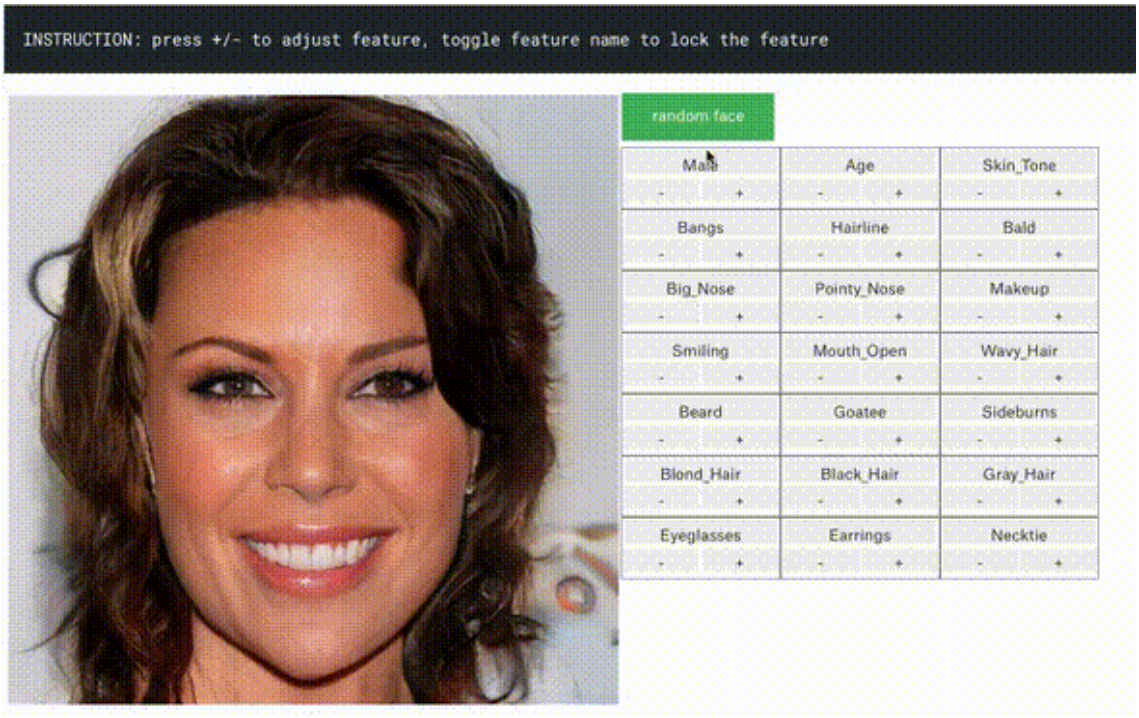
Published in Insight

Shaobo GUAN Follow

Oct 25, 2018 · 12 min read

Generating custom photo-realistic faces using AI

Controlled image synthesis and editing using a novel TL-GAN model



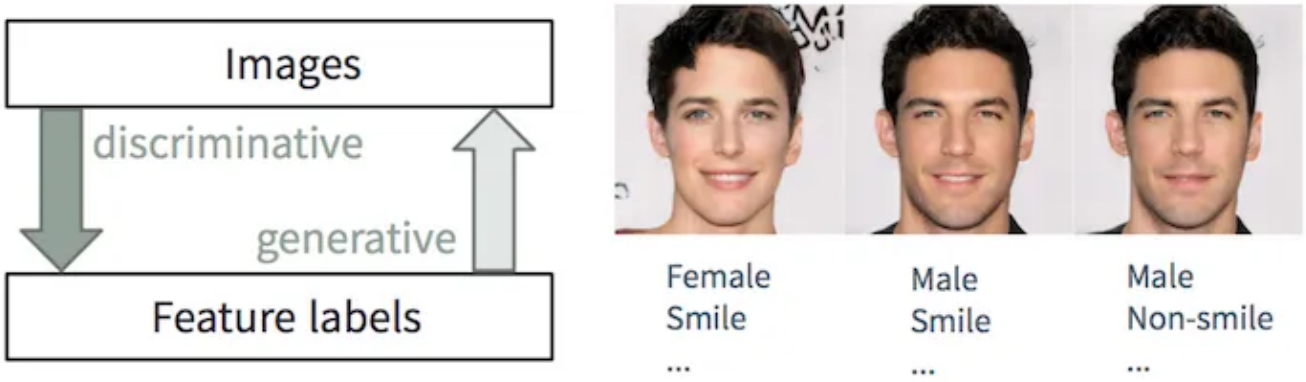
Example video of the controlled synthesis using my model (transparent latent-space GAN, or TL-GAN)

Want to learn applied Artificial Intelligence from top professionals in Silicon Valley or New York? Learn more about the Artificial Intelligence program at Insight.

