Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Yong Bin Kwon

3D Face Reconstruction Based On a Single Input Image

Master's thesis in Computer Science - MTDT
Supervisor: Theoharis Theoharis
Co-supervisor: Antonios Danelakis
June 2021

NTNU
Norwegian University of
Science and Technology

Yong Bin Kwon

3D Face Reconstruction Based On a Single Input Image

Master's thesis in Computer Science - MTDT
Supervisor: Theoharis Theoharis
Co-supervisor: Antonios Danelakis
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

3D Face Reconstruction Based On a Single Input Image

TDT4900 - Computer Science, Master's Thesis

Yong Bin Kwon

Supervisors:

Antonios Danelakis

Theoharis Theoharis

June 2021

Abstract

This project aims to accurately reconstruct 3D faces from a single image for the purpose of generating a pose and illumination invariant model. This is a topic of pivotal importance in a variety of computer vision subfields —such as face recognition—owing to the fact that regular 2D images are heavily affected by lighting conditions and pose. In recent years there has been an increased interest in these types of tasks, thanks to practical applications such as biometric authentication and missing person identification. This interest has led to further research, one of the more recent being methods that use a front- and side-facing input pair to reconstruct 3D faces. While this approach has led to greater accuracy than previously possible, it lacks in ease of use and the ability to be applied for images in-the-wild. To address this a hybrid approach using two existing methods was proposed. This method requires only a single image as input but employs a two-image input network for reconstruction. This is achieved by introducing an additional network whose task is to generate a rotated version of the original input, which in conjunction with said input make up the image-pair used for reconstruction. Although this method is expected to produce significantly more accurate results, the per-image inference time will suffer as a result of the overhead introduced by an additional network. This will not only affect the user experience but also make the system nonviable for certain real-time tasks. For that reason, it is believed that future research should focus on speed rather than accuracy.

Sammendrag

Dette prosjektet har som mål å rekonstruere et 3D ansikt fra et bilde, noe som resulterer i en modell som ikke er påvirket av ansiktsvinkel eller lys. Dette er et viktig tema for forskjellige fagfelt innenfor datasyn—slik som ansiktsgjenkjenning—grunnet at vanlige 2D bilder er høyst påvirket lys og hvor ansiktet ser i horhold til kamera. I det siste har det vært en oppsving i interesse for slike oppgaver, takket være praktiske bruksområder slik som biometrisk autentisering og identifisering av savnede personer. Denne interessen har ledet til mer forskning, hvor en av de nyligere gjennombruddene er rekonstruksjon basert på to bilder, tatt fra siden og foran. Selv om denne metoden har ført til høyere nøyaktighet enn tidligere, har den lavere brukervennlighet samt at den ikke er brukbar på oppgaver hvor man ikke har to bilder å rekonstruere med slik som video eller tilfeldige bilder funnet på nett. For å kunne rette på dette ble en sammenslåing av to allerede eksisterende metoder foreslått. Denne nye metoden krever kun ett bilde som input men bruker et to-bilders nevralt nettverk for rekonstruksjon. Dette gjøres ved å bruke et ekstra nettverk som genererer en rotert versjon av input-bildet, som sammen med det originale bildet danner input-paret brukt for rekonstruksjon. Selv om denne metoden forventes å være mer nøyaktig enn andre ett-input nettverk, forventes også kjøretiden å bli negativt påvirket, grunnet ekstra overhead som kommer av et ytterligere nettverk. Dette vil ikke bare påvirke brukeropplevelse men også gjøre systemet i ustand til å kjøre på sanntids-oppgaver. På bakgrunn av dette er det anbefalt at ytteligere forskning burde fokusere på hastighet heller enn nøyaktighet.

Preface

This thesis describes the project that has been worked on during 2020/2021 as part the computer science master's program at the Norwegian University of Science and Technology.

I would like to thank my supervisors Professor Danelakis and Professor Theoharis for their knowledge and support during this year. Your ability to respond quickly at any time of the day no matter the circumstances is something I have appreciated immensely. Secondly I would like to thank the people at IDI DRIFT for enabling me to remotely access the required hardware from home, and for always being there to fix any problems that might show up. Finally I would like to thank Kim for her emotional support not just during this year but my whole university career. Thank you for always lending an ear and cheering me up when I was feeling down. I truly could not have come this far without you.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

One of the most researched aspects in the field of computer vision are problems that relate to the human face. This is due to it being the key visual identifier that separates one person from another; a characteristic that has been taken quite literally as systems like Face ID[1] allows a great number of regular consumers to authenticate themselves using only their face.

When developing systems that attempt to solve these sort of analysis and recognition problems 3D models are to be preferred over their 2D-image counterpart, as they do not suffer from pose and illumination variations. However, this requires expensive and awkward 3D imaging devices which for most tasks are impractical and in some cases infeasible. The extreme trade-off between performance and usability makes it hard to settle for either of the options.

3D face reconstruction attempts to bridge this gap between 2D and 3D by accurately reconstructing a three-dimensional mesh from a regular image. Thus eliminating the need for expensive and slow 3D imaging systems while still producing a detailed model to be used for various computer vision tasks.

Much like the problem it attempts to solve the current state-of-the-art in 3D face reconstruction is also plagued by a trade-off between performance and practicality. The gold standard for a few years running utilizes a convolutional neural network to regress a 3D model from a single 2D image [1], and produces great results both in accuracy and runtime. Recently there was an attempt at improving the performance of the state-of-the-art by employing a network that takes as input an image pair consisting of a front and side-facing image [2]. This method did indeed manage to further increase accur-

[1] `https://support.apple.com/en-us/HT208109`

acy by a noticeable amount, but the fact that it requires two input images introduces extra constraints on the user while also being inadequate for important use-cases such as video and unconstrained images.

The method outlined in this paper attempts to use the valuable insights gained from [2], while attempting to reduce the constraints that come with a two image input network. The key contribution from this paper is that multi-view face synthesis can be used to generate an input image pair from a single 2D image. This in turn allows the use of two image input reconstruction networks using only a front or side facing image. The proposed method consists of two networks: The rotation and reconstruction network. For the former a slightly modified version of the Rotate-and-Render GAN [3] is used, whilst [2] is used for the latter. This pipeline is illustrated in Figure 1.1.

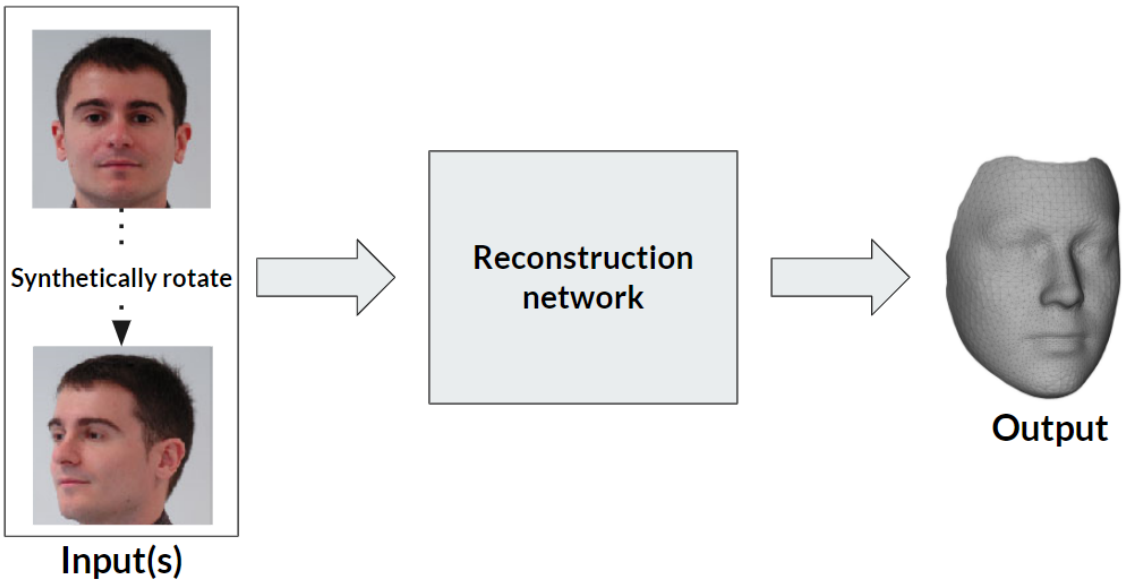


Figure 1.1: Overview of proposed method.

This Thesis has been organised in the following way: The next chapter will provide the necessary background theory that this paper is built upon. A brief overview of the related works will follow in chapter three. Chapter four will include a more detailed look at the proposed method, and eventual concerns and discussion. The fifth chapter looks at the results and a discusses of these findings. To conclude with, a look back on the work that has been done and potential improvements will make up the final chapter.

Chapter 2

Background

2.1 Convolutional Networks

2.1.1 Convolutional Neural Network

A convolutional neural network—often shortened to ConvNet or CNN—is a type of neural network that consists of at least one convolutional layer. This type of layer is composed of the same building blocks that constitute the basic fully connected layer, such as neurons and learnable weights and biases. The key difference being that it takes as input a 2D or 3D matrix of neurons as opposed to a vector. This type of matrix representation is suitable for image-related tasks, as the spatial information is kept intact by having each pixel correspond to an entry in the matrix.

Similarly, the weights and biases are also represented as matrices called kernels, or filters. To calculate the activation of a neuron in the next layer a convolution operation is performed between the kernel and the input entries it currently overlaps with–hence the name *convolutional* layer. Convolution, at least in the context of neural networks, is an operation that produces a weighted sum given a matrix of weights—the kernel—and a subset of the input that equals the kernel in size. The sum is calculated using the following formula where *i,j* is the row and column position of the output and *m,n* iterates through the whole width and height of the kernel:

$$output(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.1}$$

A more intuitive explanation of convolution is that it is the sum of element-wise multiplication between the flipped kernel and the subset of the image that it overlaps

with. In practice flipping the kernel does not do much in the way of changing the results as the kernel would just adjust itself to use the "flipped" weights in its un-flipped state. For this reason, the more commonly used function for calculating activations is cross-correlation as expressed in the following formula:

$$output(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.2}$$

This simplifies the calculations without any loss in performance, as it is just convolution without flipping the kernel. The only downside is that naming becomes a little confusing as the layer is still called convolutional even if the actual operation that is performed is cross-correlation. For simplicity sake, this paper will also be referring to the mathematical operation applied on a convolutional layer as convolution.

During a forward pass, convolution is applied multiple times as the kernel slides through the current layer until it has covered the whole width and height of it. This is illustrated in Figure 2.1. The application of convolution on local regions of the input, fittingly called a local receptive field, is what truly makes CNN's "spatial" in nature.

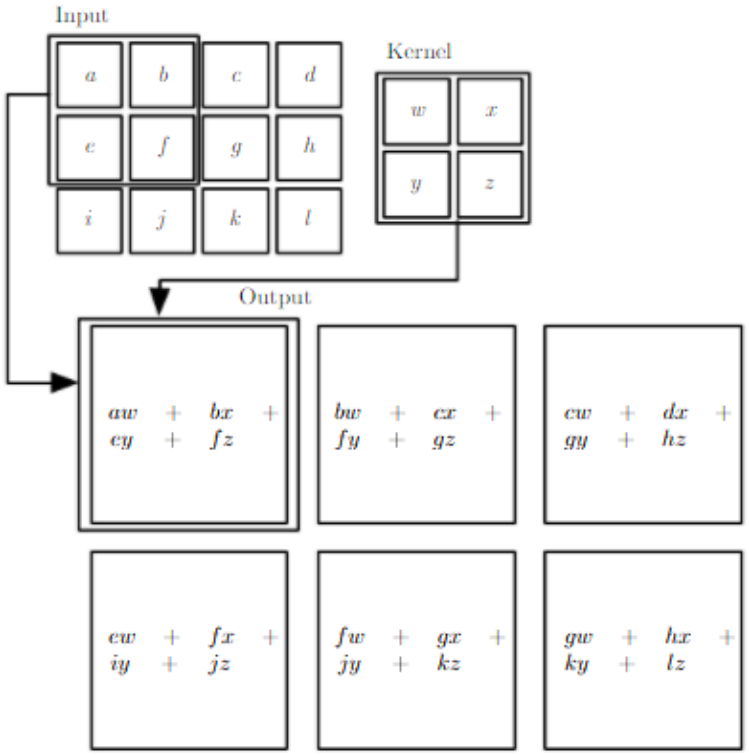


Figure 2.1: Figure from Goodfellow et al. [4] showing how a region of the input is used to calculate a neuron in the output.

2.1.2 Transposed Convolution

The name transposed convolution—or deconvolution as it is often called—correctly suggests that this is an operation closely related to the previously mentioned convolution. The biggest difference between convolution and its transposed counterpart is that whereas convolutional layers reduce the spatial resolution of the input, a transposed convolutional layer increases it. In fact, a transposed convolution will reverse the spatial downsampling following a convolution, given that stride and kernel size is the same between the two.

