FACE-GENERATING GAN

Deep Learning project report L. Kärkkäinen 61352B

1 INTRODUCTION

Images of faces make a good topic for machine learning because the kind of biological shapes and variations that occur are easily picked up by gradient descent (more so than geometrical symmetries in machine parts, for instance), and because human brains' fusiform face area makes judging the results very *intuitive*. At the same time, our detailed perception of facial features, perfected by eons of biological evolution, are not so trivial to match or fool.

There are plenty of applications not only for facial recognition, a field heavily analysed and used in the real world, but also for generation of face images that in principle look real, although they aren't real persons, or may have a real person's attributes installed with some change such as different age. Beyond maleficent purposes (AI-created social media trolls) such generated faces can be used in entertainment as done in the FaceApp mobile app. Moreover, the same methods may be applied to datasets other than faces, to provide artistic tools of style transfer.

Generation of bogus data that resembles something real is a problem tackled by traditional signal processing and probabilistic models, recurrent networks, variational auto-encoders, and very recently by generational adversarial networks. After GANs' invention in 2014 they have already proven to produce the highest quality samples in various fields. In face generation, Nvidia's StyleGAN is a notable result.

The basic idea behind GAN is to train a discriminator network that can determine if an input (in our case, a photograph) is real or generated. At the same time, a generator network is trained to output images that best fool the discriminator.

In this work I construct a model in PyTorch and optimize it to generate face images. Various approaches are tried, as described in section 2, until satisfactory results are obtained within reasonable training duration. The training is based on the UTKFace dataset of 23 708 preprocessed color photos in 200x200 resolution. A large variety of ages and ethnicities are included.

Although a lot has been published on the face-generating GANs already, training adversarial networks remains difficult. A particularly big problem is unimodality, leading to the generator always producing the one image that is most favoured by the discriminator in that moment. I present a novel training method that avoids the loss of modality: the generator is trained, via backpropagation, to maximize diversity in its data.

2 METHODS

2.1 PRELIMINARY TRIALS

At first, I implemented DC-GAN that, with minor tweaks generated somewhat good looking faces, although it was clearly overfitting the dataset and its capacity of generating realistic-looking blends between faces was existent but limited.

StyleGAN was the primary responsibility of my project partner who was too busy with other challenges and dropped out without contributing anything. However, it turned out that Nvidia's computational resources surpass those of Paniikki, and training such a network would have taken days, if not weeks, so that approach was abandoned.

2.2 NETWORK STRUCTURE

The network presented in this work is a custom design that falls in between DC-GAN and StyleGAN, although no style transfer is implemented. The structure was refined by trial, error and intuition, repeatedly observing how the results change.

The latent vectors are transformed into two formats, one "image" fed to the first layer of the generator (base_size * base_size pixels) for spatial attributes, and an "injected" layer that holds the same latent values for each pixel. The first layer is important in creating spatial structure, or otherwise the network would struggle in creating anything but flat-colored tiles. The injected layers provide a shortcut for latent attributes that might be useful in every layer, and additionally x and y coordinates are included in it for further spatial structure. Both layers contain generator_channels channels.

An UpConv layer first bilinearly resamples the current image to twice the resolution and then performs two convolutions that do not change the size. The first convolution takes twice the number of input channels in order to mix the previous layer's output with the injected channels.

The discriminator is a simple sequential convolutional network.

2.3 RGB CONVERSION

Generator and discriminator both use a large number of channels that supposedly represent different features at a particular spatial location. Separate conversion layers are used for converting generator's channels to RGB colors, and to convert image's RGB into discriminator's channels. These are the only layers in the network that are biased. The conversion is spatially limited to one pixel, to constrain the biasing effect to local changes.

2.4 BASIC GAN TRAINING

In a training round, the generator is fed a set of random latent vectors to produce a set of fake images. The fakes are fed into the discriminator, labelled *real* to obtain by backpropagation the gradients to the fake image and then further all the way to the generator's weights to obtain the updates that would best fool the discriminator. Then the same fake images are labelled *fake* and fed to the discriminator along with a randomly selected batch of *real* images, so that backpropagation can be used to obtain updates for discriminator weights. The fake image gradients are *detached* prior to using them in discriminator training to avoid backpropagation to the generator.

2.5 PROGRESSIVE TRAINING

ProgressiveGAN method is used to speed up and in a manner stabilize training. The training starts with only the two smallest layers in both networks, and gradually moves to bigger resolutions by alpha-blending during the transition. To ease implementation, all layers have the same number of channels, so that the same input/output RGB conversions may be used on any layer.

