Prediction of Facial Attributes Using a Deep Convolutional Neural Network

Addi Djikic, Christian Chamoun & Gabriel Carrizo
addi@kth.se chch@kth.se carrizo@kth.se

Royal Institute of Technology, KTH

Abstract. In this project we have retrained the final layer of a Convolutional Neural Network (CNN) called Inception to recognize if an image of a person is male or female and whether the person is wearing glasses or not.
We show an approach to this by dividing the MTFL dataset into four classes. By conducting a couple of experiments, we reach the conclusion that this is not the optimal approach because it limits the amount of training images in each class. In addition, we tried to alternate a couple of parameters in order to improve the accuracy of the network. We also tested our classifier with our own images to evaluate its generalizability. The best network we were able to train had an accuracy of 75.23% using all the four classes. We also tried a different classifier with only two classes (male and female) which had an accuracy of 85.18%.

Keywords: Artificial Neural Network, Inception, Deep Learning, Facial Attributes, MTFL, Multi-label Classification, Transfer Learning, TensorFlow

1 Introduction and Problem Formulation

Deep learning is an emerging field in both research and companies integrating different deep learning techniques in their systems. Google is a perfect example where they use deep learning for their search engine in order to give every user a unique and user-friendly experience.

One of the major fields in deep learning is recognizing different facial attributes. For example, Neeraj et al. were among the first to create a search engine based entirely on faces and their attributes[1]. Their project, among many others, bear similarities to our project in the sense that they are trying to classify multiple facial features. Another similar project is Huang, Z. et al. using facial recognition for still-to-video recognition. [2]

The strive for a robust model for determining facial attributes is an intriguing challenge for researchers globally. The problem is particularly difficult because every individual's appearance is unique and there are an infinite number of parameters that can make the same face appear different in separate images. For example, camera quality, head pose, frown, smile etc. are all factors that can make recognizing facial features problematic.

Deep learning, computer vision and artificial intelligence (AI) are interesting fields for many scientists and businesses. Face recognition is a major part of all three of these fields. Due to increased access to high computational power and the emergence of higher performing GPUs, the limits to what a computer can do are being pushed [3]. An example of what facial recognition techniques may be applied to in the future is identity detection and tracking. Hitachi is on the verge of development of such a method which implements face detection techniques.[4]

In this study, we will investigate how well we can accomplish a classifier that determines multiple attributes from a given image of a face. More specifically, we will divide the images into classes based on gender (male or female) and whether the person is wearing glasses or not. This will be done by using a pre-trained network called *Inception* (see 3.1) and retraining its final layer while leaving the remaining layers as they are (see Fig. 2 for a visualization of the network). The image dataset that will be used both for training and testing is the *Multi-Task Facial Landmark* (MTFL) dataset. The dataset is discussed further in section 3.2 and an example of it can be seen in Fig. 1. The main goal is to optimize the network's performance by investigating the effect of changing the available parameters of the network.

Fig. 1. Images from [5], which displays some examples of the images within the data set. The images are divided into four classes, "Male with glasses", "Male without glasses", "Female with glasses" and "Female without glasses".

2 Accomplishments of Previous Related Studies

With today's modern object and attribute recognition techniques, models can have a vast amount of parameters [6]. We looked at how others have implemented the same technique on similar networks.

A study by Zhang et. al. [7] was made with the same image dataset [5] as we use in this study. Their study shows how landmark detection is not a single and independent problem and that with auxiliary information (i.e. head pose and facial landmarks), the results can be greatly improved. Their results gave

a positive outcome by combining the auxiliary data together with the heterogeneous facial attributes. However, we performed our method without the pose information, we only used a selection of the attributes including the person's gender and whether they are wearing glasses or not.

