Testing the Effectiveness of Image Resizing and Decomposition Techniques on Neural Network Image Classification

Jessica Edouard (jedouard), Nebil Ibrahim (nbi),
Nathan Lovett-Genovese (nathanl)

January 2020

Code: `https://github.com/NebilIbrahim/CompressionTesting/blob/master/CompressionTesting.ipynb`

1 Introduction

The robustness of compression models is often hard to quantify. Certain compression models work well for specific types of images and poorly on other. Aside from the actual compression ratio, there are not many metrics to quantitative analyze the quality of the compression.

With the advent of big data, the need to compress images for space efficiency has become more and more important. Therefore it is necessary to have compression models whose purpose is to retain important features for analysis and downstream machine learning tasks. Compressed images are not only useful for their space efficiency, but also for their ability to make machine learning tasks more accurate by only retaining the features important for reconstructing an image. Since a potential goal of compressing images is to make machine learning tasks easier, it naturally makes sense to use their performance in a machine learning model as an effective metric for the quality of their compression. This gives way to the idea that one way to analyze the effectiveness of the compressed images is with convolutional neural networks.

Using this metric could potentially both the task of a network being able to generalize from the training data it receives at test time and to space efficient

for task where data is used primarily for machine learning models.

The different compression models we analyzed are principal component analysis, autoencoder networks, and image resizing. Our goal with this project was to test the robustness of these compression models with a classification neural network.

2 Previous Work

Serving as the main motivation for this project was the creator of the YouTube channel "CodeParade". The problem they attempt to tackle is generating human faces through a neural network. Their approach was to input yearbook photos through an auto encoder and observe how well the network is able to reconstruct the original image. The original image they input into the encoder half of their network had $144 \times 192 \times 3$ dimensions, or 82,944.

The output of their encoder squished the image into just 80 dimensions. To make sure the information stored in each of the dimensions of the compressed face output is normally distributed CodeParade uses Principal Component Analysis [2]. We wanted to build upon this concept by testing the viability of these reconstructed images in further machine learning tasks.

Zeiler and Fergus detail in their paper a similar frustration with the lack of insight into convolutional networks despite various strides made in image classification made possible with new data sets, GPU implementations, and model regularization strategies. They propose, among other methods, a "sensitivity analysis" on the output of the classifier by occluding portions of input image, emphasizing the importance of certain portions of the images to the classifier's accuracy to better visualize network insight [3]. Through this method they are able to create a visualization of the feature map and show that specific areas of the image are much stronger features than others–such as the face of a dog in an image labeled as "Pomeranian".

Here, we take inspiration from Fergus and Zeiler and their focus on how different amounts of occlusion or, in our case, noise might affect our compression. Thus, this served as further inspiration for the use of our own convolutional networks within our autoencoders.

3 Design and Implementation

The overall approach to this problem was to try a series of methods that would "compress" an original image (I) to some smaller dimension and then "decompress" it back into a new image (I') that contains the same dimensions the original image had. Then, we would take this new image, I', and evaluate it on a pre-trained convolutional network. For all but one of the compression methods, we trained these methods two different ways:

1. With the original image as both input and true label. We would compress the original image and train to decompress and receive a new image, I' that essentially aims to be a replica of the original input.

2. With a noisy original image as the input and the original image as true label. We would take the original image and add noise that was created by sampling from a Gaussian distribution that has a mean of 0 and a standard deviation 0.2. Then, we would train our compression algorithm to aim to recover an original image despite the noise it received.

