Using Unsupervised Cross-Domain Generation For Mustache Transplant

Workshop in Machine Learning Applications for Computer Graphics

Yoav Ikan, Eliran Kachlon and Ron Mokady

August 25, 2018

1 Introduction

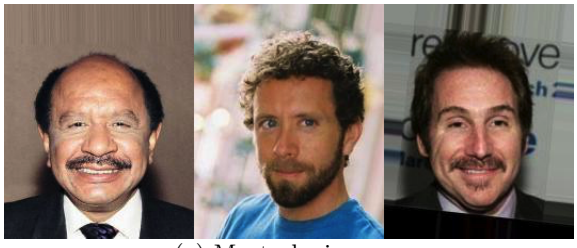
The task of changing a facial feature using image-to-image translation, that is, generating a new image from a given facial image such that the new image differs from the original only in a particular facial feature, has raised a lot of interest over the years. It has applications in many areas, including social media, and cinematic-movies. For example, in the shooting of the movie "Justice League", the studio had to digitally edit out Henry Cavills mustache.

The subject of our project is adding a mustache to a mustachesless image. Note that this problem can be thought of as a translation between two domains, a source domain that contains only mustacheless images and a target domain that contains only facial images with a mustache.

We replicate the architecture presented in [TPW16], that consists of an autoencoder and a discriminator, and the learning is unsupervised. In [TPW16], the source domain contained facial images and the target domain contained emojis. Hence, the project is not a trivial implementation of the article, as both our source and target domains are complicated, while in the article the target domain is much simpler than the source domain.

It should be mentioned that all the images we present in the text are test images, that is, images that our network was not trained on.

Organization of the text. In Section 2 we present our dataset and preprocessing. In Section 3 we introduce our various attempts to construct an autoencoder and the difference between the requirements from an autoencoder in [TPW16] and the requirements from an autoencoder in our project. In Section 4 we present the discriminator. In Section 5 we present the full network and results. Finally, in Section 6 we present our conclusions.



(a) Mustache images.

(b) Mustacheless images.

Figure 1: Example for CelebA imagess.

2 Dataset and Preprocessing

Our dataset is CelebA. As our purpose is adding a mustache to male-faces images, we used only the male faces from the dataset. We used about 70,000 pictures of men for the training of the autoencoder. For the training of the complete net, we used 7,800 pictures of men with a mustache and 7,800 pictures of men without a mustache. As the dataset attributes are not accurate, we manually checked that all the mustache images indeed contains significant mustache. We remind that we deal with unsupervised learning, thus the set of mustache images and the set of mustacheless images contain images of different people.

All original images are 178×218 RGB. In the preprocessing, we scale the images to the size of 128×128, as we explain in Section 4, and we also normalize the images. Note that the faces are not aligned, which makes the training process more difficult.

3 The Autoencoder

3.1 Original architecure

In [TPW16], the pretrained "DeepFace" net was used as the encoder. "DeepFace" maps the input images to a 256-dimensional representation, while the decoder maps a 256-dimensional representation to 64×64 RGB images through a network with 5 blocks, each consisting with an upscaling convolution, batch-normalization and Leaky-ReLU.

3.2 Decoder

At first, we implemented the decoder as described in the article, but soon realized that we get blurry pictures (see for example Figure 2). In [JAL16], the net, which is intended for super-resolution, consists of residual blocks that should improve the sharpness of the output images. Hence we decided to add 4 residual blocks which indeed improve the sharpness of the image significantly (see for example Figure 3). In the following sections, the decoder always contains residual blocks.

3.3 Pretrained Encoder

We first tried to use a pretrained implementations of "openface" and "sphereface" for the encoder, since in the article they used a pretrained net for face recognition as the encoder. We soon realized that those encoders consider only the facial features while ignoring the background (see Figure 4), because those nets were trained originally only for face recognition. It was good enough for the original article, where the target domain was emojis, which require only the facial features but don't require any background, but it is not good enough for our purpose since we also care about the background and the sharpness of the image.

Figure 2: Results of autoencoder with no residual blocks.

Figure 3: Results of autoencoder with residual blocks.

