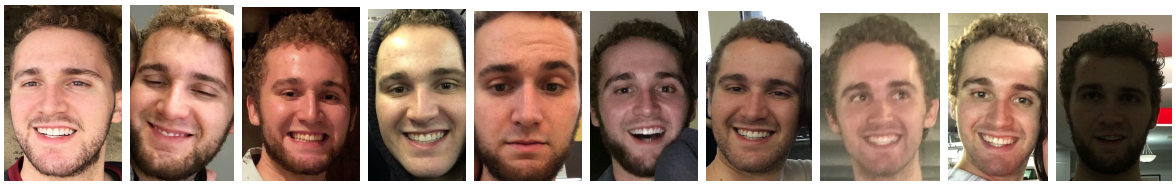
Face Swapping with an Autoencoder Network

SMU Lyle School of Engineering
CS8321 - Machine Learning and Neural Networks
Will Lacey - 45906124



ABSTRACT

Face swapping in images is a technique where the input of one face image is captured and placed onto another target face image. In the context of this swap, the features such as expression, face orientation or pose, and lighting are all transposed onto the new target image. This approach is very similar to the problem of style transfer where the objective is to transfer the style of one face to that of another. In this paper, I discuss my interest and desire to create a face swapping autoencoder as well as actually discussing how I design and showcase an autoencoder network that attempts face swapping in a low resolution. Due to time constraints and gpu resources, the merit of the output of the autoencoder network should be not judged solely on the effectiveness or quality of the results. Instead, consider the output to be a proof of concept of how a decently trained autoencoder, with an ample supply of training data, could perform.

1. INTRODUCTION

I decided to create a face swapping autoencoder because I am very familiar with an autoencoder network as well as I am interested in face swapping algorithms. I have always wanted to do face swap ever since I was first introduced to the concept through Snapchat and playing around with the app with friends. However, the level of expertise displayed through a product such as Snapchat is nothing this paper and product wishes to capture. Instead, I want to merely display the outcome and product that can be accomplished through a short amount of time and limitation on GPU resources.

In the realm of face swapping algorithms, autoencoder networks find themselves down low on the totem pole. Today, we find that Deep Learning methods absolutely stand out among the best algorithms in the face swapping problem. While an autoencoder certainly struggles to compete against modern techniques, it does however do well in interpolating multiple poses and orientations into new target faces.

In the rest of this paper, I will explore the methodologies I tested and results found through designing a face swapping autoencoder.

2. BACKGROUND

The autoencoder is a formulation of two neural networks that have the output of one fed into the input of the next. In this model, the first network's objective is to encode an input vector into some sort of meaningful information that the second network can interpret and decode. The output vector that is yielded by the second network should resemble the input vector that went into the first network. To elaborate, the first network, the encoder, must learn how to effectively reduce the input vector down in size while still retaining important features and data that the second network, the decoder, must learn to reconstruct to a copy of the input. Thus, the autoencoder network can be represented by a function consisting of two parts: an encoder represented by **h = f(x)** and a decoder **r = g(h)**, which generates the reconstruction of the input. This can be seen below in Figure 1 and elaborated upon when shown in Figure 2.

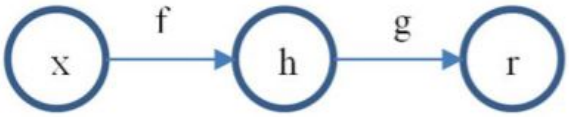


Figure 1. Flow diagram for autoencoder input.