Another study by Zhang et. al. [8] shows improved robustness through multi-task learning. They explain that different tasks imply different learning difficulties and convergence rates because of inter-class correlation. For example, if we classify 'females' in a testset, and then classify 'females with glasses' within the same testset, we are expected do get a better result for the 'female'-classification because of a larger amount of training images. Zhang et. al. [8] formulated a task-constrained loss function to take this into consideration, where they back propagate jointly to improve their generalization with landmark detection. They also discuss an 'early stopping' method which basically prevents the joint learning from overfitting.

Another analysis of the pre-trained inception model we used was done by Md. Rayed Bin Wahed et. al. [9]. They analyzed the performance of the Inception model and compared with other models on facial expressions (seven classes of expressions, 4000 steps of training). They also compared with their own implemented model and concluded that Inception-v3 model is the best performing high resolution image classifier. This made choosing the Inception model a clear choice for us.

Similar studies in gender classification have previously been conducted with different kinds of data. In the paper [10], they use people's blogs as inputs to the classifier in order to determine the gender of the author of that blog. They succeed to do this with an accuracy of 86 %. There are also studies where they attempt to modify the architecture of the Inception model in different ways [11]. We will however, use the standard V3 Inception model for our training due to the time restriction in this project.

3 Approach and Theory of Detecting Facial Attributes

Object recognition networks can take several weeks to train. Therefore, we have used a method called *transfer learning*. Transfer learning is the idea of taking a pre-trained network, and retraining it for new classes by using existing weights and filters. In other words, we will only retrain the final layer of the network to suit our purposes. Of course it is better to do a full training but transfer learning is more efficient and produces sufficient results for many applications.

In this section, we will explain how me managed to retrain the Inception model's final layer to suit our implementation.

3.1 Inception Model

The Inception model is a convolutional neural network (CNN) that is used for image classification and is therefore suitable for our task. As mentioned previously, it can take several weeks to fully train the Inception model on a super computer

and we will therefore use a pre-trained network to classify our images. The Inception model has nearly 25 million parameters and uses 5 billion multiply-add operations to classify an image [11]. By using the pre-trained model, classifying an image takes only a fraction of a second. In Fig. 2 we can see the structure of the Inception model. We can see that the Inception model has complex structure and consists of several layers. The red boxes in Fig. 2 represent two softmax outputs. the leftmost is used during training and the latter for classifying images.

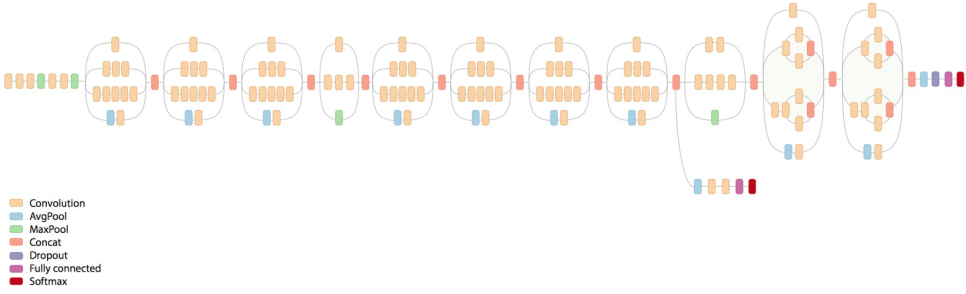

Fig. 2. Image taken from [12], which shows the structure of the Inception model.

The softmax function is used to predict the probability of the i:th class given an input vector $\boldsymbol{x}$ (the image) and the weight vector $\boldsymbol{W}$

$$P = \frac{e^{\boldsymbol{x}^T \boldsymbol{W}_i}}{\sum_{k=1}^{K} e^{\boldsymbol{x}^T \boldsymbol{W}_k}}, \tag{1}$$

where K is the number of classes, K=4 in our case.

