Facial Attribute Inverter

Large Scale Machine Learning and Deep Learning (ID2223)

Emil Juzovitski, David Diamant
Scalable Groupwork
emiljuz@kth.se | ddiamant@kth.se

January 25, 2019

1 Summary and Methods

We have have created a GAN in which facial attributes e.g. *sunglasses, hats* get inverted i.e. a person without a hat gets a hat. The GAN uses supervised learning and requires an input of data with and data without the attribute. The GAN is non-standard in its properties. Instead of having one generator we have two. One that creates a positive attribute out of a negative, and one that creates a negative picture out of a positive.

Generally speaking a GAN network consists of a generator and a discriminator. The two play a zero-sum game, in which the generator tries to create samples and the discriminator decides whether or not the sample is real. This way they can train each other, the generator becomes better at generating pictures and the discriminator gets better at distinguishing between real and generated samples.

Our network consists instead of two generators and one discriminator. All are Convolutional Neural Networks(CNN). The main idea is to only subtract/add a small part of the face; the areas connected to the attribute we wish to add. This addition or subtraction of the image is our residual image. This residual image is then added to the original picture to create a inverse attributed image. In addition to detect negative images from positive our discriminator also tries to detect if a image is generated.

The discriminators and generators are CNNs. For further details on the structure, see Table 1 respectively Table 2

Table 1: Network structure of the generator

Generator networks
Input $128 \times 128 \times 3$
5×5 conv. 64 leaky RELU. stride 1. instancenorm
4×4 conv. 128 leaky RELU. stride 2. instancenorm
4×4 conv. 256 leaky RELU. stride 2. instancenorm
3×3 deconv. 128 leaky RELU. stride 1. instancenorm
3×3 deconv. 64 leaky RELU. stride 1. instancenorm
4×4 conv. 3 leaky RELU. stride 1

Table 2: Network structure of the discriminator

Discriminator network
Input $128 \times 128 \times 3$
4×4 conv. 64 leaky RELU. stride 2. instancenorm
4×4 conv. 128 leaky RELU. stride 2. instancenorm
4×4 conv. 256 leaky RELU. stride 2. instancenorm
4×4 conv. 512 leaky RELU. stride 2. instancenorm
4×4 conv. 1024 leaky RELU. stride 2. instancenorm
4×4 conv. 1 leaky RELU. stride 1.
Flatten
Softmax Dense 3 nodes

Dual learning is a important property of the network. Essentially, g1 checks if the generated positive image from g0 looks like a positive image, and converts it back to a negative image, g0 checks if the generated image is consistent with the original picture. An implementation of dual learning required a lot o "wiring" between the different modules of the network, it required us to really understand variable scopes and reusing variables in tensorflow.

We use additional losses to calculate the different losses for the modules of the network. The loss for the generators consists of the GAN loss(Cross entropy), the dual loss(Cross entropy), the pix loss(l1 norm) and the perceptual loss(abs. difference). The discriminator loss(cross entropy). We will not go into detail on the losses.

We use bare Tensorflow as opposed to Keras as we initially planned(stated in the project plan). Due to the complicated wiring of the network where multiple "forward feeds" trough the network contribute to the gradient computation for a single part of the network.

To train the network we used a Tesla K80 GPU hosted on `https://www.floydhub.com`, where we trained the network on 1000 images for 100 epochs.

Finally we built a web-page in React and hosted it on `https://id2223glasses.herokuapp.com/`. That page uses the webcamera on the users device in order to capture images. The two generators are exported from the python code, and the webpage can then use the exported graph and data for inference using Tensorflow.js(`https://js.tensorflow.org/`).

2 Dataset

Images were taken from http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html. Each Image have attributes, e.g. it has a moustache, hat, smile. To use the dataset we divided the pictures into two parts with and without the attribute we are seeking to invert. At execution time we resize the pictures from the original size to $128x128x3$ pixels.

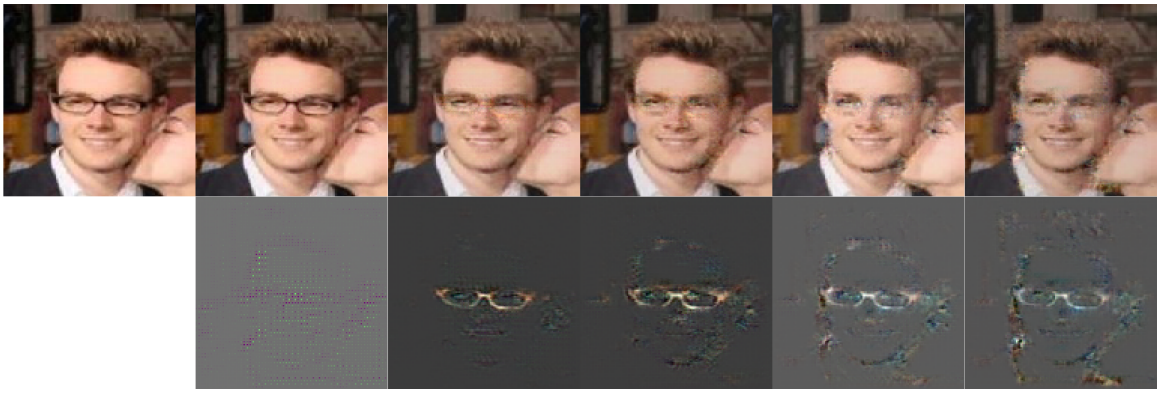
We have generated two datasets. One for glasess and one for hats. These datasets can be found at `https://www.floydhub.com/juzov/datasets/glasses` respectively,
`https://www.floydhub.com/ddiamant/datasets/hatnothat`

