Deep Convolutional Generative Adversarial Networks: The Cure for Androgenic Alopecia

Following Report Guidelines Author Names, Emails, and Group Number are Excluded

December 5, 2020

Abstract

Generative adversarial networks apply basic principles of game theory to simultaneously train both networks that can produce novel content from latent random vectors and networks that can discriminate between real world and network-generated samples. Here we implement a deep convolutional generative adversarial network in order to train a generator to produce realistic images of human faces. Using a dataset of 3,059 faces, we find that our network is able to achieve figures from random vectors that can be qualitatively recognized as faces within a few hundred epochs and can create images closely resembling faces after several thousand epochs. Through simple arithmetic operations performed on the average of random vectors representing images with visually identifiable features, we are able to demonstrate how transformations within the latent vector space results in the exchange of properties in image space. We focus our attention particularly on hair, and present a novel contribution: that hair loss can simply be averted through vector math!

Keywords Generative Adversarial Network (GAN) – Convolutional Neural Network (CNN)

1 Introduction

1.1 Problem Definition

The aim of our project is to train a neural network which can create pictures of artificial human faces from random vectors and to modify image features through vector manipulation. In order to train our network we used a dataset of 3,059 human faces freely available online.[1] The dataset includes pictures of various individuals with differing displayed emotions. In Fig. 1, we give examples of 'real world' images from the dataset which display features explored in Sec. 4. Note that we have expanded the number of images with bald people in the dataset by 100 through horizontal reflection, training on a dataset with 3,159 total images. This is primarily because, while examples in class showed how glasses can be added to people through vector math, our idea for a novel contribution is to add hair to images of bald people. By expanding the portion of the training sample with bald people, we expect a greater portion of the generated images to also be bald.

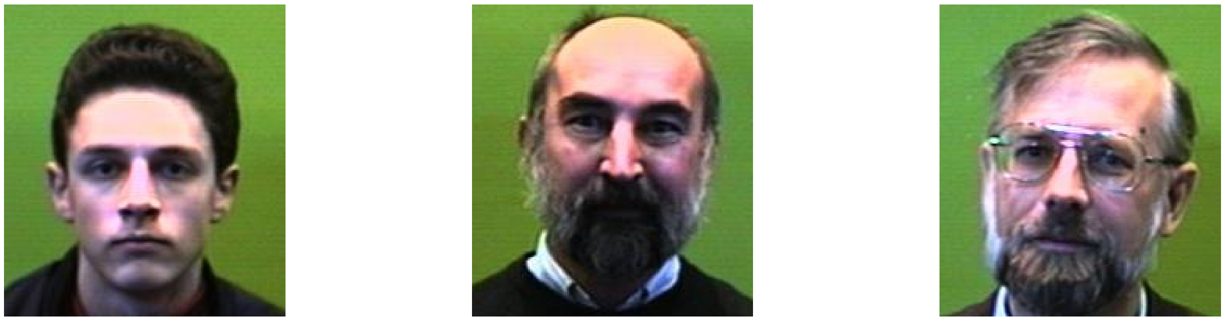

Figure 1: Images from our training dataset with features that explored among artificially generated images in Sec. 4, including hair (and the lack thereof) and glasses (and the lack thereof).

[1] https://www.kaggle.com/gasgallo/faces-data

1.2 Generative Adversarial Networks

First explored by Ref. [1], adversarial nets involve a unique method of training generative models that can make realistic images from random vectors of noise, the reverse of many convolutional networks which reduce images into categories. This is made possible by simultaneously training a discriminative model which acts as a traditional convolutional network. In such a training procedure, the discriminator attempts to accurately predict if a sample is from the training dataset or produced by the generator, while the generator attempts to decrease the accuracy of the discriminator. This can be thought of as a two-player minmax optimization game with a value function representing the success of the discriminator, which the generator seeks to minimize and the discriminator seeks to maximize.

In Sec. 2, we review how we constructed a generative network which creates images from noise vectors, a discriminative network that can distinguish real images from fake images, and the procedure for which we trained the networks against each other. In Sec. 3, we review the results of adversarial training. In Sec. 4, we explore various vector operations and their consequences in image space. Finally, in Sec. 5 we review our findings.