3.2 MTFL Data Set

Before starting any training, we have to gather a dataset. There are a lot of different datasets available for learning purposes. For our implementation, the Multi-Task Facial Landmark (MTFL) dataset [5] is suitable. MTFL contains images of faces and are labeled with four classes including gender (male/female), smiling (yes/no), wearing glasses (yes/no) and head pose (left profile/left/frontal/right/right profile). The dataset contains approximately 10000 training images and 3000 testing images. The dataset also includes a textfile that explain which classes each image belong to. In our implementation, we only focus on two attributes, gender and wearing glasses. Therefore we divided the training images into four classes: “Male with glasses” (1143 images), “Male without glasses” (4666 images), “Female with glasses (267 images) and “Female without glasses” (3928 images). Unfortunately, due to the limited amount of images in the dataset, the

amount of training images could not be equally divided amongst the classes. The textfile was also formatted so that it only contained a description of these four classes for each image. The images are not all equal in size and some of them are not color images but this will be taken care of by the Inception model since it processes the data.

3.3 TensorFlow

To build the retrainer, we used Tensorflow. Tensorflow is an open source software library that uses data flow graphs for machine intelligence tasks [13]. Tensorflow is developed by Google and can be used to build and train neural networks to detect different correlations in data. We will use it to build and train the final layer of the Inception model to classify images into the four classes presented in section 3.2.

3.4 Building the Retrainer

By using Tensorflow and Bazel which is Google's own software build tool [14], we could build the retrainer for the Inception model. The retrainer was used to retrain the Inception model on our own images.

3.5 Training

Once the retrainer was built, we could start the training process. The final layer of the Inception model was replaced by a new one that is based on the four classes described in section 3.2. The Inception model was originally trained on the classes from the ImageNet [15] dataset which did not include our images from the MTFL set. However, since we used transfer learning, we did not consider that the lower layers of the Inception model has been trained on other images, we only retrained the final layer and reused the lower layers.

The first step of the training is creating so called *bottleneck* for each image in the MTFL training set. A bottleneck is the data for each image up to the layer before the actual classification is done. These bottlenecks are stored so that computations are faster the next time training is ran. For instance, if we want to change any hyper-parameters and retrain the model, the training will be faster since the bottlenecks for each image can be reused.

When the bottlenecks are created, the final layer of the network is trained on the training images. The training was ran for 4000 steps by default. At each step, a mini-batch containing 10 images of the training set was randomly chosen, their respective bottlenecks were used as input to the final layer to get the images' predicted classes. The predicted classes were then compared to the correctly labeled classes to update the weights according to equation (2). The weights are later used during the back-propagation.

$$\boldsymbol{W}^k = \boldsymbol{W}^k - \eta \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{W}}, \tag{2}$$

where $\boldsymbol{W}$ are the weights, η is the learning rate, $\boldsymbol{J}$ is the cost function.

Since the weights were updated and used for the next step, the accuracy is improved during each iteration. After each step, the training accuracy, validation accuracy and cross entropy loss were also computed. The training accuracy is a measure of how many images that were correctly classified in the current mini-batch. The validation accuracy is the same as the training accuracy but computed on a different batch of images, separated from the training set. The network's performance can be monitored by looking at the training and validation accuracy. For instance, if the training accuracy starts to increase but the validation accuracy remains low, it means that the network is overfitting. If the network is overfitting, it means that it will focus on specific features on the training images which will not be useful for more general test images. The cross entropy is a measure of how well the learning process is going at each step. The goal is to minimize the cross entropy, as seen in (3), and therefore it is possible to monitor how well the learning is progressing by checking if the cross entropy is trending downwards after each step.

$$argmin_{\boldsymbol{W^k}} \sum_{i=1}^{N} l(f(\boldsymbol{x^k})) + \Phi(\boldsymbol{W}^k), \tag{3}$$

where N is the number of images in our dataset, $l(\cdot)$ represents the loss function, W the weight vector, $f(\boldsymbol{x^t})$ is a function of the the data $\boldsymbol{x}^t$ and Φ is denoted to the regularization term.

3.6 Evaluate the Classifier

Once the Inception model's final layer was retrained on our training images, we were able to test the classifier on our test images to get a measure of how accurate the classifier was. The classifier takes one test image as an input and returns the probabilities of the predicted classes for that image. Fig. 3 shows an example of how the classifier works.

