Generating Fashion and Flowers using Generative Adversarial Networks

Abstract

In recent years, results from generative models have substantially improved due to the introduction of Generative Adversarial Networks (GANs). GANs train two networks simultaneously: a generative network that captures the data distribution, and a discriminative network that estimates whether a sample came from the generator or from the training data. This paper applies two different GANs to two different data sets: the classic fashion-MNIST data set, containing grayscale images of fashion items, and a data set containing full-colour images of flowers. We explain the importance and power of GANs, their limitations, and elaborate on the optimal structure for the models.

1 Introduction

"What I cannot create, I do not understand" - Richard Feynman. A quote strikingly applicable to the topic of this paper: Generative Adversarial Networks. Although generative neural networks have been around for years, until recently their output could generally easily be distinguished from real data. However, in 2014 a small revolution took place with the invention of Generative Adversarial Networks (GANs). GANs can generate images, 3D-models, videos, and a lot more that are indistinguishable from real data. Given a training data set of - for example - pictures of human faces, GANs are able to generate new pictures of humans that look completely realistic, but do not actually exist. Not only that, they are also able to recognize hierarchical structures in images which enables them to e.g. perform 'arithmatic' on faces (Radford, Metz, and Chintala, 2015), an illustration of which can be found in Fig 1. Hence, it appears GANs are capable of both creating and 'understanding'.

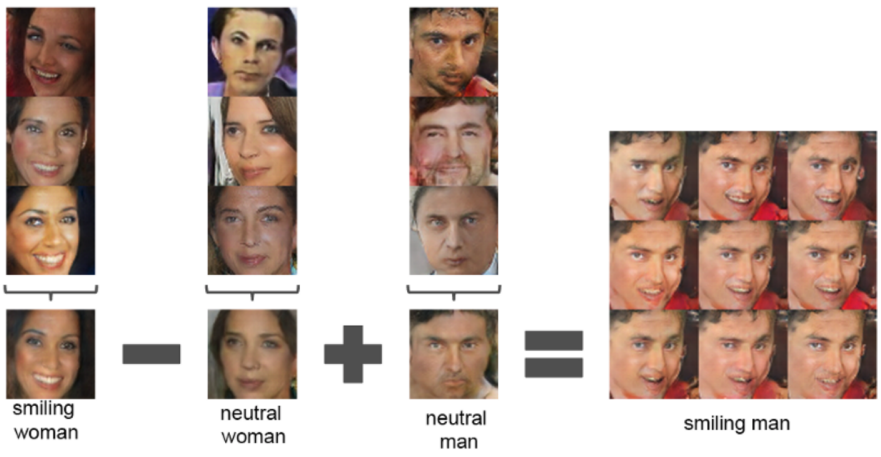


Figure 1: An illustration of the power of GANs: 'arithmatic' on generated images of human faces

Generative adversarial networks have been called "the most interesting idea in the last 10 years in machine learning" by Yann LeCun, director of AI research at Facebook.[1] Quite a statement, and a clear motivation to gain a deeper understanding of GANs. In this paper two GANs will be discussed: one will be trained on the fashion-MNIST data set, a famous data set containing greyscale images of various fashion items, and the other will be trained on a data set of full-colour flower images.

The paper is divided into the following sections: first, a brief introduction to GANs, their structure, power and (dis)advantages will be given. Next, the two GAN used in this paper will be introduced in detail, including their structure and the results. The paper concludes with a brief discussion of the results.

