving state-of-the-art results for the task of facial landmark detection on unconstrained frontal and semi-frontal face images. To this end, a system consisting of two main parts has been designed, implemented and evaluated. As stacked Hourglass Networks (HGs) have already proven their effectiveness for other landmark prediction problems such as human pose estimation, we chose to use a stacked Hourglass (HG) to produce initial predictions for each facial landmark. In order to further refine these predictions, we designed a Point Distribution Model (PDM).

We found that training the stacked HG using Wing loss and transforming the regressed heatmaps into numerical coordinates using a Differentiable Spatial To Numerical Transform (DSNT) is very effective for the task at hand. Various design choices for the stacked HG were evaluated. Regularizing the heatmaps using the Jensen-Shannon divergence proved to be helpful. Augmenting the training data by randomly rotating the face images lowered the prediction error especially for difficult images by a large factor. By applying the findings of the evaluation, our stacked HG is able to produce state-of-the-art results in terms of the point-to-point normalized error on both the 300-W and Menpo datasets. The error on Menpo was determined by a cross-dataset evaluation on a model that was only trained on 300-W.

The second part of this work focused on the PDM. We found that simple decoder networks with only two layers are able to model a mapping from a latent vector to the face shapes that can be used to improve the predictions made by the stacked HG. We furthermore have shown that initializing the shape parameters using a separate initializer network can lead to faster and more accurate predictions. Combining the stacked HG with the PDM further improves the prediction accuracy, especially for the inner facial landmarks on difficult images that show faces with strong head poses.

The stacked HG achieves state-of-the-art results on 300-W and Menpo both with and without the PDM. While adding the PDM does not lower the prediction error of the stacked HG for landmarks on the outline, it improves the localization accuracy on the inner facial landmarks of the difficult samples from 300-W. Our overall system is able to produce accurate predictions for faces in unconstrained settings, which includes faces with partial occlusions.

5.2 Outlook

Although good results have been achieved in this thesis, there are a lot of ideas that could lead to even better predictions.

The stacked HG and the PDM were trained separately in this work. As the PDM relies on the confidence measurement that is determined based on the variance of the heatmaps from the stacked HG, this is a critical point that should be further investigated. A possible improvement over the current solution could be the end-to-end training of both the stacked HG and the PDM. This would enable a more complex confidence measurement. For example, an additional confidence estimation network could be employed between the stacked HG and the PDM. This would make determining a suitable mapping function between heatmap variance and confidence obsolete and would furthermore allow to learn a different mapping for each of the landmarks. Since we observed that the typical variances lie in different ranges depending on the individual landmarks, this is an interesting direction to follow in the future.

We used the same topology for each of the individual HGs in the stack. In order to improve the stacked HG in terms of speed, model size and accuracy, different topologies for the individual HGs could be tested. Modifying the pre-processing network so that it outputs a higher resolution could also be beneficial for the accuracy of the stacked HG.

Finally, this work was restricted to semi-frontal images. An useful extension would be to modify the system to also work for profile images.

Bibliography

[1] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. “Face recognition with local binary patterns”. In: *European conference on computer vision*. Springer. 2004, pp. 469–481.

[2] Akshay Asthana, Stefanos Zafeiriou, Shiyang Cheng, and Maja Pantic. “Robust discriminative response map fitting with constrained local models”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 3444–3451.

[3] Tadas Baltrusaitis, Peter Robinson, and Louis-Philippe Morency. “Constrained local neural fields for robust facial landmark detection in the wild”. In: *Proceedings of the IEEE international conference on computer vision workshops*. 2013, pp. 354–361.

[4] Ankan Bansal, Anirudh Nanduri, Carlos D Castillo, Rajeev Ranjan, and Rama Chellappa. “Umdfaces: An annotated face dataset for training deep networks”. In: *2017 IEEE International Joint Conference on Biometrics (IJCB)*. IEEE. 2017, pp. 464–473.

[5] Peter N Belhumeur, David W Jacobs, David J Kriegman, and Neeraj Kumar. “Localizing parts of faces using a consensus of exemplars”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.12 (2013), pp. 2930–2940.

[6] Matteo Bodini. “A review of facial landmark extraction in 2d images and videos using deep learning”. In: *Big Data and Cognitive Computing* 3.1 (2019), p. 14.

[7] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. “A training algorithm for optimal margin classifiers”. In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM. 1992, pp. 144–152.

[8] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[9] Adrian Bulat and Georgios Tzimiropoulos. “Binarized convolutional landmark localizers for human pose estimation and face alignment with limited resources”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3706–3714.

[10] Adrian Bulat and Georgios Tzimiropoulos. “How far are we from solving the 2d & 3d face alignment problem?(and a dataset of 230,000 3d facial landmarks)”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1021–1030.

