Facial Landmarks Detection: A Brief Chronological Survey & Practical Implementation

Deep Learning Group Project - 'Let's Face It'

A. Bätz[1], **A. De Miguel Palacio**[1], **G. Mishra**[1]

I. Llorca Rodríguez[1], **M. Raihan Pranti**[1], **P. Satorre Mulet**[1]

Project Supervisor: H. Shahriar[1]
Prof. Dr. C. Lippert[2]

[1]Institute of Computer Science, Universität Potsdam, Germany
[2]Dept. Digital Health - Machine Learning, Hasso-Plattner-Institut (HPI), Germany

{baetz1, demiguelpalacio1, mishra}@uni-potsdam.de
{llorcarodriguez, pranti, satorremulet}@uni-potsdam.de
shahriar@uni-potsdam.de
christoph.lippert@hpi.de

Abstract

Facial recognition is widely popular due to its vast number of applications. Most facial recognition solutions rely heavily on the information provided by the location of the key landmark points of a face such as corners and center of mouth, nose and eyes. Computer science has tackled this problem from many different angles over the years, with CNN methods having the upper-hand as of late. This paper conveys a brief literature review of past and present methodology, with special emphasis on deep learning approaches. We then implement three different CNN architectures (ResNet, VGG and MobileNet) to predict the keypoints on live-stream. Results shown that VGG achieves the best performance for our problem setting.

1 Introduction

1.1 Motivation

Facial recognition plays a crucial role among our baseline social skill set. As humans, a certainly high ability to identify someone's face and the facial expressions in response of their emotional states is given for granted. It is thus not surprising that the work of properly recognizing faces has been approached for a range of purposes from different science fields - behavioral science [1], psychology, security and defense, etc. - and computer science has come to fill a much needed spot during the last couple decades [2].

Among the breakdown of necessary tasks to achieve face recognition, one of the key fundamentals is to define, identify and classify specific facial landmarks [3]. In plain words: it is often mandatory, for instance, to locate the center and left, right, top and bottom corners of an eye (and mouth, nose, etc), to accordingly define its shape before starting to judge whether it corresponds to a specific person or if this person is happy or sad. Many present-day facial analysis processes are built relying heavily on the information provided by these landmark locations. Hence, some examples of its ever-growing number of applications are: identity validation (mobile phones, ATMs, access to sensitive areas or information, forensic investigations) [4], healthcare and disease diagnosis [5], retail and marketing (personalized marketing, automatic check-in and out, payment verification) [6] or morphing and beautification.

As it will be presented in the Theoretical Background (section 2), the computer vision approaches to this problem have been numerous throughout the years, their performance measured mainly on base of their accuracy and speed. The super-fast pace at which computational power becomes both affordable and powerful is a direct cause of the spread of different methods. Amidst this mare-magnum of solutions, deep Convolutional Neural Networks (CNN) have lately gained the upper hand (see Figure 1). As an evolution within the framework of Deep Learning techniques of former direct regression methods and cascade regression methods, CNNs predict facial landmarks directly from the picture to then refine every guess separately through a variable number of convolutional and pooling layers, each with different settings and outputs [3]. The works of Wu et al. [7] and, most of all, Sun et al. in 2013 [8] are undoubtedly significant milestones worth mentioning to this regard. With those papers as theoretical fathers, many tweaks to the set of layers of the CNNs have been proposed afterwards in order to improve the algorithms' performance [9; 10; 11; 12].

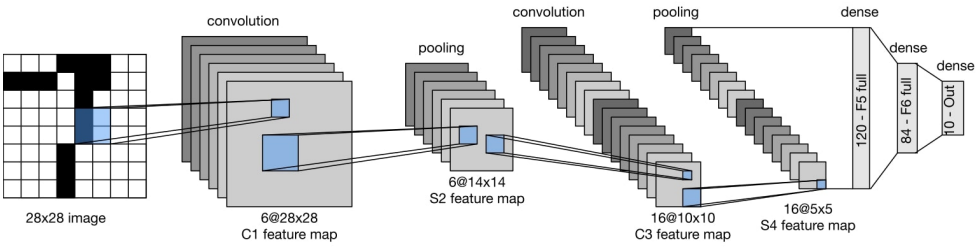

Figure 1: LeNet convolutional neural network architecture, one of the first and most iconic ones [13]

1.2 Content Summary

To give the reader a feel for the theoretical background (section 2), a brief survey of previous facial landmarks detection works will be discussed in subsection 2.1, with a special emphasis on CNN methodology in subsection 2.2. Following, a practical demonstration on live video is described in section 3, first reporting the data preprocessing techniques in subsection 3.1, and then outlining three candidate neural network architectures in subsection 3.2. To conclude this report, in subsection 4.1 we discuss the pros and cons of each model, to then decide a winner. Subsection 4.2 argues its limitations and different possibilities for further research. Finally, section 5 summarizes the findings of this collaboration.

2 Theoretical Background

This section intends to sequentially present the historic developments of facial keypoint detection approaches, briefly stopping at the most relevant milestones and authors through the years.

2.1 Non-CNN Facial Landmark Detection Algorithms

The earlier algorithms were primarily focused on how to fit a distorted face mesh [14]. They were centered on less demanding facial images without any complex variations. Most of the facial landmark detection methods have been developed over years to automatically detect the facial key points. As the time progressed and with the advancement of technology, the facial landmark detection algorithms became capable enough to handle many facial variations with specific conditions [15]. The traditional algorithms include Holistic, Constrained Local Model (CLM) methods, and regression-based methods. They can be distinguished from each other on the basis of how they are used to represent the facial aspects and shape information.

Holistic methods are specifically used for building models to represent the global facial appearance and shape information. Under them, there are two famous methods known as ASM and AAM, respectively. Active Shape Model (ASM) was developed by Cootes et.al. [16] whereas Active Appearance Model (ASM) was proposed by Taylor and Cootes Edwards et al. (1998) and Cootes et al.(2001) [17]. The ASM model was restricted to the shape modeling of images. The main difference that distinguishes AAM with ASM is that the AAM model can exhibit shape as well as the texture variability for a given dataset [16]. Unlike AAM, ASM was fast (for the time) in identifying the key points.

