Fusing animal images using deep learning

Elsa Scola Martín

Abstract

In this document I report my proposal for the application of deep neural networks to fuse animal images. The performed tasks were: Firstly, do some research about which is the most appropriate DNN architecture. I concluded the best option was to follow a Generative Adversarial Network approach. Secondly, search the appropriate datasets for the image fusion and preprocess it. After that, the GAN was built by creating the placeholders, the discriminator and generator, as well as the model loss and optimization functions. Finally, the neural network is trained and the output of the generator is periodically displayed. The results show that the neural network performs much better when the dataset is consistent. In addition, it shows the loss function does not correlate with the image quality, and therefore, using Wasserstein GANs for future improvements is proposed as a solution to this problem.

Contents

1 Description of the problem

Adversarial Networks, Variational Autoencoders and other generative DNNs allow to learn generative models able to reproduce distributions. They have been successfully applied to create artificial images that resemble real images. Convolutional neural networks have been also applied to manipulate artistic images. For example, they have been used for style transfer or image translation.

Chimeras are mythical animals formed from parts of various animals. The goal of this project is to design and implement a deep learning approach (any of those mentioned in the previous section) for naturally fusing images of different animals (e.g., cats and horses) to create a "chimera".

2 Selecting the appropriate DNN architecture

For this project, I've tried the following approaches:

Style transfer: This was my first approach. I used the pre-trained VGG19 model, which is a deep convolutional neural network built at the University of Oxford (see the paper: Very Deep Convolutional Networks for Large-Scale Image Recognition [15]).It has been trained on the ImageNet [14] dataset: 14-million images from 1,000 categories. VGG19's primary purpose is to identify objects in images.

The model has many layers, including many convolution layers.

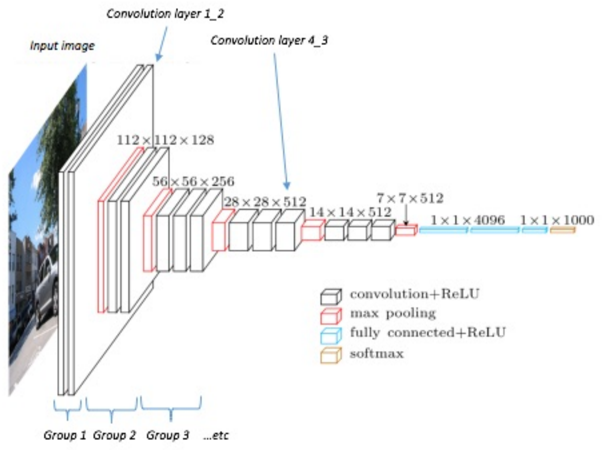


Figure 1: Diagram reproduced from the Heuritech blog.

I tried extracting the content and style from the images and then putting it together to paint both style and content, but the results weren't satisfying at all. In part, I think it didn't gave me significant results because the images weren't the right ones, instead of using two animal pictures I should have tried by using one picture of the animal, and another of just the skin of another animal.

Generative Adversarial Networks: My second approach turned out better than the previous one, in part, because I chose better images, but I will talk about the datasets later on. I decided to try this approach as it was mentioned in class and seemed to fit perfectly in this case. Many articles show experiments of image interpolation using GANs [20]. This section is shorter as I am going to explain the concepts behind GANs on the following parts.

3 Image datasets selection

In this approach my goal is to use GANs to generate images of **people mixed with dogs**. Nevertheless, before that I will ensure the neural network works properly by following the next steps:

1. Test the neural network on the MNIST dataset [16]. This data is simpler, which will allow me to see how well my model trains sooner. In total there are 60.000 written digits.
2. Test the neural network on the CelebA dataset [17]. As I'm going to be mixing people's faces with dog's faces, I will be testing the neural network with the Celebrity dataset first, which is more complex than the MNIST dataset, and I will obtain more concrete results on how is the performance of my neural network with people's faces. In total there are 200.000 person faces.
3. Once I've done the previous tests, I'm sure that neural network works properly, and therefore, I can test it with the PersonDog dataset, which is a dataset that I have created by mixing the CelebA dataset with the Flickr Dog dataset [19], and the pictures of dogs from the LHI-Animal-Faces dataset [18]. In total there are 760 dog faces plus 200.000 person faces.