2.6 NOISE-CONTROLLED DISCRIMINATOR TRAINING

At times the discriminator tends to learn too quickly. Noise is added to both generated and real images used in discriminator training. However, for generator training, no noise is added, so that cleaner gradients may be obtained and the generator effectively trains faster. The noise amplitude ε is constantly adjusted according to the difference of %-rating given to real and fake images. Most of the time the noise stays at zero or at very low values, and as such should not affect e.g. the sharpness. This proved to be more stable than the other alternatives tried, including adaptive discriminator learning rates.

2.7 LATENT SPACE

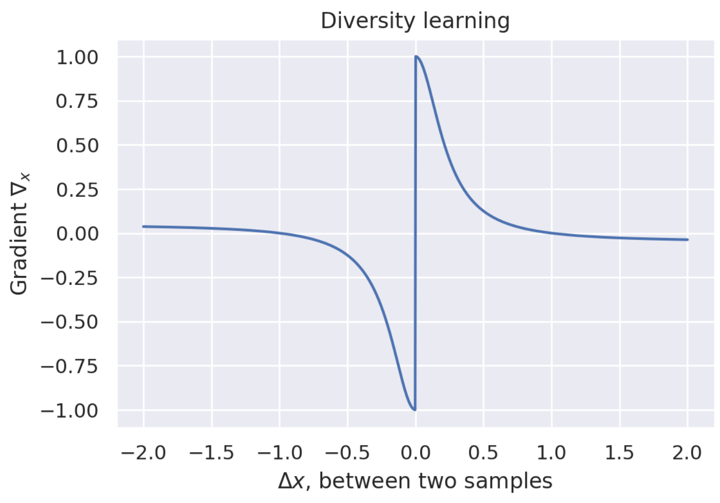
The latent vectors **z** (dim=100) are random numbers normalized to a spherical distribution rather than simple multivariate normal distribution. Since the generator network consists mostly of unbiased layers, zero input would translate to zero output, and thus a hollow sphere is a better distribution.

2.8 DIVERSITY LEARNING

Optimization by backpropagation moves the parameter values in a direction where the output loss is minimized, and in case of the generator, that means the direction that would score higher in discriminator. Full mode collapse occurs when parameters end up in a configuration that causes always the same output being generated, and any further training takes those parameters even closer that optimum. The discriminator promptly learns that image as being generated (even if an identical image exists in the dataset but appears only rarely), so the optimum shifts to something else but the mode tends to stay collapsed.

Diversity learning is a novel feature that uses backpropagation to learn a better distribution for the generator's internal state. Each sample in training uses an independent sampling of the latent vector, and should produce different values on generator's data tensors, and subsequently different generated images.

In addition to normal training, a diversity gradient is calculated on the data tensor of the final layer, from where it will backpropagate to prior layers and network weights. The gradient favours nearby values moving apart until the difference is 1. This range was chosen specifically for tanh-valued data tensors, where the difference may be at most two, and unit difference is considered optimal. The gradient is calculated among combinations of two samples within a minibatch, as follows:



$$\nabla_x = \operatorname{sign}(\Delta x) * 0.05 * \left(\frac{1.05}{(\Delta x)^2 + 0.05} - 1\right)$$

3 RESULTS

Training videos with each image in the frame corresponding, using a fixed batch of 32 latent vectors not used in training so that the faces seen stay more or less the same across rounds.

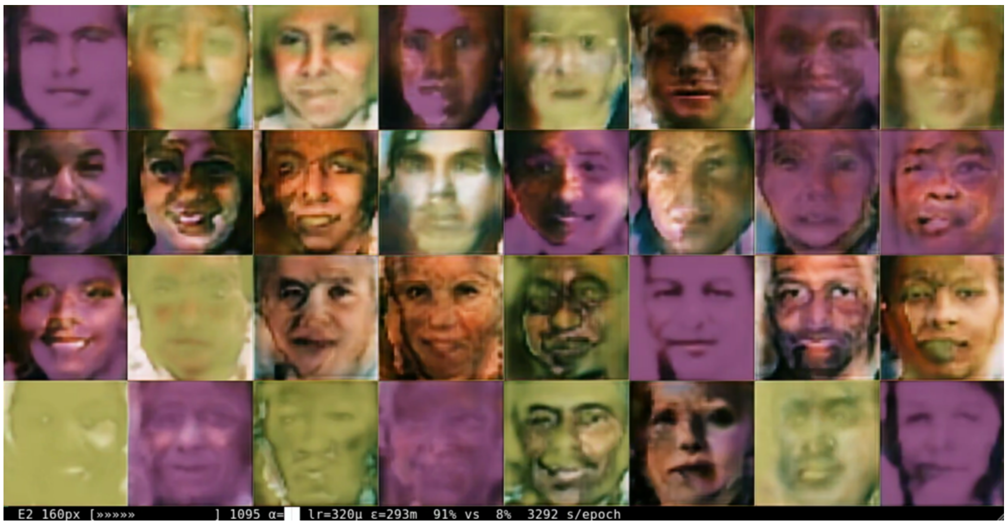
The first prototype, a slightly modified DC-GAN: https://www.youtube.com/watch?v=4kY8UlzZEkM