We conclude that there is a big difference between our domain translation and the domain translation in the original article. In the original article only the facial features were needed for the translation and the image after the translation is much different from the original image, while in our case, we need all the image features for the image translation and the image after the translation is similar to the original image (we only add a mustache).

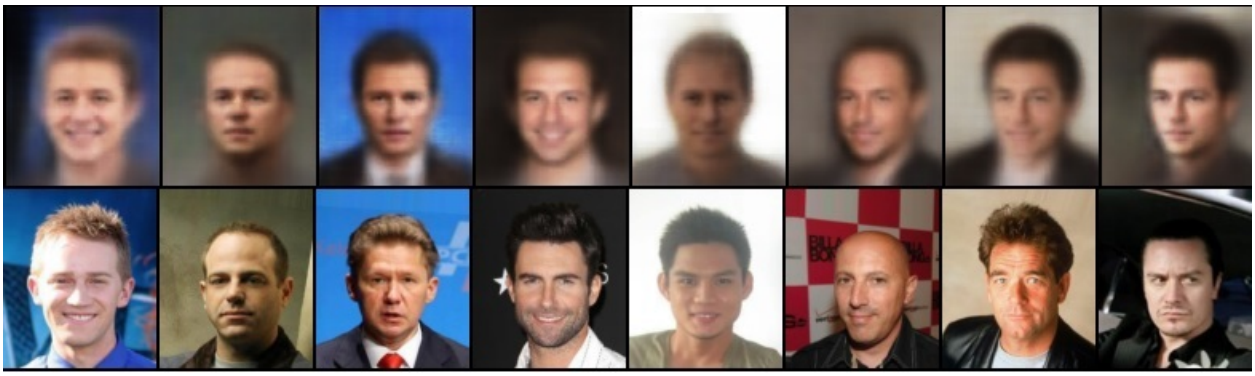


Figure 4: Results of sphereface test. The facial features are clear, but the rest of the image is blurry.

3.4 Trained Encoder

Since "openface" and "sphereface" failed, we tried to construct and train an encoder by ourselves. We took a simple encoder, consisting of 5 convolution layers with batch-normalization and Leaky-ReLU, ending with

a fully-connected layer. We also added 2 residual blocks, but we discovered that they don't improve the quality of the image. The results of the trained autoencoder were better than those of the pretrained encoder as can be seen in Figure 5. We also tried to add 1×1 convolution to each block as suggested in the article, but it didn't improve our results.

Figure 5: Results of the trained encoder, when the decoder consists of residual blocks.

3.5 Training Details and Results

We used MSE loss and Adam optimizer, where the learning rate is 0.0001. Our dataset consists of approximately $70,000$ images of male faces. We trained the net for 450 epochs. As can be seen on Figure 5, the output of our encoder is similar to the original image, but the image is a bit blurry, thus the results are not satisfactory.

We tried to improve the quality of the images by training the autoencoder with a discriminator that distinguishes between an image from the dataset and images from the output of the autoencoder, but the quality of the images did not improve.

We hoped that training the decoder with the discriminator, as done in the original paper, will improve the quality of the images, since the discriminator is required to distinguish real mustache images from images which are the output of the encoder (see Section 5 for more details). Note that emojis contain fewer details then real images, thus the task of the original autoencoder from the article is much simple then our autoencoder.

3.6 Increasing the Size of the Images

When training the discriminator, we discovered that we get better results when the images are of size 128×128 (see Section 4 for more details). Hence we increased the size of the images from 64×64 to 128×128 and trained the autoencoder again, in the same way.

4 The Discriminator

The original architecture contains a discriminator which, among the rest, distinguishes between mustache images and mustacheless images. We decided to test our discriminator before assembling the full architecture in order to make sure our discriminator indeed recognize the mustaches and doesn't suffer from overfitting. We used 1% of our images as a test dataset and the rest of the images as train dataset. The following describes our attempts to construct a discriminator that distinguishes mustache images from mustacheless images.