I've chosen to mix people and dog faces as there wasn't any animal face dataset that was big enough to obtain nice images, and my results with the celebrity dataset were pretty good.

4 Description of my approach

I organized the implementation of the project according to the tasks:

1. Download the data
2. Preprocess and Visualize the data
3. Build a Generative Adversarial Network
   (a) Function to create the Tensorflow placeholders
   (b) Discriminator function
   (c) Image generator function
   (d) Model loss function
   (e) Model optimization function
4. Train the neural network
   (a) Function to show the output of the generator
   (b) Train function
   (c) Training
5. Analysis of the results

5 Download the data

As the data I'm using is quite big, I use this *helper.py* file that functions as a wrapper and enables to download the data from my hosting service from any computer without needing to pass the whole dataset by email or cloud services, it just makes it easier. There are comments all over the file of how each function works in *helper.py*.

6 Preprocess and Visualize the data

The values of the MNIST, CelebA and PersonDog datasets will be in the range of -0.5 to 0.5 of 28x28 dimensional images. The CelebA and PersonDog dataset images will be cropped to remove parts of the image that don't include a face, then resized down to 28x28. See the method *get_image* from *helper.py*. The MNIST images are black and white images with a single color channel, while the CelebA and PersonDog images have 3 color channels (RGB color channel).

In this case I use the helper file to show a preview of the dataset I'm working with. This often helps to imagine what kind of results you are looking for. It is displayed on a 5 x 5 grid, which is the way I'll be displaying the results through all the notebook.

7 Build a Generative Adversarial Network

7.1 GANs

According to Yann LeCun, “adversarial training is the coolest thing since sliced bread.” Generative adversarial networks—or GANs, have dramatically sharpened the possibility of AI-generated content, and have drawn active research efforts since they were first described by Ian Goodfellow et al. in 2014 1.

GANs are neural networks that learn to create synthetic data similar to some known input data. They are composed of 2 separate deep neural networks competing each other: the generator and the discriminator. Their goal is to generate data points that are similar to some of the data points in the training set.

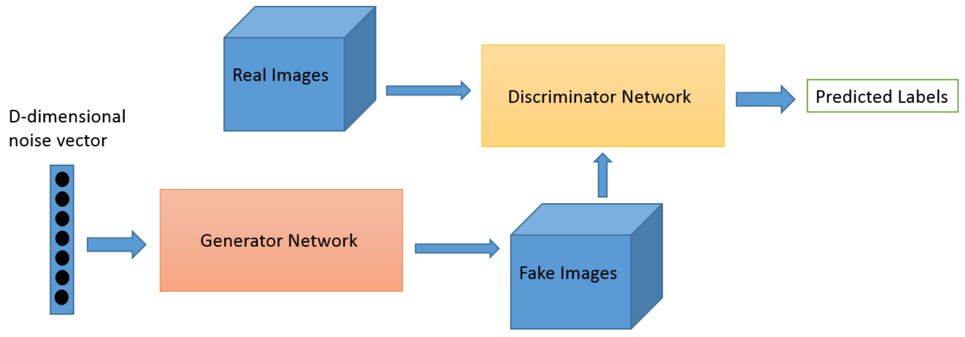


Figure 2: GAN architecture.

The discriminator model is a classifier that determines whether a given image looks like a real image from the dataset or like an artificially created image. This is basically a binary classifier that will take the form of a normal convolutional neural network (CNN).

The generator model takes random input values and transforms them into images through a convolutional neural network.

Over the course of many training iterations, the weights and biases in the discriminator and the generator are trained through backpropagation. The discriminator learns to tell ”real” images of handwritten digits apart from ”fake” images created by the generator. At the same time, the generator uses feedback from the discriminator to learn how to produce convincing images that the discriminator can’t distinguish from real images.

For this reason we say that they compete with each other.

7.2 Tensorflow placeholders

Placeholders is the building block in computation graph of any neural net (especially in tensorflow). Is a more basic structure than the variables. A placeholder is simply a variable that we will assign data to at a later date. It allows us to create our operations and build our computation graph, without needing the data. In TensorFlowterminology, we then feed data into the graph through these placeholders.