2 Generative Adversarial Networks

GANs are a relatively new type of neural networks, introduced in 2014 in a seminal paper by Goodfellow et al., 2014. The name itself already largely gives away what makes GANs special: they are both *generative*, meaning that they are able to produce new content, and *adversarial*, which means a GAN consists of two networks competing against each other. Their generative ability makes that GANs are mostly used to generate stimuli such as pictures, music, prose or speech - in fact, GANs can learn to mimic any distribution of data. And they are quite successful in doing so: a painting generated by a GAN was recently sold for $ 432,000 at Christie's.[2]

Generative models existed before Goodfellow's paper, but they were often not that great: images of generated faces tended to have missing ears, for example. Goodfellow's major realization was to pit two neural networks against each other (which is where the *adversarial* part comes from): one network produces candidates, while the second network evaluates these candidates and decides if they are real or fake. Rumour has it that Goodfellow came up with this rather revolutionary idea over a glass of beer with friends, and had a first GAN working the next morning.[3] At the moment, GAN applications are rapidly increasing, ranging from improving astronomical images (Schawinsky et al., 2017) to prediction how one's face will change with age (Antipov, Baccouche, and Dugelay, 2017).

The term 'generative model' is often used in many different ways. To avoid any confusion, in this paper the term refers to any model that takes a training set consisting of samples from some distribution p_{data} and learns to represent an estimate of the distribution, called p_{model}. Some generative models are able to estimate p_{model} explicitly, others are only able to generate samples from it. Generally, GANs focus primarily on the latter, though there have been GANs designed that can do both (Goodfellow, 2017).

Of course, one might wonder why GANs (or generative models in general) are interesting at all. After all, most GANs are only capable of producing more images, and there is generally not exactly a shortage of images in this world to start with. Still, there are several reasons why GANs are interesting, as Goodfellow also pointed out during his presentation at the NIPS 2016 conference.[4] Firstly, GANs can be trained with missing data and hence can be trained to provide predictions on missing inputs. These GAN-generated predictions are generally better than other methods that can fill in missing data. This is particularly interesting in the case of semi-supervised learning, where many of the labels of the training data are missing. Modern deep learning requires an enormous number of labeled training examples, and semi-supervised learning is a strategy to reduce this number. Furthermore, traning GANs is a great test to see how well we can represent and manipulate high-dimensional probability distributions. Such distributions play an important role in many fields of engineering and applied math. Finally, GANs can work with multi-modal outputs. Traditional machine learning models (i.e.

[1] https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning

[2] https://www.theverge.com/2018/10/23/18013190/ai-art-portrait-auction-christies-belamy-obvious-robbie-barrat-gans

[3] https://www.technologyreview.com/s/610253/the-ganfather-the-man-whos-given-machines-the-gift-of-imagination/

[4] http://www.iangoodfellow.com/slides/2016-12-04-NIPS.pdf

trained by minimizing the MSE) are not suited to deal with multiple correct outputs for a single input. GANs can.

Now that we have a better idea of what GANs are and why they are interesting, let us dive deeper into their structure. As mentioned earlier, GANs consists of two networks: a generator and a discriminator. The job of the generator is to produce stimuli (such as images) while the job of discriminator is to evaluate the generated candidates and predict if they are real or fake. So, the aim of the generator is to fool the discriminator (produce output so realistic that the discriminator is unable to tell it apart from real data) and the aim of the discriminator is to tell the truth. An analogy that is often made is that of a counterfeiter and a police officer: the counterfeiter (the generator) is learning to create fake money, and the police officer (the discriminator) is learning to detect fake money. Both of them are learning from the other and improving as a result. The counterfeiter keeps on creating better and better fake money, and the police officer gets better and better at detecting it. In an actual GAN, equilibrium is reached when the discriminator is no longer able to tell fake from real. A schematic display of a GAN can be found in Fig. 2.[5]