4.1 Avoiding Overfit

As our dataset contains only around $7,000$ mustache images, every large net can learn all the train images instead of recognizing mustaches. Thus, we tried to make it hard for the net to learn all train images in the following way. First, with probability 0.5 we flipped the image horizontally. Second, we inserted decaying Gaussian noise to the train images with expectation zero and initial variance 0.1. The decaying of the variance is linear (with slope $\frac{-0.1}{\#\text{Epochs}}$), until it gets to 0.001, where it remains constant. We have also tried exponential decay but it didn't improve the results.

4.2 Loss Criterion

The loss used in [TPW16] is cross entropy, since there are 3-categories which the discriminator should distinguish (see Section 5 for more details). For now, we are only interested in checking whether the discriminator distinguishes mustache-images from mustacheless-images, so we use the binary cross entropy loss (BCE). Hence, for now the discriminator only outputs a single number for each image, instead of a 3-tuple.

4.3 Image Size

To decrease our training time, we used 64×64 images for training the autoencoder. However, when we increased the image size to 128×128 we got better accuracy. It makes sense since in 64×64 the mustache width is sometimes only a few pixels, which make the task of detecting it harder than the case of 128×128 image in which the mustache consists of much more pixels. Thus, we train the autoencoder again with 128×128 images.

4.4 Using Initial Weights of a Pretrained Classifier

One method of shortening training time for discriminators is using initial weights of a pretrained classifier. The state of the art classifiers excels in recognizing a wide variety of generic features, which is an important attribute for our discriminator. Thus, starting the train from a pretrained classifier should have caused our discriminator to excel in recognizing the mustaches. We used the famous "resnet18", But unfortunately this discriminator performs poorly, probably due to overfitting.

4.5 Discriminator Without Fully Connected Layers

To avoid overfitting, we decided to try using a net that can't memorize all the train images. Thus, we have tried a net that doesn't contain a fully connected layer, which has a lot of weights. The net consisted of 7 convolutional layers with stride 2. After every one of those layers there is batch-norm and leaky-ReLU layers, and in addition the net contained also 4 residual blocks, in a similar way to our decoder. Finally, there is a sigmoid function in the last layer.

When training this discriminator, the loss consistently got stuck, and from the output of the discriminator, which was about the same for mustache-images and mustacheless-images, we concluded that the discriminator could not distinguish mustache-images from mustacheless images. Thus, we abandoned the idea of not using a fully connected layer.

4.6 Our Best Discriminator

Finally, we built a discriminator which consists of 7 layers of stride 2 convolution, each followed by batch-norm and leaky ReLU layers. At the end there is a fully connected layer, followed by sigmoid. This discriminator achieved over 90% success rate on both mustache and mustacheless test images. We note that the discriminator reached those achievements after 32 epochs, but also achieve 85% success rate after only two epochs, when learning rate is 0.0001.

5 The GAN

We first introduce the architecture of [TPW16]. We then show our first simplified attempt, our attempt at using the original architecture, and also an attempt to use the new architecture.

5.1 The Article Architecture

The architecture of [TPW16] consists of an autoencoder and a discriminator. The autoencoder consists of a pretrained encoder f which is not being trained, and a decoder g. In our case both f and g are pretrained, as described in Section 3, and during the training of the whole net we train only g. Namely, when we mention that we trained the generator, it means that we trained g.

The discriminator D has 3 categories to distinguish. The first is the output of the autoencoder when applied on a mustacheless-image from the dataset. The second is the output of the autoencoder when applied on a mustache-image from the dataset. The third is a mustache-image from the dataset.

We train the discriminator to distinguish between those 3 categories, while we want g to fool the discriminator.

We denote the mustache domain as t (target domain), the mustacheless domain as s (source domain) and we denote distance as d (MSE loss). The loss of the discriminator is given by,

$$L_D = -E_{x\in s}\log D_1(g(f(x))) - E_{x\in t}\log D_2(g(f(x))) - E_{x\in t}\log D_3(x),$$

while the loss of the generator is given by,

$$L_G = L_{GANG} + \alpha L_{CONST} + \beta L_{TID} + \gamma L_{TV}$$

$$L_{GANG} = -E_{x\in s}\log D_3(g(f(x))) - E_{x\in t}\log D_3(g(f(x)))$$

$$L_{CONST} = \sum_{x\in s} d(f(x), f(g(f(x))))$$

$$L_{TID} = \sum_{x\in t} d(x, f(g(x)))$$

L_{TV} is an anisotropic total variation loss which is added in order to slightly smooth the resulting image. See Figure 6 for the net's scheme as presented in the original article. Note that in the original paper $\alpha = 100$, $\beta = 1$ and $\gamma = 0.05$.