This characteristic in conjunction with its name might suggest that transposed convolution is an inverse of convolution. That is not the case as transposed convolution does not undo the effects of convolution as a whole, just the spatial downsampling. The output of this operation is reliant on learnable weights and biases, the same as a regular convolutional layer. Intuitively the difference between these operations can be understood as convolution taking an input and extracting generic features from it, while a transposed convolution takes some generic features and generates an output from it. The calculations done by a transposed convolutional layer is illustrated in Figure 2.2.

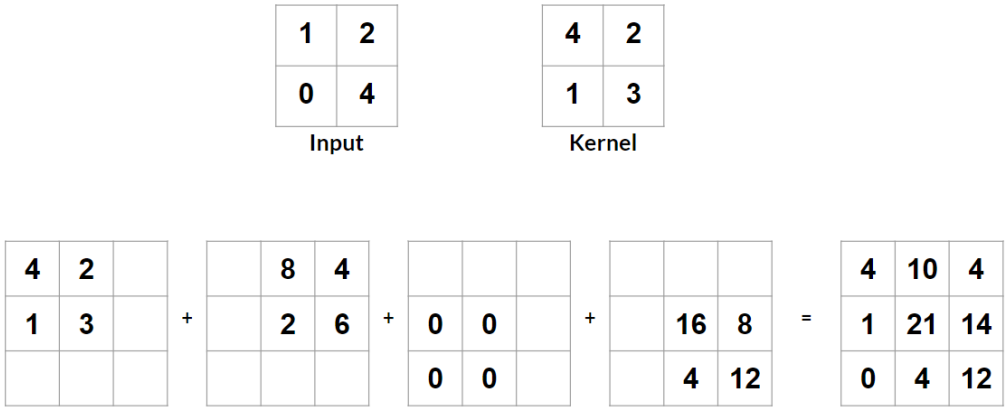


Figure 2.2: Example of transposed convolution with stride=1.

2.1.3 Generalization

The essence of neural networks is to approximate a function that maps a given input to the desired output. What is important is that this function is *generic*, meaning it works

well on all types of input, not just the ones it has seen during training. There are two situations that are emphasized in the field of neural networks as bad generalization: Overfitting and underfitting. An illustration of these cases are shown in Figure 2.3.

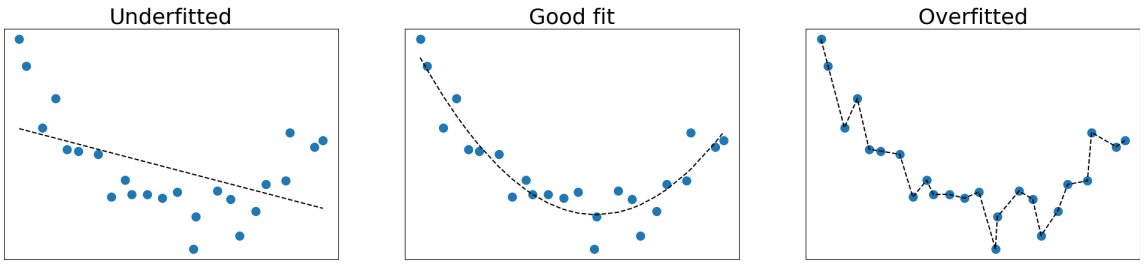


Figure 2.3: Three regressed functions of differing complexity.

Underfitting occurs when the approximated function lacks in complexity compared to that of the dataset. This lack of complexity means it is unable to capture all of the features of the input domain, and thus have to make simplifying assumptions about it. This leads to poor performance on the training data, in which case it will not perform well on never before seen data either. When underfitting do occur a more complex model or longer training might be beneficial.

Underfitting is closely related to the concept of bias in statistics. The bias of an estimator, not to be confused with the bias value applied during a forward pass, is the difference between expected and approximated value. A model is biased when its approximations, given some input, are ways apart from the true values mapped to that input. This is an apt description of an underfitting model which on average outputs undesirable values.

On the other side of the spectrum there is overfitting, where the model might be performing *too* well in some sense. A characteristic of an overfitted model is that it has great results on the training set but falls short when it is faced with unseen data. This happens because the model is trying to approximate a function that perfectly fits the training data. This leads to a function that is tailor-made for a specific sample of the input domain, the training set, resulting in it becoming too complex and too fixated on the details of each entry instead of focusing on the characteristics that define the input domain as a whole. One understanding of this phenomenon is that instead of *learning* the model is *memorizing*. When faced with overfitting it is often suggested to either decrease the complexity of the model or stop the training process earlier.

It is often said that an overfitted model has a high level of variance, which is a common concept in statistics to denote how spread out a data sample is. In this particular instance variance refers to how spread out the outputs of an approximated function are depending on the training data. An overfitted model is a prime example of high variance, as its outputs will vary wildly between training sets as illustrated on Figure 2.4. A model of high variance might suggest that it is too sensitive to the noise of each data point, and thus diluting the actual features that represent the input domain.

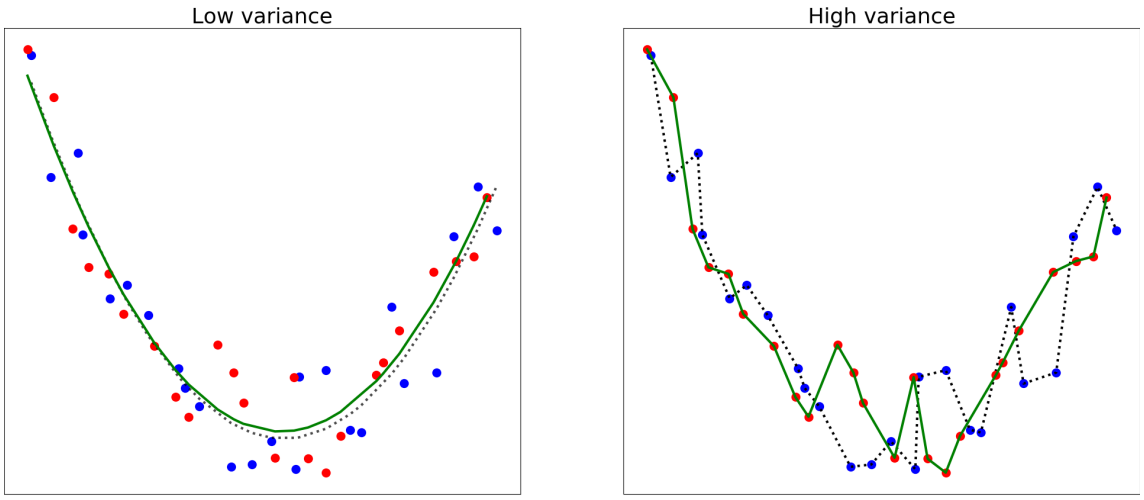


Figure 2.4: On the left the approximated functions of a well fit model and on the right those of an overfitted one. The dots represent training data from two different datasets, where the green function is trained on the red dots while the dotted black function is trained on the blue ones. Both models are trained on the same datasets.

In an ideal world one would want a model that is both unbiased *and* has a low level of variance. Unfortunately this is quite difficult to achieve due to the so-called bias-variance trade-off. As the name suggests this is a characteristic of neural networks that states that the variance of a model can be reduced by increasing its bias, and vice versa. With this in mind the training process becomes a matter of approximating a function that hits the sweet-spot that minimizes the sum of variance and bias. There are a few parameters that can be tweaked to affect this, namely model complexity and number of training iterations as was previously mentioned.

Regularization is another crucial tool when seeking to minimize error from bias and variance. Regularization as a concept is a little difficult to pinpoint as there are multiple

definitions of it. One common definition is that regularization is any technique that reduces certain coefficients of the approximated function towards zero, drastically minimizing the impact of the parameters associated with said coefficient. This in turn reduces function complexity and decreases variance. A more top-level definition can be found in *Deep Learning Book* written by Goodfellow et al.:

> Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error [4, p. 117].

The common thread among definitions is that regularization increases model generalization.

2.2 Neural Network Architectures

2.2.1 ResNet

As the name might suggest deep learning is all about *deep* neural networks, i.e networks with many layers. Intuitively each layer can be seen as an abstraction of the input image. Thus adding more layers allows the network to extract features at many different levels. This will in turn give the network a *deeper* understanding of the input space. In practice this means that the network has knowledge that allows it to perform well on data that it has never encountered during training. In contrast a shallow network given enough parameters would be able to output the desired values during training, but these results would be based on memorization as it would not have an understanding of all the intermediate features of the input— an example of bad generalization.

These extra layers allow networks to deal with increasingly complex problems but do not come without problems of their own. Normally when an additional layer is added it either improves performance or ends up creating an overfitted model. But as the network becomes deeper there comes a point in which performance decreases even when no overfitting has occurred. Instinctively one might argue that this is due to the so-called vanishing gradients problem, which occurs due to how backpropagation calculates gradients as a product of multiple partial derivatives. If certain activation functions with derivatives in the [-1, 1] range are chosen, the gradient will eventually "vanish" as the layers stack on.

This is a sentiment the authors of *Deep Residual Learning for Image Recognition* do not agree with as they note:

> This problem [vanishing gradients], however, has been largely addressed by normalized initialization and intermediate normalization layers, which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with backpropagation [5, p. 1].

and rather identifies these issues as a "degradation" problem.

The network outlined in [5] attempts to solve this issue by introducing the idea of a residual block. This is a type of module consisting of multiple layers, where the output of the module is the result of its input and the output of its final layer. The connections between input and output are called skip connections and are computationally inexpensive. Figure 2.5 illustrates one such skip connection. These modules make up the backbone of the network that is so fittingly called Residual Network—or ResNet for short.

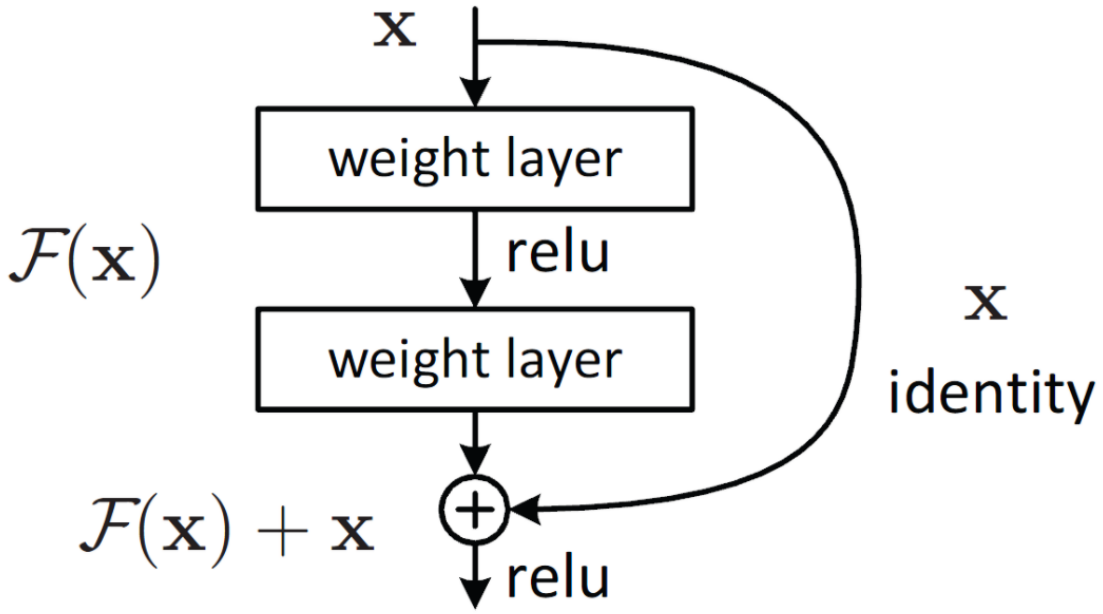


Figure 2.5: Figure from He et al. [5] of a skip connection and the resulting output.

2.2.2 MobileNetV2

MobileNet is a lightweight network architecture made with mobile and embedded devices in mind. The key contribution from its second version [6] is the building

blocks that make up the network, the inverted residual with linear bottleneck. This block takes a low-dimensional representation, upsamples it and filters it with a light-weight depthwise convolution. The output features are then projected back to a low dimensional representation. This process is illustrated in Figure 2.6.