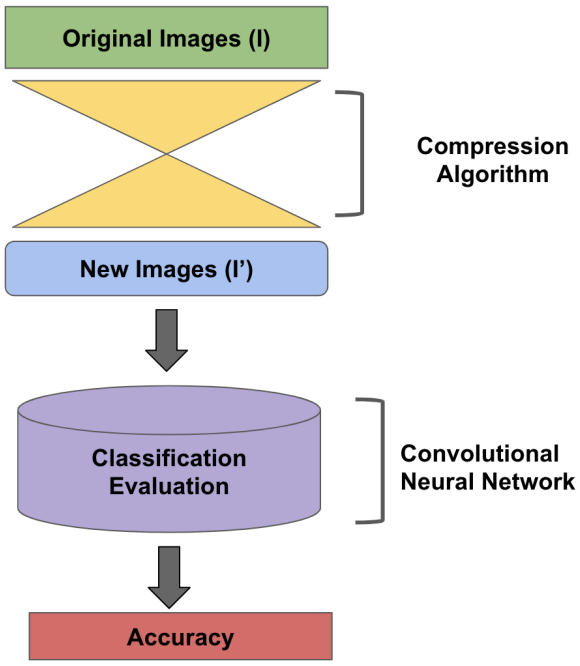

Figure 1: The pipeline from compression and decompression to evaluation

Figure 1 demonstrates a visual representation of this approach. The purpose of performing these compression algorithms and training them on different levels of Gaussian distributed noise is to receive an image that has some amount quality lost due to the compression and decompressions done. The last step of this process will then use these new I's and input them into a pre-trained convolutional neural network that has the ability to classify objects into various categories. Therefore, the result of this process will create different pipelines that have compression algorithms trained for different levels of noise.

From here, we are capable of evaluating how the pre-trained model did with the original images it classified versus the newly decompressed images it attempts to evaluate afterwards. At testing time, we then insert different images into this pipeline with no noise and also other varying levels of noise to see how these individual pipelines compare with each other in the face different noise levels.

To this end, we designed a python class that acted as an interface. It allowed us to abstract away the different image compression algorithms and easily allow additionally Gaussian distributed noise to be added later. Then, we would run the decompressed images into the pre-trained convolutional neural network and compare the results to receive new metrics about its performance.

The images we are using for this purpose are from the CIFAR10 dataset [7]. It contains 60,000 images, and 6,000 for each of the 10 labeled classes.

We divide up this section into first discussing the convolutional neural network that was trained on, and then the various compression implementations and elaborate upon them.

3.1 Convolutional Neural Network

The creation of this network was based off the tutorial that demonstrates how to create a Convolutional Neural Network using the Keras Sequential API in Tensorflow to classify CIFAR10 [5]. This network consists of Conv2D and MaxPool2D layers alternating before being flattened and processed in a Dense layer of size 64 and a final Dense layer of 10 for the 10 categories that CIFAR10 classifies.

3.2 Autoencoder networks 8810 and 8816

The neural networks we used operated as auto-encoder/decoders and had a combination of convolutional layers with larger strides to shrink the size of the image (while separately controlling the number of channels), and then a series of upsampling and normalization layers to appropriately expand the size back to the original.

As the dimensions of the image itself get far more compressed through this architecture, there is a simultaneous increase in the number of channels that each image has. As the images are 32×32, and shrinking the image past 8×8 lead to malformed output images, that extra information has to go somewhere. This is accomplished by slightly increasing the number of channels through the convolutional net. The two primary encoder/decoder architectures that we used during testing are called *8810* and *8816* for having a minimum data size of 8×8 image with 10 channels and 8×8 image with 16 channels respectively.