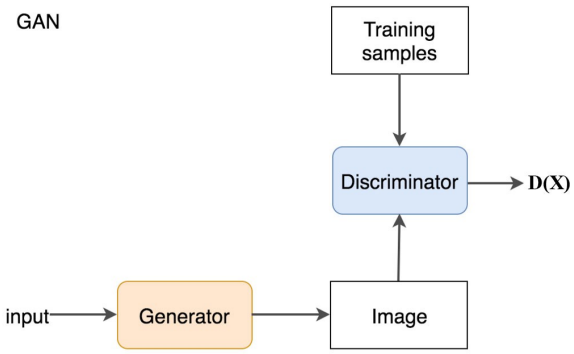


Figure 2: A simple GAN

In the simplest GAN structure, the generator starts with random data and learns to transform this noise into information that matches the distribution of the real data. Interestingly enough, the generator never sees the genuine data: instead, it learns to create realistic information by receiving feedback from the discriminator. Training the discriminator is therefore conceptually not too difficult: the discriminator takes an image from either the training set or the generator as input (without knowing the source). It outputs a number between 0 and 1: the closer to 1, the more sure the discriminator is that the image is real. The discriminator learns using traditional supervised learning techniques. Training the generator is the more interesting part: the aim of the generator is to make the discriminator output a probability as close as possible to 1 when given its generated image. To be able to do this, the generator needs feedback from the discriminator to know how to make its generated image more realistic. The discriminator tells the generator how much to tweak each pixel to make the generated image more realistic. This is done by backpropagation: the gradients of the discriminator's output are backpropagated with respect to the generated image and all the weights. However, to prevent being 'outsmarted' by the generator really quickly, the discriminator also adjusts its own weights. This is how both generator and discriminator learn.

Although the two networks are adversaries, it is very important that neither of the two networks overpowers the other: if one of the two starts to dominate, learning stops for both. If the discriminator is too good, it will return values so close to 0 or 1 that the generator will struggle to read the gradient.

[5]Image optained from https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

If the generator is much better than the discriminator, it will learn to exploit weaknesses in the discriminator that lead to false negatives. Obviously, this should be prevented: the two networks should have a similar skill level, which can be achieved by carefully tuning the nets' respective learning rates.

Not only are GANs difficult to tune, they are also computationally very intensive. Training a single simple GAN could take several hours.[6]

Besides these computational challenges, GANs have an inherent more philosophical darker side as well. They are the perfect weapon to create fake news. Currently, GANs are already being used to create fake videos of e.g. politicians, having them say things they never actually said (a phenomenon known as 'deepfake' videos).[7] Although GANs did not create this problem, they are the perfect tool to make it worse, as distinguishing these deepfakes from real videos becomes harder and harder. Goodfellow, the man who came up with the idea behind GANs, is well-aware of this threat - perhaps ironically, up to April 2019 he was leading a team at Google Brain focusing on making machine learning and AI more secure.[8] This darker side of GANs should be kept in mind. Although stopping technological process is neither desirable nor achievable, a case could be made that more effort should be put into informing the general public about the possibilities and problems of GANs and comparable networks, as it can reasonably be expected that their use in society will become increasingly prevalent.

All in all, this brief introduction to GANs has hopefully illustrated what makes these networks interesting and worth exploring more in depth. This will be done in the following sections, where we will train our own GANs on two data sets: the fashion-MNIST data set to generate new images of clothing, and a data set containing images of flowers to generate more images of flowers.

3 Generating fashion

In this section we will use a GAN to generate images of fashion items. Data set, model and results will be discussed, with an emphasis on the model structure.

3.1 The data

The network is trained on the fashion-MNIST data set, one of the classic data sets in machine learning, created by Zalando Research.[9] It consist of 70,000 images, 60k test and 10k train, categorized into 10 fashion categories. Label and corresponding description can be found in table 1. Each item is represented by a $28 * 28$ greyscale image.

This data set was chosen mainly for simplicity reasons: it is an uncomplicated and straightforward data set, included in Keras, and the items it contains can be represented by a shape of size $28 * 28 * 1$ since the images are greyscale (so no additional dimensions are needed to deal with colours).

