Recursively Trained Autoencoders

William Ray
University of Massachusetts Amherst
Amherst, MA, 01003
wray@umass.edu

Daniel Hannon
University of Massachusetts Amherst
Amherst, MA, 01003
drhannon@umass.edu

February 26, 2022

Abstract

Our Project improves upon the quality and clarity of images created by autoencoders by training them in a novel recursive manner. The model is built out of smaller stacked models that share layers. The clarity of these images was also slightly improved using batch normalization, where batch normalization layers were shared between corresponding encoder and decoder layers. These training methods produced models that recreated clearer images with a slightly lower reconstruction error than an autoencoder trained end to end with the same architecture but failed to produce new face images when used on a variational autoencoder.

1 Introduction

Our Project attempts to improve upon the quality and clarity of images created by autoencoders. From work done in GANS [1], we can see that it is possible to create crisp detailed images entirely from deep learning models. The problem we are addressing is that current autoencoders produce blurry images and lose the original image's detail. This blurriness is caused by the autoencoder failing to reconstruct details based on encoded information from the original image. In the learning process, the loss function incentivizes the model to create outputs that minimize reconstruction loss and not necessarily detailed or crisp output. Our approach to solving this problem was to train autoencoders in a new manner that we describe as "recursively." The loss would be calculated at each intermediate layer to recover better the data lost in the encoding process. We hoped that training a model this way would encourage models to learn how to recreate precise detail at higher layers while still encoding the data into a low-level latent space. We used the Labeled Faces in the Wild database [2]. We have chosen this dataset because faces lend themselves to autoencoders as they have very similar structures while having distinct features. We expected the generated faces in the dataset to be crisper and have more fine detail than the faces generated by a stacked autoencoder. To evaluate the models' performance, we compared the output between a stacked model and the recursive model when passed in the same face to reconstruct. In the case of variational autoencoders, a qualitative metric evaluated faces generated by sampling a latent vector. Images were compared and evaluated on how clear and detailed the image was and how much it resembles a face. The models were also compared on reconstruction error, a mathematical measure of how similar the input image is to the output image when performing image reconstruction.

2 Background/Related Work

Autoencoders are models used to learn dense representations of given inputs through repeated dimensionality reduction. They are unsupervised models that perform the task of learning to both encode data into lower-dimensional space and decode the encoded data into its original size.

Stacked Autoencoders are autoencoders that use multi-

ple hidden layers in the encoder and decoder. In the encoder, information is passed from one layer to the next, gradually being reduced until a maximum reduction level is reached. Figure 1 shows the structure of a stacked autoencoder. When the data is in this reduced state, it is in the coding space. Then the decoder takes this reduced data and gradually expands it until it has the same size as the input. Stacked autoencoders can use a variety of different activation functions or layer types in their architecture. The autoencoders we use are convolutional. Convolutional autoencoders use convolutional layers to reduce dimensions in the encoders and to restore dimensions in the decoder. Convolutional layers are much better at processing images compared to dense layers [1]

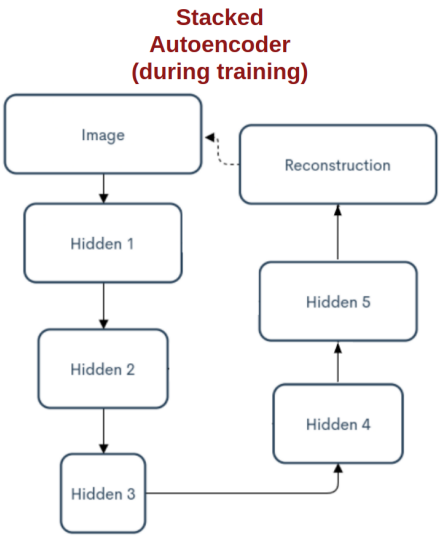

Figure 1: This is the structure of a traditional stacked autoencoder with 5 hidden layers. Boxes represent input and output tensors. Solid arrows represent convolutional layers. On the left, the first two hidden layers show the encoder, which encodes the image into an embedding layer. On the right, the last two hidden layers show the decoder, which decodes the embedding layer into a reconstruction of the original image.