Are you a company working in AI and would like to get involved in the Insight AI Fellows Program? Feel free to get in touch.

All the code and online demo are available at the project page.

Teaching computers to draw photos according to descriptions



Discriminative vs. generative tasks

Describing an image is easy for humans, and we are able to do it from a very young age. In machine learning, this task is a **discriminative** classification/regression problem, i.e. predicting feature labels from input images. Recent advancements in ML/AI techniques, especially deep learning models, are beginning to excel in these tasks, sometimes reaching or exceeding human performance, as is demonstrated in scenarios like visual object recognition (e.g. from AlexNet to ResNet on ImageNet classification) and object detection/segmentation (e.g. from RCNN to YOLO on COCO dataset), etc.

However, the other way around, g*enerating* realistic images based on descriptions, is much harder, and takes years of graphic design training. In machine learning this is a **generative** task, which is also much more challenging than discriminative tasks, as a generative model has to produce much richer information (like a full image at some level of detail and variation) based on a smaller seed input.

Despite the difficulty in creating such types of applications, **generative models** (with some control) can be extremely useful in many cases:

- **Content creation**: Imagine if an advertisement company could automatically generate attractive product images that match the content and style of the webpage where these images are inserted; a fashion designer could get inspiration by asking

an algorithm to produce 20 examples of shoe designs that are related to “leisure”, “canvas”, “summer” and “passionate”; and a new game could allow players to create realistic avatars based simple descriptions.

- **Content-aware smart editing**: We could allow a photographer to change the facial expression, amount of wrinkles and hair style of a profile photo with several clicks; and a Hollywood studio artist could transform the footage shot on a cloudy evening to look like it was shot on a sunny morning, with sunshine shedding light from the left side of the screen.

- **Data augmentation**: An autonomous driving car company could synthesize realistic videos of a particular type of accident scenario to augment the training dataset; and a credit card company could synthesize data of a particular type of fraud data that is underrepresented in the dataset to improve their fraud detection system.

In this post, we will describe our recent work called **Transparent Latent-space GAN (TL-GAN)**, which extends current cutting edge models to provide a new interface. We are currently working on a paper, that will have more technical details.

Overview of generative models

The deep learning community is making rapid progress on generative models. Among them are three promising types of models: autoregressive models, variational autoencoders (VAE) and generative adversarial networks (GAN), illustrated as the figure below. If you are interested in the details, please check out this awesome OpenAI blog post.

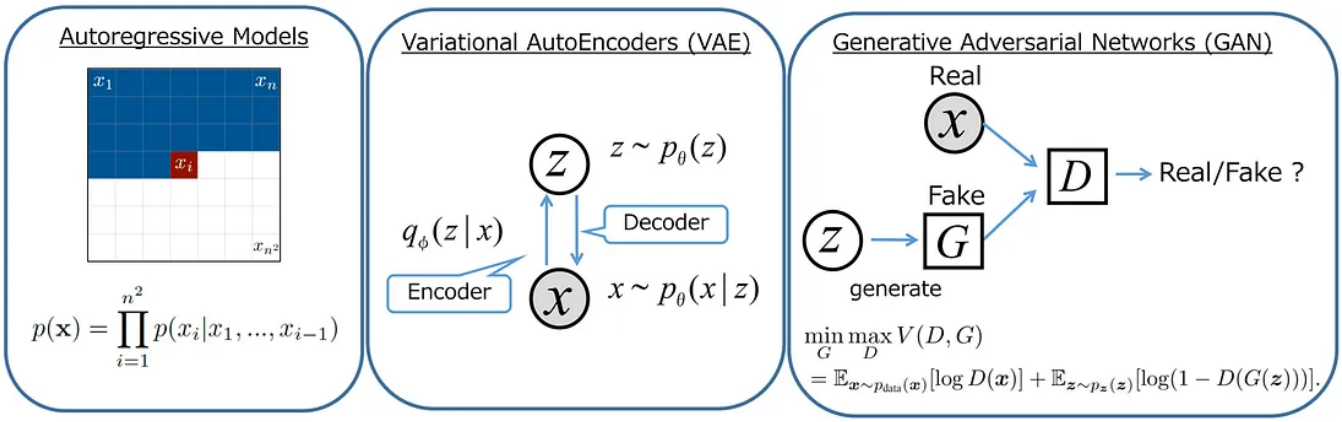


Figure: comparison of generator networks. Image from University of Waterloo STAT946F17 course

So far, GANs produce images of the highest quality (photo-realistic and diverse, with convincing details in high resolution). Look at the stunning images generated by Nvidia's recent work with pg-GAN (progressively-growing GAN). For this reason, this blog post will focus on GAN models.

Figure: synthetic images generated by pg-GAN from Nvidia. None of these images are real!