[6] https://skymind.ai/wiki/generative-adversarial-network-gan
[7] https://www.csoonline.com/article/3293002/deepfake-videos-how-and-why-they-work.html
[8] https://hub.packtpub.com/ian-goodfellow-quits-google-and-joins-apple-as-a-director-of-machine-learning/
[9] https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/

Table 1: Labels and corresponding description in Zalando Fashion-MNIST data set

Label	Description
0	T-shirt
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

3.2 The model

Our model is inspired by the paper by Radford, Metz, and Chintala, 2015 on deep convolutional GANs (DCGANs) and a tutorial on deep convolutional GANs from Google Seedbank.[10] Most GANs today are DCGANs. According to Goodfellow, 2017 some of the key characteristics of an optimal DCGAN structure are as follows:

- It is important to use batch normalization in between most layers of the discriminator and the generator. However, the last layer of the generator and the first layer of the discriminator should not be normalized, because the model needs to learn the correct mean and scale of the model distribution.
- Generally, no pooling or unpooling layers are used. When the generator has to increase the dimensions of the output transposed convolution with a stride greater than 1 is used.
- An Adam-optimizer is used for both generator and discriminator.

These settings have been found to work best for DCGANs. Since we did not want to reinvent the wheel, our model will also stick to them.

3.2.1 Setup

We run our model in Google Colaboratory, a free Jupyter Notebook environment where it is possible to run any model in the cloud using one of Google's GPUs. We use the popular machine learning library TensorFlow. In particular, we use the `tf.keras` API, which is TensorFlow's implementation of Keras. We use `tf.keras` 2.2.4.

3.2.2 Processing the data

First we normalize the input images so that each pixel has a value between -1 and 1. Although normalization is not a strict necessity, it often facilitates training and increases stability of the model.[11] Next we batch and shuffle the data. We set the buffer size to 60,000 (the total size of the training set) and the batch size to 256 (so each batch contains 256 input images, meaning $60,000/265 = 234$ batches will be passed to the model during every epoch.

[10] https://research.google.com/seedbank/seed/dcgan_with_tfkeras_and_eager

[11] https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

3.2.3 Building the model

Next we build both generator and discriminator. Both are defined using the Keras Sequential API, which builds models by stacking different layers.

The model for the generator consists of eleven layers, mostly inspired by the beforementioned tutorial and paper:

1. A standard densely-connected layer.
2. A batch normalization layer, normalizing the activations of the previous layer at each batch. This is standard procedure for all DCGANs, as already mentioned at the beginning of this section.
3. A leaky ReLU activation function. Leaky ReLU is the similar to a regular ReLU function, but allows for a small gradient when the unit is not active (hence the 'leaky' adjective). This prevents the 'dying ReLU' problem, where a neuron is stuck on the negative side and hence always outputs zero.[12]
4. A reshaping layer, which reshapes the output of the previous layer to $7 * 7 * 256$.
5. A transposed convolutional layer (also known as deconvolution) with kernel size $5 * 5$ and stride $1 * 1$ used to decrease the dimensions of the output to $7 * 7 * 128$. The padding option is set such that the convolution algorithm tries to pad evenly left and right, but if the amount of columns to be added is odd, it will add the extra column to the right (which will also be the default for every following (de)convolution layer).
6. Another batch normalization layer.
7. Another leaky ReLU activation layer.
8. Another transposed convolutional layer, this time with with kernel size $5 * 5$ and stride $2 * 2$ which changes the dimensions to $14 * 14 * 64$.
9. Another batch normalization layer.
10. Another leaky ReLU activation layer.
11. The final layer, a transposed convolutional layer with kernel size $5 * 5$ and stride $2 * 2$, using an inverse tangent activation function. This results in the desired $28 * 28(*1)$ output. This layer is not followed by a batch normalization, since the discriminator should learn the correct (unnormalized) mean and spread of the distribution.

The discriminator is a CNN consisting of seven layers:

1. A convolutional layer with kernel size $5 * 5$ and stride $2 * 2$. This layer takes as input a $28 * 28$ image (either the output from the generator or an image from the training set) and returns a $14 * 14 * 64$ shape.
2. A leaky ReLU activation function.
3. A dropout layer, which sets a random fraction of input units to zero, thereby preventing overfitting. We used a dropout rate of 0.3.
4. Another convolutional layer with kernel size $5 * 5$ and stride $2 * 2$ resulting in an output shape of $7 * 7 * 128$.
5. Another leaky ReLU activation layer.