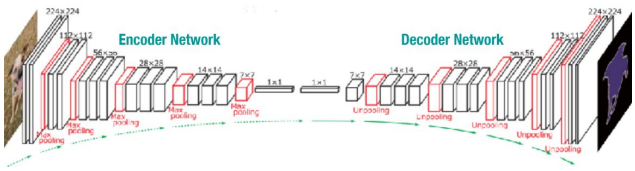


Figure 2. The architecture of the encoder and decoder network. [2]

The autoencoder is essentially a generative model which decodes an input through restoration by the hidden layer information [1].
The encoder, decoder network can be compared to a standard neural network because when training, the same technique seen in a feedforward network is used to perform backpropagation learning. Given a loss function and through using the small batch gradient descent method, the autoencoder allows for the weight values in the model to be updated. In this vein, "the autoencoder takes the input images through the network to obtain implicit information, and the decoder is responsible for decoding and producing the implicit information to restore the input images" [1]. This can be seen in Figure 3.

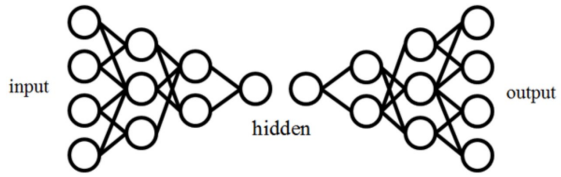


Figure 3. Network structure of input encoder and output decoder.

Like a convolutional network or a multilayer perceptron, an autoencoder framework must match the complexity of the input image task. If the size of the model does not scale properly with the input vector, then the model will fail to understand the overall distribution of the input data, and thus the autoencoder will not learn. As a result, the model must not have an encoder that is too large or too small and the optimal size must be ascertained given the input data.

3. METHOD

3.1. Autoencoder General Algorithm

The autoencoder can be used for a variety of purposes but excels at transfering the style of an input image to the content of a target output image. Again, this is where face swapping can come in as the technique for autoencoder face swapping can be effectively thought of as the style of one face being transferred to the content of another. In a face swapping autoencoder, the encoder is trained on general face images while the decoder is trained to learn the style of one particular face or person. This means that the input vector for the encoder serves as the content image to be stylized. This can be illustrated through Figure 4 and Figure 5, which are seen below.

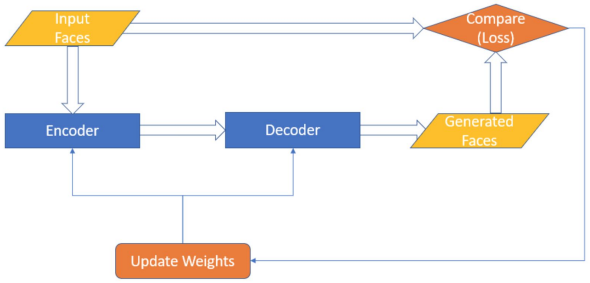


Figure 4. The autoencoder training algorithm with face images [3].

Figure 5. Face swap algorithm [3].

In Figure 4, we can see how training works for a face swapping autoencoder. Again, the encoder here is trained on a variety of face images, not focusing on one person's face in particular. The theory behind this practice is that an encoder trained on general face images will learn the features and structure behind human faces and thus be able to properly downsample the input image into a meaningful array of information that will later be able to be interpreted by the decoder. During training, the encoder should be first trained to identify people's faces in an image before it is trained in conjunction with the decoder. This way we can avoid training both the encoder and decoder simultaneously when both are in their initial stages of training outputting complete junk.
After the encoder is trained or partially trained, the decoder, as seen in Figure 5, is then trained in conjunction with the encoder. Unlike the encoder, which is trained on general face data, the decoder can only be trained on one particular face, thus for every face we would like to effectively stylize, we need to build a completely new decoder. Because we need to retrain a decoder to learn the structure for a new face to face swap onto, the autoencoder method really struggles against modern face swapping techniques.
In Figure 5, we can see a clean cut data flow where we have an input image A passed through our encoder network. After this step, we then take the decoder network trained entirely on B face images and pass the encoder output yielded from image A into it. This generates a completely new image that should have the same pose, lighting, and expression as person A had, but with person B's face.