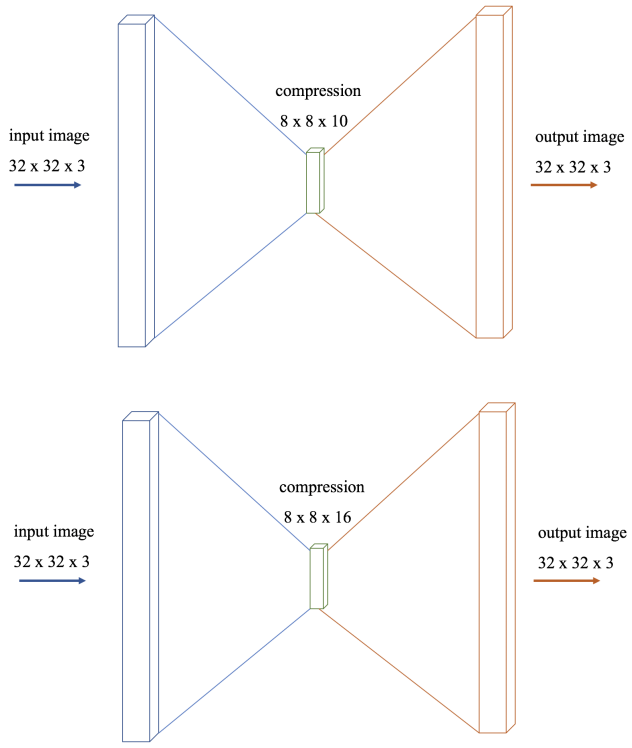

Figure 2: Simple diagram showing 8810 and 8816 architecture

One benefit of this autoencoder structure is that we can train the network to be somewhat stable when provided with input noise. Looking at figures 13 and 14 (found in the Appendix) demonstrate the differences between these models. While the model trained with noise makes the outputs fuzzier, they are inherently less noisy. This is especially visible in the horses in figure 14. The "N=0.2" refers to the standard deviation of the Gaussian distributed noise.

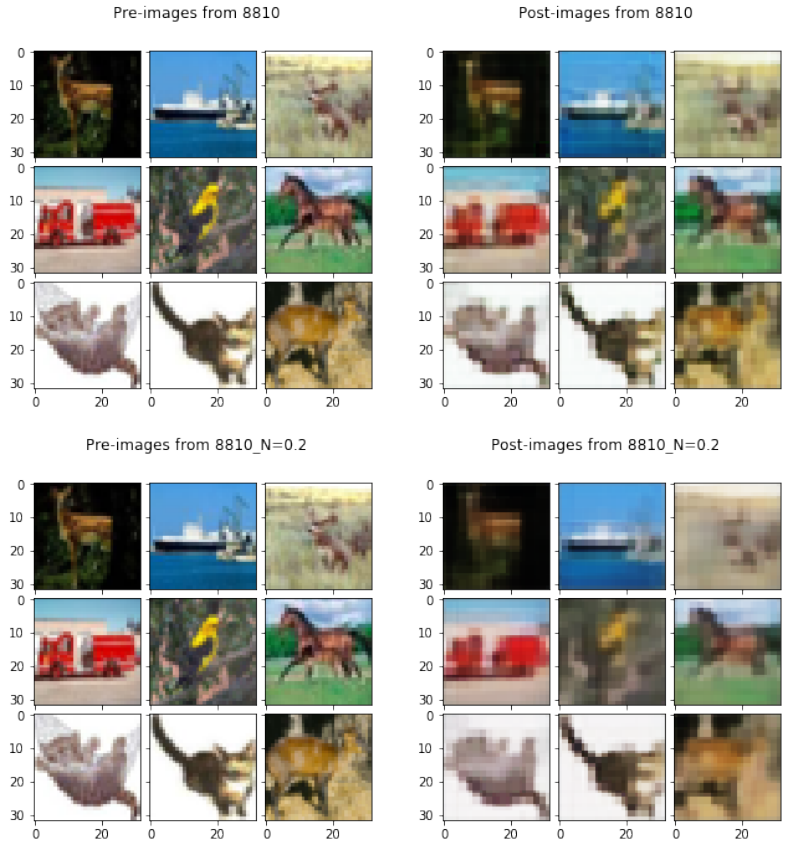

Figure 3: Pre/Post processing images for the 8810 network with and without noise during training

3.3 Principal Component Analysis

PCA compression was performed by maintaining an explained variance ratio of 0.98. An explain variance ratio is the sum of eigenvalues of the eigenvectors in the covariance matrix maintained divided by the sum of all of the eigenvalues. This method as a way to define PCA, as opposed to the number of vectors to use, serves as better metric for referring to the quality of the compression. It also transfers over to other image sets with different dimensionality better. The N=0.2 refers to the standard deviation of the Gaussian distributed noise.