My function creates 3 Tensorflow Placeholders for the Neural Network:

1. Real input images placeholder
2. Z input placeholder
3. Learning rate placeholder

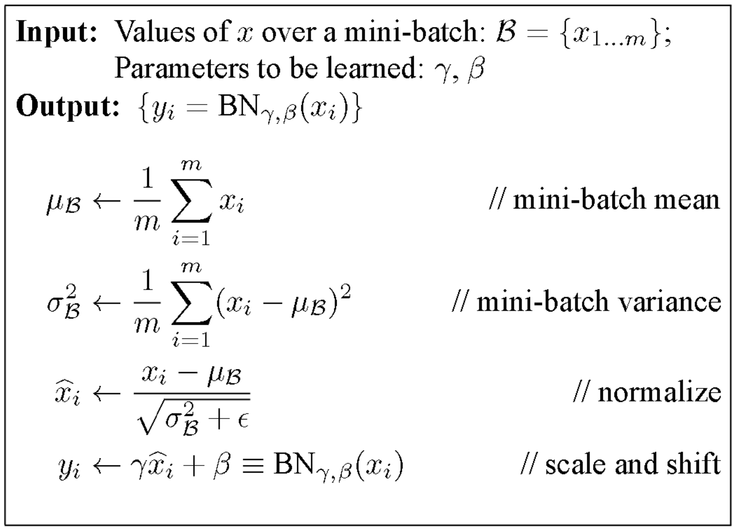
7.3 Discriminator

When given an image, the discriminator must be looking for components of the face to be able to distinguish correctly. Intuitively, some of the discriminator's hidden neurons will be "excited" when it sees things like eyes, mouths, hair, etc.

Our discriminator is a convolutional neural network. I chose not to use pooling layers to decrease the spatial size. Max pooling generates sparse gradients, which affects the stability of GAN training.

I used Leaky ReLU. We never want sparse gradients (0 gradients). Therefore, we use a leaky ReLU to allow gradients to flow backwards through the layer unimpeded.

In addition, I used Batch normalization [21]. I initialize the BatchNorm Parameters to transform the input to zero mean/unit variance distributions, but as the training proceeds it can learn to transform to x mean and y variance, which might be better for the network.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 3: Algorithm of Batch Normalization copied from the Paper by Ioffe and Szegedy mentioned above.

Finally, I also used a sigmoid at the output layer. This squeezes pixels of mixed colors, resulting in a crisper image.

7.4 Generator

The generator must try to make the discriminator outputs when given its generated image. In order to train the generator, the discriminator has to tell the generator how to tweak that generated image to be more realistic. For that, the generator must ask for suggestions from the discriminator. You do that by back-propagating the gradients of the discriminator's output with respect to the generated image.

At first, I didn't apply batch normalization to the generator, and its learning seemed to be really not efficient. After applying batch normalization layers, learning improved considerably.

7.5 Loss and optimization functions

The loss function builds the GANs for training and calculate the loss. The function returns the discriminator loss and the generator loss by using the previously implemented functions (Discriminator and Generator).

On the other side, the optimizer creates the optimization operations for the GANs by using *tf.trainable_variables* to get all the trainable variables. It filters the variables with names that are in the discriminator and generator scope names. The function returns the discriminator training operation and the generator training operation.

7.6 Train the neural network

I will list the most important lessons I have learned to train a GAN properly:

1. Normalize the inputs
   (a) Normalize the images between -1 and 1.
   (b) Use Tanh as the last layer of the generator output.
2. Avoid Sparse Gradients: ReLU, MaxPool
   (a) The stability of the GAN suffers if you have sparse gradients.
   (b) LeakyReLU works good in both Generator and Discriminator.
   (c) For Downsampling, use: Conv2d + stride
3. Apply Label Smoothing in the model loss function
4. Use the ADAM Optimizer
   (a) See Radford et. al. 2015. [22]
5. Use Dropouts in Generator in both train and test phase
   (a) Provide noise in the form of dropout (50%).
   (b) See this article [23].

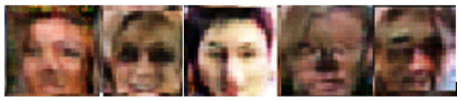
8 Analysis of the results

Analysing the three results from the three trials of the Generative Adversarial Network, we can see that the neural network much better when there is a consistent dataset. Meaning by consistent, that the data maintains a similar format. It also helps a lot for the training, to have a big dataset, because smaller ones don't give the network the appropriate time to train, and therefore, the results are poor.