Variational autoencoders are probabilistic, meaning their output is somewhat determined by chance. This type of autoencoder can be used to generate new data points or, in our case, new pictures of faces. Instead of the encoder of a variational autoencoder producing codings directly, two vectors that describe a multivariate normal distribution. One vector describes the means of the normal distributions, while the other vector describes the standard deviations. The codings are then sampled from this multivariate normal distribution and passed to the decoder. Variational autoencoders have an additional loss function in addition to their reconstruction loss. These models are penalized for producing distributions that deviate too much from a zero mean and a standard deviation of one. This penalty encourages the model to keep its coding space in a standard normally distributed region of the coding space. This allows us to randomly generate a normally distributed vector and pass it to the decoder, which will then map our vector to a data point that has similar properties to the data point in our data. If we trained our variational autoencoder on face images, the decoder would roughly produce an image of a face. [1]

Batch normalization reduces the risk of vanishing and exploding gradients.It does this by finding the mean and standard deviation of a batch and then zero-centering and normalizing the batch. Next, the batch normalization layer learns an optimal shift and scale for the data. Batch normalization is used heavily in GANs, another system for generating images. [3]

3 Approach

Our approach to solving the problem of unclear images was to train an autoencoder in a ”recursive” way. Instead of training the autoencoder as one model with one input (the image) and one output (the reconstruction), we train several models that share the same layers. Each layer in the decoder is an output layer that attempts to learn and recreate the output of the equivalent layer in the encoder. For example, in Figure 2, we would train three models. The first model would have layers: [Image, Hidden 1, Reconstruction]. The image is frozen so that it cannot be changed, and the ’Reconstruction’ layer attempts to recreate the image, given the output of ’Hidden 1’. Similarly, the second model would have layers: [Image, Hidden 1, Hidden 2, Hidden 5]. Again, the ’image’ would be frozen, as well as ’Hidden 1.’ ’Hidden 5’ attempts to recreate the output of’ Hidden 1’ given the output of’ Hidden 2’. The

third model follows the same pattern of freezing weights in the encoder layers that correspond to the higher decoder layers, while the decoder attempts to recreate the output of 'Hidden 2', given the output of 'Hidden 3'.

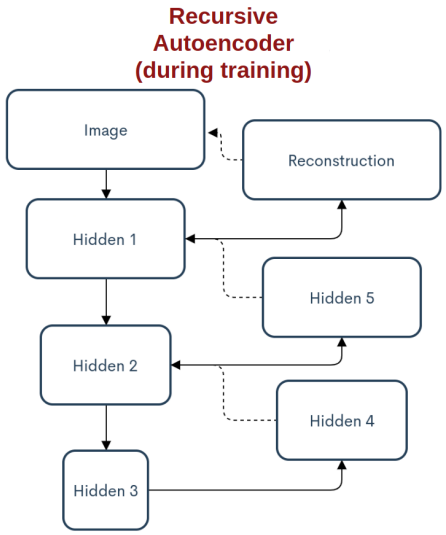

Figure 2: This is the structure of a recursively trained autoencoder with five hidden layers. On the left is the encoder, and on the right is the decoder. Boxes represent input and output tensors. Solid arrows represent convolutional layers. Dotted arrows represent mean squared error comparisons. For example, The Reconstruction tensor is compared to the Image tensor, and there is a loss proportional to the squared difference between them.

We hypothesized that this training method would produce crisper images for two reasons. First, we believed that with each decoder layer attempting to reproduce its corresponding layer's input, the decoder layers at shallower levels would be more incentivized to learn detailed feature reconstruction for things like eyes, ears, or hair, rather than high-level features like shape or color. In turn, this would recover more of the fine details lost in the encoding process, helping to improve image clarity. We expected this different training system to improve the clarity of produced images because the encoders at each layer would learn encodings that benefit the corresponding decoder layer's feature decoding task.

In a conventional stacked autoencoder, the decoder's last layer wants to minimize the pixel difference between its output and the input image. During training, this final layer learned that its representation of the image has been highly compressed and lacks information. The layer produces a blurry image to hedge its bets and minimize the loss to compensate for this. The last layer in the recursively trained model was never trained on inputs that came all the way from the bottom of the autoencoder, so it assumes there is a lot more information in its input than may actually exist. Because the layer does not assume missing information, it does not produce a blurry image in an attempt to hedge its bets.