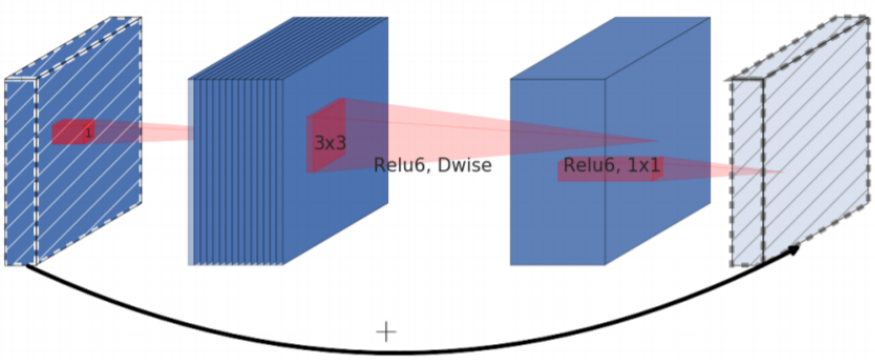


Figure 2.6: Figure from Sandler et al. [6] illustrating the workings of an inverted residual layer.

2.2.3 Encoder-Decoder Network

An encoder-decoder network is a neural network consisting of two components: The encoder and the decoder. The encoder attempts to extract the core features of the input. In the world of computer vision this often means to propagate an input image through convolutional and pooling layers. The decoder uses these features to generate the desired output. The resolution of the input is typically upscaled in a way that mirrors how the encoder downscales the initial input. This is most commonly achieved by the use of transposed convolutional layers. One such encoder-decoder network is shown in Figure 2.7.

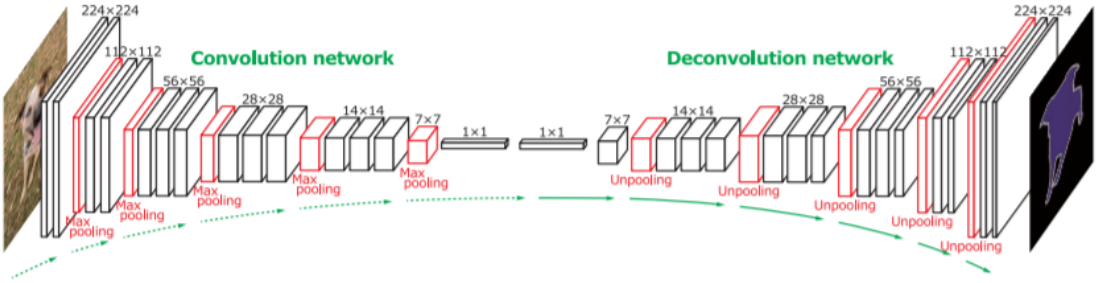


Figure 2.7: Figure from Noh et al. [7] illustrating an encoder-decoder network architecture used for semantic segmentation.

Chapter 3

Related Works

3.1 3DMM - 3D Morphable Model

3DMM is a technique for the synthesis of textured 3D faces, and was first introduced as early as 1999 [8]. Instead of focusing on the individual vertices and their texture values, 3DMMs draw their attention towards general characteristics that make up a human face. To generate a model with certain characteristics—or principal components as they are called—a database of captured 3D face models are utilized. Generation is then simply a case of finding the correct linear combination of models that matches the desired principal component values. Figure 3.1 illustrates different 3DMM generated models.

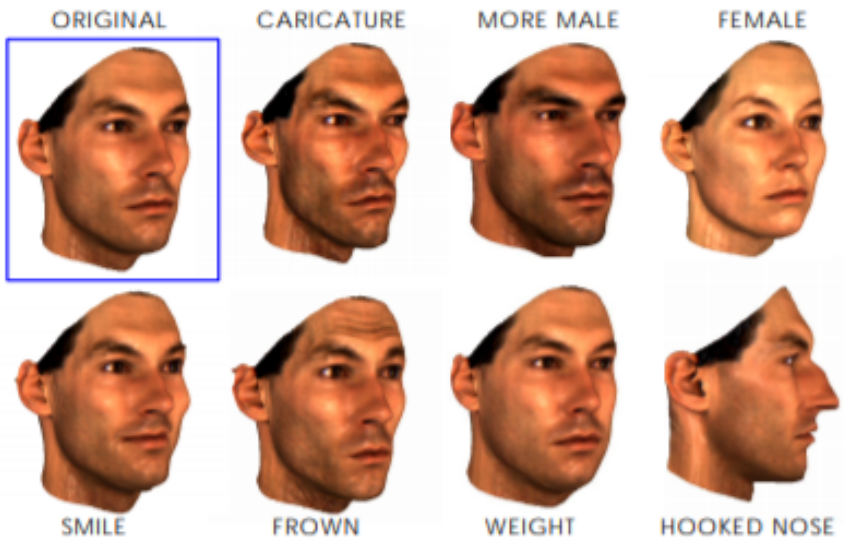


Figure 3.1: Figure from Blanz et al. [8] illustrating the effect of modifying different principal components.

The Basel Face Model 2009 (BFM) [9] is one such 3DMM and is among the ones that have seen most use. BFM describes its models as indicated by Equation 3.1 and 3.2.

$$S = \bar{S} + A\alpha \tag{3.1}$$

$$T = \bar{T} + B\beta \tag{3.2}$$

Where S is the shape, $\bar{S}$ the average shape, A the principle shape components and α the standard deviations of each shape component. Similarly, T is the texture, $\bar{T}$ the average texture, B the principle texture components and β the standard deviations of each texture component.

3.2 300W-LP: Training Dataset

300W-LP is a dataset that contains images across a large range of poses, in addition to their 3D model counterparts [10]. The problem with other datasets is that they for the most part only contain front-facing images, which is problematic for networks that seeks to reconstruct on unconstrained images. 300W-LP looks to amend this by introducing the so-called face-profiling method, which synthetically rotates a face to generate large pose-variants of it. This results in a big dataset across a large range of poses. The paper in question also introduces a way to regress BFM parameters from an image, which is how the ground truth 3D models are represented in 300W-LP. Some example images from this dataset are shown in Figure 3.2.

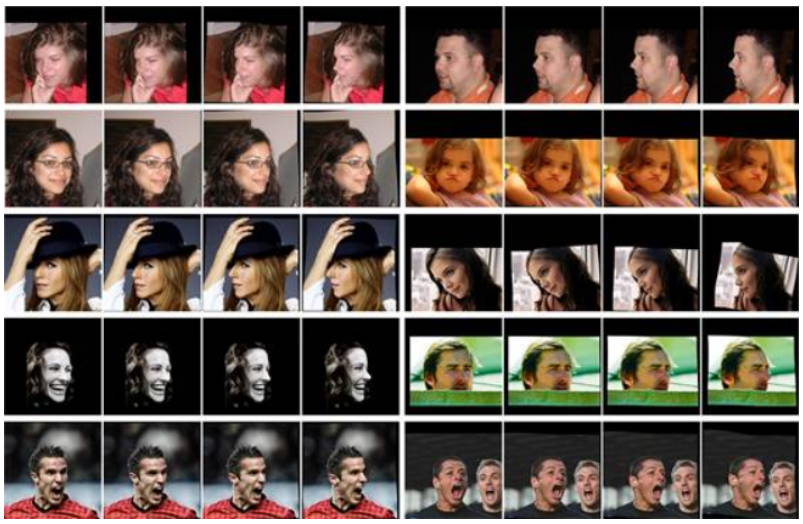


Figure 3.2: Different images from 300W-LP and their corresponding synthetically rotated variants.

3.3 PRNet

In 2018 Feng et al. published *Joint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network* [1] which introduced a new type of network made with face alignment and 3D face reconstruction in mind. This network goes under the name position map regression network—or PRNet for short. It was, and still is, considered the state-of-the-art both in terms of accuracy and inference time, handily beating the competition on both metrics at the time it was published.

One of the key issues these other networks faced was representing 3D faces in a way that could be regressed. One such representation was to concatenate the positions of every vertex into a one-dimensional vector. The issue with this kind of approach is two-fold: For one the spatial adjacency information between vertices is lost, which is something that could be utilized by CNNs and their local receptive fields. The second issue is that predicting a 1D vector requires fully connected layers at the latter part of the network which increases the number of parameters by quite a bit. PRNet is able to avoid both of these issues by using a UV position map as a 3D model representation. This is a 2D map where each integer coordinate represents one vertex on the 3D face model, and the R, G and B value at said position in UV space corresponds to the x, y and z coordinates. Figure 3.3 illustrates one such UV position map.

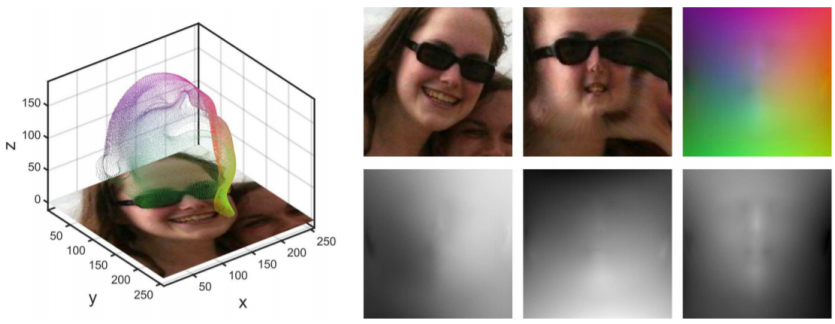


Figure 3.3: Figure from Feng et al. [1]. The image to the left illustrates an image in UV space and its corresponding ground-truth point cloud. The first row of the image to the right shows the initial image, extracted UV texture map and its UV position map. The second row shows the values of the RGB channels on the UV position map.

When considering different ways of representing a face, it is important that the training set—in this case 300W-LP—is also represented in the same manner. The first thing to consider is the size of the UV map, as it should be big enough to fit every vertex of the face model. The UV map size chosen is 256x256 which is more than enough to fit

the 53K vertices in a BFM model. The more important part is how to actually construct a UV position map from a 3D model. This is done by "flattening" the 3D model onto the UV-map using Tutte embedding, which was originally a way of projecting graphs onto a plane. In this case, the graph vertices are the vertices of the 3D model and the plane is the UV position map. An intuitive illustration of this process is shown in Figure 3.4[1].

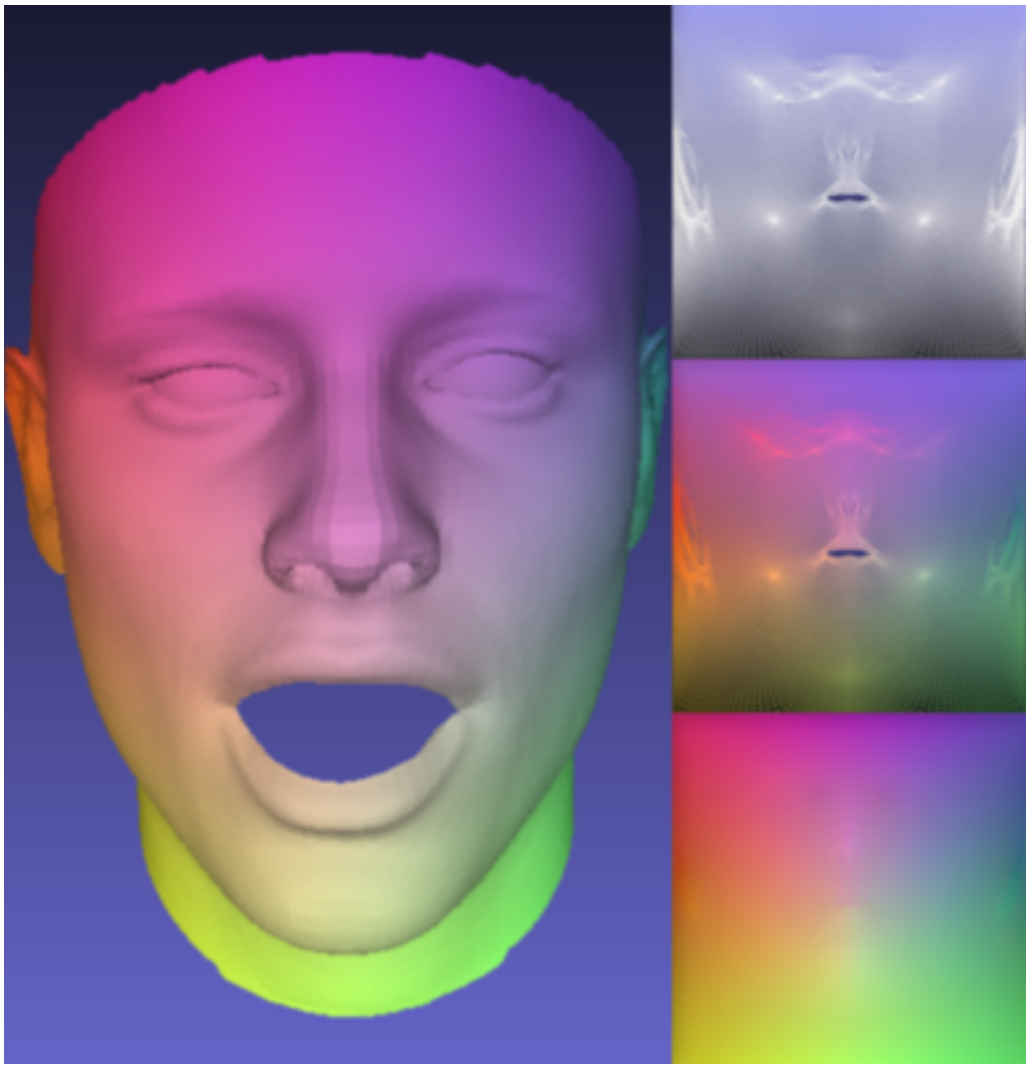


Figure 3.4: On the left a 3D model with colors corresponding to vertex positions. On the right from top to bottom: The flattened model, flattened model where each vertex has an RGB value corresponding to the left picture, "smoothened" UV map by interpolation.