[11] Adrian Bulat and Georgios Tzimiropoulos. “Super-FAN: Integrated facial landmark localization and super-resolution of real-world low resolution faces in arbitrary poses with GANs”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 109–117.

[12] Xavier P Burgos-Artizzu, Pietro Perona, and Piotr Dollár. "Robust face landmark estimation under occlusion". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1513–1520.

[13] Xudong Cao, Yichen Wei, Fang Wen, and Jian Sun. "Face alignment by explicit shape regression". In: *International Journal of Computer Vision* 107.2 (2014), pp. 177–190.

[14] Dong Chen, Gang Hua, Fang Wen, and Jian Sun. "Supervised transformer network for efficient face detection". In: *European Conference on Computer Vision*. Springer. 2016, pp. 122–138.

[15] Joon Son Chung and Andrew Zisserman. "Lip reading in the wild". In: *Asian Conference on Computer Vision*. Springer. 2016, pp. 87–103.

[16] Timothy F Cootes, Gareth J Edwards, and Christopher J Taylor. "Active appearance models". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 6 (2001), pp. 681–685.

[17] Timothy F Cootes, Christopher J Taylor, David H Cooper, and Jim Graham. "Active shape models-their training and application". In: *Computer vision and image understanding* 61.1 (1995), pp. 38–59.

[18] Timothy F Cootes, Gareth J Edwards, Christopher J Taylor, et al. "Comparing active shape models with active appearance models." In: *Bmvc*. Vol. 99. 1. Citeseer. 1999, pp. 173–182.

[19] David Cristinacce and Timothy F Cootes. "Feature detection and tracking with constrained local models." In: *Bmvc*. Vol. 1. 2. Citeseer. 2006, p. 3.

[20] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *international Conference on computer vision & Pattern Recognition (CVPR'05)*. Vol. 1. IEEE Computer Society. 2005, pp. 886–893.

[21] Matthias Dantone, Juergen Gall, Gabriele Fanelli, and Luc Van Gool. "Real-time facial feature detection using conditional regression forests". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 2578–2585.

[22] Jiankang Deng, Anastasios Roussos, Grigorios Chrysos, Evangelos Ververas, Irene Kotsia, Jie Shen, and Stefanos Zafeiriou. "The Menpo Benchmark for Multi-pose 2D and 3D Facial Landmark Localisation and Tracking". In: *International Journal of Computer Vision* 127.6 (June 2019), pp. 599–624. ISSN: 1573-1405. DOI: `10.1007/s11263-018-1134-y`. URL: `https://doi.org/10.1007/s11263-018-1134-y`.

[23] C. Fabian Benitez-Quiroz, Ramprakash Srinivasan, and Aleix M. Martinez. "EmotioNet: An Accurate, Real-Time Algorithm for the Automatic Annotation of a Million Facial Expressions in the Wild". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[24] Zhen-Hua Feng, Josef Kittler, Muhammad Awais, Patrik Huber, and Xiao-Jun Wu. "Wing loss for robust facial landmark localisation with convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2235–2245.

[25] Yoav Goldberg. "A primer on neural network models for natural language processing". In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 345–420.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[27] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[28] Ralph Gross, Iain Matthews, Jeffrey Cohn, Takeo Kanade, and Simon Baker. “Multi-pie”. In: *Image and Vision Computing* 28.5 (2010), pp. 807–813.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[30] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[31] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[32] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. “Spatial transformer networks”. In: *Advances in neural information processing systems*. 2015, pp. 2017–2025.

[33] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[34] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[36] Vuong Le, Jonathan Brandt, Zhe Lin, Lubomir Bourdev, and Thomas S. Huang. “Interactive Facial Feature Localization”. In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 679–692. ISBN: 978-3-642-33712-3.

[37] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[38] Sijin Li, Zhi-Qiang Liu, and Antoni B Chan. “Heterogeneous multi-task learning for human pose estimation with deep convolutional neural network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, pp. 482–489.

[39] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. “A survey of deep neural network architectures and their applications”. In: *Neurocomputing* 234 (2017), pp. 11–26.

[40] David G Lowe et al. “Object recognition from local scale-invariant features.” In: *iccv*. Vol. 99. 2. 1999, pp. 1150–1157.

[41] Diogo C Luvizon, Hedi Tabia, and David Picard. "Human pose regression by combining indirect part detection and contextual information". In: *arXiv preprint arXiv:1710.02322* (2017).

[42] Iacopo Masi, Stephen Rawls, Gerard Medioni, and Prem Natarajan. "Pose-Aware Face Recognition in the Wild". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[43] Uwe Meier, Rainer Stiefelhagen, Jie Yang, and Alex Waibel. "Towards unrestricted lip reading". In: *International Journal of Pattern Recognition and Artificial Intelligence* 14.05 (2000), pp. 571–585.