Controlling the output of GAN models

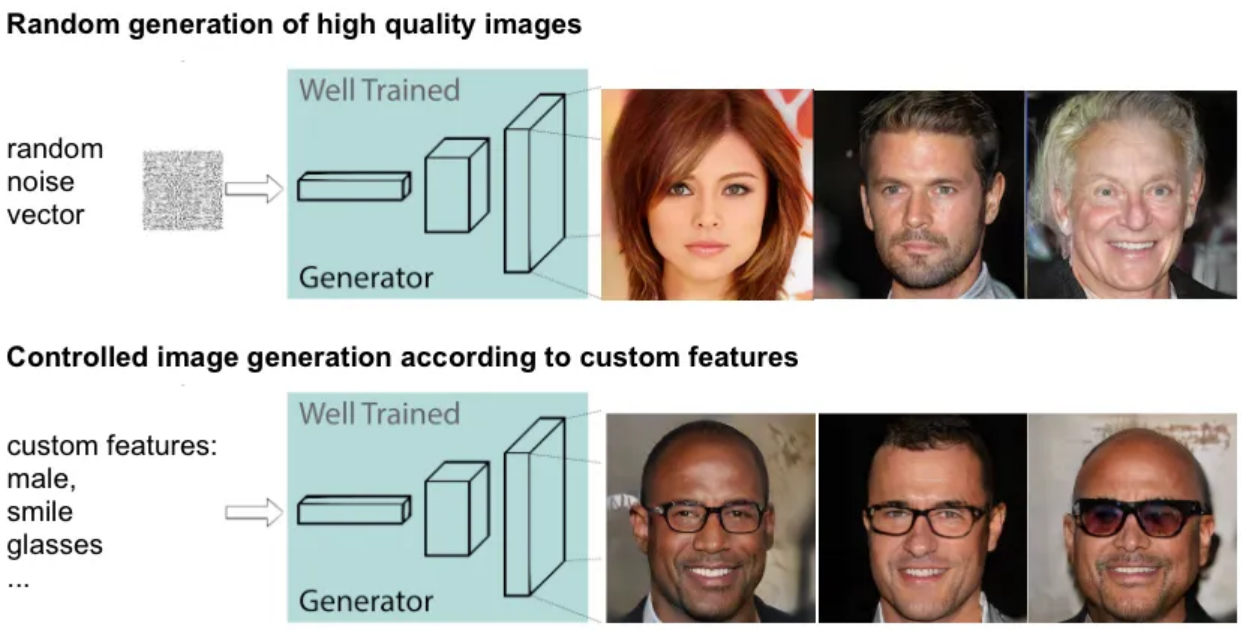


Figure: random image generation vs. controlled image generation

The original version of GAN and many popular successors (like DC-GAN and pg-GAN) are **unsupervised** learning models. After training, the generator network takes random noise as input and produces a photo-realistic image that is barely distinguishable from the training dataset. However, we cannot further control the features of the generated images. In most applications (such as the scenarios described in the first section), users would like to generate samples with **custom features** (like age, hair color, facial expression, etc), and ideally, tuning each feature continuously.

To achieve controlled synthesis, numerous variants of GAN have been created. They can be roughly divided into two types: style-transfer networks and conditional generators.

Style-transfer networks

Style-transfer networks, represented by CycleGAN and pix2pix, are models trained to translate image from one domain to another (e.g. from horse to zebra, from sketch to colored images). As a result, we cannot continuously tune one feature gradually between two discrete states (eg. add slightly more beard on the face). Also, one network is dedicated to one type of transfer, so it requires ten different neural networks to tune 10 features.