The network itself has an encoder-decoder structure where the encoder and decoder are made out of residual layers and transposed convolutional layers respectively. A

[1] `https://medium.com/@hyprsense/bridging-the-academia-gap-an-implementation-of-prnet-training-9fa035c27702`

simplified illustration of the network structure is illustrated in Figure 3.5. The reason such a structure is used as opposed to a fully convolutional network without any spatial downsampling is simply due to convolutions on large input sizes being computationally expensive, doubly so at the deeper parts of the network where each layer has a greater depth resolution.

Figure 3.5: figure from [1] illustrating PRNet network structure.

3.4 3D Face Reconstruction from Two Images

3D Facial Reconstruction from Front and Side Images is a Thesis written by Lium, Ola [2], and was a successful attempt at increasing the accuracy of PRNet lowering NME by 45%. Although several changes were made to the original network, the key contribution from this Thesis is the use of an image pair consisting of a front- and side-facing image to reconstruct a model from. Intuitively this gives the network a greater understanding of the depth of the face, thus resulting in a more accurate reconstruction. The other change from the original PRNet is replacing the residual blocks with inverted residual blocks. The network architecture is illustrated in Figure 3.6.

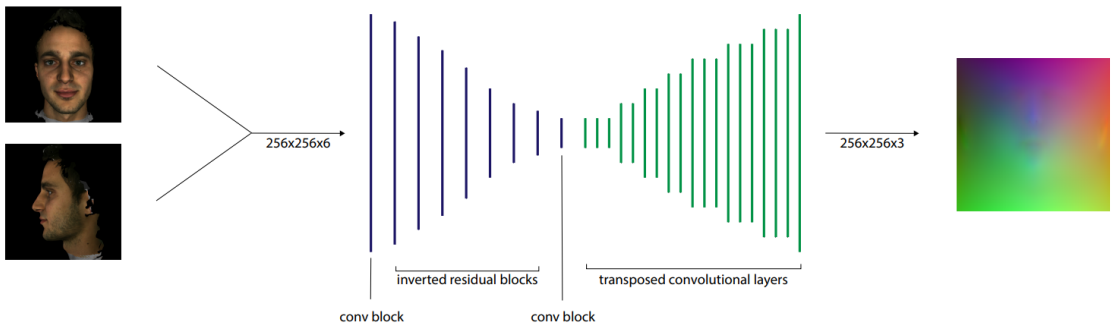


Figure 3.6: Figure from [2] illustrating the network structure.

This method also made some changes to how training was performed. More specifically the network is pre-trained on synthetic data generated from Facegen[2], which is then followed by training on 300W-LP. Synthetic data is a great way of expanding the training set when the existing datasets are lacking, but does come with its own flaws. One problem is that synthetic data will never be as accurate as real, and the error from the synthetic data might propagate to the network itself which is quite undesirable. The trade-off between increased data but decreased data quality needs to be taken into account not only when deciding whether to include synthetic data at all, but also when considering how large the synthetic dataset should be.

The upside of using Facegen is mainly its consistency factor. The program first generates a 3D model to then render into a 2D image. This ensures that there are no discrepancies between the 2D image used as input during training and the ground truth mesh, as no approximations have to be made or data synthesis performed when projecting 3D onto a 2D plane. 300W-LP on the other hand regresses its 3D models from 2D images, and because of this there are bound to be some inaccuracies as the regression function is just an approximation. The same applies to large pose images, where in the case of Facegen models it is just a matter of rotating the 3D model before rendering, whereas for 300W-LP the face profiling function have some obvious imperfections as illustrated in Figure 3.7.

(a) Original. **(b)** Rotated.

Figure 3.7: Images of the same subjects from 300W-LP.

[2] `https://facegen.com/`

3.5 Rotate-and-Render

Rotate-and-Render is a GAN-based unsupervised method for synthetically rotating a 2D image of a face in the range [-90, 90] degrees [3]. The key feature of this network is how it manages to train without any ground truth image to compare its output to. This is quite important as virtually every dataset that has large-pose images are either taken in heavily controlled environments [11], which affects generalization, or are synthetic [10], which propagates the error from dataset to network. One such dataset is illustrated in Figure 3.8.

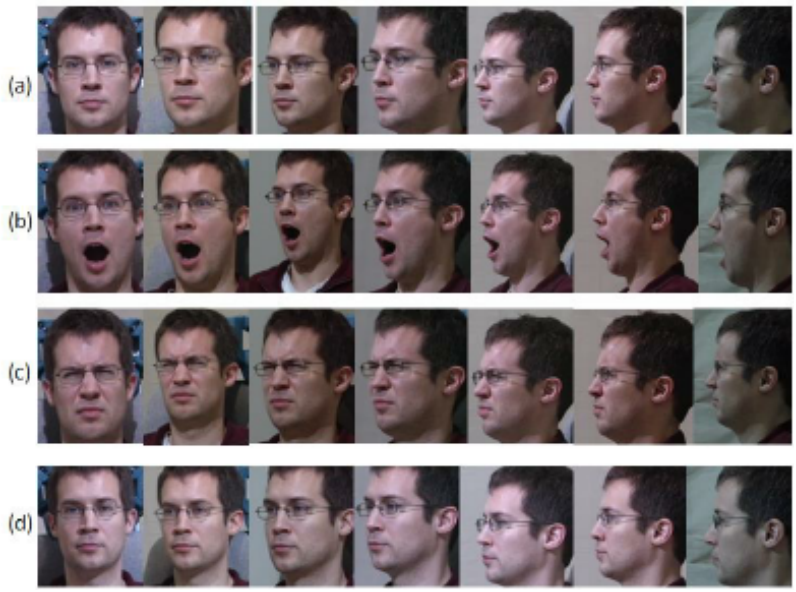


Figure 3.8: Image samples from Multi-PIE [11], a dataset containing large-pose faces.

The training process of this network is as following: Reconstruct the image to a 3D model, rotate it, render it in 2D. When this has been done it reverses the whole process by again reconstructing the image to 3D, rotate it back and render it in 2D. By doing this the model can compare the back-and-forth rotated face to the original image as a form of self-supervision. This process is illustrated in Figure 3.9.

The model achieves very good results both based on metrics and eye-test compared to the competition. The only downside is that it has a decently slow runtime and is very much non-viable for real-time applications on consumer-grade GPUs.

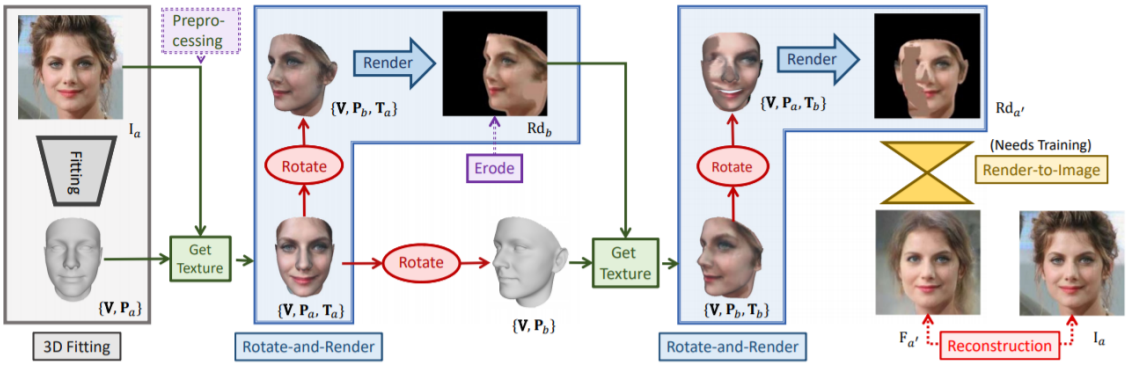
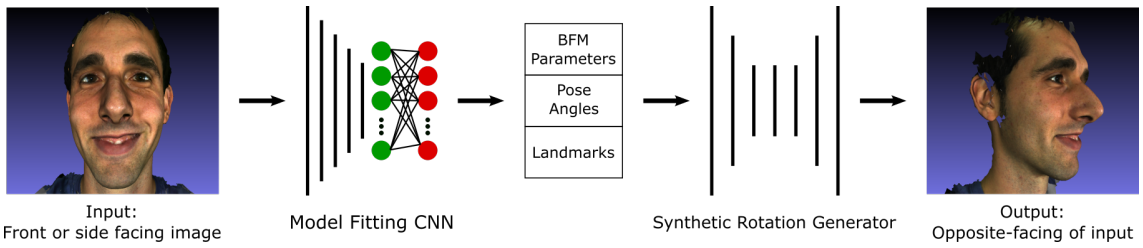


Figure 3.9: Figure taken from Zhou et al. [3] going through the two rotate-and-render operations that make up the pipeline.

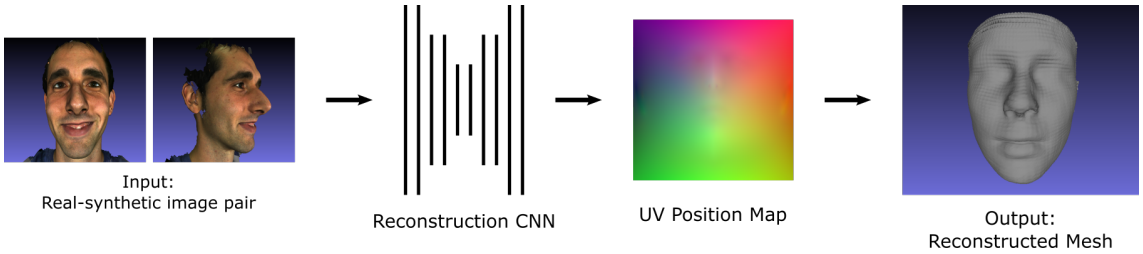
Chapter 4

Methodology

There are two main components that make up the proposed pipeline: The network that rotates a given input image, and the network that reconstructs a 3D face from the real-synthetic input image pair. The way this image pair is fed into the network is by concatenating them depthwise, meaning the input will have dimensions WxHx6, where depth 1-3 are the RGB channels for the real image and 4-6 are the RGB channels for the synthetic image. The width and height can be any value, but it will be resized to 256x256 during pre-processing. As most of the inner workings of these modules have been explained in the previous chapter this chapter will include more of a discussion on what has been done and potential concerns or points of discussion. The proposed pipeline is illustrated in Figure 4.1.



(a) Synthetic rotation pipeline.



(b) Reconstruction pipeline.

Figure 4.1: The proposed pipeline.

4.1 Synthetic Rotation

The idea for this module was inspired by the large-pose synthesis method applied on images in the 300W-LP dataset. But due to its slow inference time, Rotate-and-Render was chosen instead for its good performance and reasonable runtime.

4.1.1 Modifications to Rotate-and-Render

Most of the work on Rotate-and-Render went into enhancing the interoperability between it and the reconstruction system. The way Rotate-and-Render originally works is that it first regresses BFM parameters for a batch of images. Those BFM parameters are then made into a 3D model and rotated in 3D space. Finally this rotated model is rendered back into 2D. These processes are performed separately, which is somewhat of a usability concern as this means that the user would have to manually call multiple processes to get a rotated image, and finally call the reconstruction network to get a 3D face. To amend this the rotation network is modified to do everything in one pass, as well as taking a more object-oriented approach to coding as to make the rotation system more modular and thus fit nicely into the reconstruction pipeline. This results in reconstruction being done in just one call by the user, who only needs to point to the image they want to reconstruct.

In addition to this, a system that automatically determines what direction and how much the input is to be rotated has been implemented. Initially, the user had to manually input the desired yaw-angle of the rotated image. For the purposes of this system, a simple function that automatically rotates from front to side-facing or vice versa is utilized. As long as the pose of the subject is in the range [-90°, 90°] this will net a valid rotation. The amount of rotation can be defined by the user but for this particular system a rotation of 30° has been chosen. Figure 4.2 illustrates the "auto rotation" process.