[44] Daniel Merget, Matthias Rock, and Gerhard Rigoll. "Robust Facial Landmark Detection via a Fully-Convolutional Local-Global Context Network". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[45] Kieron Messer, Jiri Matas, Josef Kittler, Juergen Luettin, and Gilbert Maitre. "XM2VTSDB: The extended M2VTS database". In: *Second international conference on audio and video-based biometric person authentication*. Vol. 964. 1999, pp. 965–966.

[46] Erik Murphy-Chutorian and Mohan Manubhai Trivedi. "Head pose estimation in computer vision: A survey". In: *IEEE transactions on pattern analysis and machine intelligence* 31.4 (2008), pp. 607–626.

[47] Alejandro Newell, Kaiyu Yang, and Jia Deng. "Stacked hourglass networks for human pose estimation". In: *European Conference on Computer Vision*. Springer. 2016, pp. 483–499.

[48] Aiden Nibali, Zhen He, Stuart Morgan, and Luke Prendergast. "Numerical coordinate regression with convolutional neural networks". In: *arXiv preprint arXiv:1801.07372* (2018).

[49] Panagiotis Perakis, Georgios Passalis, Theoharis Theoharis, and Ioannis A Kakadiaris. "3D facial landmark detection under large yaw and expression variations". In: *IEEE transactions on pattern analysis and machine intelligence* 35.7 (2012), pp. 1552–1564.

[50] *Python*. 2019. URL: `https://www.python.org/` (visited on 06/13/2019).

[51] *PyTorch*. 2019. URL: `https://pytorch.org/` (visited on 06/13/2019).

[52] Deva Ramanan and Xiangxin Zhu. "Face detection, pose estimation, and landmark localization in the wild". In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Citeseer. 2012, pp. 2879–2886.

[53] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. "Face alignment at 3000 fps via regressing local binary features". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1685–1692.

[54] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1.

[55] C. Sagonas, E. Antonakos, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. "300 faces In-the-wild challenge: Database and results". In: *Image and Vision Computing (IMAVIS), Special Issue on Facial Landmark Localisation "In-The-Wild"*. 2016.

[56] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. "300 Faces in-the-Wild Challenge: The first facial landmark localization Challenge". In: *300 Faces in-the-Wild Challenge (300-W).* Proceedings of IEEE International Conf. on Computer Vision (ICCV-W). Dec. 2013.

[57] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. "A semi-automatic methodology for facial landmark annotation". In: *5th Workshop on Analysis and Modeling of Faces and Gestures (AMFG 2013).* Proceedings of IEEE International Conf. Computer Vision and Pattern Recognition (CVPR-W). June 2013.

[58] M Saquib Sarfraz and Olaf Hellwich. "Head Pose Estimation in Face Recognition Across Pose Scenarios." In: *VISAPP (1)* 8 (2008), pp. 235–242.

[59] M Saquib Sarfraz and Olaf Hellwich. "On head pose estimation in face recognition". In: *International Conference on Computer Vision and Computer Graphics.* Springer. 2008, pp. 162–175.

[60] Jie Shen, Stefanos Zafeiriou, Grigoris G Chrysos, Jean Kossaifi, Georgios Tzimiropoulos, and Maja Pantic. "The first facial landmark tracking in-the-wild challenge: Benchmark and results". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops.* 2015, pp. 50–58.

[61] Rainer Stiefelhagen, Jie Yang, and Alex Waibel. "Estimating focus of attention based on gaze and sound". In: *Proceedings of the 2001 workshop on Perceptive user interfaces.* ACM. 2001, pp. 1–9.

[62] Rainer Stiefelhagen, Jie Yang, and Alex Waibel. "Tracking eyes and monitoring eye gaze". In: *Proc. Workshop on Perceptual User Interfaces.* 1997, pp. 98–100.

[63] Yi Sun, Xiaogang Wang, and Xiaoou Tang. "Deep convolutional network cascade for facial point detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2013, pp. 3476–3483.

[64] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. "Deepface: Closing the gap to human-level performance in face verification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2014, pp. 1701–1708.

[65] Zhiqiang Tang, Xi Peng, Shijie Geng, Lingfei Wu, Shaoting Zhang, and Dimitris Metaxas. "Quantized densely connected u-nets for efficient landmark localization". In: *Proceedings of the European Conference on Computer Vision (ECCV).* 2018, pp. 339–354.

[66] George Trigeorgis, Patrick Snape, Mihalis A Nicolaou, Epameinondas Antonakos, and Stefanos Zafeiriou. "Mnemonic descent method: A recurrent process applied for end-to-end face alignment". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 4177–4187.