Conditional generators

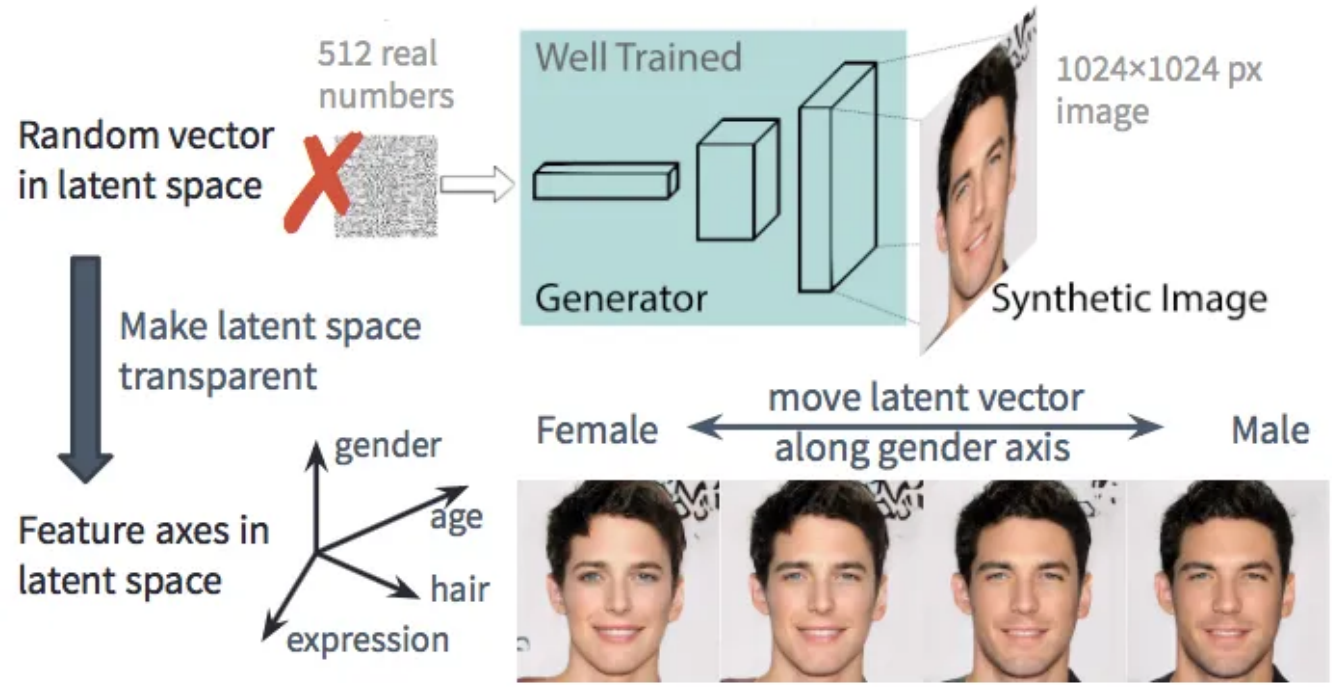
Conditional generators, represented by conditional GAN, AC-GAN, and Stack-GAN, are models that jointly learn images with feature labels during training time, enabling the image generation to be conditioned on custom features. Therefore, when you want to add new tunable features to the generation process, you have to retrain the whole GAN model, which takes an enormous amount of computing resources and time (e.g. days to weeks on a single K80 GPU with the perfect set of hyper-parameters). In addition, you have to rely on a single dataset that contains all the custom feature labels to perform the training, instead of leveraging different labels from multiple datasets.

Our model, which we call **Transparent Latent-space GAN** (TL-GAN), addresses these problems of existing methods by approaching controlled generation task from a novel angle. It offers users the ability to **gradually tune one or multiple features using a single network.** Besides, adding new tunable features can be done very efficiently in less than one hour.

TL-GAN: a novel and efficient approach for controlled synthesis and editing

Making the mysterious latent space transparent

We will leverage NVIDIA's pg-GAN, the model that generates the photo-realistic high resolution face images as shown in the the previous section. All the features of a generated 1024px*1024px image are determined solely by a 512-dimentional noise vector in the latent space (as a low-dimensional representation of the image content). Therefore, **if we could understand what the latent space represents (i.e., making it transparent), we could completely control the generation process.**



Motivation of TL-GAN: understand latent space to control generation process

By experimenting with the pre-trained pg-GAN, I found that the latent space actually has two good properties:

- It is well populated, meaning that most points in the space will generate reasonable images
- It is quite continuous, meaning the interpolation between two points in the latent space usually leads to a smooth transition of corresponding images.

With this in mind, my intuition was that it is possible to find directions in the latent space that are predictive of features that we care about (eg. male-female). If so, we can use unit vectors of these directions as the feature axes for controlling the generation

process (more male-like or more female-like).

Approach: uncovering the feature axes

To find these feature axes in the latent space, we will **build a link between a latent vector *z* and the feature labels *y*** through supervised learning methods trained on paired *(z,y)* data. Now the problem becomes how to get such paired data, since existing datasets only contain images *x* and their corresponding feature labels *y*.