Note that the discriminator is required to distinguish the output of the autoencoder from mustache-images from the dataset. Hence, we hoped that when training the generator with the discriminator with those losses, the quality of the images of the autoencoder will improve.

It should be mentioned that in all the following training methods we used decaying Gaussian noise and horizontal-flipping, as mentioned in Section 4.

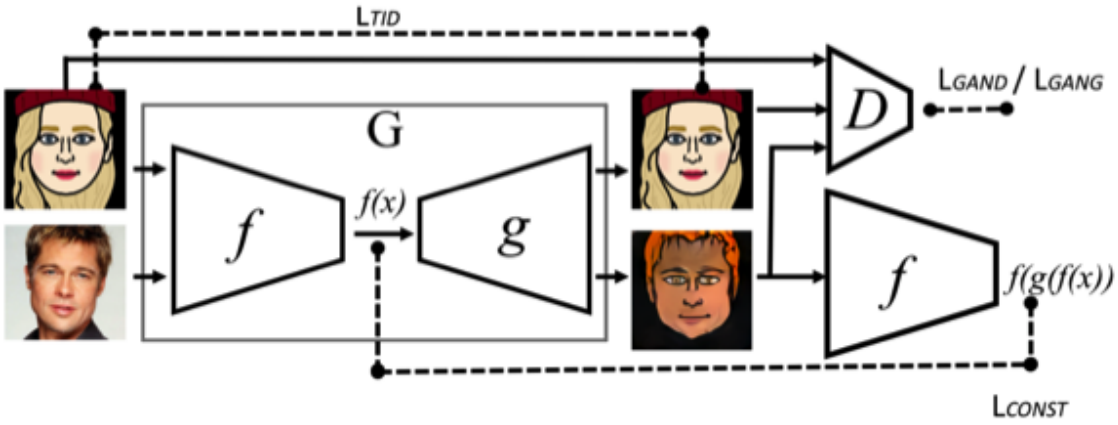


Figure 6: Architecture scheme from the original paper, [TPW16], in our case the real image represent mustacheless image and the emoji represent mustache image.

5.2 Simpler Architecture

Since the autoencoder output blurry images, we were concerned that the discriminator will easily distinguish between the autoencoder outputs and images from the dataset which are not blurry. Thus, for simplicity, at first we required from the discriminator only to distinguish the output of the autoencoder when applied on a mustache-image from the output of the autoencoder when applied on a mustacheless-image, and we required the generator to fool it.

Therfore, we decided to begin with lighter architecture which consists of L_{TID}, L_{CONST} and a variation the L_{GANG}:

$$L_D = -E_{x \in s} \log D_1(g(f(x))) - E_{x \in t} \log D_2(g(f(x)))$$

$$L_{GANG} = -E_{x \in s} \log D_2(g(f(x)))$$

Where $\alpha = 100$ and $\beta = 1$. After 9 epochs we got impressive mustaches, as can be see in Figure 7, but the images were still blurry. In addition, as the training progressed the faces slowly deteriorated, as can be seen in Figure 8.

As the output of the autoencoder became less and less similar to the original image, we decided to increase the hyperparameters α and β. We increased β in order to increase the weight of L_{TID}, i.e. the MSE loss. Similarly, we increased α in order to make the output of the generator close to the original image in the latent space, hoping that it will stop the deterioration. We discovered that increasing α did not change much in the output of the image, while increasing β did improve the quality of the image. We got the best results with $\alpha = 130$ and $\beta = 150$, where after about 500 epochs the results were better, but the images were still blurry, see Figure 9.