The PersonDog dataset has less than 800 dog faces and over 200.000 person faces. Obviously, there isn't enough dog data, but I have prioritized quality versus quantity after doing several trials with much bigger datasets, where the majority of the images had a lot of noise, and where it didn't only appear the dog's face, but also its body and the environment (usually with many other elements in it). This situation accurs for other animal dataset too, and for this reason, I've chosen to mix this small "good quality" dataset with a big and consistent dataset, which is the Celebrity dataset.

The results could definitely be improved, by creating a consistent dataset of dog faces, which had only one specie's data, because the changes between the different species make the data more inconsistent, and therefore the results are worse.

Nevertheless, the results are pretty decent. In the following images I will show my favourite Chimeras results (that look a bit like a werewolf):



Additionally, I will show some good results from the MNIST and CelebA datasets:

Regarding the standard loss function, we can also appreciate that the quality of generated images does not correlate with the loss of either the generator or the discriminator. Since both are competing against each other, the losses fluctuate. For future major improvements here is an article [24] that explains how GANs have always had problems with convergence and, as a consequence, there is no way to know when to stop training them. In other words, the loss function doesn't correlate with image quality.

This is a big problem because:

1. you need to be constantly looking at the samples to tell whether you model is training correctly or not.
2. you don't know when to stop training (no convergence).
3. you don't have a numerical value that tells you how well are you tuning the parameters.

This interpretability issue is one of the problems that Wasserstein GANs aims to solve.

9 Conclusion

We can observe, that in the context image generation, GANs perform remarkably well when having the appropriate data. In conclusion, from the results obtained during this project, we can infere that this technique could be used in many companies, for example, Game development and animation production are expensive and hire many production artists for relatively routine tasks. GANs can auto-generate and colorize Anime characters, which would decrease those production costs.

10 Implementation

All the project steps were implemented in Python. I used the *helper.py* for loading the data and Tensorflow for the implementation of the neural network. I illustrate how the implementation works in the Python notebook `Project52-Scola-Notebook.ipynb`. Inside of the compressed file that I submitted, the following elements can be found:

- the Jupyter Notebook
- an HTML file version of the Jupyter Notebook
- this report
- *helper.py*
- and an *img* folder

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014.

[2] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. CoRR, abs/1312.6114, 2013.

[3] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and J¨urgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In IJCAI Proceedings-International Joint Conference on Artificial Intelligence, volume 22, page 1237. Barcelona, Spain, 2011.

[4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia, pages 675–678. ACM, 2014.

[5] Yoon Kim. Convolutional neural networks for sentence classification. CoRR, abs/1408.5882, 2014.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[8] Alex J Champandard. Semantic style transfer and turning two-bit doodles into fine artworks. CoRR, abs/1603.01768, 2016.

[9] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576, 2015.

[10] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2414–2423, 2016.

[11] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. arXiv preprint arXiv:1703.07511, 2017.

[12] Hao Dong, Paarth Neekhara, Chao Wu, and Yike Guo. Unsupervised image-to-image translation with generative adversarial networks. arXiv preprint arXiv:1701.02676, 2017.

[13] He Zhang, Vishwanath Sindagi, and Vishal M Patel. Image de-raining using a conditional generative adversarial network. arXiv preprint arXiv:1701.05957, 2017.

[14] http://image-net.org/

[15] Karen Simonyan and Andrew Zisserman,Very Deep Convolutional Networks for Large-Scale Image Recognition,CoRR, abs/1409.1556,2014.

[16] http://yann.lecun.com/exdb/mnist/

[17] https://www.kaggle.com/jessicali9530/celeba-dataset

[18] http://www.stat.ucla.edu/ zzsi/HiT/exp5.html

[19] http://www.recod.ic.unicamp.br/ rwerneck/datasets/flickr-dog/

[20] http://www.k4ai.com/dcgan/

[21] Sergey Ioffe and Christian Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, CoRR, abs/1502.03167, 2015

[22] Alec Radford and Luke Metz and Soumith Chintala, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, CoRR, abs/1511.06434, 2015.

[23] Phillip Isola and Jun-Yan Zhu and Tinghui Zhou and Alexei A. Efros, Image-to-Image Translation with Conditional Adversarial Networks, CoRR, abs/1611.07004, 2016.

[24] Martin Arjovsky, Soumith Chintala, Léon Bottou, Wasserstein GAN, abs/1701.07875, 2017.