Generating Cats

CAS ADS M3 Project

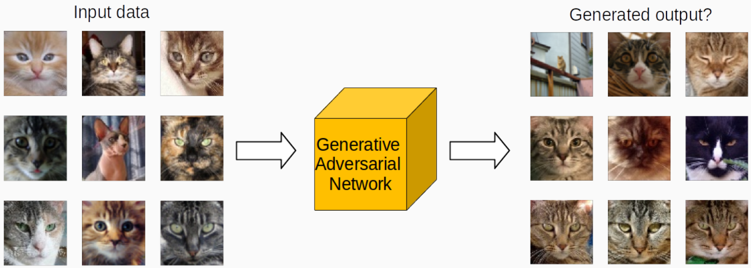
Petra Müller

20. März 2020

Project Goals

Project Goals

Training a Generative Adversarial Network (GAN) to generate realistic looking cat pictures.



Project workflow

1. Understand Generative Adversarial Networks
2. Find and prepare training data
3. Build the models
4. Train the models
5. Generate Cats!
6. Conclusions

Understand Generative Adversarial Networks

Generative Adversarial Networks

GANs were introduced by Ian J. Goodfellow and co-authors in 2014[1]. These networks consist of two neural networks - a generator and a discriminator - which compete with each other in a 'game' (thus the "adversarial"):

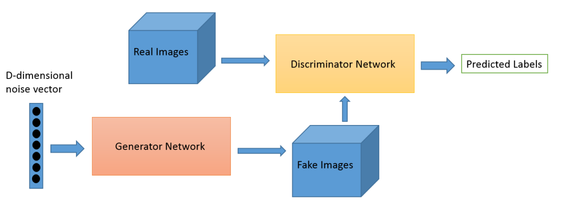


Figure 1: GAN Schema. Source: https://www.oreilly.com/content/generative-adversarial-networks-for-beginners/

The generator model is a *deconvolutional neural network* that transforms random input values (noise) into images. The discriminator model is a *binary (convolutional) classifier* that evaluates whether a given image is a real image from the training data set or a fake image created by the generator. The discriminator learns to tell real images apart from fake images created by the generator. At the same time, the generator learns how to produce images that the discriminator can't distinguish from real images.

Find and prepare training data

Training Data

For the sake of this project I only wanted to generate cat faces and not entire cats. I obtained a collection of 29843 images of cat faces of size 64x64 from GitHub user Federico Ferlito (Ferlix)[1].

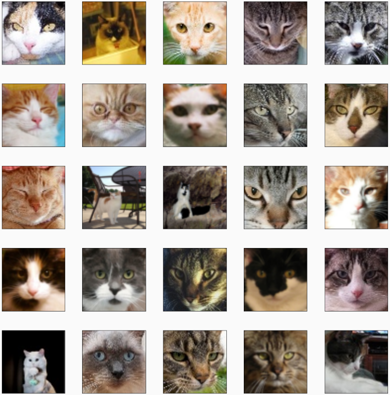


Figure 2: 25 Examples of cat faces from the training data set

[1] https://github.com/Ferlix/Cat-faces-dataset

Training Data

Potential problem: The training data is noisy
Many images in the training data set are not centered on the cat's face, and some don't contain a cat at all.

For this project, I ignored this issue. Instead, I pre-processed the data as following:

1.
```
allcats_norm = (allcats - 127.5) / 127.5
print(np.max(allcats_norm[3,:,:,:]), np.min(allcats_norm[3,:,:,:]))    # print max and min values

0.9058823529411765 -0.9764705882352941
```

   Scale images to the range [-1, 1]: It is recommended to use the hyperbolic tangent activation function (tanh) for the generator model output, which gives values ranging from [-1, 1]. Therefore, it is also recommended that real images used to train the discriminator are scaled so that their pixel values are in the range [-1, 1].

2.
```
def flip(x: tf.Tensor) -> (tf.Tensor):
    '''
    takes a tensor of images and flips the images horizontally with 50% chance
    using tensorflow's random_flip_left_right function. returns a tensor with
    flipped images.
    '''
    x = tf.image.random_flip_left_right(x)
    return x

[ ] # create tensorflow type dataset with flipped images:
    cat_features_data = tf.data.Dataset.from_tensor_slices(cat_images_tf).shuffle(SAMPLE_SIZE).map(flip).batch(BATCH_SIZE)
```

   When sampling training batches, flip 50% of the images horizontally: to enlarge the training data set.

Build the models

The Generator Model

I first built a generator model following the tutorial on tensorflow.org[2]. After some reading[3] and tweaking, my final generator network looked like this:

Input: noise vector, 128

- Fully connected layer, reshape
- 5x Deconvolution layers, each followed by **Batch Normalization** and **relu** activation
- Fully connected layer with **tanh** activation function

Output: 'image' of 64x64 pixels and 3 channels.

```
Model: "Cat Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2048)              264192
_________________________________________________________________
reshape (Reshape)            (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 4, 4, 512)         1638400
_________________________________________________________________
batch_normalization (BatchNo (None, 4, 4, 512)         2048
_________________________________________________________________
re_lu (ReLU)                 (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 8, 8, 256)         3276800
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
re_lu_1 (ReLU)               (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 16, 16, 128)       819200
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
re_lu_2 (ReLU)               (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 32, 32, 64)        204800
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
re_lu_3 (ReLU)               (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 64, 64, 32)        51200
_________________________________________________________________
batch_normalization_4 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
re_lu_4 (ReLU)               (None, 64, 64, 32)        0
_________________________________________________________________
dense_1 (Dense)              (None, 64, 64, 3)         99
=================================================================
Total params: 6,258,659
Trainable params: 6,256,675
Non-trainable params: 1,984
```