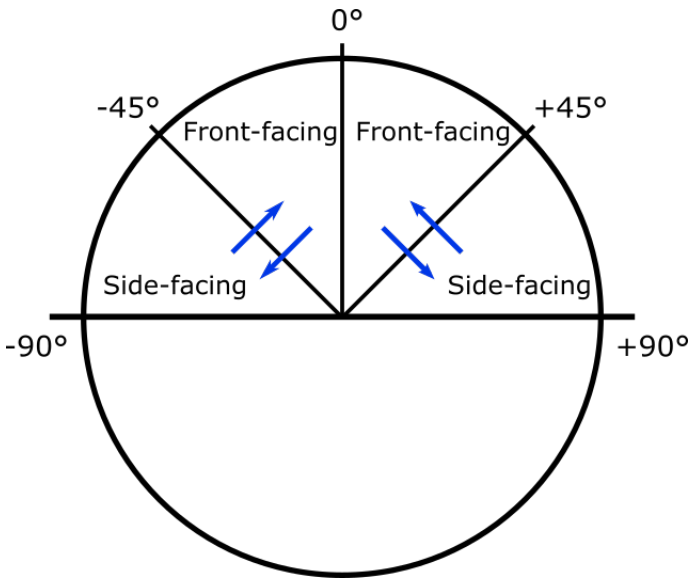


Figure 4.2: Look at a human face from bird's-eye view. The blue arrows indicate what direction the automatic rotation function will rotate the face to, given a yaw angle.

4.1.2 Concerns and Discussion

There are two main concerns regarding this part of the system: Runtime and general redundancy of the system. The first concern is pretty straightforward. The per-image inference time on an RTX 2080TI—a high-end consumer-grade GPU—is around 0.6s, which completely dwarfs the run time of the reconstruction network.

The second concern is less of a real concern and more a feeling that the system is a little "clunky" for lack of a better word. The synthetic rotation network already reconstructs the image once—twice during training— which is then followed by the actual reconstruction network. Which means that the system as a whole performs a perfectly fine reconstruction, just to do an additional reconstruction that is a little more accurate. Simply put, doing reconstruction twice seems a little redundant. Ideally a more elegant solution that has a more cooperative relationship between the rotation network and reconstruction network would be preferred. As of right now they are very much two independent systems that are forcefully put together.

4.2 Reconstruction

While at a first glance not much has changed about this part of the pipeline, there are some fundamental differences from what is described in [2]. First of all instead of using two real images the network will use one real and one synthetic image. Secondly this revised network will try to reconstruct facial hair in addition to the face. Both of these changes are tied closely to the training process, which is made clearer by the fact that no change has been made to the actual network architecture as seen on Table 4.1.

Input	Layer	Kernel	Stride	Output
256 x 256 x 6	Convolution	3	2	128 x 128 x 32
128 x 128 x 32	Inverted Residual	3	-	64 x 64 x 96
64 x 64 x 96	Inverted Residual	3	-	32 x 32 x 144
32 x 32 x 144	Inverted Residual	3	-	16 x 16 x 192
16 x 16 x 192	Inverted Residual	3	-	8 x 8 x 576
8 x 8 x 576	Convolution	3	2	8 x 8 x 512
8 x 8 x 512	Transposed Convolution	4	1	8 x 8 x 512
8 x 8 x 512	Transposed Convolution	4	2	16 x 16 x 256
16 x 16 x 256	Transposed Convolution	4	1	16 x 16 x 256
16 x 16 x 256	Transposed Convolution	4	1	16 x 16 x 256
16 x 16 x 256	Transposed Convolution	4	2	32 x 32 x 128
32 x 32 x 128	Transposed Convolution	4	1	32 x 32 x 128
32 x 32 x 128	Transposed Convolution	4	1	32 x 32 x 128
32 x 32 x 128	Transposed Convolution	4	2	64 x 64 x 64
64 x 64 x 64	Transposed Convolution	4	1	64 x 64 x 64
64 x 64 x 64	Transposed Convolution	4	1	64 x 64 x 64
64 x 64 x 64	Transposed Convolution	4	2	128 x 128 x 32
128 x 128 x 32	Transposed Convolution	4	1	128 x 128 x 32
128 x 128 x 32	Transposed Convolution	4	2	256 x 256 x 16
256 x 256 x 16	Transposed Convolution	4	1	256 x 256 x 16
256 x 256 x 16	Transposed Convolution	4	1	256 x 256 x 3
256 x 256 x 3	Transposed Convolution	4	1	256 x 256 x 3
256 x 256 x 3	Transposed Convolution	4	1	256 x 256 x 3

Table 4.1: Layers of reconstruction network.

4.2.1 Dataset Generation

The dataset used for training is synthetic and generated by Facegen[1]. This is a software for generating 3D face meshes. To achieve generation of varied faces Facegen utilizes a custom 3DMM consisting of 273 3D scans[2].

The specifics of Facegen data generation is as follows: First a random face is generated, that is, a set of Facegen 3DMM parameters. These are then applied to the base BFM mesh to generate a 3D model. This model will have the shape and texture as defined in the face previously generated but will also have BFM mesh topology, i.e same vertex count, same indices etc. Each Facegen model—or subject—in the dataset is rendered at yaw angles [-90°, ±67.5°, ±45°, ±22.5°, 0°] plus a random noise factor sampled uniformly from the range [0, 22.5]. This is illustrated in Figure 4.3. Each rendering also has a random roll and pitch pose in the range [-25°, 25°].

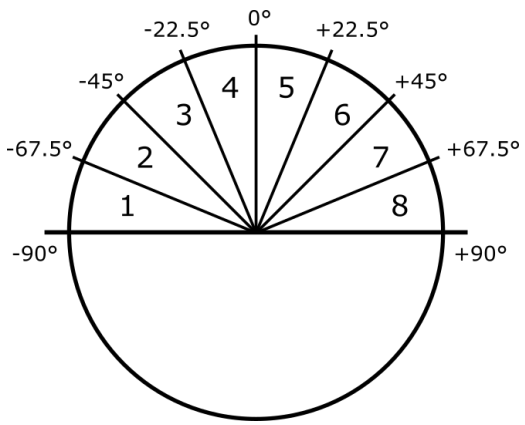
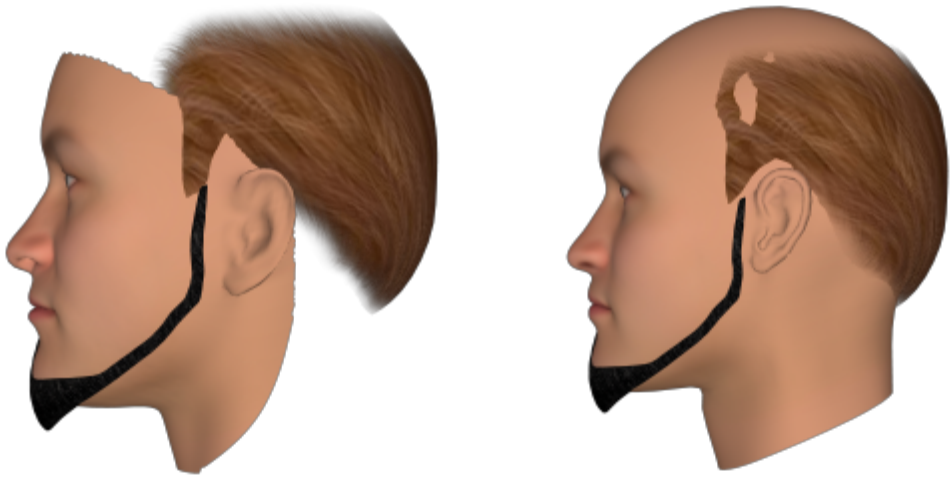


Figure 4.3: Look at a human face from bird's-eye view. Each number indicates the yaw-angle range for each of the eight renderings.

When rendering this way there were a few obvious improvement areas. First of all, while BFM meshes are fairly detailed it also only models the identifiable parts of the human face, that is, the front. This might not be a problem for front-facing images, but for large poses this leads to rendered images looking fairly strange as part of the back and side of the head is not modeled. Secondly, since the top of the head is missing from these models it can cause a situation where hair looks like it is floating depending on the hairstyle, since it is meant to be sitting at the top. Both of these cases are illustrated in Figure 4.4a.

[1] `https://facegen.com/`
[2] `https://facegen.com/faq.htm`



(a) BFM mesh topology. **(b)** Alternative mesh topology.

Figure 4.4: Large pose rendering of the same Facegen subjects with two different mesh topologies.

To fix these issues an alternative rendering method was used. Instead of using a BFM mesh for rendering, another mesh topology that models the whole face employed, which can be found in Figure 4.4b. The BFM model will still need to be generated though as it is still used to generate the ground truth UV position map. Because of this, the new base mesh is aligned with the BFM base mesh, so that the position of the vertices on the rendered image is the same as in the UV position map.

Initially the idea was also to include the 300W-LP dataset into the training process, but was opted out of for two reasons: First of all the synthetic rotation network had issues recognizing some of the larger-pose faces in 300W-LP due to their synthetic nature. This is especially bad since a face needs to be detected as part of its model-fitting step to generate landmarks and BFM parameters. This would have made the 300W-LP part of the dataset biased towards front-facing images and negatively affected the reconstruction network's level of generalization.

It is worth mentioning that this issue is somewhat relieved by the synthetic rotation system. This is due to it generating a rotated image, allowing the reconstruction network to receive both a front and side facing image anyway. With that being said this would still train the reconstruction network overwhelmingly in the direction of receiving a real front facing image and a synthetic side facing one as input, which certainly is not the case for images-in-the-wild.

The second reason 300W-LP is not included in training is simply that the synthetic rotation network perform somewhat poorly on large-pose images in this dataset. This is most likely due to these having been synthetically rotated once already.

In the end the extra data gained from adding 300W-LP was not considered worth increasing the bias towards frontal images in addition to decreasing the quality of large-pose images.

4.2.2 Pre-processing

In the broadest sense pre-processing means to do some sort of transformation to data before it is used in the main process. In machine learning a pre-processing step most commonly occurs before training, but there can also be a separate pre-processing step when the network is used outside training., which is the case for this particular network

The objective of the pre-processing step differs depending on whether it is ran during training or testing. When running the network outside training the aim would be to generate the input that corresponds to the best possible results. This is not necessarily the case during training, where the pre-processing step would gladly generate a sub-optimal input if it results in the best performing network when training is finished. That being said the input during training should generally mirror the ones from real-life application to provide consistency.

Cropping

One of the transformations that is shared between training and testing is that of cropping, although it is especially important during the latter. When running the network on images in-the-wild the size and position of the face on the image might vary wildly. Cropping ensures that the network can always expect a centred face of a certain scale as input. In contrast, if cropping is not performed the network would have to account for variations in position and scale of the face, in addition to actually performing the reconstruction. This is quite undesirable as it would add a lot of complexity to the problem.

Another benefit of cropping is that it would make most of the input consist of the actual face. It is essential that the input contains as much relevant information as pos-

sible especially on such limited resolution. It could be argued that this is a moot point as images are very much able to have faces that are contained in 256x256 pixels or more. If the face is smaller than this upsampling would be performed, where the extra pixels added from this would not be "real" data but rather interpolated. That being said if cropping is not performed the network would have to take as input a higher resolution image to have the same performance as a 256x256 network with cropping.

To perform the cropping a lightweight face recognition network included in the dlib[3] library is used. One might argue that if the reconstruction network increases in complexity it might be able to do the work of the recognition network as well as perform reconstruction in one pass, but there are good reasons for not doing this.

First, creating datasets for two networks with well-defined tasks is far easier than that of a single complex structure. This is especially the case for the recognition network, as it would be limited to using only typical 3D-face datasets when there is an immense amount of non-synthetic data for recognition out there such as CelebA [12], VGGFace2 [13], UMDFaces [14] and plenty of others.

Second, training with a single loss-function for what is essentially two different tasks might make it unclear what is working and what is not. It would be hard to tell exactly how well or poorly a network is able to recognize faces or reconstruct based on a single loss and/or evaluation of the output. One could say that this does not matter as it is incredibly difficult to understand what a neural network is doing "under the hood", and most likely would not be doing something as straightforward as face recognition into reconstruction separately. The point still stands that allocating the tasks between two networks gives a clearer idea of potential issues as an intermediate output is available.

For this particular network only the non-synthetic image is cropped. This is due to the face detector having a difficult time recognizing faces on synthetically rotated images. It is also worth mentioning that the output from the synthetic rotation network is fairly constant in the sense that it is always centred and more or less the same scale. Because of this there is no real performance loss from not cropping the synthetically rotated image.

[3] http://dlib.net/

Data augmentation

Data augmentation is a group of regularization techniques where some alteration, such as rotation or noise injection, is applied to the training data. The intuition is that applying one or more transformations to the original input image generates a brand new input. This in turn expands the dataset which results in better generalization.