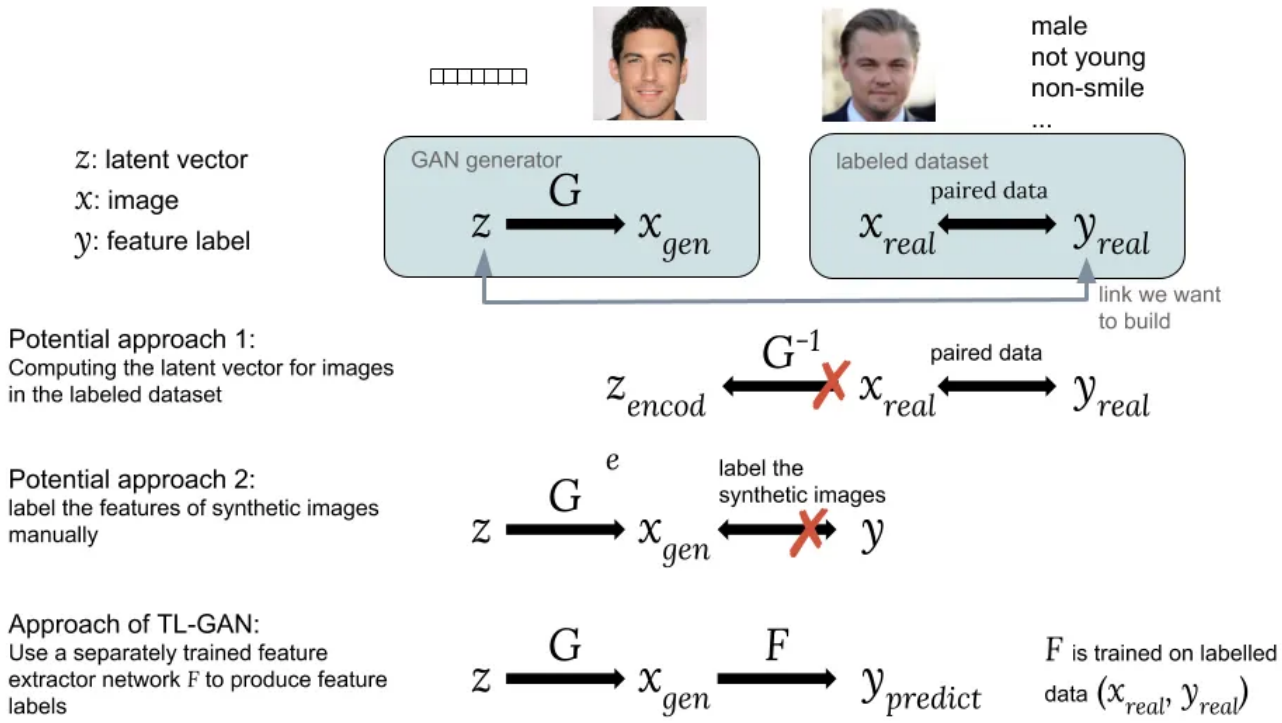


Figure: approaches to link latent vector z with feature label y

Potential approaches:

> *One potential approach is to compute the corresponding latent vectors* z *of images* x_real *from an existing dataset labeled with features of interest* y_real. *However, the GAN network does not offer an easy way to compute* z_encode=G^(−1)(x_real)*, making this idea difficult to implement.*
>
> *A second potential approach is to generate synthetic images* x_gen *using GAN from a random latent vector* z *as* x_gen=G(z). *The problem here is that the synthetic images are unlabeled, and we can not easily leverage the available labeled dataset.*

To solve this problem, the key innovation of our TL-GAN model is to **train a separate**

feature extractor (a classifier for discrete label or regressor for continuous label) model *y=F(x)* using an existing labelled image dataset *(x_real, y_real)*, and then couple the trained GAN generator G with the feature extractor network F. Once this is done, we can predict the feature labels *y_pred* of the synthetic images *x_gen* using the trained feature extractor network, and thus establish the link between z and y through synthetic images as *x_gen=G(z)* and *y_pred=F(x_gen)*.

Now that we have the paired latent vector and features, we can train a regressor model *y=A(z)* to uncover all feature axes for controlling the image generation process.