2 Methods

2.1 Generator Architecture

It is nothing short of remarkable that a convolutional network 'generator' can artificially create recognizable images from random noise. Such random vectors are not in themselves meaningful, rather the ability arises from weights being trained to assign regions of the latent vector space to represent image properties. The specific architecture of our generator therefore plays a large role in its ability to recognize vectors as images. In Table 1, we review this architecture.

We employed a Keras Sequential model expecting a vector of 100 randomly generated values. The model resembles a traditional convolutional network, but reversed to increase dimensionality into an image. We first used a densely connected layer of of 131072 nodes ($\equiv 32\times32\times128$, each connected to 100 inputs and a bias for 13238272 weights) employing the rectified linear unit (ReLU, $f(x) = x$ if $x > 0$ else $f(x) = 0$)) activation function and reshaped the output to be $32 \times 32 \times 128$. We further increased the dimensionality through upsampling to $64 \times 64 \times 128$. Just as convolutional windows are important in detecting features, we employ a convolutional layer with a 3×3 window and unit stride in order to build features. With a filter depth of 128, the output is $64 \times 64 \times 128$. We again use the ReLU activation function and, following Ref. [4], we normalize the activation of the layer for each training batch with a momentum of 0.8. After upsampling again to $128 \times 128 \times 128$, we again use a convolutional layer with 3×3 window size and unit stride, but decrease the depth to 64 (since we are building up to a RGB image), leading to a $128 \times 128 \times 64$ output. We use the same activation function and batch normalization as the first convolutional layer. We use a final convolutional layer (with the same window size and stride but with a $f(x) = \tanh(x)$ activation function) to reduce the filter depth to 3, resulting in an image we hope can resemble a face after training.

Table 1: A summary of the architecture of our generator as detailed in Sec. 2.1.

Layer	Output Shape	Parameters
Input Latent Random Vector	100	0
Densely Connected Layer with ReLU Activation	131072	13238272
Reshape	$32 \times 32 \times 128$	0
2D Upsampling	$64 \times 64 \times 128$	0
2D Convolutional Layer	$64 \times 64 \times 128$	147584
Batch Normalization	$64 \times 64 \times 128$	512
ReLU Activation	$64 \times 64 \times 128$	0
2D Upsampling	$128 \times 128 \times 128$	0
2D Convolutional Layer	$128 \times 128 \times 64$	73792
Batch Normalization	$128 \times 128 \times 64$	256
ReLU Activation	$128 \times 128 \times 64$	0
2D Convolutional Layer	$128 \times 128 \times 3$	1731
Hyperbolic Tangent Activation	$128 \times 128 \times 3$	0

2.2 Discriminator Architecture

On their own, 'discriminator' networks that can distinguish between real content and that generated by computers might have increasing importance if computer generated content becomes more prevalent. While our stated goals are to examine input vectors and images produced by the generator, a discriminative network that is good at discerning faces from our sample and faces from the generator is critical for training our generator. An under-performing discriminator will not push the generator to develop more realistic images, whereas a discriminator that continually improves in detecting artificial images will allow for continued generator improvement throughout training. In Table 2, we review the architecture of our discriminator.

We employed a Keras Sequential model expecting a 128 × 128 × 3 input layer for our RGB images. Our first layer after the input had a depth of 32 and used a 3 × 3 convolutional window with a stride of 2 resulting in 64 × 64 × 32 output. We employed a 'leaky' version of the rectified linear unit activation function (LeakyReLU) following Ref. [2],

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{1}$$

where we set α to 0.2 for each instance of LeakyReLU. Then, since randomly excluding nodes has proven benefits to network training [3], we use a dropout of 25%. A subsequent convolutional layer was similar to the first with the same window size and stride but with a filter depth of 64, resulting in a 32 × 32 × 64 output which we padded with a border of zeros. We use the same fractional dropout and activation function and normalize activations for each training batch with a momentum of 0.8. Our third convolutional layer had the same stride and window size with a filter depth of 128, resulting in a 17 × 17 × 128 output, and employ the same bath normalization, dropout, and activation function as the previous to layers. Our final convolutional layer used a filter size of 256 and a stride of 1 rather than 3, resulting in 17 × 17 × 256 output. After batch normalization and dropout, a dense layer of 73,984 node connections and a bias led to a single output node with a sigmoid activation function ($f(x) = 1/(1 + e^{-x})$) which determined whether or not the input image was 'real.'

Table 2: A summary of the architecture of our discriminator as detailed in Sec. 2.2.