[67] Georgios Tzimiropoulos. "Project-out cascaded regression with an application to face alignment". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2015, pp. 3659–3667.

[68] Paul Viola, Michael Jones, et al. "Rapid object detection using a boosted cascade of simple features". In: *CVPR (1)* 1 (2001), pp. 511–518.

[69] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. “Phoneme recognition using time-delay neural networks”. In: *Backpropagation: Theory, Architectures and Applications* (1995), pp. 35–61.

[70] Robert Walecki, Ognjen Rudovic, Vladimir Pavlovic, and Maja Pantic. “Copula Ordinal Regression for Joint Estimation of Facial Action Unit Intensity”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2016.

[71] Nannan Wang, Xinbo Gao, Dacheng Tao, Heng Yang, and Xuelong Li. “Facial feature point detection: A comprehensive survey”. In: *Neurocomputing* 275 (2014), pp. 50–65.

[72] Xinyao Wang, Liefeng Bo, and Li Fuxin. “Adaptive Wing Loss for Robust Face Alignment via Heatmap Regression”. In: *arXiv preprint arXiv:1904.07399* (2019).

[73] Wayne Wu, Chen Qian, Shuo Yang, Quan Wang, Yici Cai, and Qiang Zhou. “Look at boundary: A boundary-aware face alignment algorithm”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 2129–2138.

[74] Xuehan Xiong and Fernando De la Torre. “Supervised descent method and its applications to face alignment”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2013, pp. 532–539.

[75] Jiaolong Yang, Peiran Ren, Dongqing Zhang, Dong Chen, Fang Wen, Hongdong Li, and Gang Hua. “Neural Aggregation Network for Video Face Recognition”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* July 2017.

[76] Jing Yang, Qingshan Liu, and Kaihua Zhang. “Stacked hourglass network for robust facial landmark localisation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.* 2017, pp. 79–87.

[77] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. “Recent trends in deep learning based natural language processing”. In: *ieee Computational intelligenCe magazine* 13.3 (2018), pp. 55–75.

[78] Fisher Yu and Vladlen Koltun. “Multi-scale context aggregation by dilated convolutions”. In: *arXiv preprint arXiv:1511.07122* (2015).

[79] Amir Zadeh, Tadas Baltrusaitis, and Louis-Philippe Morency. “Convolutional Experts Constrained Local Model For Facial Landmark Detection”. In: *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017.

[80] Amir Zadeh, Yao Chong Lim, Tadas Baltrusaitis, and Louis-Philippe Morency. “Convolutional Experts Constrained Local Model for 3D Facial Landmark Detection”. In: *The IEEE International Conference on Computer Vision (ICCV) Workshops.* Oct. 2017.

[81] Amir Zadeh, Yao Chong Lim, Paul Pu Liang, and Louis-Philippe Morency. “Variational Auto-Decoder”. In: *CoRR* abs/1903.00840 (2019). arXiv: `1903.00840`. URL: `http://arxiv.org/abs/1903.00840`.

[82] Stefanos Zafeiriou, George Trigeorgis, Grigorios Chrysos, Jiankang Deng, and Jie Shen. “The menpo facial landmark localisation challenge: A step towards the solution”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.* 2017, pp. 170–179.

[83] Jie Zhang, Shiguang Shan, Meina Kan, and Xilin Chen. "Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment". In: *European conference on computer vision*. Springer. 2014, pp. 1–16.

[84] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. "Joint face detection and alignment using multitask cascaded convolutional networks". In: *IEEE Signal Processing Letters* 23.10 (2016), pp. 1499–1503.

[85] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. "Facial landmark detection by deep multi-task learning". In: *European conference on computer vision*. Springer. 2014, pp. 94–108.

[86] Shizhan Zhu, Cheng Li, Chen Change Loy, and Xiaoou Tang. "Face alignment by coarse-to-fine shape searching". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4998–5006.

[87] Xiangyu Zhu, Zhen Lei, Xiaoming Liu, Hailin Shi, and Stan Z Li. "Face alignment across large poses: A 3d solution". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 146–155.

List of abbreviations

CNN Convolutional Neural Network

SVM Support Vector Machine

PDM Point Distribution Model

DSNT Differentiable Spatial To Numerical Transform

HG Hourglass

AAM Active Appearance Model

ASM Active Shape Model

CLM Constrained Local Model

PCA Principal Component Analysis

FCNN Fully Convolutional Neural Network

STN Spatial Transformer Network

VAE Variational Auto Encoder

VAD Variational Auto Decoder

IOD Inter-Ocular Distance

RMSE Root Mean Squared Error