We expected that when an autoencoder trained in this new recursive way reconstructed images of people's faces, the reconstructions may look less like the original person; however, we expected the reconstruction to look crisper and more clear as an image of a person's face. Every autoencoder loses information, so our approach attempts to trade higher-level information for lower-level information to achieve higher quality images. We also expected variational autoencoders trained recursively to produce more believable faces that were not as blurry as faces produced by a stacked variational autoencoder.

4 Experiment

4.1 Dataset Statistics

We used the Labeled Faces in the Wild dataset for our experiments, which contained a collection of 13,233, 250 x 250 x 3 colored images of different people's faces. The faces vary in age, race, ethnicity, lighting, and angle. We used this dataset as it presents a robust set of faces that all have wildly varying details and shapes yet are all considered faces.

To preprocess, we cropped them to the center 128 pixels. All the image values were divided by 255, so every value was mapped to one between 0 and 1. Our model took in images of size 128 x 128 x 3, with values between 0 and 1 as input, and produced images of size 128 x 128 x 3, with values between 0 and 1 as output.

4.2 Basic Recursive Model

The first experiment conducted was to build the recursive model from scratch and compare it to a conventional, stacked autoencoder for facial reconstruction. We used a standard structure for these models. The models were comprised of fourteen layers, seven in the encoder and seven in the decoder. All of the encoder layers were convolutional with a stride of two, "same" padding, a kernel size of 3, and had SELU activation. All decoder layers were convolutional transpose layers with the same parameters as the encoder layers, except for the first encoder layer and the last decoder layer, which had a kernel size of 5. The last decoder layer used the sigmoid activation function instead of SeLU to produce an image with pixel values between 0 and 1. The number of filters is doubled after every encoder layer and halved after every decoder layer. We implemented all models in Keras [4].

The faces produced from the recursively trained version of this model appeared to be clearer and less blurry than the images produced by the stacked autoencoder, but with a bit of extra graininess. Unlike our hypothesis, the recursive autoencoder appears to do at least as good a job reconstructing the original face as the stacked autoencoder with a slightly better reconstruction error as well. The outputs of all the models' facial reconstructions can be seen in Figure 5, and the average reconstruction loss of all the models can be seen in Figure 6. We conducted further experiments to attempt to improve the clarity of the image reconstructions further. We tested the addition of batch normalization and experimented with a ReLU activation function between layers.

4.3 Batch Normalization

The first experiment included a standard batch normalization. We created a standard batch normalization layer, as seen in [1][5]. We added these layers between all the layers in the encoder and decoder. We found that the images produced by this method of batch normalization left images completely washed out and hardly recognizable as if the faces were covered in frost. The stacked model with this method of batch normalization produced faces that were still blurry and more generic but were still recognizable as faces. Both of these models had an increased reconstruction error compared to their unnormalized counterparts.

Given the experiment results, we decided that the best course of action for training a recursive autoencoder with batch normalization was to share the Batch Normalization layer between the respective layers of the encoder and decoder. The structure of a model trained recursively with shared batch normalization layers can be seen in Figure 3. This method of sharing the normalization parameters between the encoding layers and decoding layers proved to be more effective than the traditional batch normalization when paired with the recursive training structure. The faces reconstructed with this model had a higher level of clarity than the stacked autoencoder and were not as grainy as the faces reconstructed by the recursive autoencoder without batch normalization. However, the images have a higher reconstruction error than both the non-batch normalized stacked and recursive models.

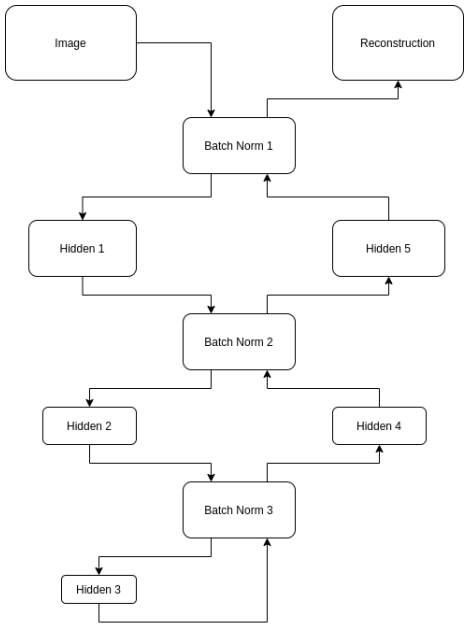

Figure 3: The structure of a recursive model with shared batch normalization. On the left are the encoder layers, and on the right are the decoder layers. In the middle are the batch normalization layers, which are shared and used by respective layers in the encoder and decoder.