[12] https://www.tinymind.com/learn/terms/relu

6. Another dropout layer, again with a dropout rate of 0.3.

7. A flattening layer, which flattens the input to an array of size $7 * 7 * 128 = 6272$.

8. A final densely-connected layer where it is specified that the output should be of size 1. The discriminator will output a number between 0 and 1, denoting the probability that the input image is real, according to the discriminator.

3.2.4 Training the model

Now that we have defined the two networks, the next step is to train them. However, to be able to do so we first still have to specify an optimizer and a loss function. The discriminator loss is defined by how well the discriminator is able to distinguish real images from fake ones: if 1 denotes a real image and 0 a fake one, the discriminator's predicted probability should be as close as possible to 1 if the image is real and as close as possible to 0 if it is fake. For the generator, this is different: the generator's loss is defined by how well it can trick the discriminator. How well the discriminator is able to classify real images as real is not of importance to the generator. Hence, the generator is doing its job well the closer the prediction of the discriminator is to 1 when classifying an image generated by the generator. The closer it is to 0, the greater the loss for the generator. For both networks we opt for a cross-entropy loss function. Since both networks are trained separately, we also need to specify two optimizers: both networks are trained with an Adam-optimizer, since this is the standard for DCGANs.

The model can now be trained. We train the model for 300 epochs. Each epoch takes approximately 15 seconds.

The training begins by generating a random input which is fed to the generator. The generator will then produce an image, which is fed to the discriminator. The discriminator then classifies this image as real or fake. For both generator and discriminator the loss is calculated, and the corresponding gradients are used to update the weights of both generator and discriminator. This process is repeated for the specified number of epochs.

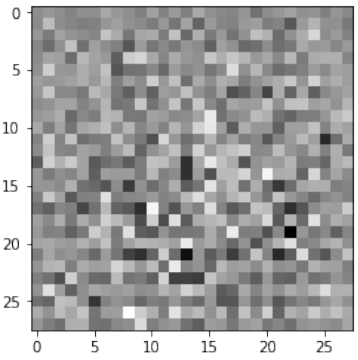


Figure 3: Output of generator before training

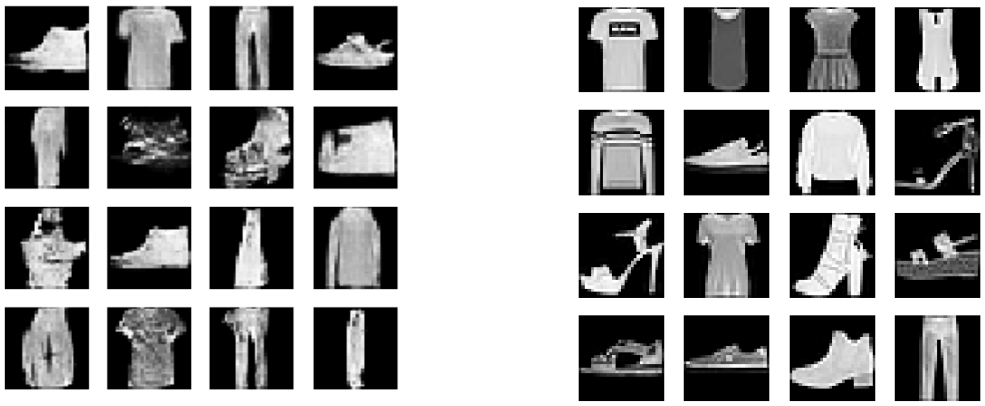
3.3 Results

Fig. 3 shows an image from the generator before being trained: clearly, this is just a collection of random pixels.

Fig. 4a displays a grid of 16 images, all generated by the generator after 300 epochs of training. Just like the training set, all images are in black and white.