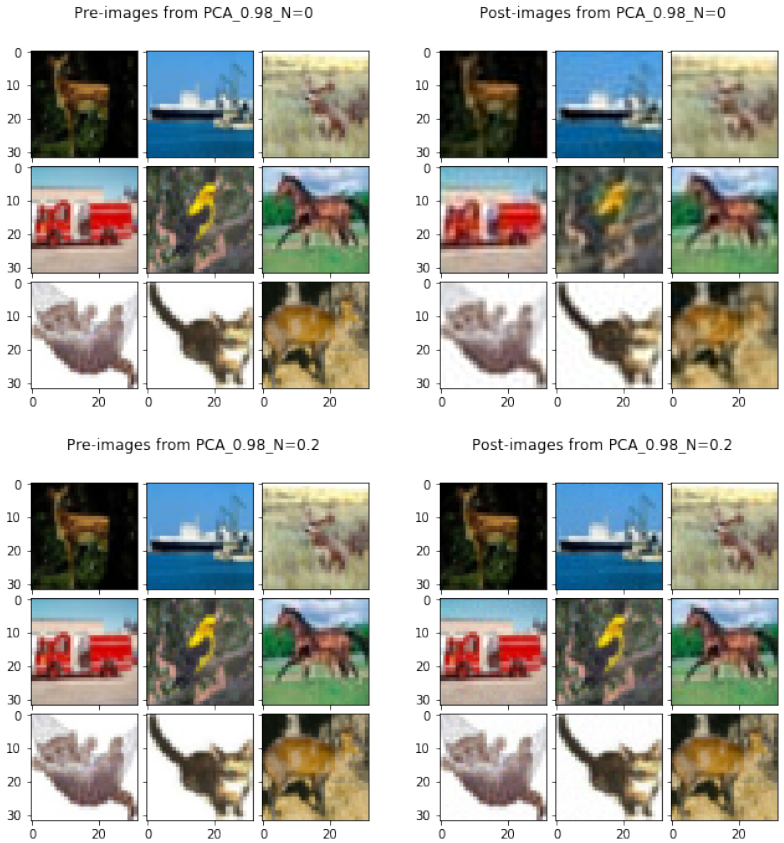

Figure 4: Pre/Post processing images for the PCA network with and without noise during training

3.4 Resizing

This is the one compression method that we used that is not "trained" on Gaussian distributed noise. The function is that is used to do this is the skimage.transform.resize [6]. It performs interpolation to resize our images in both directions. We first resize the image down by 0.8 and then resize it back up to the same dimensions the image was in. Below is the result of this method.

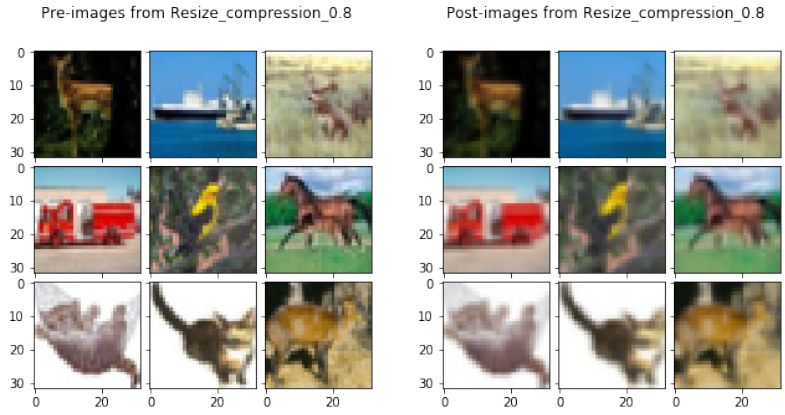

Figure 5: Pre/Post processing images for resizing

4 Results

We compare the accuracies of the compression methods trained with noise and without noise on various inputs. The results are separated by compression method subsections and presented here. The models were evaluated on the full $10,000$ images in the CIFAR-10 dataset.

4.1 Autoencoder Networks

The autoencoders preform very comparably given any amount of noise where $0 \leq \sigma \leq 0.3$. Even though 8816 has 1.6 times less compression, the network only performs marginally better for $\sigma < 0.2$. For $\sigma \geq 0.2$ the difference in accuracies becomes much more prominent. Both auto encoders perform worse than the original input does for $\sigma \leq 0.05$, but for $\sigma > 0.05$, the drop in accuracy is much slower than the original with an increase in noise.