4.4 Activation Functions

Based on our readings on stacked convolutional autoencoders in [1][6], we decided to use the SELU activation function for our models. The main apparent advantage of the SELU over the standard ReLU activation function was that SELU was self normalizing [1][5]. After our success with the shared batch normalization model, we attempted to improve upon its high reconstruction error by using a standard ReLU activation function instead of SELU because we believed we no longer needed the self normalization as we were using batch normalization. The model created was a recursively trained model with shared batch normalization and a ReLU activation function. The model proved to have an incredibly high reconstruction error, and the images reconstructed by the recursively trained model were not very face-like and had a grid pattern. The ReLU activation did not work, and SELU activations were used in all experiments moving forward.

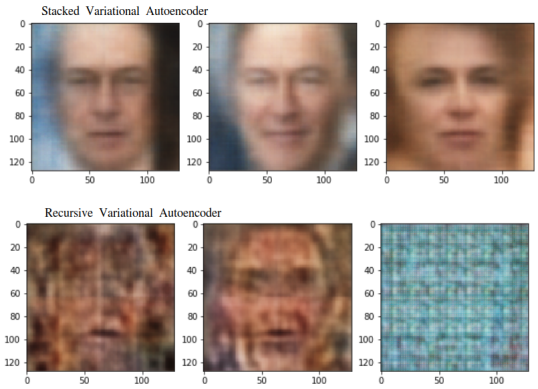

Figure 4: Figure 4: Three faces produced by the different variational autoencoders. On top are the stacked variational autoencoder outputs, while on the bottom are the recursive variational autoencoder outputs.

4.5 Variational Autoencoders

In the last experiment, we trained a variational autoencoder recursively. We hypothesized that because the recursive training style proved beneficial to facial reconstruction, it could also improve the quality of faces produced by a variational autoencoder. We trained a variational autoencoder without the shared batch normalization in both a stacked and recursive fashion. As expected, the stacked autoencoder generated images that were blurry and lacked fine detail. The faces produced from random points in the recursive autoencoder's latent space did not look like human faces at all. Most points just produced vaguely human-colored noise, while others produced faces that looked like monsters. Some example faces produced by each variational autoencoder can be seen in Figure 4.

4.6 Results

The model that reproduced images with the smallest reconstruction error was the recursively trained autoencoder using SELU and no batch normalization. This model was closely followed by the stacked autoencoder using SELU and no batch normalization. The other models had a significantly higher reconstruction error with the recursively trained model using ReLU and the recursively trained model with standard batch normalization having the highest. All of the reconstruction errors can be seen in table 1.

Model Type	Reconstruction error
Recursive With Batch Norm	7831
Stacked With Batch Norm	467
Stacked No Batch Norm	258
Recursive No Batch Norm	239
Stacked Shared Batch Norm	643
Recursive Shared Batch Norm	455
Stacked Shared BN with ReLU	643
Recursive Shared BN with ReLU	2020
Recursive Shared Batch Norm	455

Table 1: Reconstruction error of evaluated models. Sum of the mean squared error between input image and reconstruction over averaged 100 input images.

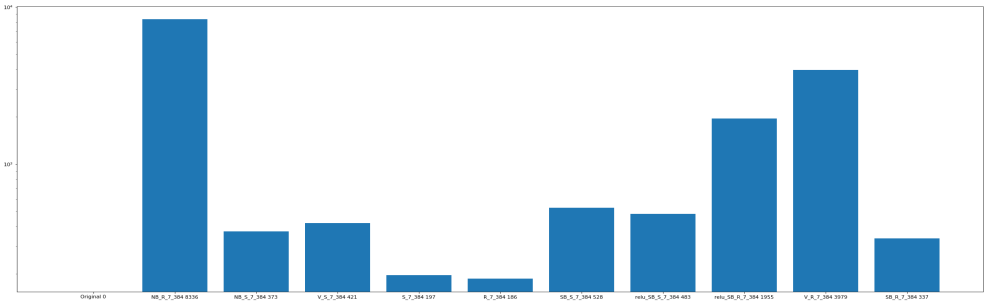

Figure 5: The average reconstruction error of each model across 100 faces. From left to right, recursive non-shared batch norm, stacked non-shared batch norm, no batch norm stacked, no batch norm stacked, shared batch norm stacked, shared batch norm stacked with ReLU activation, shared batch norm recursive with ReLU activation, shared batch norm recursive.