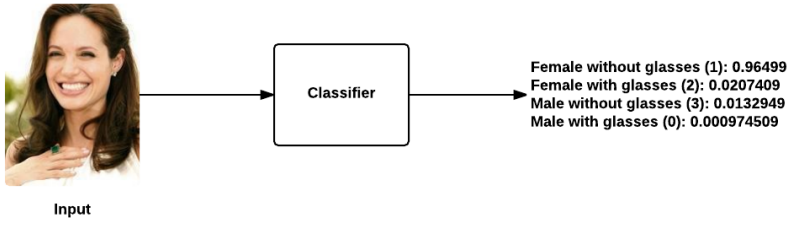

Fig. 3. An example of how the classifier takes an image as input and returns the probabilities of the predicted classes. *Note that the illustration is an example and not our result, see section 4 for our results on the test set.*

We can see in Fig. 3 that the classifier is $\approx 96.5\%$ certain that the input is a female without glasses, which is a correct classification.

Even if this is a good way of checking the accuracy of the classifier, we would like to input all our images in the testset (approximately 3000 images) and evaluate how many correctly classified images it outputs. Therefore, we wrote a script that inputs all the images to the classifier, stores the output predicted classes, compares the predicted classes to the labeled classes and counts how many correctly classified images it got.

The final accuracy of the classifier can then be calculated as the rate between the number of correctly classified images and the total number of images in the test set. Fig. 4 shows an example of the programs output.

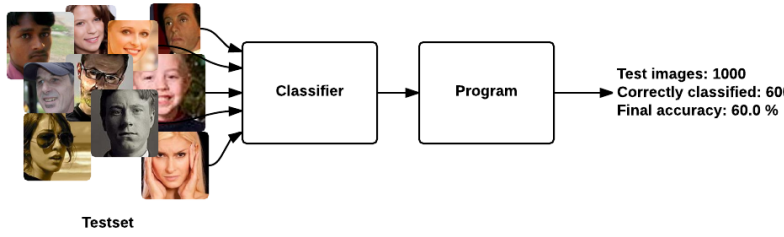

Fig. 4. An example of how the program evaluates the classifier on a set of test images. *Note that the illustration is an example and not our result, see section 4 for our results on the test set.*

4 Experimentation and Results

There are several parameters that can be adjusted in order to improve the network's accuracy. The *training steps* parameter can be adjusted to either train the network for a longer or shorter period. There are also a couple of parameters including: *random crop*, *random scale*, *random brightness* and *flip left/right* that could be enabled to deform, crop or brighten the training images in random ways before training. By distorting the training images, the classifier will learn to deal with real-life noise in the images. The final parameter that could be adjusted is the *learning rate*. The learning rate controls the magnitude of the updates, if it is small, the training will take more time but might improve the accuracy.

Our initial experimentations were performed with the default parameter settings for the network seen in Table 1.

4.1 Comparison of two Classifiers

The first experiment we conducted was to compare two different classifiers:

Table 1. Default parameters of the Inception net

Paramters	Values
Training steps	4000
Learning rate	.01
Flip left/right	0%
Random crop	0%
Random scale	0%
Random brightness	0%

(i) Classifier with four classes, Male with glasses (1143 images) Male without glasses (4666 images), Female with glasses (267 images) and Female without glasses (3928 images).
(ii) Classifier with two classes, Male (5807 images) and Female (4193 images).

As mentioned in section 3.2, dividing the training images into four classes as in classifier (i) made some of the classes contain significantly less images then the others.

Therefore, we wanted to conduct this experiment to find out both how the number of classes and the difference in sizes between the classes might affect the network's accuracy.

4.1.1 Classifier (i) With classifier (i) the final accuracy was 75.23%, computed on all test images and with the default parameter setting in Table 1.

Fig. 5 shows the training and validation cross entropy loss for all training steps.