On the contrary to holistic approaches, the CLM methods proposed first by Cristinacce and Cootes in 2006 [18] and improved by Saragih et al. in 2011 [19] were used to deduce the landmark positions based on the global facial shape patterns. They could also recognise autonomous local appearance particulars around each landmark and thereby were fast in identifying any obstacle or brightness [15]. It has been observed that, the mentioned algorithms made use of statistical methods as their foundation thereby leading to a good prognosis accuracy [14].

Lastly, regression-based methods implicitly capture the facial shape and appearance information. In recent times, the deep learning methods gained prominence over traditional ones and thereby took the command for detecting facial landmarks. To support the same, CNN models are an advancement within the regression-based methods as it will be explained in the upcoming section.

Table 1 below gives an overview of how the above mentioned traditional algorithms differ from each other with respect to different parameters.

Algorithms	Appearance	Shape	Performance	Speed
Holistic method	Whole face	Explicit	Poor generalization/good	Slow/fast
Constrained Local Method (CLM)	Local patch	Explicit	Good	Slow/fast
Regression-based method	Local patch/whole face	Implicit	Good/very good	Fast/very fast

Table 1: Comparison of the major types of facial landmark detection algorithms.

2.2 Development of Facial Landmark Detection with CNN

One could trace back the very first effective CNN based method to 2005 [20]. S. Duffner and C. Garcia proposed in his pioneer work what we would today consider a simple neural network architecture, consisting of just 6 layers - including input and output (see Figure 2). Their application proved to perform unusually well in conditions of varying lighting and pose when compared to the back then state of the art methods. This is not surprising knowing that previous Holistic and Constrained Local Model (shape-based) methods tried to find within the images a specific pre-built face shape model [15] - that might indeed be very different to the silhouette of a face that is looking sideways. Nevertheless, the results were not relevant enough to produce further research, nor to

target any sort of commercial application. After all, Deep Learning algorithms could only be run in CPUs, with the consequent drawback on performance and making them unable to compete in many problem settings. This was the main cause of a drought in the matter of research for the following years.

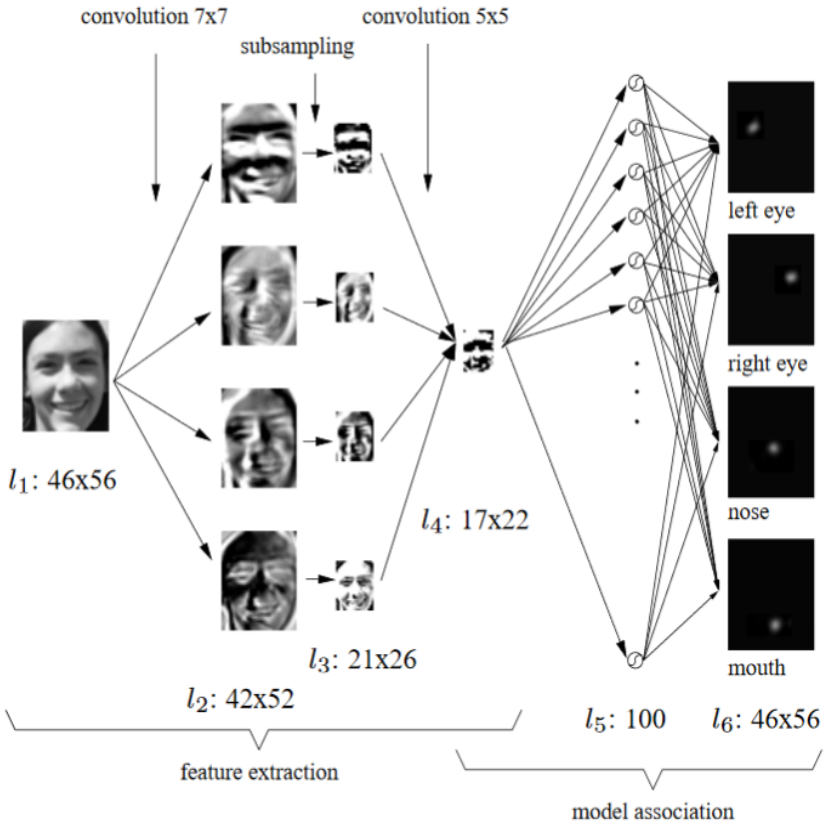

Figure 2: Duffner's Primitive CNN Model for Facial Keypoint detection [20]

In 2010, Dollár et al. [21] presented his Cascaded Pose Regression algorithm to detect poses of objects in images. While it was not specifically designed to address faces, it was definitively applicable. Then, a major disruption in the Deep Learning field took place: in 2012 AlexNet broke in, opening the possibility for CNN algorithms to run on GPUs [22], hence accomplishing significantly better results. This caused a wave of researchers to turn their heads into Deep Learning and we see the use of CNN for facial recognition resumed: Zhu et al. [23] and Luo et al. [24] reintroduced neural networks, this time adding Restricted Boltzman Machines in each layer. These works arguably paved the way for what came in 2013. As previously mentioned in the introduction, the research of Wu et al. [25] first and most importantly Sun et al. [8] later signified a major turning point, changing what is considered top notch facial landmark recognition methodology from then on.

Sun et al. could build on the same basis a model substantially more complex than that proposed 8 years before by at al.: 3 levels with different layer configurations (as much as 10 in the first and most important of them - Figure 3) where cascaded in their design for a neat and yet affordable coarse-to-fine prediction. As in many other computer vision solutions, convolution and max pooling layers were interspersed repeatedly reducing the size of the layers while increasing the number of channels to then end up the blocks with 2 fully connected layers. Their accuracy results improved by a considerable margin the best performing methods of the time [26; 27; 28; 29], generally based on regression trees.