Layer	Output Shape	Parameters
Input Image	128 × 128 × 3	0
2D Convolutional Layer	64 × 64 × 32	896
LeakyReLU Activation	64 × 64 × 32	0
Dropout	64 × 64 × 32	0
2D Convolutional Layer	32 × 32 × 64	18496
2D Zero Padding	33 × 33 × 64	0
Batch Normalization	33 × 33 × 64	256
LeakyReLU Activation	33 × 33 × 64	0
Dropout	33 × 33 × 64	0
2D Convolutional Layer	17 × 17 × 128	73856
Batch Normalization	17 × 17 × 128	512
LeakyReLU Activation	17 × 17 × 128	0
Dropout	17 × 17 × 128	0
2D Convolutional Layer	17 × 17 × 256	295168
Batch Normalization	17 × 17 × 256	1024
LeakyReLU Activation	17 × 17 × 256	0
Dropout	17 × 17 × 256	0
Flatten	73984	0
Densely Connected Layer with Sigmoid Activation	1	73985

2.3 The Minimax Optimization Game

Our implementation of the adversarial procedure involves both networks training from a binary cross entropy loss function (which is analogous to the value function described in Sec. 1.2)

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_{true,\,i}\log\left(y_{predict,\,i}\right) + \left(1 - y_{true,\,i}\right)\log\left(1 - y_{predict,\,i}\right), \tag{2}$$

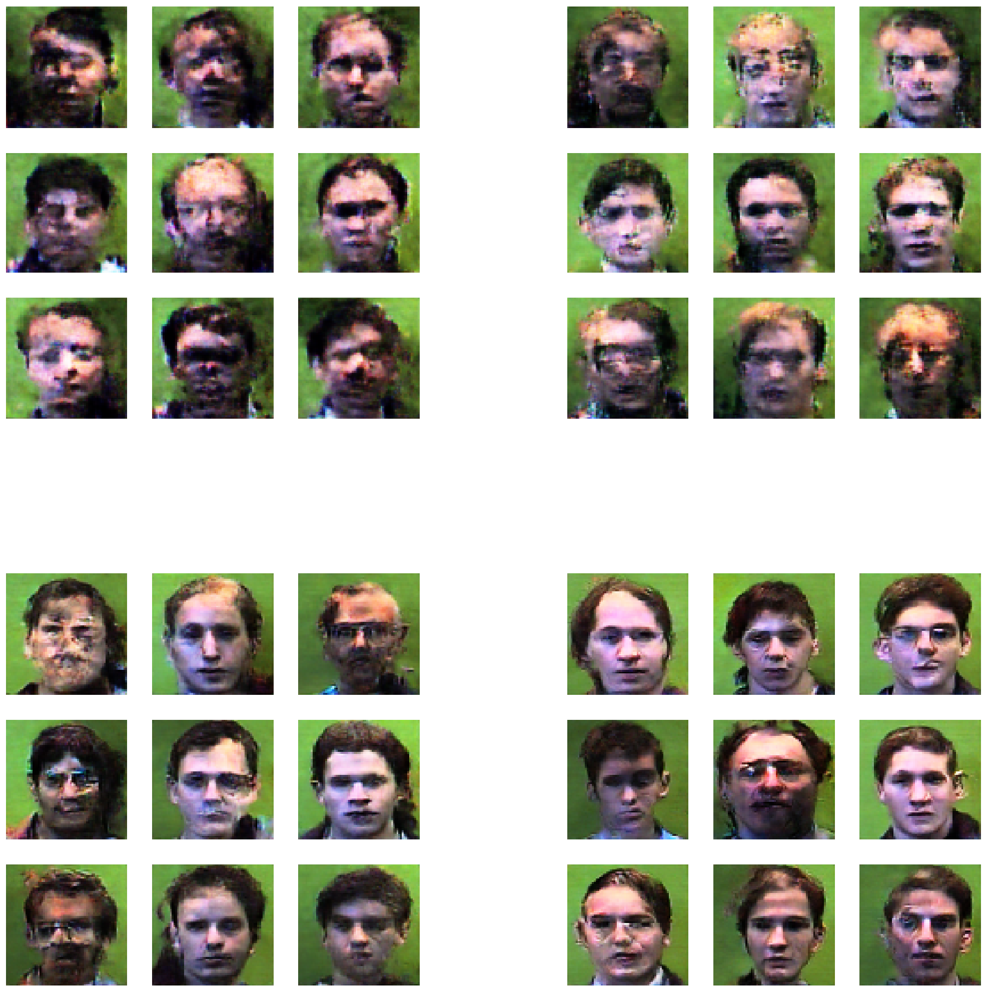

Figure 2: Generated images at various stages of the training process. Top left are 9 images from random noise vectors after 400 epochs, top right are 9 images from different random vectors after 800 epochs, bottom left are images after 3,000 epochs, bottom right are images after 9,000 epochs.