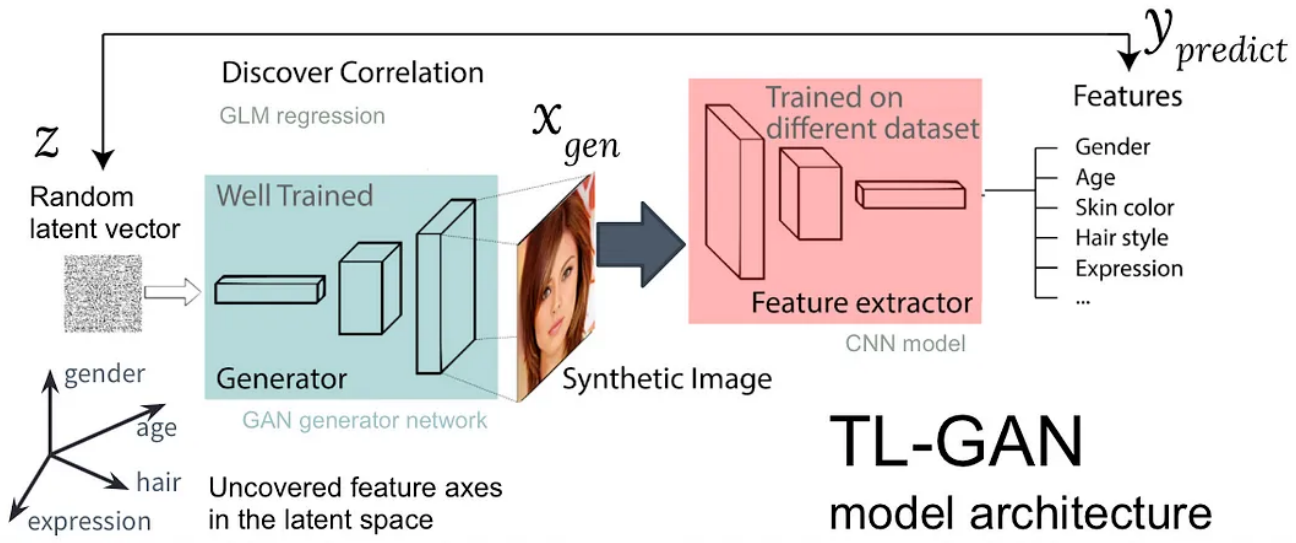


Figure: architecture of our TL-GAN model

The above figure shows the architecture of the TL-GAN model, which contains five steps:

1. **Learning the distribution:** Choose a well-trained GAN model and take the generator network. I chose the well-trained pg-GAN (provided by Nvidia), which offers the best face generation quality.

2. **Classification:** Choose a pre-trained feature extractor model (could be a convolutional neural network or other computer vision models), or train your own feature extractor network using a labelled dataset. I trained a simple convolutional neural network on the CelebA dataset (which contains 30,000+ face images with 40 labels each).

3. **Generation:** Generate a number of random latent vectors, pass through the trained

GAN generator to produce synthetic images, then use a trained feature extractor to produce features for every image.

4. **Correlation:** Use a Generalized Linear Model (GLM) to perform regression between latent vectors and features. **The regression slope becomes the feature axes.**

5. **Exploration:** Start from one latent vector, move it along one or more feature axes, and examine how this affects the generated images.

I made this process very efficient; once we have a pre-trained GAN model, identifying feature axes **only takes one hour** on a single-GPU machine. This is achieved by several engineering tricks including transfer learning, downsampling image size, pre-cache synthetic images, etc.

Results

Let us see how this simple idea works.

Moving latent vector along feature axes

First, I tested whether the discovered feature axes can be used to control the corresponding feature of the generated image. To do this, I generated a random vector z_0 in the latent space of GAN, and produced a synthetic image x_0 by passing it through the generator network $x\_0=G(z\_0)$. Next, I moved the latent vector along one feature axis u (a unit vector in the latent space, say, corresponding to the gender of the face) by distance λ, to a new location $x\_1=x\_0+\lambda u$, and generated a new image $x1=G(z1)$. Ideally, the corresponding feature of the new image would be modified toward the expected direction.

The results of moving the latent space vector along several example feature axes (gender, age, etc) are shown below. This works surprisingly well! We can **smoothly morph** the image between male ⟷ female, young⟷old, etc.