The above training of the generator and the discriminator was done in the following way. We first trained the discriminator until its average-loss over an epoch was low, to a point where we were sure that it distinguishes mustache-images from mustacheless images. We then trained the generator to a point where its averages-loss was low, and so on (From now on, we refer to this way of training as way I.). This method may be referred to as "balance loss via statistics" or "find a schedule to uncollapse training".

We also tried to train both the discriminator and the generator together. At first, we tried to alternate between the generator and the discriminator in each batch (that is, we trained the discriminator for one batch, then the generator, and so on), but got bad results. We assumed that the discriminator was much better than the generator so we tried to train the generator more than the discriminator (up to a 1:100

Figure 7: Using simple architecture we succeeded in adding mustaches, but the images are still blurry.

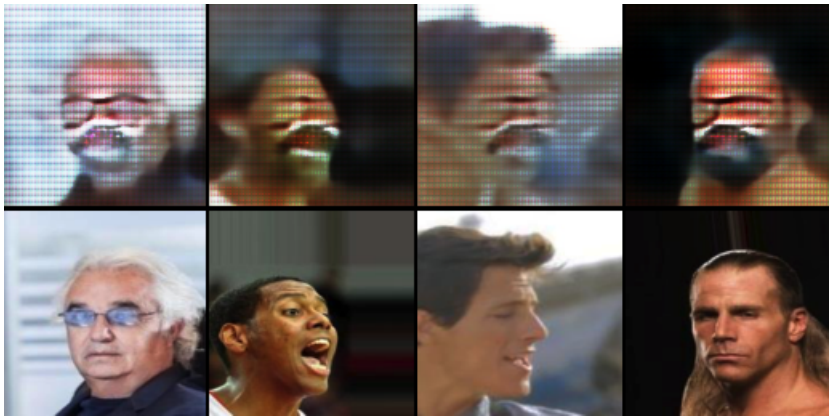


Figure 8: As we continue to train using the simple architecure, the images slowly deteriorates.

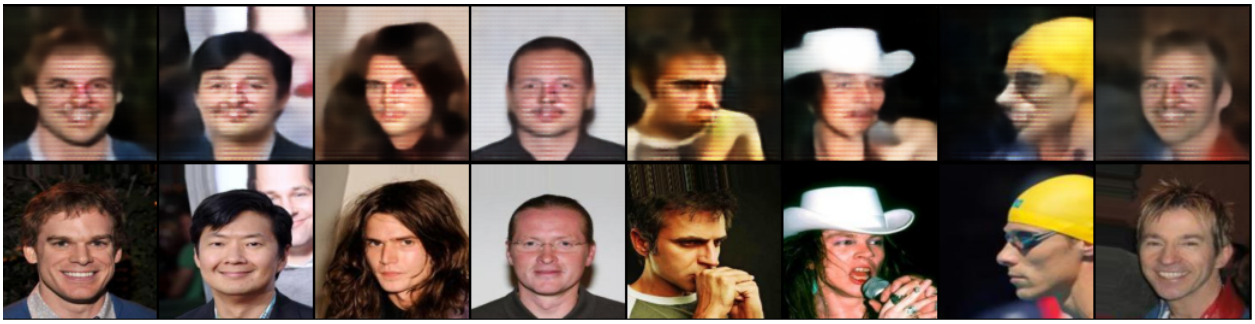


Figure 9: The increaing of the hyperparametrs stop the deterioration, but the mustaches are less significant.

ratio), it improved the results, but the results were still unsatisfactory as you can see in Figure 10. (From now on, we refer to this way of training as way II.)

As we were unable to improve the quality of the images in this architecture, we moved to the original architecture, hoping that the real L_D and L_{GANG} will improve the quality of the images.

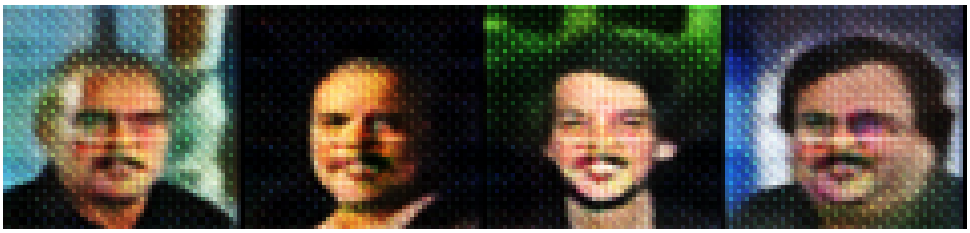


Figure 10: Bad result using way II training.