An alternative point of view is that data augmentation is detrimental to the training of a network since it dilutes the dataset with synthetic data. As mentioned in Section 3.4 this might be one of the biggest points of concern when dealing with synthetic data like this. Luckily, the most commonly used data augmentation techniques are generally very basic and mimic real life situations well.

For this particular network two data augmentation techniques are applied, the first of which is colour channel scaling. In computer graphics the most widely used method of representing colour is the RGB model. This is a model where each colour is denoted as a linear combination of red, green and blue. Accordingly, an image in RGB representation will have three colour channels, each comprised of the weights for R, G and B values for each individual pixel. Colour channel scaling simply means to multiply each of these channels with a separate scalar. For this implementation the scalar is random and uniformly sampled from the range $[0.6, 1.4]$. An illustration of what this might look like can be found in Figure 4.5a. In practice what colour channel scaling achieves is modifying the overall colour of the image without altering the shapes.

Ideally a 3D reconstruction network is able to regress a model based on the contour and shapes of a human face. This is exactly why colour channel scaling is such a good fit in this particular instance. Due to its shape-preserving nature it makes sure that no discrepancies between input and ground truth are present. It also penalizes reliance on colours, increasing generalization between different lighting conditions and camera settings.

The other augmentation that is applied is called dropout and might not be as intuitive as the former. What dropout essentially does is it ignores certain neurons during the training process. The neurons that get ignored changes with each iteration of training. Dropout is often applied inside a neural network at different layers during a forward pass, although in this case it is only applied on the input layer. Figure 4.5b illustrates the application of dropout on an image. Dropout leans heavily into the definition of regularization that defines it as reducing the coefficients of certain input parameters

of the model. Intuitively this means that the model will learn not to be too fixated on specific pixels but rather the image as a whole.

(a) Colour channel scaling. **(b)** Dropout.

Figure 4.5: Images of the same subjects from 300W-LP with differing augmentations.

4.2.3 Training Pipeline

Training of the reconstruction network is pretty standard, and goes through the various steps mentioned in this section. The network starts training with pre-trained weights from [2], which is trained on Facegen models and 300W-LP. Then the image is cropped, synthetically rotated and the ground truth UV-position map is calculated from the ground truth BFM Facegen mesh. The resulting input image pair and ground truth UV map is saved for training. Colour channel scaling and dropout is applied to the input image pair right before loss and gradients are calculated during training. The actual training process itself is fairly standard, using mean squared error as a loss function.

One of the biggest dilemmas surrounding this process was whether to synthetically rotate after every augmentation had been applied or not. The former would definitely be more realistic, but there is a big downside to this approach which is that the rotation network is fairly slow, and since augmentations are applied on a per-epoch basis it would mean that the rotation network would have to be ran at a rate proportional to the number of epochs. Thus the decision was made to only run the rotation network once for every image and save the results.

4.2.4 Concerns and Discussion

One of the main points of improvement as noted in [2] is the addition of facial hair onto the synthetic data generated by FaceGen. The issue with this approach is that Facegen does not come with a great variety of facial hair options for their models, only eight in total. With this few options it is hard for the network to properly learn how to model facial hair, if at all. There definitely is potential as several real-life use cases, especially in video games such as Football Manager[4] and Skyrim[5], are able to model a great variety of facial hair using Facegen. But sadly those resources were not available for this particular project.

In the same vein there is doubt of whether Facegen can be the sole dataset that the network is trained on and still generate good results. In [2] the network is pre-trained on a Facegen dataset before going over to 300W-LP. The performance of this pre-trained model definitely left something to be desired as it was not able to out-perform PRNet.

There is also a worry that the Facegen 3DMM might not be able to model all types of faces equally as well. Looking at the demographics of people that were captured to create the Facegen 3DMM, it seems as if some have a far higher level of representation. Histograms illustrating these demographics are listed in Figure 4.6.

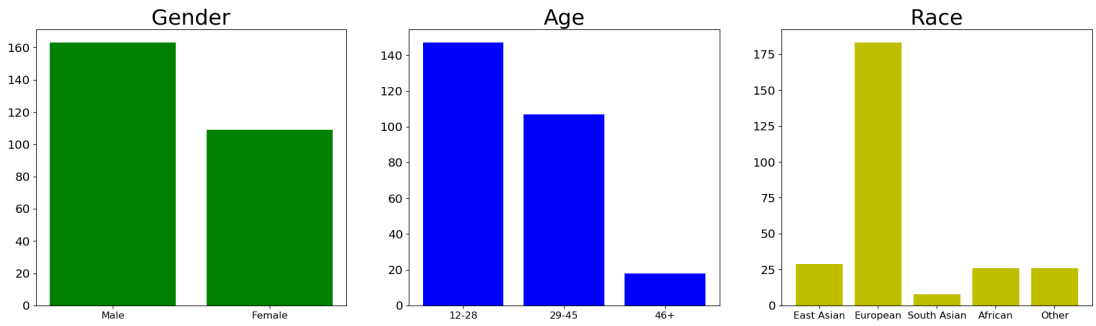


Figure 4.6: Gender, age and race of Facegen 3D scans.

Usually when discussing data bias the concern is that certain demographics or characteristics are overly favoured in the dataset. This is not actually the case for the Facegen dataset, as the generated subjects have a completely random gender, age and race, all sampled uniformly. The issue in this particular case is that it might not be able to model every demographic adequately. When looking at the low amount of non-Europeans re-

[4] `https://www.footballmanager.com/#desktop`

[5] `https://elderscrolls.bethesda.net/en/skyrim`

cruited for this 3DMM there is a real concern that the principal component analysis is not able to capture the characteristics of those particular races properly. The same goes for older people and to a lesser extent females. This can create a discrepancy in data quality between subjects depending on demographics. Worst-case, it might lead to the reconstruction network not being able to model certain people as well as others.

Chapter 5

Results

5.1 Synthetic Rotation

The modified Rotate-and-Render seems to be working as well as the original version. The network performs best when the input image is a slightly rotated front-facing image, as this is the angle that least occludes the parts of the face that are included in the synthetic rotation. The network also seems to gain performance the closer pitch rotation is to zero. This is most likely due to the fact that the network sets pitch rotation to zero degrees for its output, and as a result the more rotation the original image has in this direction the more the network has to rotate. A few example images are shown in figure 5.1.

Figure 5.1: Synthetic rotation of images in 300W-LP.

5.2 Reconstruction

5.2.1 Evaluation Dataset

The evaluation dataset chosen is the MICC Florence dataset [15], consisting of 2D videos and high-resolution 3D scans of 53 subjects. This dataset is a good way of testing face reconstruction from various angles to simulate usage in-the-wild. It is worth mentioning that some subjects do have facial hair, and this is accurately portrayed in the 3D models as well. This means that a network that is trained on data without facial hair will have significantly higher error than is expected.

Each Florence subject is rendered in total 27 times, at yaw angles [±80°, ±60°, ±40°, ±20°, 0°] each for three different pitch angles [-15°, 0°, 15°].

5.2.2 Evaluation Metric

The goal of a metric is to measure performance in an accurate an robust manner, and for the task of 3D face reconstruction such a metric will calculate the deviations between ground truth and generated model. To this end the metric that was chosen was the normalized mean error (NME) of the euclidean distance between vertices of the reconstructed and ground truth meshes:

$$NME = \frac{1}{N}\sum_{i=1}^{N}\frac{\|(p_i - q_i)\|_2}{d} \tag{5.1}$$

Here d denotes the bounding box size of the generated mesh, and N the number of vertices. The NME is calculated for every mesh, then averaged out as follows:

$$meanNME = \frac{1}{K}\sum_{j=1}^{K} NME_j \tag{5.2}$$

Where K is the number of meshes that has been evaluated.

5.2.3 Evaluation Performance

Two different models are evaluated, as well as PRNet for comparison. These models are as follows: Model using weights from [2] and the model trained as described in Section 4.2.

Evaluation is performed by reconstructing a face, then aligning it with the ground truth model using ICP. The results are illustrated on Figure 5.2, while NME values can be found on Figure 5.3 and Table 5.1.

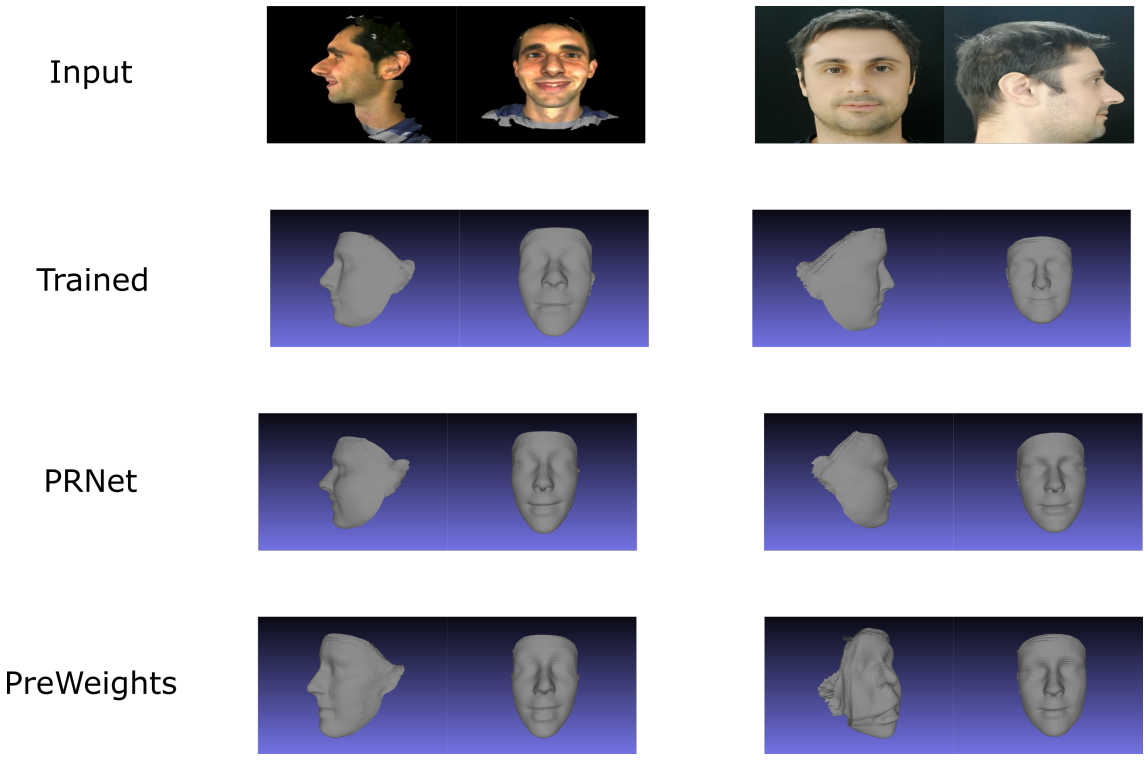


Figure 5.2: Reconstruction results from different models. Trained" is the proposed method trained as described in 4.2, "PreWeights" is the proposed method using only the weights from [2]. Image to the left is from MICC Florence while the one on the right is from a proprietary Norwegian University of Science and Technology dataset.

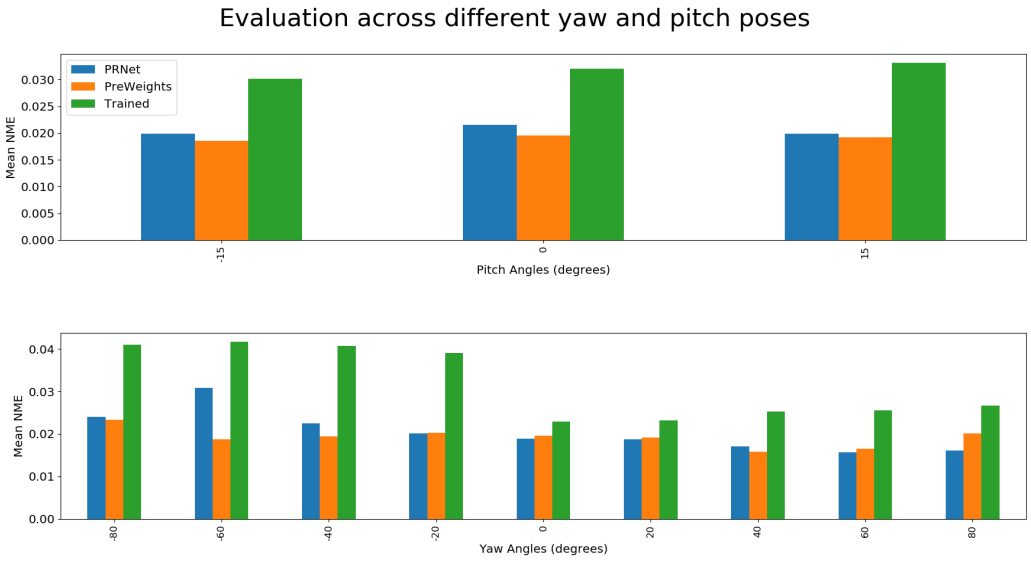


Figure 5.3: Mean NME of different models at different angles.