As a refererence point, Fig. 4b shows some examples from the input data (the pictures which the generator tries to resemble as closely as possible). Clearly, our current GAN is not yet capable

of producing images that are indistinguishable from the training set: the generated images look like blurred versions of the real images.



(a) Generated images after 300 epochs

(b) Real images

Figure 4: A comparison of the real and (generated) fake images

As can be expected, the GAN makes most progress during the first epochs. This is especially clear when we display the output of 10, 150 and 300 epochs next to each other, as done in Fig. 5.

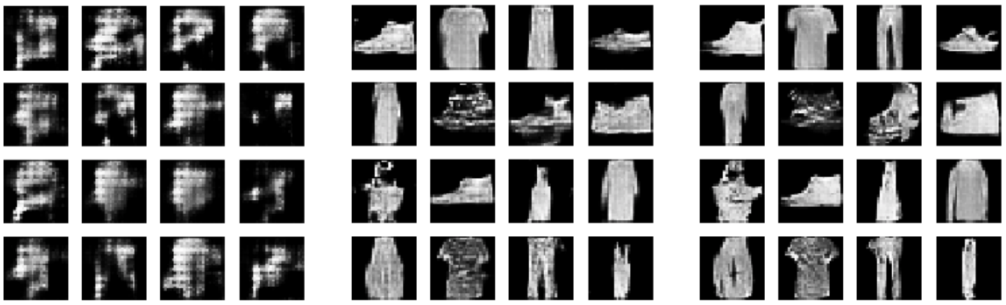


Figure 5: A comparison of the output generated after 10, 150 and 300 epochs

4 Generating flowers

In this section we will create another GAN to generate pictures of coloured flowers. This problem is slightly more complicated because the inputs have one additional dimension; the pictures are now in full-colour instead of greyscale (so the dimension is $28 * 28 * 3$).

4.1 The data

The data set used contains 4,242 images of flowers labeled in 5 different categories: daisy, dandelion, rose, tulip and sunflower. We obtained this data set from Kaggle[13]. It was originally created as a training set for Convolutional Neural Networks to perform flower classification. Looking into the data

[13]https://www.kaggle.com/alxmamaev/flowers-recognition

set, we can see that it is not yet fit for our (generative) purposes. Some pictures share the same label despite being quite different (e.g. with respect to colour) which would confuse our relatively simple GAN. The data therefore needs to be pre-processed, which will be discussed in the next section.

4.2 The model

This model is inspired by a Kaggle kernel 'GAN n' Roses' produced by Peterson Katagiri Zilli [14]. It is largely similar to the model in the previous section, but some adjustments were made to be able to handle the additional dimension in this data set.

The model for the generator consists of the following layers:

1. A standard densely-connected layer of dimension 256.
2. A leaky ReLU activation function.
3. A batch normalization layer.
4. Repeat step 1, 2, and 3 twice: once with a densely-connected layer of dimension 512 and the second time with a densely-connected layer of dimension 1024.
5. Another densely-connected layer of dimension $28 * 28 * 3$.
6. A inverse tangent activation function.
7. A reshaping layer, which reshapes the output of the previous layer to $28 * 28 * 3$.

The model for the discriminator consists of the following layers:

1. A flattening layer, which flattens the input to an array of size $28 * 28 * 3 = 2352$.
2. A standard densely-connected layer of dimension 1024.
3. A leaky ReLU activation function.
4. A batch normalization layer.
5. Repeat step 1, 2 and 3 twice: once with a densely-connected layer of dimension 512 and the second time with a densely-connected layer of dimension 256.
6. A final densely-connected layer where it is specified that the output should be of size 1. The discriminator will output a number between 0 and 1, denoting the probability that the input image is real, according to the discriminator.

4.2.1 Setup

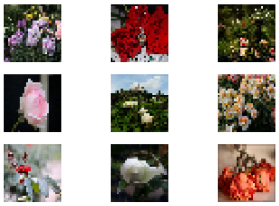
We once again run our code on Google Colab. We use `tf.keras` 2.2.4.