5.3 Back to the Original Architecture

We changed the output of the discriminator to output a 3-tuple, and changed the losses according to the original architecture (we changed the loss of the discriminator and the generator to cross-entropy).

We started with $\alpha = 100$ and $\beta = 1$ like in the article, and at the beginning, we took $\gamma = 0$. The results of both ways of training were bad, and while the loss of the discriminator was low, the loss of the generator kept growing.

As before, we tried to improve the results by increasing β. Unfortunately, this didn't help. For low β we got the results shown in Figure 11. We increased β again to get the results in Figure 12. As one can see, in both cases the results are unsatisfactory, and no mustache was added. In both training methods, the discriminator loss was very low, while the loss of the generator kept growing. We also tried to increase γ a little, but this did not improve the results.

We conclude that the original architecture doesn't fit the task of adding a mustache to an image.

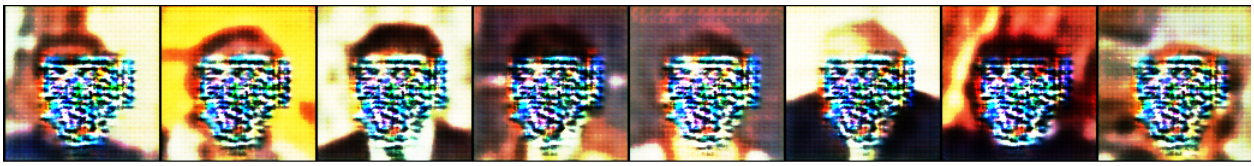


Figure 11: Bad results using the original architecture.

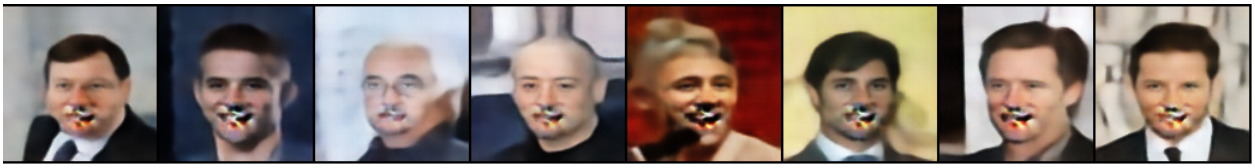


Figure 12: The image is more clear as β is larger, but no mustache was added.

5.4 New Architecture

One of the main weaknesses in our attempts so far was the fact that the autoencoder outputs blurry images as presented in Section 3. Hence, we conceive an architecture in which we don't use an autoencoder, however we use a generator which "creates a mustache", i.e. gets as input an image such that the subtraction of the generator's output from the input image is a mustache image. Namely, the generator generates only a mustache, which we transplanted on the image as can be seen in Figure 13. We train the discriminator to distinguish between mustache images and images created by substraction of the generator's output from mustacheless image. We train the generator to fool the discriminator and also to make sure that mustache

image remain the same after "adding the mustache". The losses are defined as follows:

$$L_D = -E_{x \in s} \log D_1(x - G(x)) - E_{x \in t} \log D_2(x)$$

$$L_G = L_{GANG} + \beta L_{TID}$$

$$L_{GANG} = -E_{x \in s} \log D_2(x - G(x))$$

$$L_{TID} = \sum_{x \in t} d(x, x - G(x))$$

Since there is no autoencoder, the results are sharp as the dataset images. But, unfortunely the generator added unwanted red marks as can be seen in Figure 14.