3.2. Autoencoder Implementation

For my autoencoder, I used the VGG19 pretrained weights, generated from Francios Chollet's source code [4], to initialize my encoder. I did this because VGG19 has captured a lot of meaningful data trained from thousands of content images. Since I was low on time, I didn't want to train my own encoder and because VGG19 had been trained on face and human data, I knew it wouldn't be all too bad to work with. If I wanted to repeat this implementation of an autoencoder face swap, I would

absolutely want to train an encoder off of purely face images.

For my decoder, I entirely used my own face, since it was the most accessible to me. In conjunction, I trained my decoder with an Adam optimizer using a learning rate of 3e-5 and betas of 0.4 and 0.999. These were the variables I chose as recommended by Yanzong Guo et al. in his paper [1].

My training data consisted of 449 face images where each image consisted of a resolution of 64 by 64 pixels using RGB channels. I trained for 5000 epochs for my decoders using a batch size of 64 and a 6000 epochs for the ones with a batch size of 32. I did this to explore two different batch size parameters during the small window of time I had. Each training window took about a day to completely train, which felt like an absolute eternity.

A step that I skipped for the sake of simplicity was cropping each of my input face images to increase training convergence on meaningful face data. I did this to save some time programming and creating face swap images. If I were to do this again, I would absolutely use several thousand face input images to train my decoder and also create a face tracking auto crop to ensure better training convergence.

4. EVALUATION

I created five decoder networks that each stemmed from the original 449 images of my own face. Each used the encoded embeddings from the VGG19 network and each had varying degrees of success. Below is a concatenated list of output images generated through the autoencoder network.

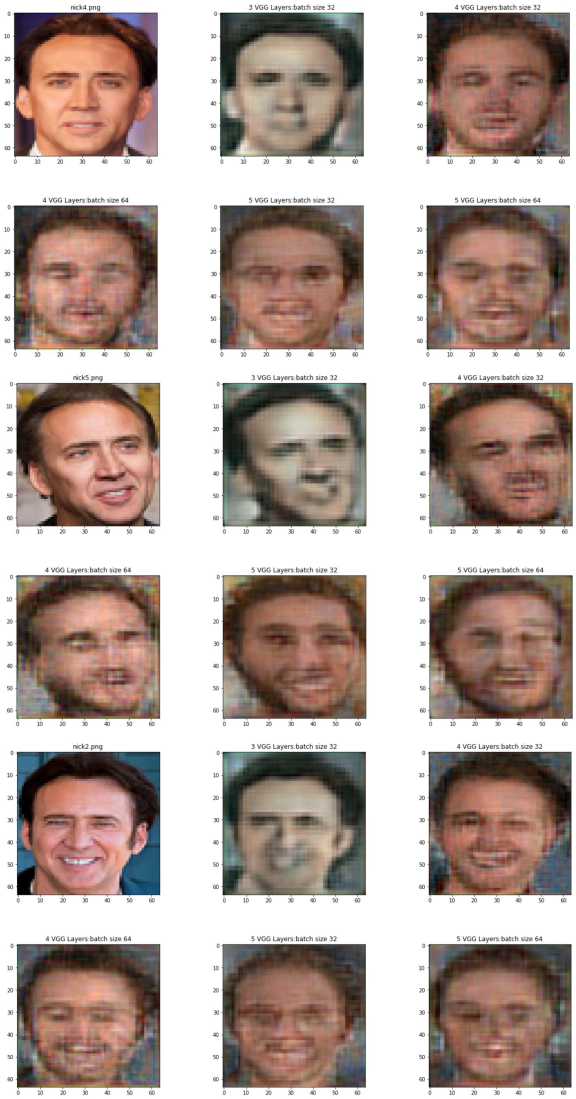


Figure 6. Nicolas Cage's Pose to Will Lacey's Face

Here we see that the top middle decoder was too shallow of a network to properly perform the face swap algorithm. This decoder would not really benefit from training longer, but instead suffers from the problem mentioned before of being insufficiently complex.