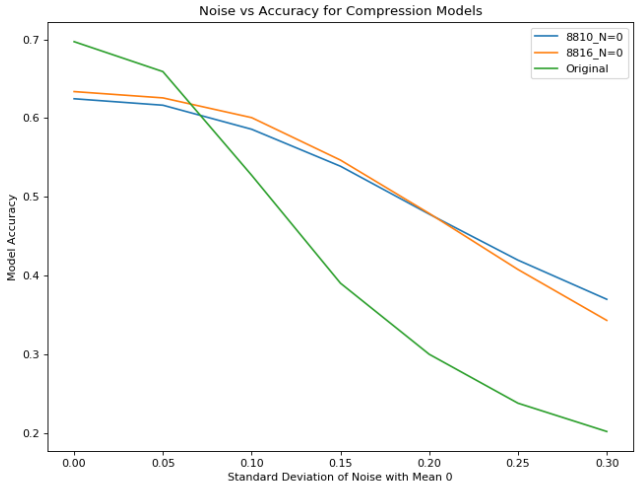

Figure 6: Accuracy of the autoencoder networks and the original images as noise is added

When the autoencoders are trained with noise ($\sigma = 0.2$) there is a relatively constant relationship between noise and accuracy. Even where noise is greater the noise the autoencoders were trained on, the accuracy of the models has a small decrease. Between the two autoencoders, the difference in accuracy is small, but 8816 always performs better than 8810.

The accuracy of the autoencoders trained with noise performs worse than the autoencoders trained without noise for $\sigma < 0.15$, but the accuracy of the autoencoders trained without noise drops dramatically while the ones trained with noise hold steady.

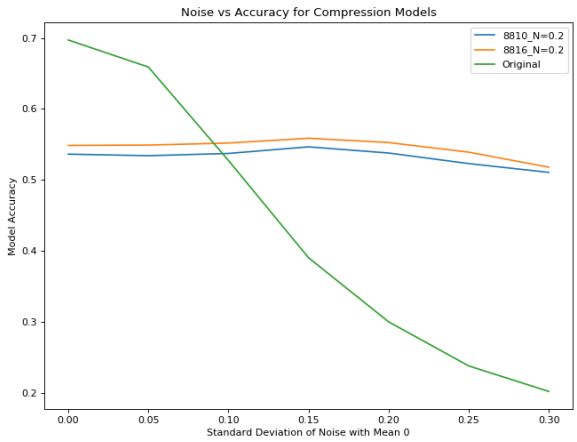

Figure 7: Accuracy of the autoencoder networks trained with noise and the original images as noise is added

4.2 PCA

The PCA model trained with no noise starts out with an accuracy similar to that of the original. When $\sigma > 0$ the original is not nearly as robust to noise.

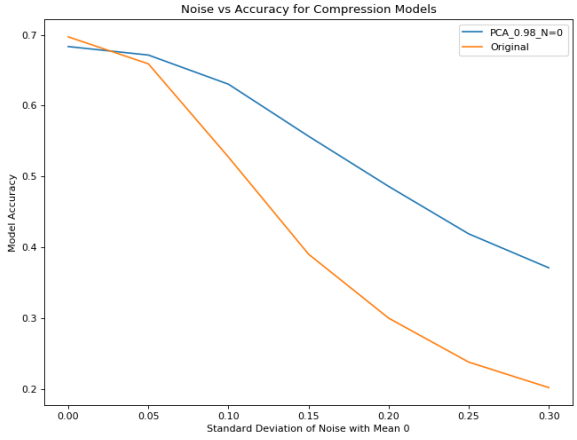

Figure 8: Accuracy of the PCA model and the original images as noise is added

The effect of a PCA model trained with noise ($\sigma = 0.2$) appears to be minimal. As noise increases there is virtually no difference between the classification accuracies of the original and the PCA model in this case.

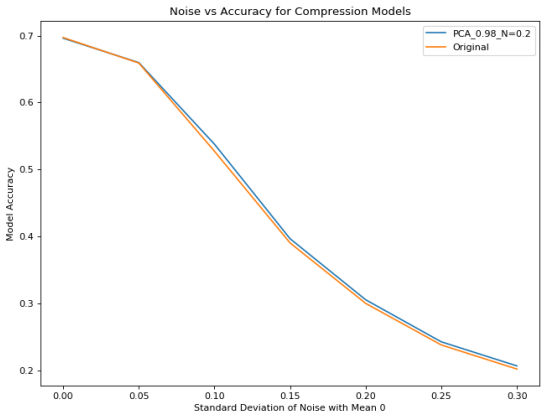

Figure 9: Accuracy of the PCA model trained with noise and the original images as noise is added

4.3 Resizing

Resizing by compressing the images to 0.8 of their original width and height. For $\sigma \geq 0$ the difference in accuracy drastically increases and remains mostly constant for $\sigma > 0.15$.