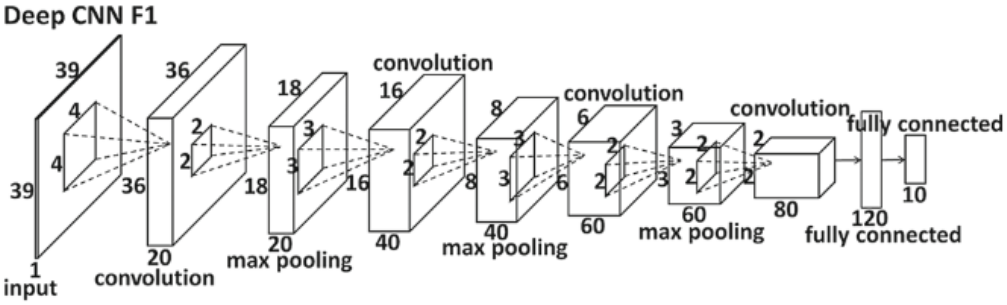

Figure 3: Architecture of the first level of layers proposed by Sun et al. in 2013

Since then, the majority of developments in facial keypoint detection have been born from deep learning based methods. Among the most relevant authors, Zhang et al. [9] demonstrates that the prediction accuracy improves greatly when getting involved other more or less correlated auxiliary attributes such us gender or expression, which additionally has a positive effect on reducing bias. Ranjan et al. [10] improves the method in the same direction. Also worth mentioning is the approach of Trigeorgis et al. in 2016 [11], implementing successfully for the first time Recurrent Neural Networks after a typical CNN solution in their Mnemonic Descent Method to improve state of the art results. For the last couple years, cutting edge models surpass one another quite frequently. From these, the work from Guo et al. [12] argues to be one of the best rounded ones in terms of accuracy (also in the wild) and speed.

3 Implementation

The objective of this work is to predict facial keypoint positions on live-streams. This is attained by first training a CNN model on a dataset of face images containing the x- and y-coordinates of 15 key facial landmarks, being capable of predicting these coordinates on unseen images and then, using computer vision techniques, extending the model to perform facial detection on a live-stream. Once the face is detected the predictions must follow the key facial landmarks as the person moves in real-time.

In the current section, the implementation of this work will be outlined. We describe the used dataset, the preprocessing techniques employed and the reasoning behind these transformations. The CNN architectures utilised for the model are then presented. Lastly, we explain how the trained model is put to work on live video.

3.1 Data Preprocessing

The 'Facial Keypoints Detection' dataset used was taken from 'Kaggle Competitions' [30], containing 7049 images of size (96×96) pixels. Each predicted keypoint is specified by a (x, y) pair of coordinates in the space of pixel indices. There are 15 facial keypoints that represent the following elements of the face: the "left and right eyes' centers", the "left eye's inner and outer corners", the "right eye's inner and outer corners", the "left eyebrow's inner and outer ends", the "right eyebrow's inner and outer ends", the "nose tip", the "mouth's left and right corners" and finally the "mouth's top and bottom lips' centers" - see top left image in Figure 5 ('left' and 'right' here refers to the point of view of the subject within the image). These key facial landmarks were the ones this work aimed to predict and detect during live-streams.

Deep neural networks necessitate vast amounts and quality of training data so that the model can learn convincingly and avoid overfitting. These factors have massive repercussions on the results of face related problems [31]. Unfortunately the initial dataset saw itself reduced from 7049 images to a scarce 2140, after it was decided to drop the images that contained at least one missing x- and/or y-coordinate for a key facial landmark, as it was too complex to implement an imputation method that could meaningfully replace and fill-up those missing values (see subsection 4.2). Finally, from these scarce remaining 2140 images an 80:20 train-test-split was applied leaving behind the training dataset with 1712 input images and the test dataset with 428 images. To compensate for this reduction of data, it was decided to implement *data augmentation*, as it is reliable and relatively straightforward to code. Data augmentation (DA) overcomes this issue by artificially inflating the training dataset with transformations, as explained by Taylor L. and Nitschke G. [32]. In plain words: it is a cheap way to create additional, accurate data. Another reason that convinced this group to opt for DA was that, in recent years, there has been an increased use of general DA methods, whose results demonstrated to improve CNNs' performance and accuracy [31; 32]. This work validated such statement once a final CNN model was chosen - see Figure 4.

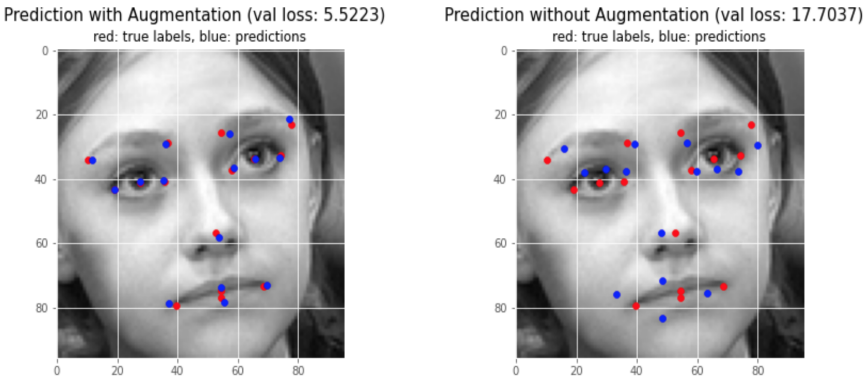

Figure 4: Validation loss comparison of the key facial landmarks' prediction on the trained images with (left image) and without (right image) data augmentation.