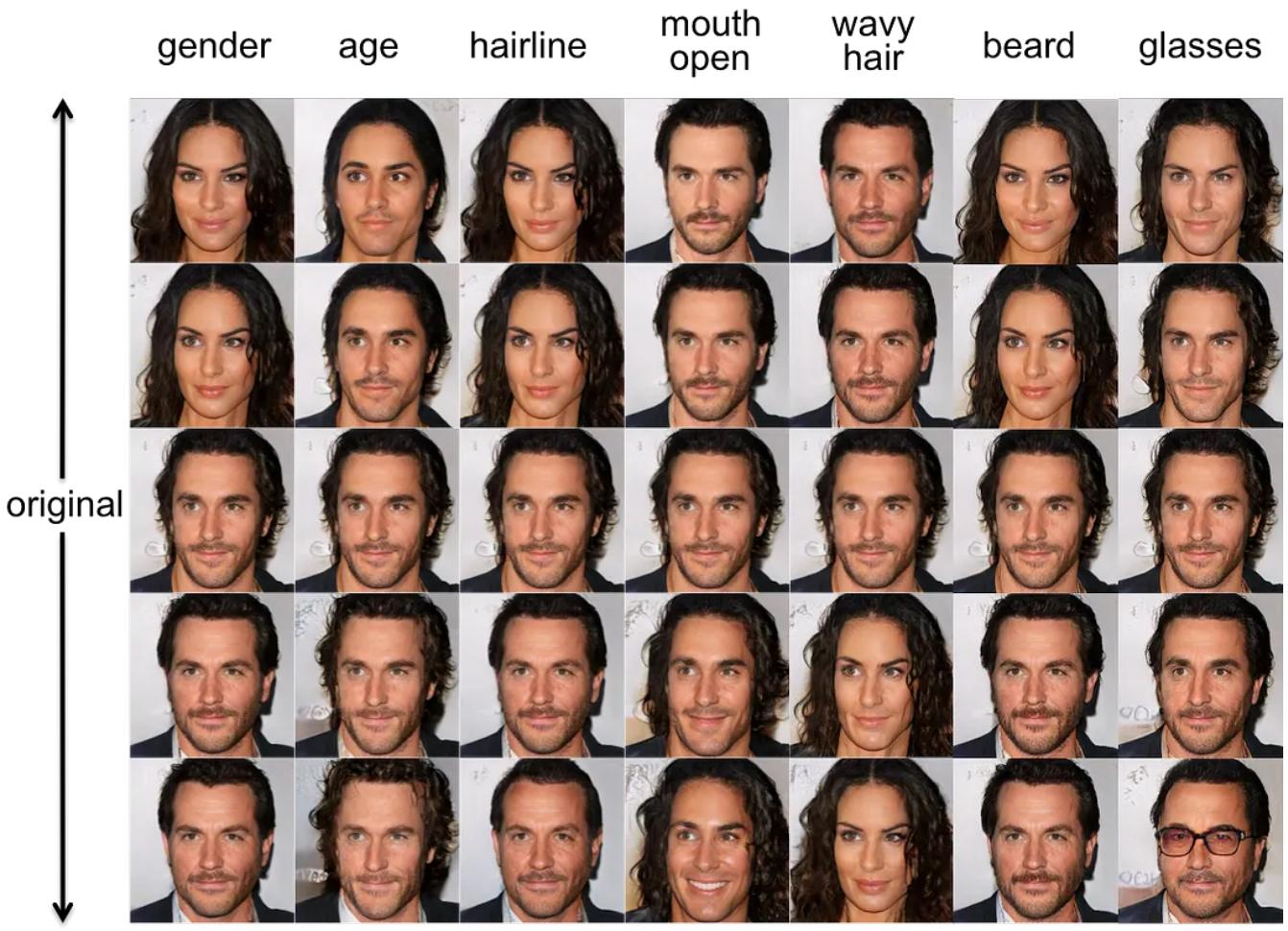


Figure: Initial result of moving latent vector along example entangled feature axes

Disentangling correlated feature axes

The examples above shows a shortcoming of the initial method, i.e., the **entangled feature axes**. For instance, when I intended to reduce the amount of beard, the generated faces became more female-like, which is not what a user would expect. This problem is due to the fact that the gender feature and the beard feature are by nature **correlated**, modifying one leading to the corollary change of the other. Similar things happened to other features like hairline and wavy hair. As is illustrated in the figure below, the original beard axis is not perpendicular to the gender axis in the latent space.

To solve this problem, I used straight-forward linear algebra tricks. Specifically, I projected the beard axis to a new direction that is orthogonal to the gender axis, which effectively removes their correlation, and thus could potentially disentangle these two features of generated face images.