for a sample of N images. For the discriminator, we calculate this loss for 128 real images with $y_{true,\, i \ldots N} = 1$ and 128 images from the generator with $y_{true,\, i \ldots N}$=0 every epoch. For the generator, we calculate this loss for 128 'fake' images but use $y_{true,\, i \ldots N}$=1 every epoch, training the generator to trick the discriminator. We accomplish this by creating a combined generator and discriminator model which will only update weights for the generator. In this way, the discriminator is only updated based on its own loss function, but the generator can be trained based on the inverted loss function using the discriminator with up-to-date weights. Each model uses the Adam optimizer [5] with a learning rate of 0.0002.

3 Results

As suggested in Sec. 2.2, having a specialized discriminator employing dropout, batch normalization, and advanced activation functions and optimizers allowed for the model to keep up with improvements in the generator, thereby allowing for improvements in the generator even at a high number of epochs. In Fig. 2, we give example images generated from different random vectors at 400, 800, 3,000, and 9,000 epochs, which display recognizable figures at relatively early epochs and clear faces as training progresses.

4 Discussion

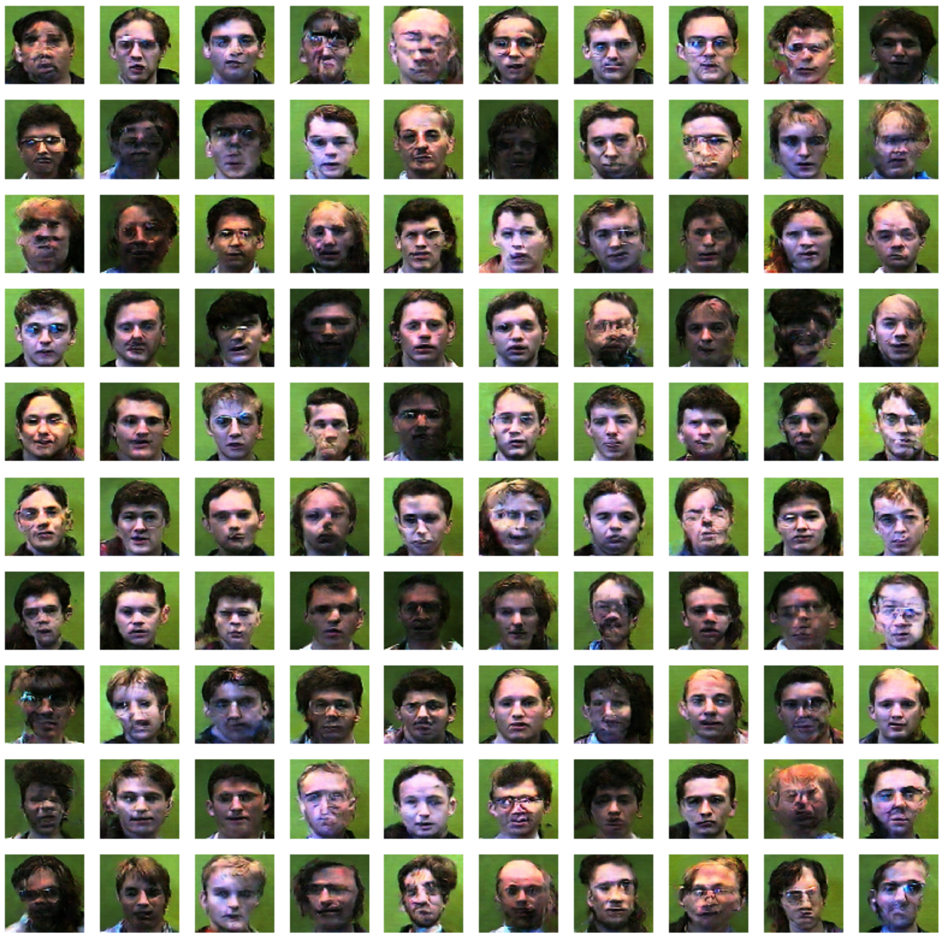

Figure 3: The sample of 100 images made by the generator from 100 random vectors after 10,000 training epochs which we used to select features from.