Fig. 5. Image provided by Tensorboard, visualizing the entropy loss over 4000 training steps with classifier (i). The data has been smoothed to make it more distinct.

Fig. 6 shows how the training and validation accuracy changes for each training step.

Fig. 6. Image provided by Tensorboard, visualizing the accuracy over 4000 training steps with classifier (i). The data has been smoothed to make it more distinct.

By extracting the number of missclassified images from our program we can get an idea of which classes that were difficult to predict by the classifier. In Table. 2, the result from this experiment can be visualized.

Table 2. Result of checking missclassified images for classifier (i).

Class	No. of images	No. of missclassified	Percentage
Male with glasses	1143	173	15.1 %
Male without glasses	4666	203	4.4 %
Female with glasses	267	162	60.7 %
Female without glasses	3928	203	5.2 %

4.1.2 Classifier (ii) With classifier (ii) the final accuracy was 85.18%, computed on all test images and with the default parameter setting in Table 1.

Fig. 7 shows the training and validation cross entropy loss for all training steps.

Fig. 7. Image provided by tensorboard, visualizing the entropy loss over 4000 training steps with classifier (ii). The data has been smoothed to make it more distinct.

Fig. 8 shows how the training and validation accuracy changes for each training step.

Fig. 8. Image provided by tensorboard, visualizing the change in training and validation accuracy over 4000 training steps with classifier (ii). The data has been smoothed to make it more distinct.

By extracting the number of missclassified images from our program we can get an idea of which classes that were difficult to predict by the classifier, as we did for classifier (i). In Table. 3, the result from this experiment can be visualized.

Table 3. Result of checking missclassified images for classifier (ii).

Class	**No. of images**	**No. of missclassified**	**Percentage**
Male	5807	274	4.7 %
Female	4193	169	4.0 %

See section 5 for a discussion about the results of this experiment.

4.2 Testing Classifier (ii) on our own Images

The second experiment that we conducted was to test the classifier on our own images. We wanted to see how well the classifier worked on images that we provided ourselves. For this experiment we only used classifier (ii) with two classes, male and female. Before we inputted the images to the classifier they were re-sized to 250×250. We used 22 test images of our friends and 18 of them were correctly classified by our classifier, this gives a final accuracy of 81.81 %. Some of the images resulted in surprisingly high accuracy, between $95 - 99.5\%$.

4.3 Distortions and Occlusions

The third experiment that we conducted was to adjust the different distortion and occlusion parameters stated in Table 1. We wanted to see if the network's accuracy could be improved by adjusting these parameters. During this experiment, the learning rate was kept to its default value such that the result would not be influenced by it. However, the training images are being distorted in this experiment which means that the bottlenecks can not be reused and they have to be recreated for every new setting. Therefore, the training steps were lowered to 100 steps for faster computations.

Several parameter settings were tested. The different settings that were used can be seen in Table 4 below.

Table 4. Parameter settings.

Setting	A	B	C	D
Training steps	100	100	100	100
Learning rate	.01	.01	.01	.01
Flip left/right (%)	10	0	0	0
Random crop (%)	0	10	0	0
Random scale (%)	0	0	10	0
Random brightness (%)	0	0	0	10

When the network is training it will select some random images from the training set that will not be used during training. After the training is finished, it will use these images to test the network and provide a final accuracy on these images. During this experiment, we used this final accuracy to evaluate the effect of changing the parameters instead of computing the final accuracy on our own test set for every new setting.

The experiment was done on classifier (i) with four classes and the final accuracy for each setting is stated in Table 5 below.

Table 5. Final accuracy for the different settings.

Setting	Default	A	B	C	D
Final Accuracy	60.1%	56.6%	47.2%	59.0 %	60.1 %

4.4 Learning Rate and Training Steps

The last experiment we did was to adjust the learning rate and training steps to see if the network's accuracy could be improved further. The advantage with this experiment compared to changing the distortion parameters is that the bottlenecks could now be reused for every new run which was time-saving. This experiment was performed on both classifier (i) and (ii).