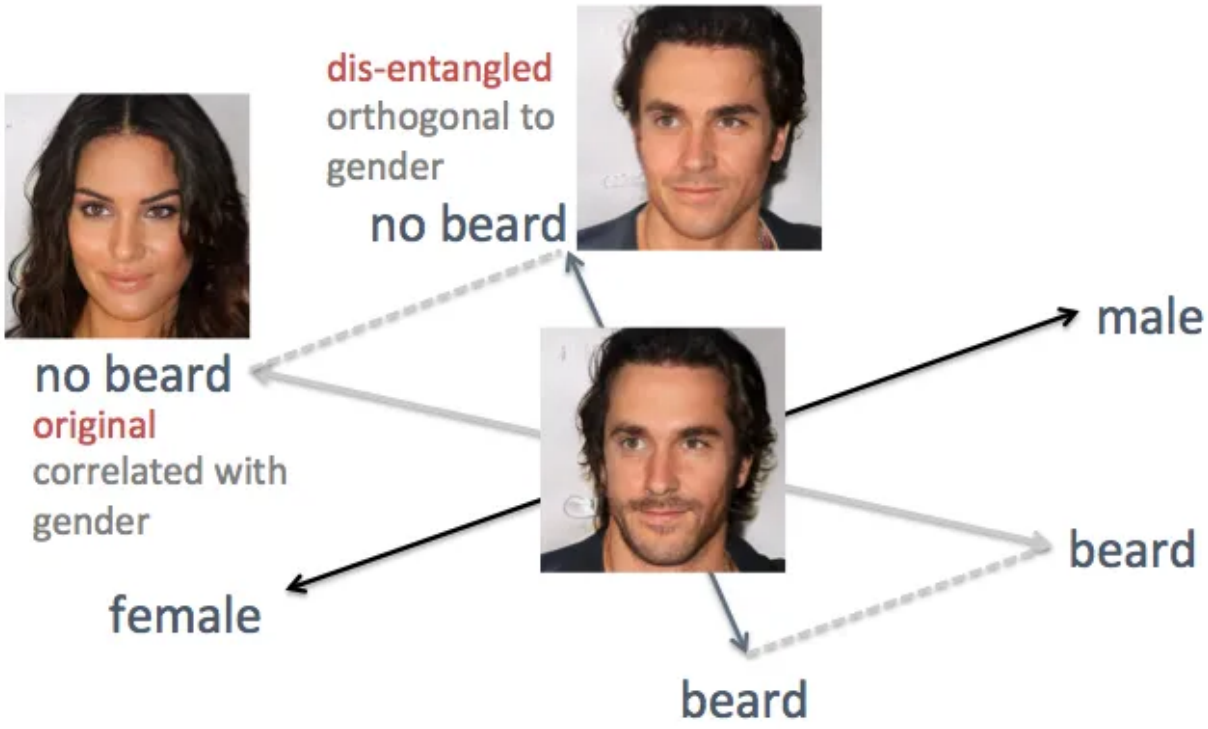


Figure: disentangle correlated feature axes using linear algebra tricks

I applied this feature disentangling method to the same example face. This time I used gender axis and age axis as reference features, projected all other feature axes so that they became orthogonal to gender and age, and examined the generated images when the latent vector moved along the newly generated feature axes (shown in the figure below). As we expected, the features including hairline, wavy hair and beard now modify the gender of face anymore, which behaves as expected.

Figure: Improved result of moving latent vector along example disentangled feature axes

Flexible interactive editing

To see how flexibly our TL-GAN model can control the image generation process, I built an interactive GUI to explore the effect of gradually tuning feature values along different feature axes, as shown below:

Video: interactive editing of face images using TL-GAN

Again, the model works surprisingly well when I use the features axes to control the generated images!

Recap

This project provides a novel method to control the generation process of an unsupervised generative model like GAN (generative adversarial network). Using an already well-trained GAN generator (Nvidia's pg-GAN), I made its latent space transparent by discovering the meaningful feature axes in it. When a vector moves along a feature axis in the latent space, the corresponding image morphs over this feature, which enables controllable synthesis and edit.

This method has clear advantages:

1. Efficiency: To add a new feature tuner for the generator, you do not need to re-train the GAN model, thus it only takes <1h to add 40 feature tuners with our method.

2. Flexibility: You can use any feature extractor trained on any dataset to add more feature tuners to the well-trained GAN.

A word about ethics

This work allows us to have fine grained control over image generation, but it still relies heavily on the features of our dataset. Training on photos of celebrities of Hollywood means that our model will be very good at generating photos of a predominantly white and attractive demographic. In turn, this would lead to users only being able to generate these specific kind of faces which are only representative of a small subset of people. If we were to deploy this as an application, we would want to make sure that we have augmented our initial dataset to take into account the diversity of our users.

In addition, while this tool could be a huge creative help, we should ask ourselves how this model could be used for nefarious purposes. If we can generate realistic looking faces of any type, what are the implications for our ability to trust in what we see. These sort of issues are important to tackle today. As we have seen with the recent applications of deepfakes, the capability of AI methods is increasing at a fast pace, so it is vital for us to start conversations about how to best deploy them.

Online demo and code

All the code and online demo of this work are available at this Github project page.

If you want to play with the model in your web-browser

You do not have to download any code, model or data. Just follow the instructions on this section of the GitHub README page. You can tweak faces in your web-browser as is demonstrated in the the above Video.

If you want to try my code

Just follow the README page of the GitHub repository. The code is built on Anaconda Python 3.6 with Tensorflow and Keras.

If you want to contribute

Machine Learning Insight Ai Artificial Intelligence Programming Creativity

About me

I recently finished PhD in computational and cognitive neuroscience from Brown

University, with a concurrent master degree in computer science focusing on machine learning. In the past, I studied how neurons in the brain collectively process information to achieve high-level functions like visual perception. I am enthusiastic about the algorithmic approach to understand, mimic, and implement intelligence, as well as apply them to solve challenging real-world problems. I am actively looking for ML/AI researcher positions in th tech industry.

This work was done as the project for the Insight AI fellow program over three weeks. I give my thanks to the program director Emmanuel Ameisen and Matt Rubashkin for their gener guidance, especially Emmanuel Ameisen for his suggestions and editing work on this blog post. I also give my thanks to all Insight staff for providing a great learning environment, ar to Insight AI fellows from whom I learned a lot. Special thanks to Ruobing Xia for providing numerous inspirations as I decided on project directions, and for the enormous help structuring and editing this blog post.

This passage was originally posted on my personal blog.