	Mean NME
PRNet	0.02043
PreWeights	0.01908
Trained	0.0.03179

Table 5.1: Mean NME for various models.

Surprisingly, the best performing model was not the one that was trained specifically with synthetic rotation in mind, but rather the one trained for the purposes of [2]. There were a few issues with the synthetic dataset that led to such poor performance. First of all it seems as if the synthetic data is simply not realistic enough to provide top-of-the-line results. As mentioned in Section 4.2.4 this was a real concern with the network as [2] had illustrated sub-par performance when only training on a Facegen generated dataset.

The poor level of detail on Facegen subjects propagates to other parts of the system as well. The synthetic rotation network seemingly had very inconsistent results, depending on the subject composition and pose. This is illustrated in Figure 5.4. Synthetic rotation has a tendency to generates results such as the one on Figure 5.4b when the pose gets large, but this is not nearly as frequent on real data than it is on the Facegen dataset. This creates a discrepancy between the data it is trained on versus the data it is tested on.

(a) Original. **(b)** Rotated. **(c)** Original. **(d)** Rotated.

Figure 5.4: The same subject from two different poses and its synthetic rotation. On the left an unnatural result and on the right a "proper" result.

Lastly, the model was not able to reconstruct facial hair properly. A big reason for training on the Facegen dataset in the first place was training the model to include facial hair into its predictions, as existing datasets for 3D face reconstruction does not include facial hair in their ground truth meshes. Because of a lack of variety and real-

ism on the facial hair meshes provided by Facegen, it seems the model was not able to learn to model these properly. In which case the inclusion of facial hair might have been detrimental to the performance of the network, as a non-optimal network might have been chosen after training had concluded due to it performing better on subjects with facial hair, even though it did not possess the ability to actually model facial hair.

The fact that the "PreWeights" model was able to perform better than PRNet, even if only slightly, demonstrates the capability of the system if trained on "real" datasets such as 300W-LP. Especially since it is prone to some high-error outliers as is illustrated on Figure 5.2 on the bottom row, third column. It is believed that going forward this would be be the preferred training dataset for this system. To circumvent the issues detailed in Section 4.2.1 it is thought to be better to not perform synthetic rotation during training and instead use one front -and side-facing image from 300W-LP as an input pair to the system. The image that is supposed to be the synthetically rotated image in the image pair would need to be modified to better mimic the output of the synthetic rotation network. A few examples would be: Using the same background colouring method as Rotate-and-Render, setting pitch pose to zero, cropping/scaling the image in the same way Rotate-and-Render would. The synthetic rotation network could also be modified to better mimic the data from 300W-LP. The most notable changes would be to retain the pitch pose when rotating instead of setting it to zero and cropping the output image in the same way data from 300W-LP would be cropped.

5.3 Concerns and Discussion

5.3.1 MICC Florence

During evaluation it is important that the dataset is as detailed and realistic as possible, but also that it represents the general populace in a proper manner. The concern with MICC Florence is that certain demographics are over-represented while some are non-existent. Histograms illustrating the demographics of the evaluation dataset are found on Figure 5.5.

The issue with data bias on an evaluation set is that it will not be able to capture the proper level of generalization for the models that it evaluates. A model that only performs well on Caucasian's would seemingly get great results when evaluated on this particular dataset, even though it would have bad generalization across races. This in turn might create an inaccurate picture of the performance of a model.

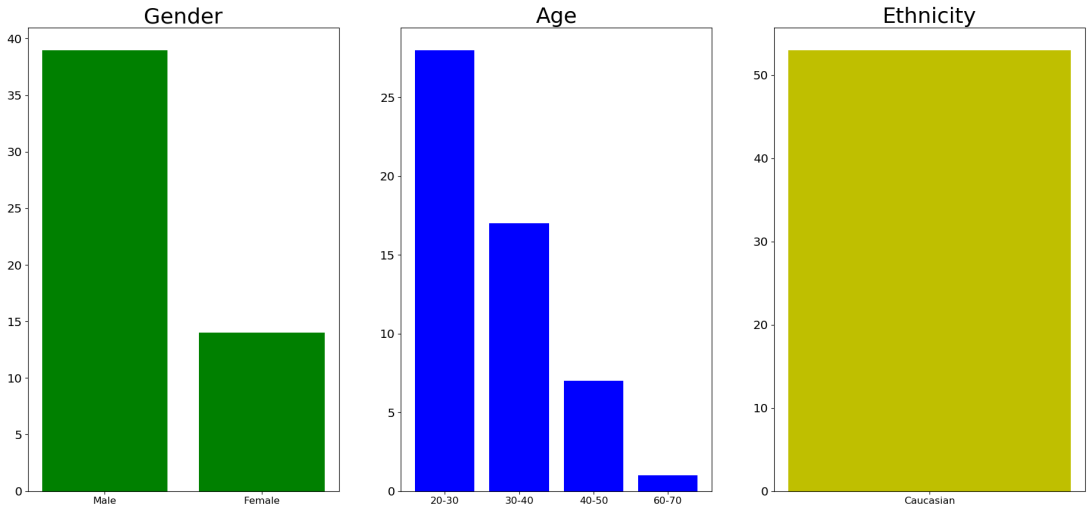


Figure 5.5: Demographic of subjects captured in the MICC Florence dataset.

Another issue with this dataset is that there is no set standard for rendering the models. This means that there might be deviations between the evaluation found between different experiments. As an example; the NME of PRNet in this particular Thesis is quite a bit lower than what was found in [1]. The differences in rendering that may cause these discrepancies might be related to lighting, position of the face on the image, scale etc. And since the models are not aligned to look straight forward the rendering of meshes at certain poses requires some approximations, which introduces another potential error. Since every model is evaluated on the same renderings per experiment this is not necessarily a problem. But could cause some confusion when comparing with other experiments.

5.3.2 NME

There is some uncertainty regarding the metric for evaluating the different models. As seen in [2] the network gets a mean NME that is close to half that of PRNet, but when looking at the 3D model produced by PRNet and [2] the results are not that different. Although the "eye-test" is not the most accurate of metrics there is still some value to be gained from such evaluation. One alternative metric that could be considered is to test different networks on face recognition models that utilize 3D models, and compare the resulting accuracy.

5.3.3 ICP Alignment

As previously mentioned, ICP is used to align the ground truth 3D mesh to the reconstructed face so the NME can be properly calculated. The issue with ICP is that it is just an approximation, as there is no "correct" alignment between two different meshes. This might cause some strange alignments, doubly so if there are significant deviations between the meshes. If ICP is not able to align the meshes properly it might lead to some egregious outliers when calculating NME, which might negatively affect the perceived performance of the network.

5.3.4 Runtime

The runtime of the network is on average 0.6 seconds per image on an RTX 2080TI. This makes it completely unviable for applications such as videos where a stream of images need to be reconstructed in real time. Even with the release of a new generation of consumer-grade GPUs it seems unlikely that this system will ever manage true real-time use.

Chapter 6

Conclusion

6.1 Conclusion

The method proposed in this Thesis showed that synthetic data generation is not yet at the level where it can match real datasets. While the results were not spectacular it managed to measure a slightly lower NME than PRNet using weights from another network. This illustrates the true potential of multi-view synthesis in 3D face reconstruction, and with further work a more impressive showing might be expected.

6.2 Further Work

Further work should first and foremost focus on the improvements outlined in Section 5.2.3. In addition to this the outliers found during testing should be looked at closer. The hypothesis is that this is due to inadequate synthetic rotations as was illustrated in Figure 5.4b.

Outside of this the most obvious improvement area would be reducing the runtime. Since the majority of the processing time is used by the synthetic rotation network it would make sense to look for ways to improve this part of the system. A simple modification would be to change the model-fitting network from 3DDFA [10] to something that has a lower inference time. 3DDFA_V2 [16] or even PRNet would net some improvement on this end. But given how much improvement would be required to make the system real-time viable, some drastic changes would have to be made. Whether that be to switch to a more lightweight synthetic rotation network, or try to create a system from the ground up that aims to rotate and reconstruct in one seamless pass.

Appendix A

Facegen Dataset Generation

```
def generate_facegen_dataset(dataset_root_folder, save_root_folder, file_list, last_end):
    bfm = MorphabelModel('Data/BFM/BFM.mat')
    uv_coords = face3d.morphable_model.load.load_uv_coords('Data/BFM/BFM_UV.mat')
    uv_coords = process_uv(uv_coords)

    synthesizer = Synthesize()
    image_h, image_w = 256, 256

    num_subjects = len(os.listdir(dataset_root_folder))
    with open(file_list, 'a') as f:
        for i in range(1+last_end, num_subjects+1):
            subject_number = str(i)
            print(f"subject␣{subject_number}")
            obj_fp = os.path.join(
                dataset_root_folder,
                subject_number,
                f"subject_{subject_number}.obj"
            )
            save_folder = os.path.join(save_root_folder, subject_number)
            for i in range(-4, 4):
                if not os.path.exists(save_folder):
                    os.makedirs(save_folder)
                current_img_fp = os.path.join(
                    dataset_root_folder,
                    subject_number,
                    f"render_{i}.png"
                )
                img_save_fp = os.path.join(save_folder, f"orig_{i}.png")
                rotated_img_fp = os.path.join(save_folder, f"rotated_{i}.png")
                posmap_fp = os.path.join(save_folder, f"posmap_{i}.npy")
                cropping_tform = generate_posmap_facegen_bfm(
                    bfm,
                    uv_coords,
                    current_img_fp,
                    obj_fp, posmap_fp,
                    save_image=True
                )
```

```
                current_img = io.imread(current_img_fp)
                current_img_cropped = skimage.transform.warp(
                    current_img,
                    cropping_tform.inverse,output_shape=(image_h, image_w),
                    preserve_range=True)
                img = apply_random_background(current_img_cropped.astype(np.uint8))
                io.imsave(img_save_fp, img, check_contrast=False)

                try:
                    synthesizer.synthesize_image(img_save_fp, rotated_img_fp)
                except TypeError:
                    print("bricked"+img_save_fp)
                    continue


                f.write(img_save_fp + ' ' + rotated_img_fp + ' ' + posmap_fp + '\n')
```