By studying the result provided by tensorboard in Fig. 5, 6, 7, and 8 we were able to find a reasonable values for the training steps. In the figures, it can be observed that 4000 training steps is unnecessary for both classifier (i) and (ii) since the training converges in an earlier state. Approximately the same accuracy was achieved when lowering the training steps to 1500 for classifier (i) and 500 for classifier (ii).

After finding reasonable values for the training steps we were able to experiment with a range of values for the learning rate. Table 6 shows the results for classifier (i) and (ii). Note that the accuracy was computed on a random set of the training images that the network provided after training and not on all the images in the MTFL testset.

Table 6. The results from adjusting the learning rate for both classifiers.

Classifier (i)		
Setting	***Learning rate***	***Accuracy***
a	1	47.1 %
b	0.1	72.6 %
c	0.01 (Default)	71.5 %
d	0.001	59.4 %
e	0.0001	52.4 %

Classifier (ii)		
Setting	***Learning rate***	***Accuracy***
a	1	85.6 %
b	0.1	86.8 %
c	0.01 (Default)	86.9 %
d	0.001	84.5 %
e	0.0001	81.4 %

5 Discussion

5.1 Comparison Between Classifier (i) and (ii)

The results from comparing classifiers (i) with four classes and (ii) with two classes turned out as we predicted. Classifier (ii) got a final accuracy of 85.18 %, computed on all our test images from the testset and classifier (i) got 75.23 %.

This shows that the classifier with only two classes got approximately 10 % better accuracy. By checking the number of missclassified images in each class for both classifiers we got a clear indication of why the accuracy is better for

classifier (ii). The reason can be visualized by studying the results in Table 2 and Table 3. For classifier (i), we can see in Table 2 that the percentage of missclassified images increases depending on the number of images in the class. It is clear that the "Female with glasses" class that only contains 267 images got the highest missclassified percentege of 60.7 %. This is because the network does not have enough of training images for that class in order to learn and be able to accurately predict images belonging to that class. This means that the overall final accuracy of the classifier will be affected negatively. If we investigate the result in Table 3, we can see that the number of images in each class are almost equal, therefore the percentage of missclassified images only differ by 0.7 % which results in a better classifier with higher final accuracy.

However, considering that classifier (i) has unequal amount of images in each class it still produces good results and can recognize more attributes compared to classifier (ii) which only recognizes gender.

5.2 Testing with our own Images

The result from testing with our own provided images turned out to be pretty good, we got a final accuracy of 81.81 %. This shows that classifier (ii) was trained well and was able to correctly classify completely unknown images. It is also an indication that the learning process was not overfitted.

However, the result is not fully reliable since we only had 22 test images to evaluate the result. The difference in accuracy between providing our own test images and using the test images from the MTFL set is approximately 5 % in favor of the MTFL testset. This might be because the test images in the MTFL set are all profile pictures that are taken at close range but our own images were taken from different ranges.

5.3 The Effect of Distorting the Images

In Table 5 we can see the result of distorting the images. The experiment showed that the distortion had a negative effect in every case except for the random brightness distortion. We think this is because we already have limited our dataset too much by dividing 2 classes in to 4, so the distortion only limited our already sparse dataset even further. With a larger dataset we think this would help generalize the classifier.

In addition, since the distortion of the images happens randomly, we saw that the accuracy could alternate with approximately $\pm 5\%$ for the same setting. However, due to the timescale of this experiment, we settled with taking the first value for each setting instead of perhaps taking the average of a couple of runs which might have been better. Furthermore, we only tested the recommended value of 10 % for each parameter at the time. The results might have been positive if we had tried different values and also the combination of different distortions together. A *random search* would most probably help improve the results.

5.4 The effect of adjusting the learning rate and training steps

Lowering the training steps for both classifier (i) and (ii) turned out to produce approximately the same accuracy. It is useful to lower the training steps since the training process becomes shorter without affecting the result.