3 Results

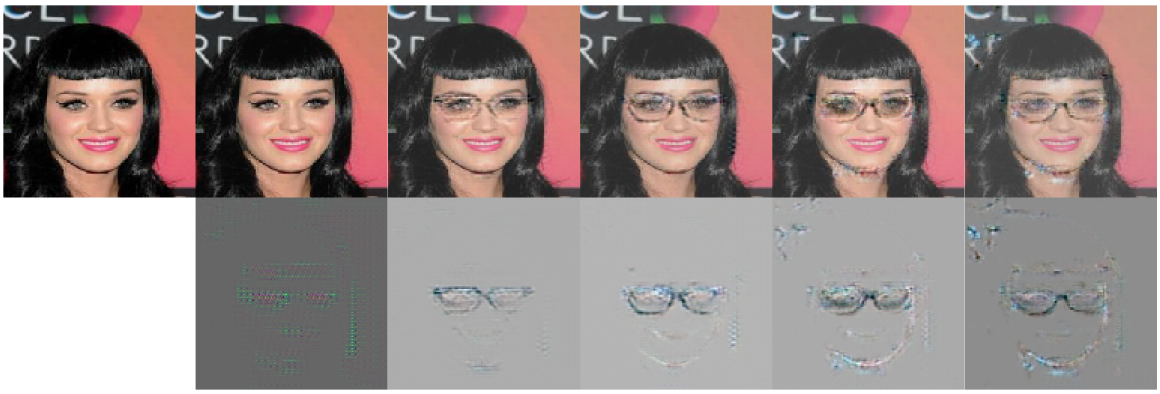
Due to the nature of our project it is hard to say how well the model performs. When working with GANs two usual metrics are *Inception Score* and *Fréchet Inception Distance*. However, for our limited study we have not used any metrics, rather we will simply look at the pictures and evaluate what we see. The web-app also helps to understand how well the model generalizes.

Looking at the results we can say that the model is somewhat successful on the celeb A test images, see Figure 1. The models generalizing capabilities leaves us with some wishes. For example, while using the web-app, the model struggles on adding glasses to our own faces. A better result is obtained by removing glasses from our own faces. This is highly dependant on the lighting in the room, and the quality of the camera used. This can be seen in Figure 2.

We see that the residual image keeps brighting the picture, additionally we see that the same residual does include areas which should not be included in the feature. The latter problem can be eliminated through spatial attention(link). The former requires a fix on how we manage the residual.



(a) Resulting Removed attribute image (above) and Residual image (below) (Orig pic, 20-, 40-, 60-, 80-, 100- epochs)



(b) Resulting added attribute image (above) and Residual image (below) (Orig pic, 20-, 40-, 60-, 80-, 100- epochs)

Figure 1: Result on a sampled images from testing set

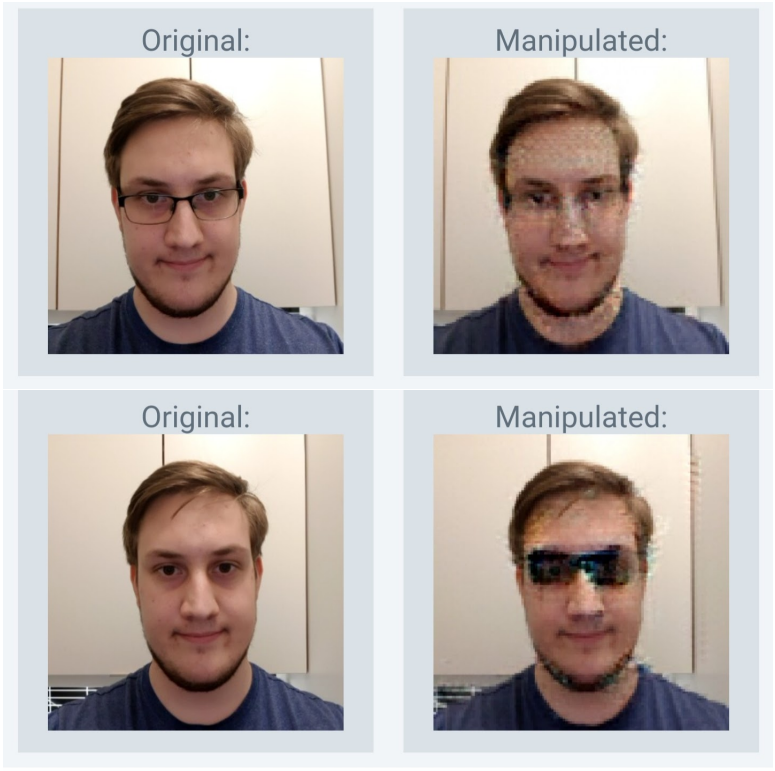


Figure 2: Example of using the model on pictures outside of Celeb-A

4 How To Run The Code

4.1 Python dependencies

We use Python 3.6 along with Tensorflow no other dependencies are necessary.

4.2 Pre-execututions

Download the Classification dataset from floydhub. `https://www.floydhub.com/juzov/datasets/glasses`
extract it to the ML directory.

4.3 Executution

Simply run the following line in the ML-folder:

```
python3.6 model.py
```

4.4 Webapp

If you want to run the web-app, cd into /webb:

```
npm install
npm run-script build
npm start
```

Alternatively, just use the hosted website. `https://id2223glasses.herokuapp.com/`