The generic DA techniques can be dissected into two distinct categories: geometric transformations and photometric transformations. The aim of geometric transformations is to modify the image's geometry by translating image pixel values to new positions, to in consequence make the CNN invariant to changes in orientation and position [31; 32]. The geometric transformations used in this work were rotation, flipping, shifting and scaling. The aim of photometric transformations is to reform the RGB colour channels of the images. This is achieved by shifting pixel colours to new values, to obtain a CNN invariant to change in lighting (tonality) and colour [31; 32]. As this work dealt with gray-scale images, the photometric transformations used were lighting perturbations (modification of brightness of images) and filtering (blurring of images).

For the practical implementation of DA, the Python 'Albumentations' library [33] was employed in this work. This library provides the necessary basic DA transformations (e.g. flipping, rotation) that one can combine for more complex transformations. Every image in the reduced training dataset gets transformed three times, hence inflating the (training) data to four times its size. In rare occasions, one or several data points were located outside the frame of the image after shifting or rotating. Such images had to be dropped, narrowing the final amount of training images to 6736. This means that we achieved a set that is 95.6% the size of the original one (before null deletion). Furthermore, the CNN will also be trained on different variations of angles, orientations, brightness and blurriness, unlike the original dataset where all faces were clear, centered and straight.

The three implemented complex DA transformations are depicted on Figure 5. All three are the result of combining to varying degrees these transformations: translation, scaling, rotation, horizontal flip, blurring and brightness and contrast change.

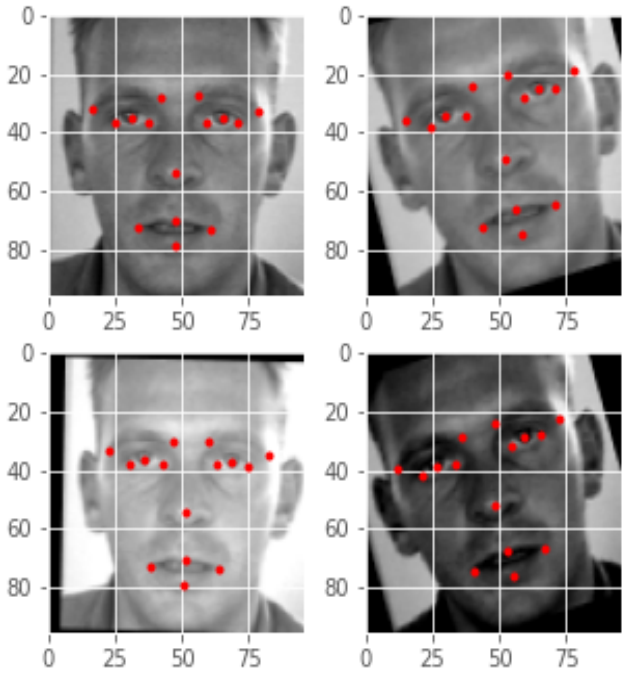

Figure 5: Diagram showing an arbitrary training image on the top left and its corresponding augmented elements surrounding it.

3.2 Architectures

This work commenced by studying in depth the code found on 'DebuggerCafe' [34] in order to get an impression of how to approach this work's task. The code of this work still has some similarities to the script shown on [34], but over the course of time the code became gradually unique.

Once a clear idea on how to tackle this work's task was acquired, this group investigated three distinct deep neural network (NN) architectures in parallel to implement the most effective one for our final live-stream detection. *VGG*, *MobileNet* and *ResNet* were the architectures taken into consideration - the layer-structure of each are tabulated hereunder on Tables 2, 3 and 4 respectively. All three NN architectures were trained using 400 epochs, a learning rate of 1×10^{-4}, batch sizes of 256 and the *Adam* (Adaptive Moment Estimation) as an optimiser, as it is well known for being efficient for CNNs and tasks requiring large datasets and vast amounts of trainable parameters [35], which is the case of this work. Since the Adam optimiser adapts the learning rate during training, it was decided not to tune it on our own.

As for the objective function, the *MSE* (Mean Squared Error) was implemented for the loss term, but no explicit regularization term was applied. As an alternative, we used *Dropout* and/or *Batch Normalization* . Dropout, proposed by N. Srivastava, G. Hinton et al. [36], is a simple regularisation technique that prevents overfitting by randomly dropping some units (but only temporarily) during training time. Training deep NNs is complicated and making them converge in a reasonable amount of time can be complex. Batch Normalization, a heuristic proposed by S. Ioffe and C. Szegedy [37], is an immensely convenient, effective technique found to improve the reliability and accelerate the convergence when training deep NNs. This is achieved by normalising the activations of hidden layer nodes, by subtracting their mean and diving by their standard deviation, estimating both based on the current mini-batch [37].

If the reader is interested in a deeper explanation of the code implemented for all the different operations on each of the NNs' layers, please refer to PyTorch's documentation [38].

3.2.1 VGG's NN

In 2014 K. Simonyan and A. Zisserman [39] from the VGG (Visual Geometry Group) of Oxford proposed a considerably deep CNN (16 layers) suited for large-scale image and video recognition, using small (3×3) convolution kernels. VGG always employs padding to ensure the input and output sizes are equivalent. Subsequently max. pooling with a (2×2) pooling window is utilised to reduce the resolution. VGG mainly helps to reduce the need for hyperparameter-tuning of traditional CNNs. Some of the code used by this collaboration to implement VGG's CNN was taken from [40].

Layer	Operation(s)

1	`Conv2d(1, 32, kernel_size = 3, padding = 2)`
	`LeakyReLU`
	`BatchNormalization`

2	`Conv2d(32, 32, kernel_size = 3, padding = 2)`
	`LeakyReLU`
	`BatchNormalization`
	`MaxPooling`

3	`Conv2d(32, 64, kernel_size = 3, padding = 2)`
	`LeakyReLU`
	`BatchNormalization`

4	`Conv2d(64, 64, kernel_size = 3, padding = 2)`
	`LeakyReLU`
	`BatchNormalization`
	`MaxPooling`

...	...