We can see in Table 6 that adjusting the learning rate gave either worse or similar results to the default value for both classifiers. Therefore, the default value was suitable for these classifiers. However, the learning rate was adjusted while the training steps where held constant, so a random search for the optimal combination of the parameters might have resulted in a different outcome.

5.5 Future Work

Another possible implementation is "k-networks". Since we got sufficient results from dividing the dataset into two classes, one could implement k-networks where each network classifies two classes. For instance, network A classifies male/female, network B classifies glasses/no glasses and network C classifies smiling/not smiling. These networks could then be combined so that one image is classified first by A, then B and lastly C in order to get an accurate classification of the input image.

6 Conclusion

This study has shown that it is possible to take a pre-trained network, retrain its final layer for new classes in order to classify images of your own choice.

The Inception model turned out to be suitable for this purpose. It was previously trained on about 1000 different classes but it was possible for us to use it to classify our own images that were divided into both two and four classes with a relatively high accuracy in both cases.

The MTFL data set was mostly useful for our classifier with two classes. We saw that increasing the number of classes affected the final accuracy negatively. This is because the data set only contains 10000 training images so dividing it into four classes resulted in classes with different number of images in each of them. In order to experiment with more than four classes and more attributes, such as smiling/not smiling, we would have needed more images in the data set. Due to our experiment, we are certain that increasing the number of classes and attributes further would result in an even worse accuracy since we clearly could observe a pattern of how the number of images in a class is correlated with the number of missclassified images for that same class.

In conclusion, this type of transfer learning that we studied in this paper is an efficient way of creating your own classifier. With access to other dataset there are endless of possibilities to explore. We saw how the classifier could be used on our own images of our friends and by experimenting with all the available parameters, the optimal settings could be found to boost the accuracy of the classifier even further.

References

1. Kumar, N., Belhumeur, P., Nayar, S.: Facetracer: A search engine for large collections of images with faces. In: European conference on computer vision, Springer (2008) 340–353
2. Huang, Z., Zhao, X., Shan, S., Wang, R., Chen, X.: Coupling alignments with recognition for still-to-video face recognition. In: Proceedings of the IEEE International Conference on Computer Vision. (2013) 3296–3303
3. Hof, R.: Is artificial intelligence finally coming into its own - mit technology review, https://www.technologyreview.com/s/513696/deep-learning/ (2017)
4. Smith, M.: Hitachi reveals new ai for real-time identity detection and tracking, http://www.networkworld.com/article/3184345/security/hitachi-reveals-new-ai-for-real-time-identity-detection-and-tracking.html (2017)
5. Zhang, Z., Luo, P., Change Loy, C., Tang, X.: Facial landmark detection by deep multi-task learning, http://mmlab.ie.cuhk.edu.hk/projects/tcdcn.html (2017)
6. TensorFlow: How to retrain inception's final layer for new categories, https://www.tensorflow.org/tutorials/image-retraining (2017)
7. Zhang, Z., Luo, P., Loy, C.C., Tang, X.: Learning deep representation for face alignment with auxiliary attributes. IEEE transactions on pattern analysis and machine intelligence **38**(5) (2016) 918–930
8. Zhang, Z., Luo, P., Loy, C.C., Tang, X.: Facial landmark detection by deep multi-task learning. In: European Conference on Computer Vision, Springer (2014) 94–108
9. Wahed, M.R.B.: Comparative Analysis between Inception-v3 and Other Learning Systems using Facial Expressions Detection. PhD thesis, BRAC University (2016)
10. Bartle, A., Zheng, J.: Gender classification with deep learning. (2003)
11. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 2818–2826
12. Research Blog: Train your own image classifier with inception in tensorflow, https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.htm (2017)
13. TensorFlow: Open-source software library for machine intelligence, https://www.tensorflow.org/ (2017)
14. Bazel: Bazel.build, https://bazel.build/ (2017)
15. ImageNet: Large scale visual recognition challenge, http://www.image-net.org/challenges/lsvrc/ (2017)