4.1 The Feature Sample

After training the network over 10,000 epochs, we provided 100 random numbers to the generator which produced 100 artificial images resembling faces to serve as a sample for use in vector manipulation. From these, we visually identified various categories including ‘people with hair’, ‘bald people’, and ‘people wearing glasses.’

4.2 Feature representations

From thee sample in Fig. 3 we identified 3 ‘people with hair’, 3 ‘bald people’, and 3 ‘people wearing glasses’. These are displayed in Fig. 4, Fig. 5, and Fig. 6, respectively, along with averages of each category. In order to understand their vector representation, we first calculated the distances between vectors of the same category and compared

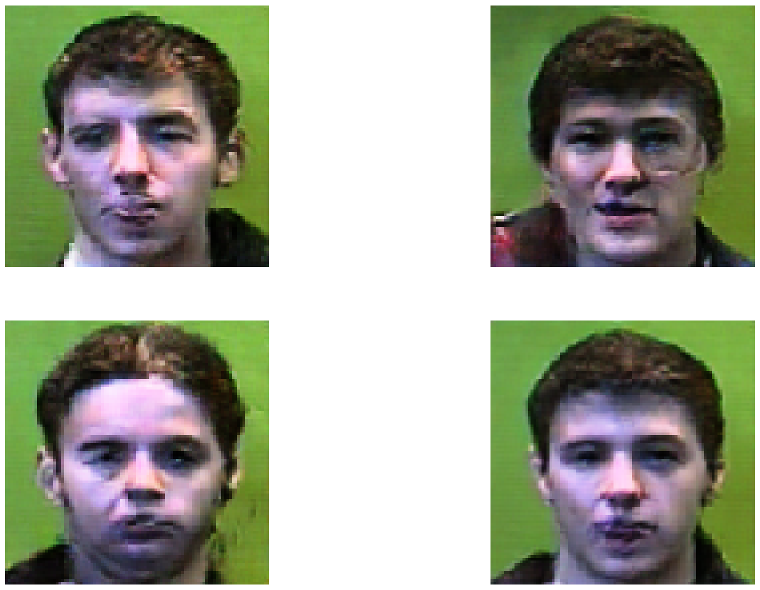

Figure 4: Three artificially generated faces with hair identified from Fig. 3. The bottom right picture is the average of the three selected images.

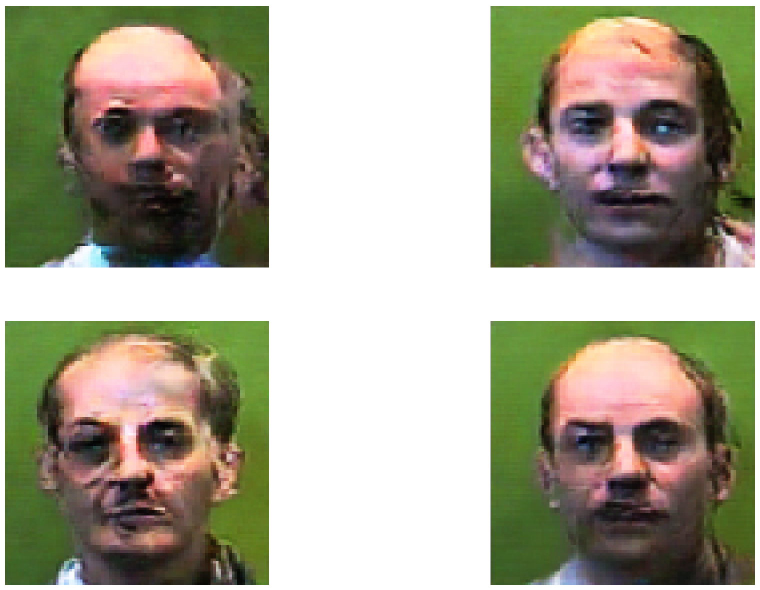

Figure 5: Three artificially generated faces of bald people identified from Fig. 3. The bottom right picture is the average of the three selected images.