A trial run with an early version of the current architecture "rev2": https://www.youtube.com/watch?v=3dfXcv_O7Jo

Long training run of a later version "rev3": https://www.youtube.com/watch?v=uGlphey7TFw

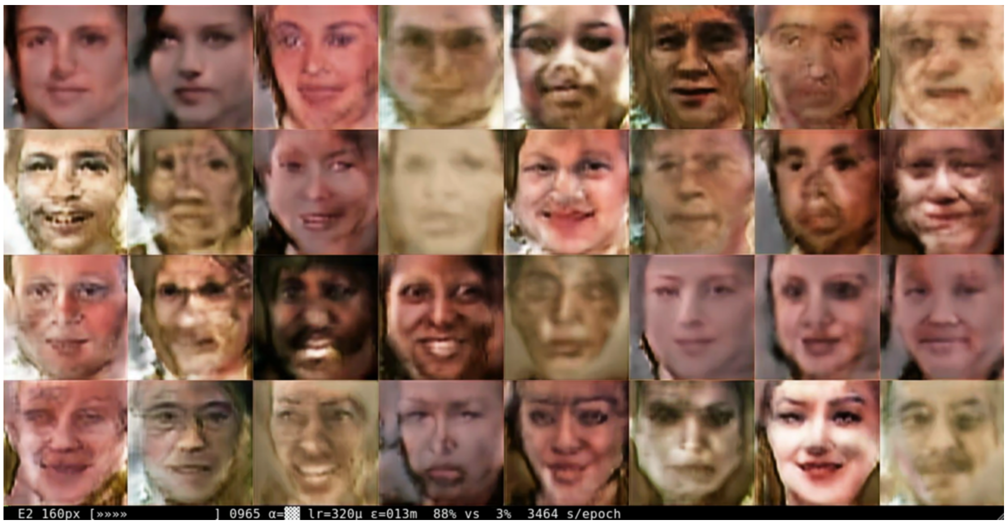
Final revision (git master): https://www.youtube.com/watch?v=E3xHI4bF234

Partial bimodality seen in training:



```
 E2 160px [»»»»»            ] 1095 α=█  lr=320µ ε=293m  91% vs  8%  3292 s/epoch
```

The final revision training with diversity learning:



```
 E2 160px [»»»»             ] 0965 α=▒  lr=320µ ε=013m  88% vs  3%  3464 s/epoch
```

4 DISCUSSION

The model learned to produce acceptable face images after just two hours of training on a Quadro P5000 (a GPU launched in 2016), suggesting that the used network structure performs well, although no heavy claims on that should be done without rigorous analysis, and in particular, verifying that the generated images do not correspond to those in the dataset.

Smooth morphing displayed in chin shapes and other features during training suggest that the GAN is performing properly and not simply learning bitmaps from the dataset, in which case blending or selection between images would occur instead. A lot of colour oscillation occurs even with decreased learning rates on later epochs, and this should be reduced by further tuning (starting with investigation on where it originates).

Overall, this has been an useful exercise into the world of generative networks, and it has become quite clear that training them successfully is a delicate issue. Simply changing the number of channels could completely break anything, so a lot of time was spent on re-tuning the parameters whenever necessary.

The biggest contribution of this work might be diversity learning which helps to reduce mode collapse in GAN training. Certainly similar methods must have been attempted before but a brief literature review revealed none. Multiple approaches were tried until ending up with the equation presented in section 2.8 due to its sound basis and good practical performance. More research on this is certainly required, in particular with relation to proper scaling when used in all and not just the final layer, as well as quantitative performance comparisons against training with no diversity learning.

5 USE THE SOURCE

Publicly available via https://version.aalto.fi/gitlab/ljkarkk2/deeplearn

Download and extract to UTKFace/ the 107 MB dataset from https://susanqq.github.io/UTKFace/, then run python gantrain.py. FFMPEG is used to produce training.mkv. Networks are saved once per epoch.

- gantrain.py: main program, training loop
- networks.py: module definitions
- latent.py: latent vector creation
- facedata.py: loading of dataset
- visualization.py: video output

6 REFERENCES

Progressive Growing of GANs for Improved Quality, Stability, and Variation https://arxiv.org/abs/1710.10196

A Style-Based Generator Architecture for Generative Adversarial Networks https://arxiv.org/abs/1812.04948

GAN — DCGAN (Deep convolutional generative adversarial networks) https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f