11	`Conv2d(256, 512, kernel_size = 3, padding = 2)`
	`LeakyReLU`
	`BatchNormalization`

12	`Conv2d(512, 512, kernel_size = 3, padding = 2)`
	`LeakyReLU`
	`BatchNormalization`

13	`adaptive_avg_pool2d` `reshape`
14	`dropout`
15	`Linear(512, 512)`
16	`Linear(512, 30)`

Table 2: *VGG* deep neural network's layer-structure. In 'Conv2d(x, y, ...)', 'x' represents the number of input channels and 'y' the number of output channels. The layers in the gap '...' are of the same pattern as the ones before. On the other hand, in 'Linear(x, y)', 'x' represents the number of input features and 'y' the number of output features.

3.2.2 MobileNet NN

The MobileNet NN, proposed in 2017 by A. G. Howard et al. [41], is a CNN that was developed on the previously mentioned VGG in aim of making it adequate and more agile for devices with less computing power. MobileNet uses (depth-wise) separable convolutions, instead of the traditional convolution techniques, to build light weighted deep NNs. In a depth-wise separable convolution, the first convolution performed is a depth-wise convolution, which does not reduce the number of channels (i.e. every channel is multiplied by the corresponding channel in the kernel). In this work a (3×3) sized kernel was used for this NN. The channel size of the kernel is the next layer's desirable channel size. Then the second convolution performed is a point-wise convolution with a (1×1) sized kernel.

This deep CNN is well known for being suitable for object detection and face attributes, therefore giving this collaboration a good reason to try to implement this it. With the use of MobileNet, the number of trainable parameters is drastically reduced. In consequence, the predictions' calculations can be efficiently carried out in less powerful devices such as mobile phones, thus inheriting such name. The layer-structure is tabulated in Table 3.

Layer	Operation(s)
1	Conv2d(1, 1, kernel_size = 3, padding = 2, groups = 1) Conv2d(1, 32, kernel_size = 1) LeakyReLU BatchNormalization MaxPooling
2	Conv2d(32, 32, kernel_size = 3, padding = 2, groups = 32) Conv2d(32, 64, kernel_size = 1) LeakyReLU BatchNormalization MaxPooling
...	...
5	Conv2d(128, 128, kernel_size = 3, padding = 2, groups = 128) Conv2d(128, 256, kernel_size = 1) LeakyReLU BatchNormalization MaxPooling
6	Conv2d(256, 256, kernel_size = 3, padding = 2, groups = 256) Conv2d(256, 512, kernel_size = 1) LeakyReLU BatchNormalization MaxPooling
7	adaptive_avg_pool2d reshape
8	dropout
9	Linear(512, 512)
10	Linear(512, 30)

Table 3: *MobileNet* deep neural network's layer-structure. As previously mentioned, the layers in the gap '...' are of the same pattern as the ones before.

3.2.3 ResNet NN

In 2017 Wu et al. [42] proposed a deep residual CNN (ResNet) in the setting of key facial landmarks detection. To implement the CNN structure represented in the paper written by Wu et al., the PyTorch code on ResNets supplied from D2L [13] was used. Residual NNs were introduced to deal with the problem of degradation of the training accuracy when learning very deep networks [43]. This is attained by including *residual blocks*. These blocks contain identity mappings, i.e. short-cut connections, that skip some convolutional layers. The residual block used in this work is visualised in Figure 6. Residual NNs are simpler and more efficient to optimise, but most importantly accuracy is improven as depth of the NN is increased considerably [43].

Initially, the model architecture proposed by Wu et al. [42] was implemented. In order to improve the model, different changes were applied and the validation loss was examined. As this work required training the model several times, the number of training epochs for this task was reduced to 300.

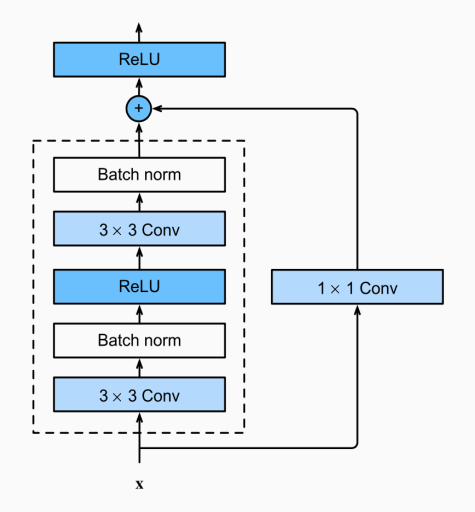

Figure 6: *ResNet*'s residual block [13]. These are used in layers 2-5 in the ResNet NN, as shown in Table 4.

One feature that offered much potential for improving the model was the number of residual blocks. The question was how many residual blocks are optimal for our network. We gradually increased its number to examine the change in the training and validation loss. As a consequence, the quantity of trainable parameters rose as well. The latter is important because it affects both training time and prediction time. Since the intention was to perform additional hyperparameter tuning, the goal was to limit the training time. As the final goal was real-time prediction (during live-streams), the aim was to minimise prediction time as much as possible.

Once investigated, it was decided to use either 4 or 5 residual blocks. It is appreciated in Figure 7 that additional residual blocks would offer only marginal improvements while imploding the number of trainable parameters and in consequence also the training and prediction times.

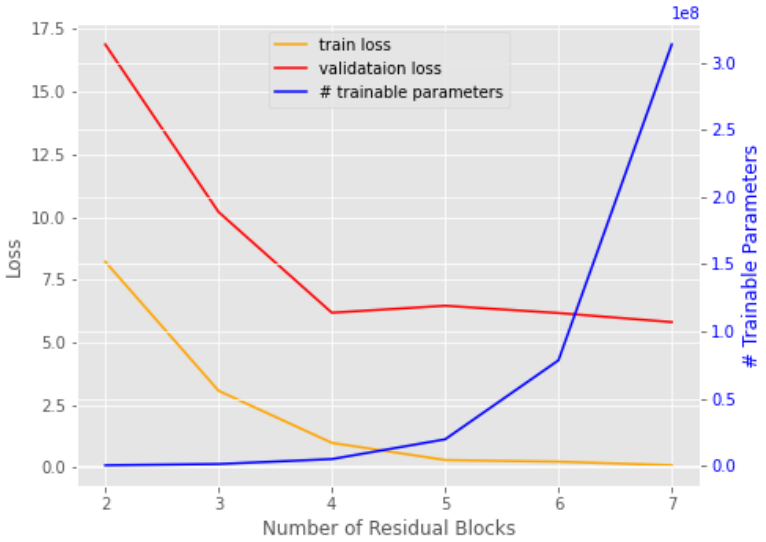

Figure 7: *ResNet*'s number of trainable parameters, training and validation loss comparisons depending on the number of residual blocks employed within the CNN's architecture.