The two decoders in the bottom middle and bottom right both suffer from similar problems, neither of these models were sufficiently trained and as a result continuously fail to properly display the input image as a new style.

The remaining two decoders do a decent job of transfering the content image, but would most likely benefit from using a face encoder rather than a VGG19 encoder. Either way, you can definitely see with limited training and limited resources that the face swapping algorithm worked, in the very least, as a proof of concept.

These next images showcase some problems where the input image is less similar to the decoder training set in terms of face shape and gender.

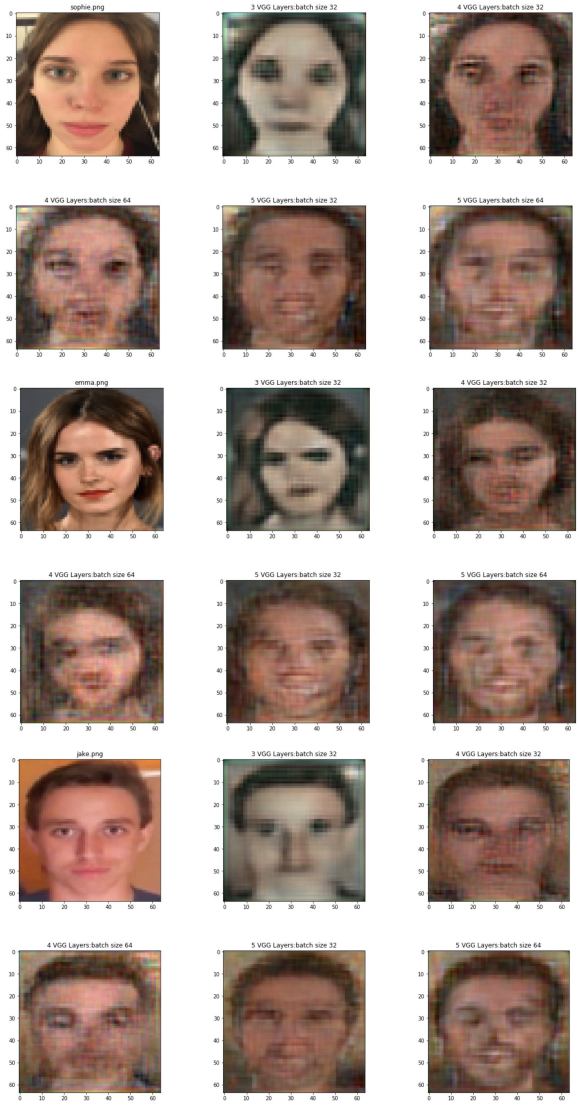


Figure 7. Other Faces' Pose to Will Lacey's Face

These faces are definitely horrible although their results are still not quite complete failures. We can still see glimmers of my own face in the output.

5. CONCLUSION

The face swapping autoencoder was a fun introduction to face swapping algorithms. I definitely hope to explore more in this vein as my understanding of deep learning methods continue. If I were to do this implementation again, I would definitely use a custom built face encoder to capture different races and genders in my face swap algorithm as well as train for much longer.

6. REFERENCES

[1] Yanzong Guo, Wangpeng He, Juanjuan Zhu, Cheng, Li. A Light Autoencoder Networks for Face Swapping. In CSAI '18: Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence. Dec. 2018.

[2] Eric Larson. Lecture Notes for Neural Networks and Machine Learning: CNN Visualization and Fully Convolutional Learning. At Southern Methodist University. Spring. 2020.

[3] "Faceswap." [Guide] Training in Faceswap - Faceswap Forum - The Place to Discuss Faceswap and Deepfakes, 29 Sept. 2019, forum.faceswap.dev/viewtopic.php?t=146.

[4] Francois Chollet. Deep Learning Models. From Github. Sept. 2017. https://github.com/fchollet/deep-learning-models

[5] Justin Ledford. Universal Style Transfer Keras. From Github. Apr. 2018. https://github.com/8000net/universal-style-transfer-keras

[6]