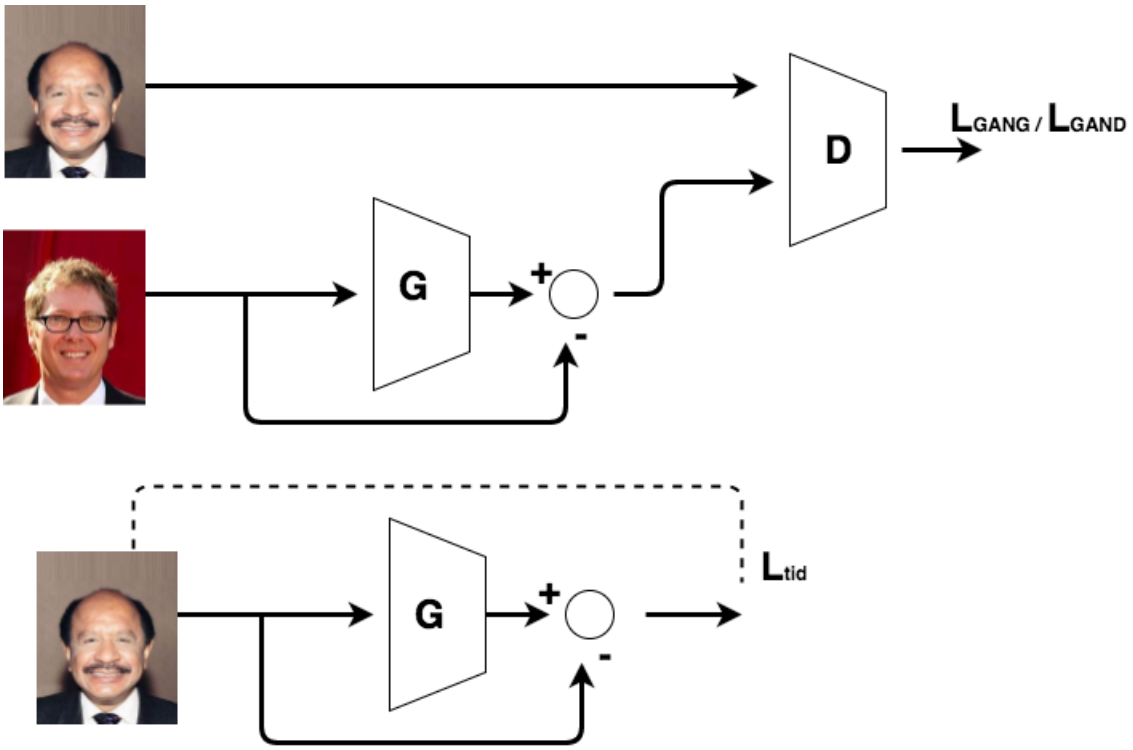


Figure 13: The new architecture diagram.

6 Conclusions and Future Work

During the work we observed some key-points. First, while in the original article it was enough to use DeepFace as the encoder, this was not enough for our purpose, because there is a difference between our domain translation and the domain translation in the original article. While in the original article only the facial features were needed for the translation and the image after the translation is much different from the original image, in our case, we need all the image features for the image translation and the image after the translation should be similar to the original image, as we only add a mustache which is a small feature compared to the whole image. Also, as we mentioned earlier, adding residual blocks improve the autoencoder sharpness significantly.

Secondly, the added Gaussian noise and probabilistic horizontal-flipping were crucial in our attempt to avoid over-fitting.

Figure 14: Using the new architecture we got sharp images, but also red marks.

Thirdly, the performance of the discriminator improved drastically when we increased the image-size from 64×64 to 128×128 since in 64×64 the mustache is only a few pixels wide, which make the task of recognizing it harder.

We've also discovered that contrary to expectations, we got the best results in way I of training, i.e. train first the discriminator until its loss is low, only then the generator until loss threshold is achieved and so on.

In addition, we've got better results using a simpler architecture than using the original architecture, which implies that the original architecture doesn't fit for this task, mostly because the domain translation is significantly different from the original paper domain translation.

An interesting direction for improving the results would be to take a better autoencoder. As our autoencoder outputs blurry images, the task of identifying an output of the autoencoder and a real image is easy for the discriminator, but with better autoencoder the results may improve. Another improvement can be achieved by increasing the dataset size, as discriminator can easily learn $7,000$ images. We've also didn't try to align the images, what may improve the results further.

References

[JAL16] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[TPW16] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *CoRR*, abs/1611.02200, 2016.