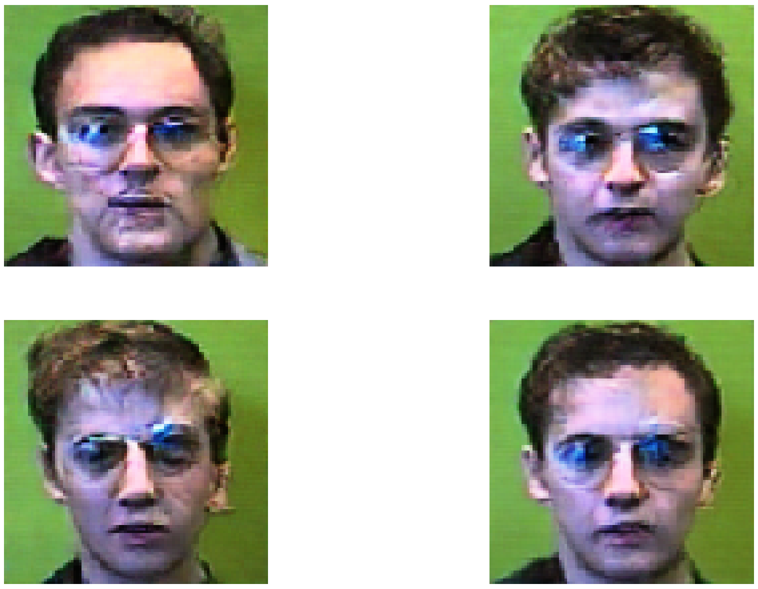

Figure 6: Three artificially generated faces with glasses identified from Fig. 3. The bottom right picture is the average of the three selected images.

them to distances from vectors of the other categories, where distance is calculated as

$$r_{1,2} = \sqrt{\sum_{i=1}^{N=100} (x_{1,\,i} - x_{2,\,i})^2}. \tag{3}$$

We found that the three bald people were separated from each other by distances of ∼13.4, ∼13.0, and ∼14.4, an average of 13.6, whereas the three images were separated from the 6 images selected from people with hair and people with glasses by an average distance of 13.7. While it is initially surprising that images from Fig. 5 are about as close to each other in vector space as they are to images from Fig. 4 and Fig. 6, it is important to consider that vector representations include 100 dimensions, while we are only considering a small number of features. In certain dimensions that map to specific features the distance might be great, but this does not necessarily mean the whole vectors will be separated by great distances; classifying the vector space is not as straightforward as one might initially guess.

4.3 Manipulations within the Latent Vector Space

Through simple vector math, we posit that we can extract vector representations of features. In order to obtain a vector representation of hair, we subtract the mean of the images in Fig. 5 of people without hair from images of people with hair from Fig.4, i.e. $x_{hair} = x_{avg.\,person\,with\,hair} - x_{avg.\,person\,without\,hair}$. Similarly, in order to obtain a vector representation of glasses, we subtract the mean of the images in Fig. 4 of people with hair from images of people with glasses from Fig.6, i.e. $x_{glasses} = x_{avg.\,person\,with\,glasses} - x_{avg.\,person\,with\,hair}$.

We selected one artificially generated bald person on the bottom left of Fig.5 and one generated person with hair on the top right of Fig.4 to serve as test subjects for feature manipulation. In Fig.7, we added the hair vector to the bald test subject. In addition, we included 3 other images in which we weighted the original result by 0.75 and added a vector of noise weighted by 0.25. In Fig. 8, we add the glass vector to the bald test subject and again add 3 additional images with an 0.25 factor of noise. In Fig. 9, we follow the same procedure to add glasses to our haired test subject.