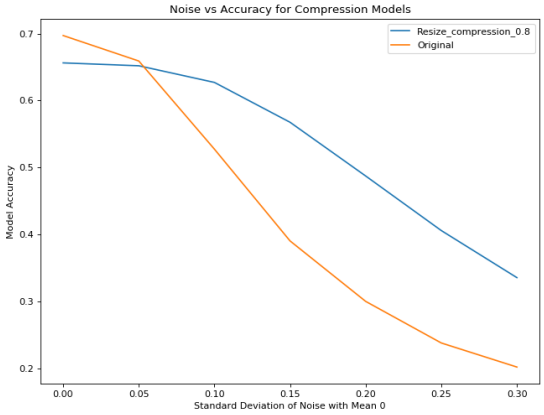

Figure 10: Accuracy of the resized model and the original images as noise is added

4.4 Summary

The compression ratio for all models is fixed to the type and architecture of model except for resizing and PCA. The compression ratio is defined simply as the ratio of the original width to the compressed width multiplied by the ratio of the original height to the compressed height. The compression. PCA compression ratio is defined as the total number of features in the input image divided by the number of components needed to meet the explained variance ratio which in the case of the models used is 0.98.

Table 1: Model Accuracy vs Noise

Model Name	**Compression Ratio**
$8810_{N=0}$	4.8
$8816_{N=0}$	3
$PCA0.98_{N=0.2}$	1.08
$PCA_0.98_{N=0}$	6.92
Resize compression 0.8	1.56
$8810_{N=0.2}$	4.8
$8816_{N=0.2}$	3

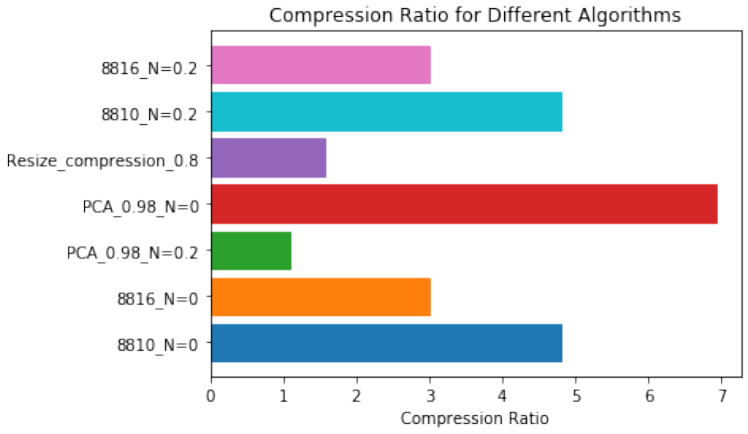

Figure 11: Compression ratios of the various models used

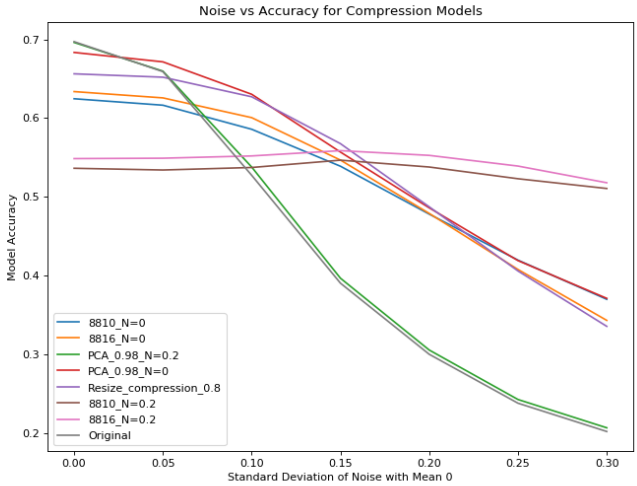

Figure 12: Compression model accuracy for all of the models and the original

5 Discussion

The compressions that we're using in this experiment can be used in multiple possible downstream applications as a form of dimension reduction (and not always as a compression/extraction combination). To see how the different compressions might handle different downstream tasks, the results of this survey