Appendix B

MICC Florence Rendering

```
from operator import pos
import os
import shutil
# switch to "osmesa" or "egl" before loading pyrender
os.environ["PYOPENGL_PLATFORM"] = "egl"

import numpy as np
import math
import sys
np.set_printoptions(threshold=sys.maxsize)
import pyrender
import trimesh
import glob
from image_synthesis.model_fitting.model_fitting import load_3ddfa, get_param
from image_synthesis.model_fitting.utils.estimate_pose import angle2matrix
from skimage.io import imsave
import argparse
from math import sqrt

def get_rotation_matrix(rotation_directions):
    rotation_matrix = angle2matrix(rotation_directions)
    col4 = np.array([
        [0.0],
        [0.0],
        [0.0]
    ], dtype=np.float32)
    rotation_matrix = np.hstack((rotation_matrix, col4))
    row4 = np.array([0.0, 0.0, 0.0, 1.0], dtype=np.float32)
    rotation_matrix = np.vstack((rotation_matrix, row4))
    return rotation_matrix

def scale_triplet(triplet, scalar):
    return (triplet[0]*scalar[0], triplet[1]*scalar[1], triplet[2]*scalar[2])

def negate_pose_list(pose_list):
    return (-pose_list[0], -pose_list[1], -pose_list[2])
```

```
def calculate_score(triplet):
    score = [abs(triplet[0]), abs(triplet[1]), abs(triplet[2])]
    scale = [10.0 if abs(pose)>0.2 else math.ceil(50.0*abs(pose)) for pose in triplet]
    scale = list(map(lambda x: (x+(x%2))/10.0, scale))
    return score, scale

def rotate_mesh(mesh_fp, save_fp, rotation_direction, yfov):
        with open(mesh_fp) as f:
                mesh_lines = f.readlines()
        aligned_mesh_file = open(save_fp, "w")
        xmax = 0
        xmin = 0
        ymax = 0
        ymin = 0
        z_list = np.array([])
        y_acc = 0
        n_vertex = 0
        rotation_matrix = angle2matrix(rotation_direction)
        for line in mesh_lines:
            if line.startswith("v␣"):
                n_vertex += 1
                vertex = list(map(float, line.split("␣")[1:]))
                rotated_vertex = rotation_matrix.dot(np.array(vertex))
                y_acc += rotated_vertex[1]
                xmax = rotated_vertex[0] if rotated_vertex[0] > xmax else xmax
                xmin = rotated_vertex[0] if rotated_vertex[0] < xmin else xmin
                ymax = rotated_vertex[1] if rotated_vertex[1] > ymax else ymax
                ymin = rotated_vertex[1] if rotated_vertex[1] < ymin else ymin
                z_list = np.insert(
                    z_list,
                    z_list.searchsorted(rotated_vertex[2]),
                    rotated_vertex[2]
                )
                aligned_mesh_file.write(f"v␣{'␣'.join(map(str,␣rotated_vertex))}\n")
            else:
                aligned_mesh_file.write(line)
        aligned_mesh_file.close()
        height = (ymax-ymin)
        z_offset = z_list[round(n_vertex*0.85)]
        distance = (height/(2*np.tan(yfov)))
        xmid_point = xmin+((xmax-xmin)/2.0)

        return (distance, z_offset), xmid_point, ymin+(height/2.0)

parser = argparse.ArgumentParser(description='Generate␣png␣from␣micc␣mesh.')
parser.add_argument('-m', '--mode', default='gpu', type=str, help='gpu␣or␣cpu␣mode')
parser.add_argument('--bbox_init', default='two', type=str,
                            help='one|two:␣one-step␣bbox␣initialization␣or␣two-step')
parser.add_argument(
    '--path_prefix',
    default='/lhome/yongbk/eval_florence/Original',
    type=str,
    help='location␣of␣folder␣containing␣the␣meshes'
```

```
)

args = parser.parse_args()
model, alignment_model = load_3ddfa(args)

class Subject:
    def __init__(self, render, yfov, subject_number):
        subject_number_str = \
            f"0{str(subject_number)}" if subject_number<10 else str(subject_number)
        self.mesh_directory_path = \
            f"{args.path_prefix}/subject_{subject_number_str}/Model/frontal1/obj"
        mesh_fp = [fn for fn in glob.glob(f"{self.mesh_directory_path}/*.obj")
            if not os.path.basename(fn).startswith('aligned') and
            not os.path.basename(fn).startswith('rotated')]
        print(mesh_fp)
        self.render = render
        eval_img_dir = f"{self.mesh_directory_path}/eval_img"
        if os.path.exists(eval_img_dir):
            shutil.rmtree(eval_img_dir)
        os.mkdir(eval_img_dir)
        self.yaw_step_size = np.radians(20)
        self.pitch_step_size = np.radians(15)
        self.yfov = yfov
        self.save_fp_prefix = f"{eval_img_dir}/subject_{subject_number_str}"


        score = [1.0, 1.0, 1.0]
        scale = [1.0, 1.0, 1.0]
        first_iteration = True
        self.aligned_mesh_fp = f"{self.mesh_directory_path}/aligned.obj"
        current_pose = (0.0, 0.0, 0.0)
        while not all(x <= 0.01 for x in score):
            if first_iteration:
                distance, xmid_point, self.cam_height = \
                    rotate_mesh(mesh_fp[0],
                    self.aligned_mesh_fp,
                    negate_pose_list(current_pose),
                    yfov
                )
                first_iteration = False
            else:
                distance, xmid_point, self.cam_height = \
                    rotate_mesh(
                        self.aligned_mesh_fp,
                        self.aligned_mesh_fp,
                        negate_pose_list(scale_triplet(current_pose, scale)),
                        yfov
                    )
            self.render.change_mesh(self.aligned_mesh_fp)
            self.render.change_camera_pose(xmid_point, self.cam_height, distance)
            current_pose = self.render(f"{self.save_fp_prefix}_0_0.jpg", False)
            score, scale = calculate_score(current_pose)
            print(current_pose)
```

```
            print(scale)
        self.render(f"{self.save_fp_prefix}_0_0.jpg")
        print("align done")


    def render_subject(self, pitch_step, yaw_step):
        yaw_target = yaw_step*self.yaw_step_size
        pitch_target = pitch_step*self.pitch_step_size
        rotated_mesh_fp = f"{self.mesh_directory_path}/rotated.obj"
        save_fp = f"{self.save_fp_prefix}_{pitch_step}_{yaw_step}.jpg"
        current_rotation_direction = (
            yaw_target,
            pitch_target,
            0.0
        )
        distance, xmid_point, self.cam_height = rotate_mesh(
            self.aligned_mesh_fp,
            rotated_mesh_fp,
            current_rotation_direction,
            self.yfov
        )
        self.render.change_mesh(rotated_mesh_fp)
        self.render.change_camera_pose(xmid_point, self.cam_height, distance)
        self.render(save_fp)

        return save_fp, current_rotation_direction


class Render:
    def __init__(self, yfov):
        self.r = pyrender.OffscreenRenderer(256, 256)
        self.camera_node = \
            pyrender.Node(camera=pyrender.PerspectiveCamera(yfov=yfov, aspectRatio=1.0))
        light = pyrender.DirectionalLight(color=[1,1,1], intensity=12e3)
        self.mesh_node = None
        self.scene = pyrender.Scene(bg_color=(0.0, 0.0, 0.0))

        self.scene.add_node(self.camera_node)
        self.scene.add(light, pose=np.eye(4))

    def change_mesh(self, mesh_fp):
        if self.mesh_node != None:
            self.scene.remove_node(self.mesh_node)
        self.mesh_node = pyrender.Node(mesh=pyrender.Mesh.from_trimesh(trimesh.load(mesh_fp)))
        self.scene.add_node(self.mesh_node)

    def change_camera_pose(self, x, y, z):
        camera_pose = np.array([
            [1.0, 0.0, 0.0, x],
            [0.0, 1.0, 0.0, y],
            [0.0, 0.0, 1.0, 3.0*z[0]+z[1]],
            [0.0, 0.0, 0.0, 1.0],
        ])
```

```
        self.scene.set_pose(self.camera_node, camera_pose)


    def __call__(self, save_fp, check=True):
        #self.scene.set_pose(self.mesh_node, get_rotation_matrix(rotation_directions))
        color, _ = self.r.render(self.scene)
        imsave(save_fp, color)
        _, _, _, current_pose = get_param(model, alignment_model, save_fp, args)
        return current_pose



if __name__ == '__main__':
    render = Render(np.pi / 3.0)
    faultyOBJs = [3, 30]
    for i in range(1, 54):
        if i in faultyOBJs:
            continue
        else:
            print(i)
            subject = Subject(render, np.pi / 3.0, i)
            transformation_file = open(f"{subject.mesh_directory_path}/transformations.txt", "w")
            for pitch_step in range(-1, 2):
                for yaw_step in range(-4, 5):
                    save_fp, rotation_from_init = subject.render_subject(pitch_step, yaw_step)
                    transformation_file.write(
                        f"{save_fp} {' '.join(map(str, rotation_from_init))}\n"
                    )
            transformation_file.close()
```

Bibliography

[1] Y. Feng, F. Wu, X. Shao, Y. Wang and X. Zhou, 'Joint 3d face reconstruction and dense alignment with position map regression network,' in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 534–551.

[2] O. Lium, '3d facial reconstruction from front and side images,' Master's Thesis, NTNU, Jun. 2020.

[3] H. Zhou, J. Liu, Z. Liu, Y. Liu and X. Wang, 'Rotate-and-render: Unsupervised photorealistic face rotation from single-view images,' in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5911–5920.

[4] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[5] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, 'Mobilenetv2: Inverted residuals and linear bottlenecks,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[7] H. Noh, S. Hong and B. Han, 'Learning deconvolution network for semantic segmentation,' in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1520–1528.

[8] V. Blanz and T. Vetter, 'A morphable model for the synthesis of 3d faces,' in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999, pp. 187–194.

[9] P. Paysan, R. Knothe, B. Amberg, S. Romdhani and T. Vetter, 'A 3d face model for pose and illumination invariant face recognition,' in *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, Ieee, 2009, pp. 296–301.

[10] X. Zhu, Z. Lei, X. Liu, H. Shi and S. 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.

[11] R. Gross, I. Matthews, J. Cohn, T. Kanade and S. Baker, 'Multi-pie,' *Image and Vision Computing*, vol. 28, no. 5, pp. 807–813, 2010.

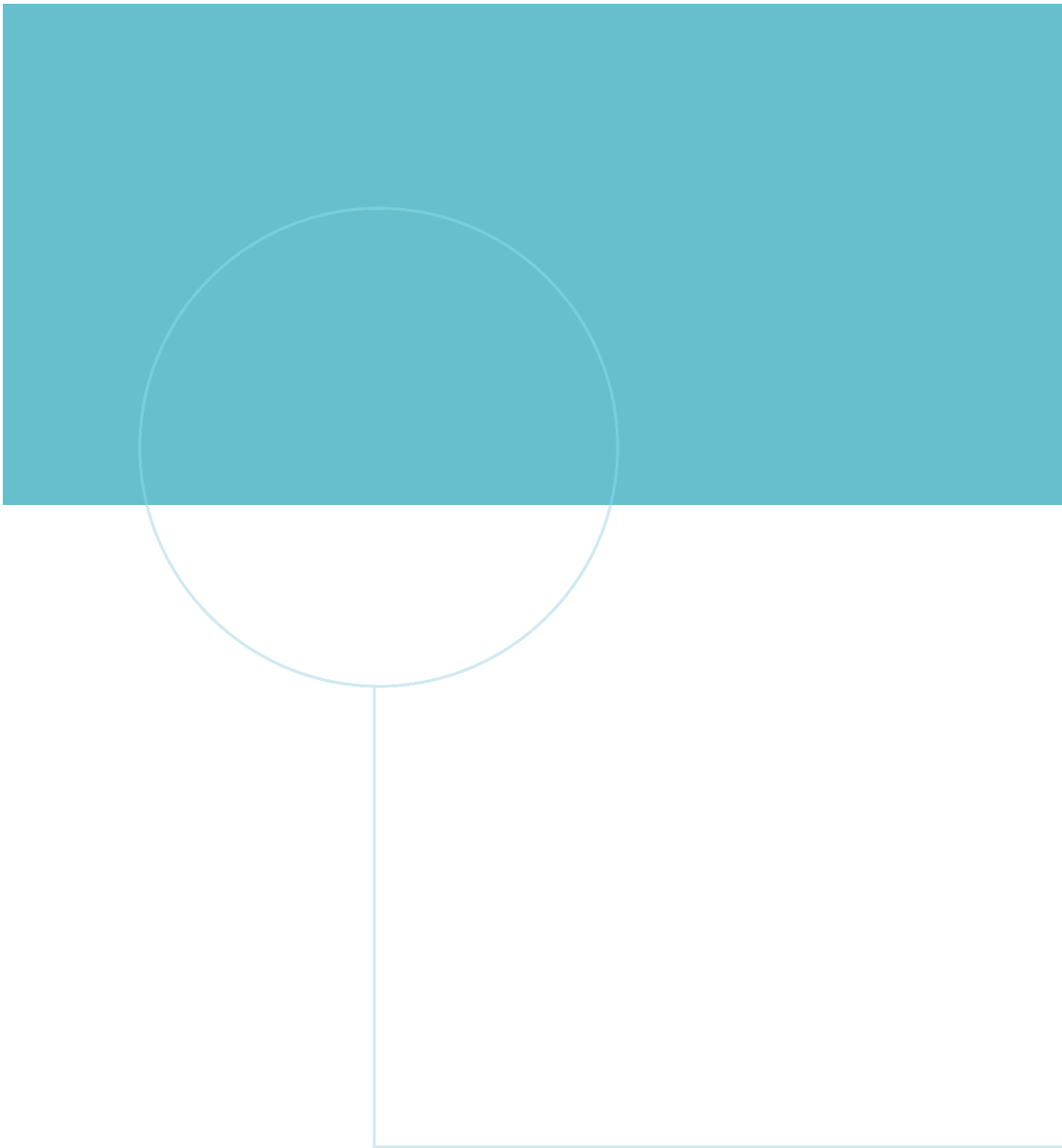
[12] Z. Liu, P. Luo, X. Wang and X. Tang, 'Deep learning face attributes in the wild,' in *Proceedings of International Conference on Computer Vision (ICCV)*, Dec. 2015.

[13] Q. Cao, L. Shen, W. Xie, O. M. Parkhi and A. Zisserman, 'Vggface2: A dataset for recognising faces across pose and age,' in *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*, IEEE, 2018, pp. 67–74.

[14] A. Bansal, A. Nanduri, C. D. Castillo, R. Ranjan and R. Chellappa, 'Umdfaces: An annotated face dataset for training deep networks,' *arXiv preprint arXiv:1611.01484v2*, 2016.

[15] A. D. Bagdanov, A. Del Bimbo and I. Masi, 'The florence 2d/3d hybrid face dataset,' in *Proceedings of the 2011 joint ACM workshop on Human gesture and behavior understanding*, 2011, pp. 79–80.

[16] J. Guo, X. Zhu, Y. Yang, F. Yang, Z. Lei and S. Z. Li, 'Towards fast, accurate and stable 3d dense face alignment,' *arXiv preprint arXiv:2009.09960*, 2020.



NTNU

Norwegian University of
Science and Technology