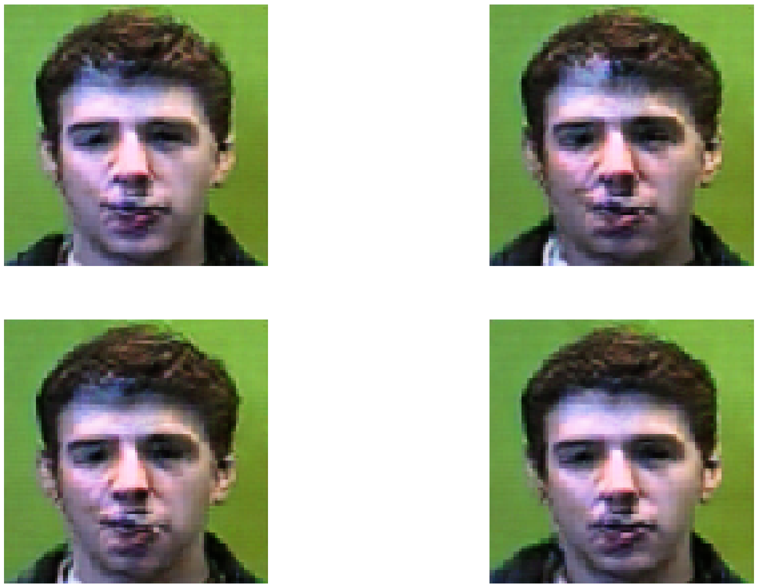

Figure 7: Here we add the 'hair' vector (calculated as described in Sec. 4.3) to the third bald face, which is located at bottom left of Fig. 5. We also include 3 other images (top right, bottom left, and bottom right) which are 25% random noise.

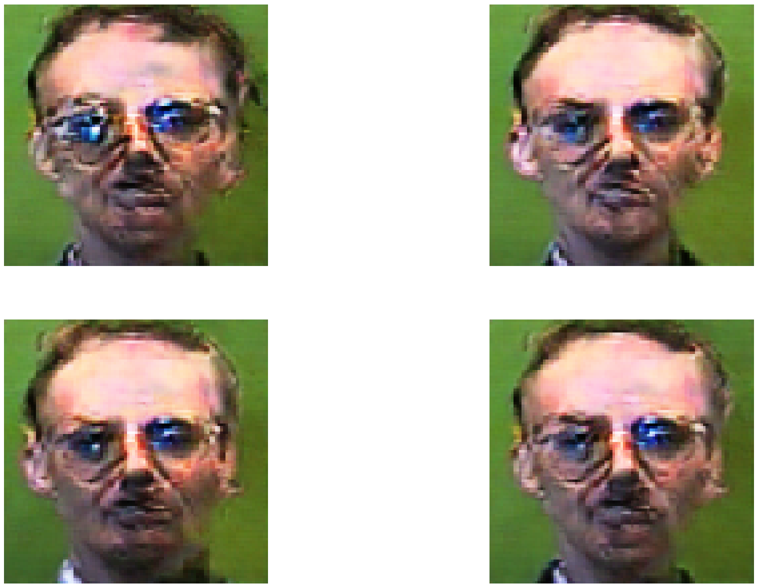

Figure 8: Here we add the 'glasses' vector (calculated as described in Sec. 4.3) to the third bald face, which is located at bottom left of Fig. 5. We also include 3 other images (top right, bottom left, and bottom right) which are 25% random noise.

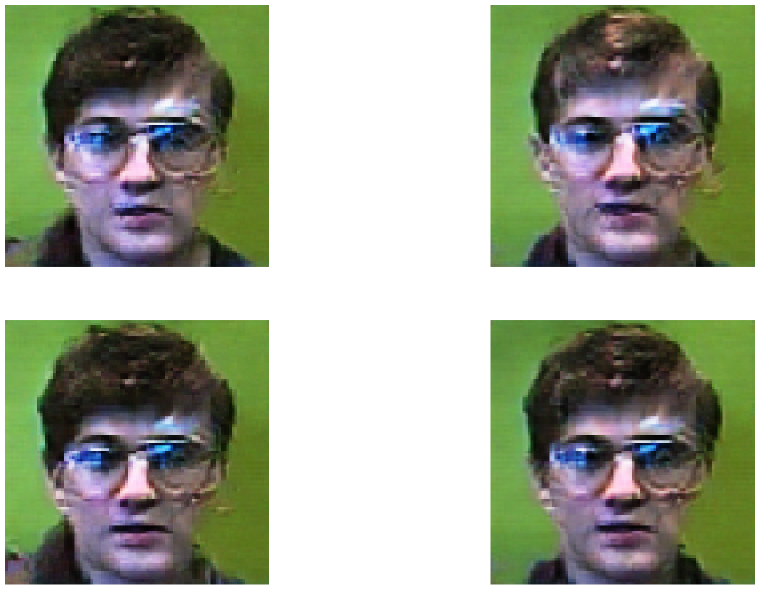

Figure 9: Here we add the 'glasses' vector (calculated as described in Sec. 4.3) to the second face with hair, which is located at the top right of Fig. 4. We also include 3 other images (top right, bottom left, and bottom right) which are 25% random noise.