Figure 3: Model Summary of the generator network

[2] https://www.tensorflow.org/tutorials/generative/dcgan

[3] https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/

The Discriminator Model

My final discriminator model looked like this:

Input: ’image’ of 64x64 pixels and 3 channels

- 4x convolution layers, each followed by **Batch Normalization** and **leaky relu** activation
- Flatten, fully connected layer with **sigmoid** activation function

Output: a number between 0 and 1

```
Model: "Cat Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        3072
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        65536
_________________________________________________________________
batch_normalization_5 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         131072
_________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 256)         524288
_________________________________________________________________
batch_normalization_7 (Batch (None, 4, 4, 256)         1024
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 4, 4, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 4097
=================================================================
Total params: 729,857
Trainable params: 728,961
Non-trainable params: 896
```

Figure 4: Model Summary of the discriminator network

Parameters, Optimizer and Loss Functions

Model Hyperparameters:

```
# MODEL HYPERPARAMETERS
BATCH_SIZE = 128                                                        # minibatch size
WEIGHT_INIT_STD = 0.02                                                  # make the std more narrow than default helped for some reason i don't know
WEIGHT_INIT_MEAN = 0.0
LEAKY_RELU_SLOPE = 0.2
DOWNSIZE_FACTOR = 2
SCALE_FACTOR = 4 ** DOWNSIZE_FACTOR
GENERATOR_LR = 0.0002                                                   # learning rates
DISCRIMINATOR_LR = 0.0002
NOISE_DIM = 128                                                         # size of the noise vector to give the generator as input

WEIGHT_INITIALISER = tf.keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STD, mean = WEIGHT_INIT_MEAN, seed = 42)
```

Optimizer

Adam

Model Loss:

- Generator Loss: cross entropy between a tensor of ones and the discriminator prediction of generated images.
- Discriminator Loss: the sum of cross entropy between a tensor of ones and the discriminator prediction of real images AND cross entropy between a tensor of zeros and the discriminator prediction of fake images.

Train the models

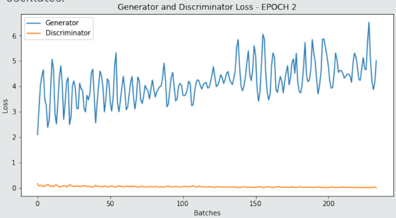
Training routine

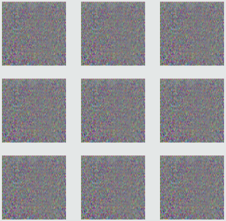
- for every epoch:
  - for every batch:
    - Sample noise (batch size, noise dim)
    - Generate images based on noise
    - Evaluate generated images
    - Evaluate real images
    - Calculate Generator Loss
    - Calculate Discriminator Loss
    - Apply gradients from optimizer
  - Collect losses from batches
  - Display epoch losses for certain epochs
  - Generate and display images for certain epochs without training

Model Training

The first epochs (1-2):

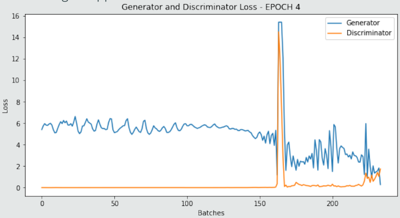
Discriminator Loss reaches 0 quickly while the Generator Loss oscillates:

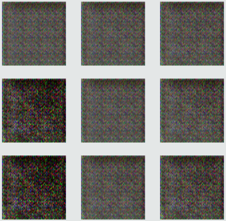




Around epoch 3-4:

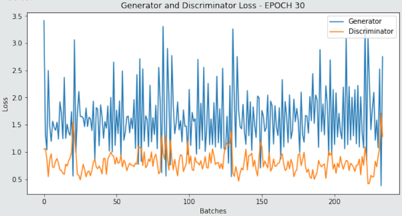
Some magic happens!

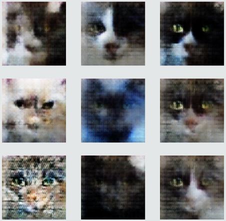




Around epoch 30:

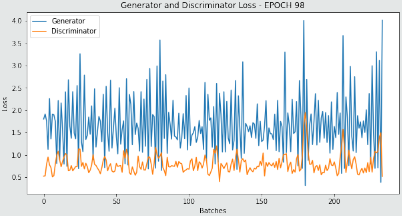
Cats!

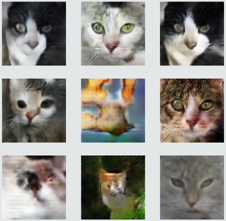




Around 100 epochs:

Some cats start to look good!





Training Considerations

Duration:
On Google Colab GPU backend, 1 epoch took about 1 minute. Training this model for 100 epochs thus takes about 1h 35min.

When to stop training?
With training GAN's, there is no 'early stopping'. One could basically train these Models forever. Stopping the training is done by examining the output and decide that it is 'good enough'. I stopped training at 201 epochs.

Visualising the training process:
It is useful to take the same noise input to evaluate the generator (plotting generated images without training) - this allows one to 'see' what the network is doing. To get an idea of what happens during the training process, one can combine the generated outputs to a .gif: `https://imgur.com/R4Yr05G`

Generate Cats!

Hundreds of Cats!

Figure 5: 100 generated cats after epoch 201

Conclusions

Conclusions

- Training GAN's is interesting and fun but takes a long time.
- Cleaning up the training data could be quite fruitful.
- I tried adding noise to labels but it did not improve training.
- I could try more 'gan hacks'[4] such as adding Dropout.

[4] https://github.com/soumith/ganhacks

Questions?

References i

Ian J. Goodfellow et al.
Generative adversarial networks.
Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014), pages 2672–2680., 2014.