4.2.2 Processing the data

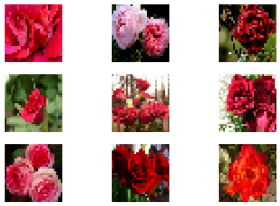
We downloaded the data set (225 MB) from Kaggle to a local drive. As mentioned before, the data set first has to be modified. Our goal is to generate flowers from one specific type (daisies, dandelions, roses, tulips or sunflowers). We decided to manually select the pictures of flowers for each type that generally resemble each other. For example, we chose to keep only the red roses from the subset of roses. Fig. 6a shows a sample from the roses in the data set before this manual selection. Fig. 6b shows a sample from the training set after selection.

This inevitably reduces the number of images our network works with, but it also decreases the complication level of the images and hence helps the generator. We found that working with the original data was not feasible for our relatively simple GAN.

[14] https://www.kaggle.com/pkzilli/gans-n-roses/



(a) Batch of roses before selection



(b) Batch of roses after selection

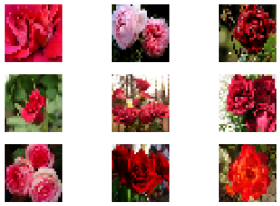
Figure 6: Processing of the data

4.2.3 Training the model

The discriminator receives batches of 32 images consisting of 16 real images and 16 fake images generated by the generator. The output of the discriminator is the probability that the observed image is real. The generator learns to get the discriminator to output values closer to 1, while the discriminator learns to distinguish fake images and real ones. We run the training for 2,000 epochs. The learning rate is set to 0.0002. It is advisable to also add some random noise to each of the inputs to stabilize the training.[15]

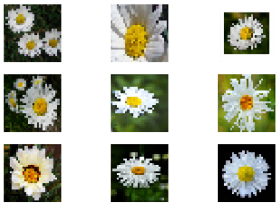
4.3 Results

We chose to work with 3 types of flowers: roses, sunflowers and daisies. Fig 7 shows an extract of each subset and their respective number of items.



(a) Batch of roses (38)

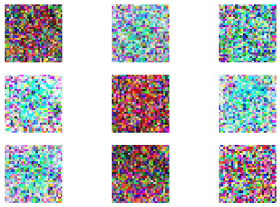
(b) Batch of sunflowers (120)



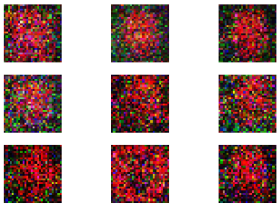
(c) Batch of daisies (53)

Figure 7: Processed subsets of flowers

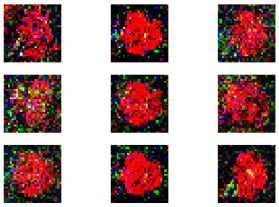
For each subset we tested the generator after 300, 1000 and 1700 epochs and recorded the results:



(a) 300 epochs



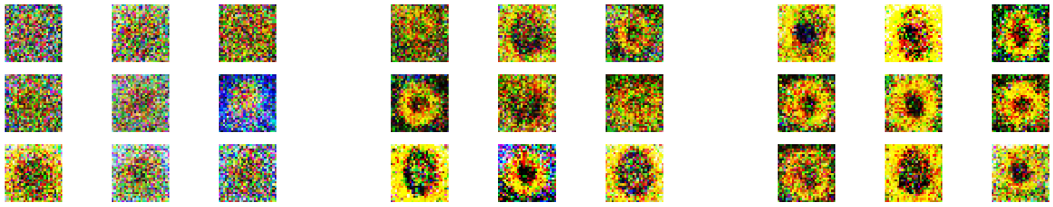
(b) 1000 epochs



(c) 1700 epochs

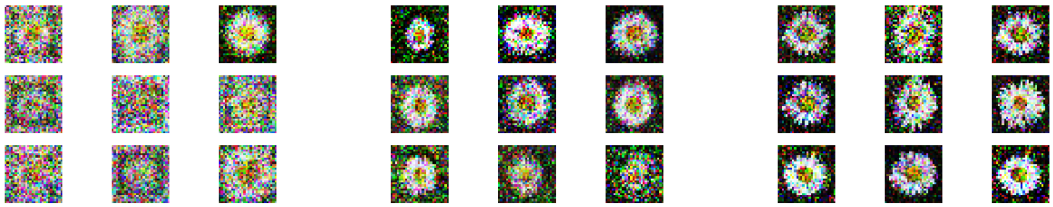
Figure 8: Generated roses

[15] https://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/



(a) 300 epochs (b) 1000 epochs (c) 1700 epochs

Figure 9: Generated sunflowers



(a) 300 epochs (b) 1000 epochs (c) 1700 epochs

Figure 10: Generated daisies

The generated sunflowers look relatively 'noisy'. However, our generated daisies and roses look quite clean. We can almost distinguish each white ray florets of the daisies' and each one of the roses' petals. But still, the model is simple and these generated images can be easily distinguished from real ones. It is worth noting that training the model for more epochs does not improve the quality of the images. It would ultimately lead to the collapse of the model and make the generator create one same single image over and over again.

5 Discussion and conclusion

GANs are a fascinating topic, definitely a worthy topic of a paper such as this one. However, they are not easy to implement. GANs are computationally intensive and anything beyond a very simple vanilla GAN takes quite some effort to program well, and quite a lot of time to train. We therefore decided to start with a relatively simple data set, the fashion-MNIST, so that we could spend most of our energy and time on the model itself. When we extended to the more complicated (because in full-colour) flower data set, we found that our generated pictures are not even close to being indistinguishable from real pictures. Our recommendation would therefore be to work with GANs only if there is enough computing power and enough training data available. Especially the generated flowers would probably look better if more training data would have been available. But even with this limited subset of data the generator is capable of producing output that bears a clear resemblance to the real flowers.

Our results show that it is possible to 'make your own GAN', even though the results may not be perfect (which can be expected, as both the time for this assignment and the computing power available are limited). GANs currently are a state-of-the-art deep learning technique, and building a GAN yourself gives much more insight into their structure and power. This is especially relevant regarding the ability of GANS to generate deep fake videos or other 'fake news' imagery.[16] GANs are here to stay, and our research on them has made us more aware of their possibly disruptive abilities. Regardless of whether you know how to code one or not, society as a whole should be made more

[16] For a striking example, see: https://themitpost.com/deepfakes-and-dead-celebrities-into-the-world-of-gans/

aware of the fact that even videos cannot be 'trusted' anymore, which makes a critical attitude more important than ever. As one recent paper (Chesney and Keats Citron, 2019) puts it: "Harmful lies are nothing new. But the ability to distort reality has taken an exponential leap forward with 'deep fake' technology." We wholeheartedly agree with this. Hence, another obvious suggestion would be to look further into generating videos with GANs. Although the scope of this would be large enough for it to be a topic for a thesis, it would certainly be interesting and relevant to further investigate.

References

Antipov, G., M. Baccouche, and J. Dugelay (2017). *Face Aging with Conditional Generative Adversarial Networks*. arXiv: 1702.01983.

Chesney, R. and D. Keats Citron (2019). *Deep Fakes: A Looming Challenge for Privacy, Democracy, and National Security*. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3213954.

Goodfellow, I. et al. (2014). *Generative Adversarial Networks*. arXiv: 1406.2661.

Goodfellow, I. (2017). *NIPS 2016 Tutorial: Generative Adversarial Networks*. arXiv: 1701.00160.

Radford, M., L. Metz, and S. Chintala (2015). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. arXiv: 1511.06434.

Schawinsky, K. et al. (2017). *Generative Adversarial Networks recover features in astrophysical images of galaxies beyond the deconvolution limit*. arXiv: 1702.00403.