Our results in Fig. 8 and Fig. 9 turned out lovely. Our test subjects now have glasses and presumably better eyesight. However, Fig.7 looks more like the average image of people with hair rather than our test subject having been gifted hair. This is quite a disappointing result, since we already reviewed in class an example of giving a generated image of a woman glasses. Thus, in Fig. 10 we explore adding additional weights (factors of 2, 3, and 4) to our test subject compared to the hair vector, i.e. $y = x_{hair} + 2x_{specific\,bald\,person}$, $y = x_{hair} + 3x_{specific\,bald\,person}$, and $y = x_{hair} + 4x_{specific\,bald\,person}$. The result is fantastic! Both the images weighted by factors of 2 and 3 are exactly as one would imagine adding hair to the test subject, shown in the top left of the image, might look.

5 Conclusions

Here we have shown how a generative network can learn to produce artificial images from random noise by mapping regions of vector space to facial features. After pitting a discriminative model against a generative model in a minmax optimization game, we found that our generator was quickly able to make recognizable figures and, after several thousand epochs, ones that closely resembled faces. By isolating features such as glasses and hair through latent vector subtraction, we were able to show how such features manifested in pictures of faces. In particular, we find that finely tuned weighting can produce convincing images of a bald person who has grown hair, most convincingly demonstrated in Fig. 10. Alone, the idea that computers can create novel images of faces is marvelous; the results of manipulation of facial features through vector arithmetic is doubly so. It is no wonder generative adversarial networks have been dubbed "the coolest idea in machine learning in the last twenty years."[2]

[2]LeCun, Yann, RL Seminar: The Next Frontier in AI: Unsupervised Learning

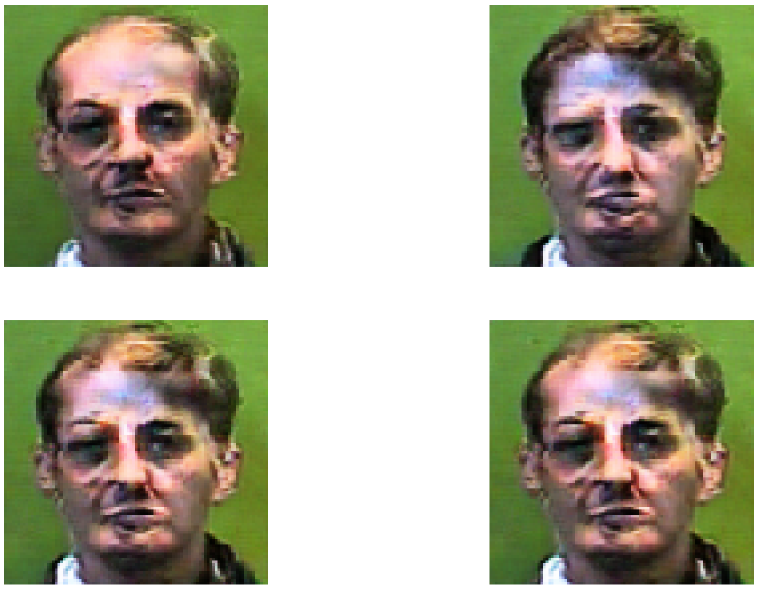

Figure 10: A demonstration of an improved technique to add hair to a bald person. The top left image is the specific generated bald image we have selected to manipulate. In the top right image, we add the hair vector (calculated as described in Sec. 4.3) to the top left image weighted by a factor of two ($y = x_{hair} + 2x_{specific\,bald\,person}$). In the bottom left, we weight the bald image by a factor of 3, and in the bottom right image we weight it by a factor of 4.

References

[1] Goodfellow, I. et al. Generative Adversarial Nets. NIPS 2014.

[2] Maas, A. et al. Rectifier Nonlinearities Improve Neural Network Acoustic Models Overfitting. WDLASL 2013.

[3] Srivastava, N. et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR 2014.

[4] Ioffe, S. et al. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. JMLR 2014.

[5] Kingma, D. et al. Adam: A Method for Stochastic Optimization. ICLR 2015.