Wu et al. [42] proposed an initial convolutional layer with a (11×11) sized kernel before the residual blocks. In order to examine the effect of this kernel size on the validation loss and also to decide on either 4 or 5 residual blocks a small grid search was applied. The results are depicted in Figure 8. It can be observed that the models with 5 residual blocks have less variance in their validation loss, but on average don't offer an improvement in the loss compared with the models with 4 residual blocks. Since the latter have the advantage of containing less trainable parameters, it was concluded that 4 residual blocks were the optimal choice for our network architecture. Furthermore, Figure 8 depicts that the two smallest validation losses were observed with a (3×3) sized kernel, once with 4 and once with 5 residual blocks. Thus, this kernel size was adopted in this work's ResNet NN model - see Table 4.

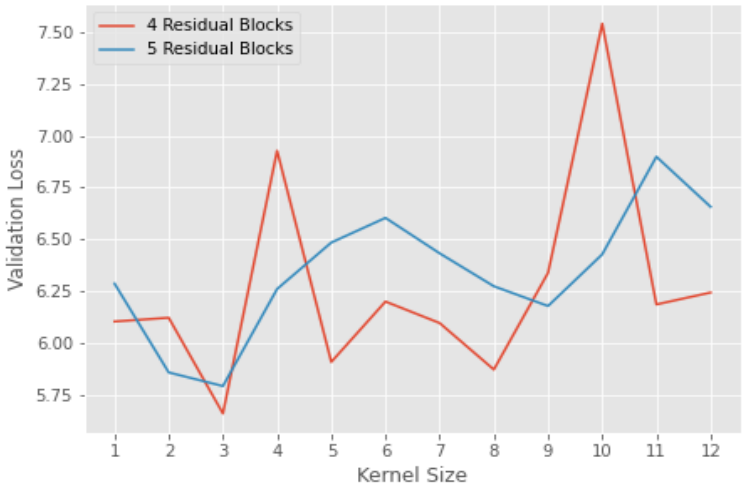

Figure 8: Validation loss of *ResNet* depending on the kernel size used for the first convolutional layer.

Layer	Operation(s)
1	`Conv2d(1, 32, kernel_size = 3, padding = 'same')` `MaxPooling`
2	`Residual(32, 64)`
3	`Residual(64, 128)`
4	`Residual(128, 256)`
5	`Residual(256, 512)`
6	`adaptive_avg_pool2d` `reshape`
7	`Linear(512, 256)`
8	`Linear(256, 30)`

Table 4: *ResNet* deep neural network's layer-structure. The 'Residual(x, y)' layers are the ResNet's residual blocks depicted in Figure 6. As for 'Conv2d(x,y)', 'x' and 'y' represent the number of input and output channels respectively.

3.3 Prediction on Live-Stream

To finish the implementation description, it is shortly explained how the trained model was placed to work on live-stream, as it is the source of the biggest limitations of this work. Prior to predicting key facial landmarks, the region(s) from the video with the face(s) in it must be cropped. This can then be passed to the CNN.

As building a face recognizer was beyond the scope, an existing application [44] was employed. Initially, this repository makes use of the Haar Cascade face detection model [45]. This model is slightly outdated and the performance was not satisfactory, so it was decided to be exchanged for a reportedly better one based on the Python OpenCV's module for deep NNs [46]. Although this solved most problems and achieved better accuracy, certain prediction delay remained unfortunately unchanged. The resulting consequences and possible solutions are explained in 4.2.

4 Model Evaluation & Discussion

This section comments the advantages and disadvantages of each of the architectures presented on subsection 3.2, to then reason a winning model. We then argue its limitations and what further improvements could be implemented.

4.1 Model Evaluation

The comparison of the loss of the three architectures (on the validation set) is represented below (Figure 9). Please, note that one of the graphs is plotted in logarithmic scale, while the other is not. Similarly, the second one is cut to show only the last (and most relevant) 300 epochs. As it has been stated many times through the research, not only the prediction accuracy is important in order to proclaim a champion, but also the time it takes for each model to obtain these predictions. Hence, see in Table 5 the lapse of time needed by each architecture for such task.

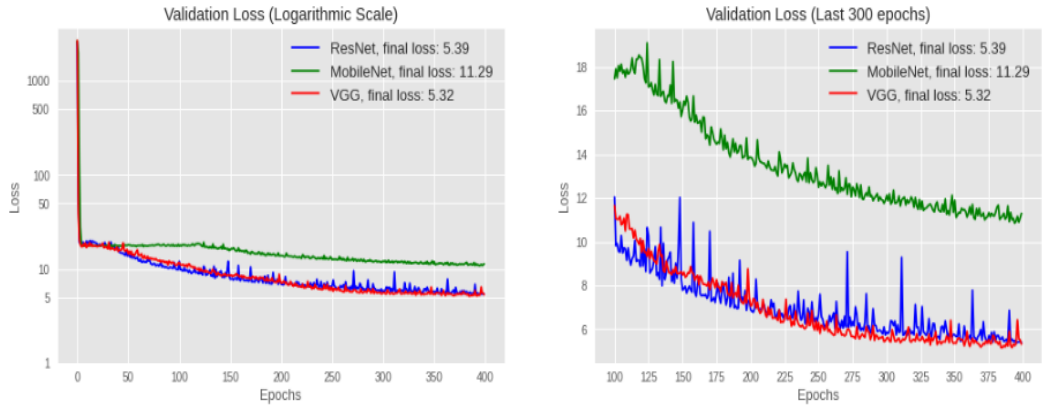

Figure 9: Validation loss of the three implemented models

CNN Architecture	Training Time	Loss	Prediction Time
MobileNet	$\approx$ 15 min.	11.29	2.34 s
VGG	$\approx$ 45 min.	5.32	3.48 s
ResNet	$\approx$ 20 min.	5.39	4.96 s

Table 5: Validation Loss: models trained on 6736 instances with 400 epochs; prediction on 1783 images.