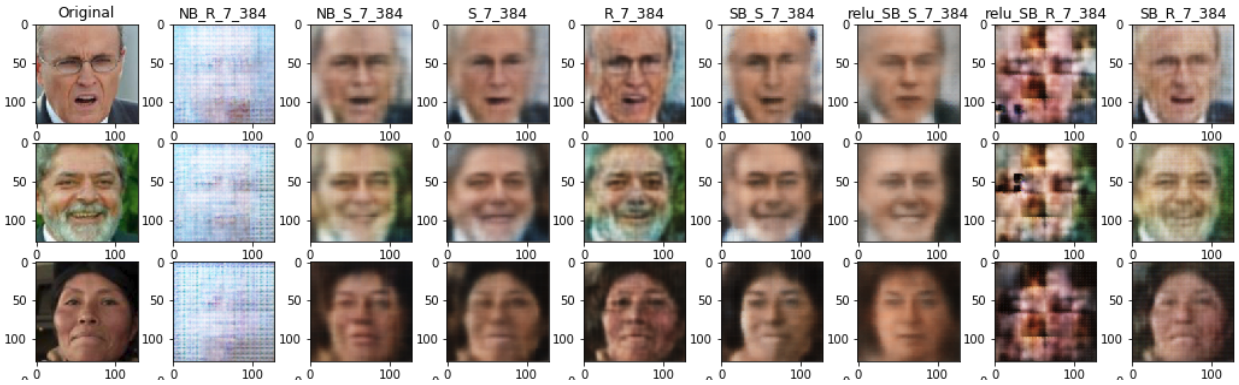

Figure 6: The reconstruction of three given faces across all of the models. From left to right, recursive non-shared batch norm, stacked non-shared batch norm, no batch norm stacked, no batch norm stacked, shared batch norm stacked, shared batch norm stacked with ReLU activation, shared batch norm recursive with ReLU activation, shared batch norm recursive.

The other metric to compare the quality of each model is the clarity of each image produced. We believe that the clearest facial reconstructions are from the recursively trained model with shared batch normalization and the recursively trained model with no batch normalization. All of the model's reconstruction of a few faces are given in figure 6.

5 Conclusion

We have learned that by recursively training autoencoders, we can improve the detail of faces reconstructed without impacting the reconstruction error. We also learned that models trained in this way tend to be more sensitive, and slight changes can have a much more significant impact than they do in autoencoders trained conventionally. This sensitivity to initial conditions can be seen in the recursive autoencoder's drastic reconstruction error with ReLU or standard batch normalization. Both of these slight changes completely ruined the output of the recursive model. We also learned that because of this sensitivity, the recursive training method is also not useful for variational autoencoders, as the faces produced from the embedding space are not very human at best, and at worst, complete noise.

There are several ways recursive autoencoder could be explored further. Perhaps upsampling layers could be used instead of the convolutional transpose layers in the decoder. Autoencoders have performed well with upsampling in other works [1]. Similarly, max-pooling layers could be used after every layer in the encoder, and the encoder layers could be modified to have a stride of 1.

These max-pooling layers would reshape the tensors in the same way and may allow for more efficient data flow through the network [1]. Another interesting modification that could be made would be instead of training a model for every layer in the decoder, only train one model with, but include a loss for every layer in the decoder. The loss would be proportional to the mean squared error between the decoder's layer output and the corresponding encoder layer output. The different losses could even have different weights allowing for the exploration of the idea that recursive training may be more useful near the model's terminals than the center of the model or vice versa. Finally, a more thorough hyperparameter search could be performed with the basic recursively trained model.

References

[1] A. Geron, *Hands-on machine learning with Scikit-Learn and TensorFlow*, 2nd ed., 2019.

[2] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007.

[3] A. R. et al., "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015.

[4] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[6] J. M. et al., "Stacked convolutional auto-encoders for hierarchical feature extraction," 2011.