This information is enough to extract several conclusions. First of all, one can observe that the loss of MobileNet seems to keep going subtly down at 400 epochs. Nonetheless, it is very unlikely that it will ever match the numbers of the other two competitors. We thought unnecessary to keep testing with wider epoch ranges. Additionally, the time it takes to train the models in our machines is in between 15 and 45 minutes, which makes it undesirable to perform futile tests. Taking all of this into account, MobileNet is our first discarded model, in spite of being the fastest one. The final decision is then in between ResNet and VGG. Both their MSE losses draw an almost identical trajectory, but VGG is 30% faster. On top of that, its loss is notably more stable. Therefore, VGG takes the definitive first place.

It cannot go without mention that all of the models present overfitting, the difference of their train and test losses being relatively low. The main cause of this surely has to do with the small size of our training data, which even after the augmentation techniques we put it through did not even reach 7000 samples.

4.2 Discussion

4.2.1 Limitations

It feels fair to remark that this work did not intend to improve the performance of any current commercial application, nor to add to the existing state of the art literature. There are many pre-built facial keypoint recognition solutions that can be run straight out of the box if one wants a fast and reliable live-prediction.

With that in mind, the biggest limitation within the reach of our project is the prediction time. Despite the fact that the keypoint detection is rather accurate, the model takes too long to show the predictions in webcam-captured video (around 3,5 s). This makes our application unfortunately unsuitable for live-stream detection. The problem does not seem to be on the CNN. Our suspicions lean towards the face detector we used to trim a square with the face from the live-video before plugging it into the model. We did not code this face recognizer ourselves. Instead, we implemented an external solution whose integration appears to cause some trouble, thus producing the delay in the prediction. We tried to approach this problem by tweaking the settings of the face detector, playing around with the fps ratio and other parameters; but none of them shed a relevant improvement.

We used Google Colab in Google Drive for the majority of the coding of the project. During the later stages of the development we observed that running the code on other environments (namely Spyder) predicted the face considerably faster. This makes us suspect that the multiple cloud integrations involved in Colab might also be part of our delay problem. We have not investigated further due to time constraints, but it looks definitively promising.

4.2.2 Possible Improvements & Further Research

As in almost every machine learning problem, one straight improvement would be increasing the size of the dataset in which we train our model. In most cases (unless the added images were exceptionally misleading), this would result into better predictions and less overfitting. In a similar way, we found stunning that almost all of the solutions to this challenge treated the null values by just dropping them, thus reducing the training data set to approximately a third. We played around the idea of dealing with this problem by implementing a loss function that, based in MSE, ignores the prediction on the missing points. Something similar can be seen in Wu et al. 2017 [42]. The biggest roadblock for these had to do with the difficulty of implementing the backpropagation for these specific points.

Improvements and further research in the sense of the neural network composition can get beyond imagination. Some of the most frequently quoted papers in this work exceed ours in complexity by far, despite being 8 years old. Furthermore, during the last years we have been witnesses of big tech companies racing in the construction of parallel-computing machines that allow them to perform training on immensely deep neural networks in order to win this or that image-classification challenge. This is of course one direction in which our work could be enhanced.

Closer to earth, we believe there is room for further research regarding the choice of the loss function, for instance. As it is stated previously, we end up using MSE loss because it shed the best results and because of the ease it offers for drawing comparisons. Other loss functions that are proven strong for our problem setting were tried out and discarded due to their poor performance or the excessive complexity of their implementation. Among the first group it is worth to mention the Wing Loss [47], which tries to improve accuracy by amplifying the impact from small and medium range errors. Unfortunately this did not happen in our case. Some examples of the functions that we tried to use but found out of scope are the nameless loss function proposed in the infamous work by Sun in 2013 [8], or the Enhanced-Normalized Mean Error Loss [48], which intends to improve the Wing-Loss. Most probably our model could perform better with some of these complex losses.

5 Conclusion

Facial recognition in general and facial landmark detection in particular have been hot topics of research during the last couple of decades due to the vast number of applications for which they are necessary. While shape-based methods (holistic and CLM) had the upper hand in the past, Deep Learning solutions are undoubtedly the go-to choice since 2013.

Our practical demonstration is yet another proof that, with current technology, deep neural networks can be easily implemented for computer vision problems such as facial landmark detection. Applications that require fast prediction in devices without high computational power are not an exception anymore. Moreover, some of the most problematic conditions for the methodologies used in the past like occlusion, lighting or face pose are fairly well solved with CNN.

References

[1] Cohn JF. Advances in Behavioral Science Using Automated Facial Image Analysis and Synthesis [Social Sciences]. IEEE Signal Processing Magazine. 2010;27(6):128–133.

[2] Ouanan H, Ouanan M, Aksasse B. Facial landmark localization: Past, present and future. In: 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt); 2016. p. 487–493.

[3] Yan Y, Naturel X, Chateau T, Duffner S, Garcia C, Blanc C. A survey of deep facial landmark detection. In: RFIAP. Paris, France; 2018. Available from: https://hal.archives-ouvertes.fr/hal-02892002.

[4] Galbally J, Marcel S, Fierrez J. Biometric Antispoofing Methods: A Survey in Face Recognition. IEEE Access. 2014;2:1530–1552.

[5] Kruszka P, Porras AR, Sobering AK, Ikolo FA, La Qua S, Shotelersuk V, et al. Down syndrome in diverse populations. American Journal of Medical Genetics Part A. 2017;173(1):42–53. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/ajmg.a.38043.

[6] Kumar V, Rajan B, Venkatesan R, Lecinski J. Understanding the Role of Artificial Intelligence in Personalized Engagement Marketing. California Management Review. 2019;61(4):135–155. Available from: https://doi.org/10.1177/0008125619859317.

[7] Wu Y, Wang Z, Ji Q. Facial Feature Tracking Under Varying Facial Expressions and Face Poses Based on Restricted Boltzmann Machines. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2013. .

[8] Sun Y, Wang X, Tang X. Deep Convolutional Network Cascade for Facial Point Detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2013. .

[9] Zhang Z, Luo P, Loy CC, Tang X. Learning Deep Representation for Face Alignment with Auxiliary Attributes. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2016;38(5):918–930.

[10] Ranjan R, Patel VM, Chellappa R. HyperFace: A Deep Multi-Task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017;41(1):121–135.

[11] Trigeorgis G, Snape P, Nicolaou MA, Antonakos E, Zafeiriou S. Mnemonic Descent Method: A Recurrent Process Applied for End-to-End Face Alignment. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2016;2016-December:4177–4187.

[12] Guo X, Li S, Yu J, Zhang J, Ma J, Ma L, et al. PFLD: A Practical Facial Landmark Detector. 2019 feb. Available from: http://arxiv.org/abs/1902.10859.

[13] Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning. arXiv preprint arXiv:210611342. 2021.

[14] Khabarlak K KL. Fast Facial Landmark Detection and Applications: A Survey; 2021. .

[15] Wu Y, Ji Q. Facial Landmark Detection: A Literature Survey. International Journal of Computer Vision. 2019 feb;127(2):115–142.

[16] Timothy F Cootes GJECJT. Comparing Active Shape Models with Active Appearance Models; 1999. .

[17] T F Cootes GJECJT. Active Appearance Models; 1998. .

[18] Cootes DCT. Feature Detection and Tracking with Constrained Local Models; 2006. .

[19] Jason M Saragih SLJFC. Deformable Model Fitting by Regularized Landmark Mean-Shift; 2011. .

[20] Duffner S, Garcia C. A connexionist approach for robust and precise facial feature detection in complex scenes. In: Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium. vol. 2005. IEEE Computer Society; 2005. p. 316–321.

[21] Bradley D. Analysis and Improvement of Facial Landmark Detection. 2019. Available from: https://www.researchgate.net/publication/332866914.

[22] Alom MZ, Taha TM, Yakopcic C, Westberg S, Sidike P, Nasrin MS, et al.. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches; 2018.

[23] Zhu X, Ramanan D. Face detection, pose estimation, and landmark localization in the wild. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition; 2012. p. 2879–2886.

[24] Luo P, Wang X, Tang X. Hierarchical face parsing via deep learning. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition; 2012. p. 2480–2487.

[25] Wu Y, Wang Z, Ji Q. Facial feature tracking under varying facial expressions and face poses based on restricted boltzmann machines. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition; 2013. p. 3452–3459.

[26] Liang L, Fang RX, Sun WJ. Face Alignment via Component-based Discriminative Search;.

[27] Valstar M, Martinez B, Binefa X, Pantic M. Facial Point Detection using Boosted Regression and Graph Models;.

[28] Microsoft Face SDK;.

[29] Luxand Face SDK;.

[30] Kaggle Competitions - Facial Keypoints Detection;. Available from: https://www.kaggle.com/c/facial-keypoints-detection.

[31] Wang X, Wang K, Lian S. A Survey on Face Data Augmentation. CoRR. 2019;abs/1904.11685. Available from: http://arxiv.org/abs/1904.11685.

[32] Taylor L, Nitschke G. Improving Deep Learning using Generic Data Augmentation. CoRR. 2017;abs/1708.06020. Available from: http://arxiv.org/abs/1708.06020.

[33] Buslaev A, Iglovikov VI, Khvedchenya E, Parinov A, Druzhinin M, Kalinin AA. Albumentations: Fast and Flexible Image Augmentations. Information. 2020;11(2). Available from: https://www.mdpi.com/2078-2489/11/2/125.

[34] Getting Started with Facial Keypoint Detection using Deep Learning and PyTorch;. Available from: https://debuggercafe.com/getting-started-with-facial-keypoint-\detection-using-pytorch/.

[35] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization; 2017.

[36] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 2014;15(56):1929–1958. Available from: http://jmlr.org/papers/v15/srivastava14a.html.

[37] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift; 2015.

[38] torch.nn;. Available from: https://pytorch.org/docs/stable/nn.html.

[39] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition; 2015.

[40] VGG's CNN Code;. Available from: https://www.kaggle.com/chr9843/myfacialkeypointsnb.

[41] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al.. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications; 2017.

[42] Wu S, Member S, Junhong Xu I, Zhu S, Guo H. A Deep Residual Convolutional Neural Network For Facial Keypoint Detection with Missing Labels; 2017. Available from: http://www.elsevier.com/open-access/userlicense/1.0/.

[43] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition; 2015.

[44] theAEGuysCode. colab-webcam;. Available from: https://debuggercafe.com/getting-started-with-facial-keypoint-\detection-using-pytorch/.

[45] OpenCV. Haar Cascade Face Detection;. Available from: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

[46] OpenCV. DNN;. Available from: https://github.com/opencv/opencv/tree/master/samples/dnn.

[47] Feng ZH, Kittler J, Awais M, Huber P, Wu XJ. Wing Loss for Robust Facial Landmark Localisation with Convolutional Neural Networks;.

[48] Lai S, Chai Z, Li S, Meng H, Yang M, Wei X. Enhanced Normalized Mean Error loss for Robust Facial Landmark detection;.