Table 2: Model Accuracy vs Noise

Model	σ**=0**	σ**=0.05**	σ**=0.1**	σ**=0.15**	σ**=0.2**	σ**=0.25**	σ**=0.3**
Original	**0.6973**	0.6592	0.5274	0.3902	0.2998	0.2376	0.2016
$8810_{N=0}$	0.6247	0.6165	0.5859	0.5389	0.478	0.4195	0.3697
$8816_{N=0}$	0.6338	0.6258	0.6007	0.5467	0.4789	0.4076	0.3429
$PCA0.98_{N=0}$	0.6836	**0.6716**	**0.6304**	0.5569	0.486	0.4189	0.3709
Resize 0.8	0.6564	0.6521	0.6272	**0.5676**	0.4873	0.4057	0.3353
$8810_{N=0.2}$	0.5363	0.5341	0.5373	0.5466	0.5379	0.523	0.5106
$8816_{N=0.2}$	0.5486	0.5491	0.552	0.5587	**0.5527**	**0.5391**	**0.5179**
$PCA0.98_{N=0.2}$	0.6963	0.6598	0.5379	0.3965	0.3052	0.2423	0.2063

boil down to 3 essential questions.

- Does the compression maintain information essential to downstream machine learning use?
- How small is the compressed data with respect to the original?
- How resistant to input noise is the data compression?

Each of the measurements that we took indicate an answer to each of these questions. We can characterize the amount of maintained information by the successful classification rate of decompressed images, we can directly measure the compression rate, and we can characterize a compression model's resistance to noise based on the classification rate after compressing/decompressing noisy images.

Figures 6 - 10 show how the various compression tactics maintain important information as noise increases.

The autoencoder/decoder networks (trained with 0 noise) preserve less information at low noise, but are more resistant against higher levels of noise. This tradeoff is exaggerated in the autoencoder/decoder networks trained with noise. They perform practically constantly no matter the noise of input, but are significantly less accurate than the original image (at least at low noise levels).

The PCA trained without noise is very similar in performance to autoencoder/decoder compression, while the PCA trained with noise performs nearly identically with the raw input image. Because of this, PCA trained without noise dominates PCA trained with noise.

The basic resizing seems to have a similar starting point as the autoencoder

models, but worse capacity to deal with noise, and given it's worse compression factor, it seems to be dominated by the autoencoder/decoder.

Looking at figure 11 and 12, we can see that the PCA trained without noise is by far the most compressed. The 8810 and 8816 autoencoders also have a good compression ratio.

From these two factors, low noise tasks (less testing noise relative to training noise), the PCA dominates, but with high noise images (more testing noise relative to training noise), the autoencoder dominates. So for future work with data dimension reduction, expected noise of the input is key in deciding which of these two architectures to use.

References

[1] Fei-Fei Li, Justin Johnson and Serena Yeung. *Training CNNs, part 2.* https://www.cs.princeton.edu/courses/archive/fall19/cos429/slides/cos429_fall2019_lecture19_DL5_training.pdf

[2] CodeParade. *Computer Generates Human Faces.* https://www.youtube.com/watch?v=4VAkrUNLKSo

[3] Matthew D. Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks.* https://arxiv.org/pdf/1311.2901.pdf

[4] Ryan Flynn. *Cifar-Autoencoder* https://github.com/rtflynn/Cifar-Autoencoder

[5] *Convolutional Neural Network (CNN) : TensorFlow Core* https://www.tensorflow.org/tutorials/images/cnn

[6] *Module: transform - skimage v0.17.dev0 docs* https://scikit-image.org/docs/dev/api/skimage.transform.htmlskimage.transform.resize

[7] Alex Krizhevsky *Learning multiple layers of features from tiny images* https://www.cs.toronto.edu/ kriz/cifar.html

6 Appendix

This section includes more figures.

This figure shows pre-images and post images from the 8816 autoencoder trained with and without noise. The N=0.2 refers to the standard deviation of the Gassuain distributed noise.

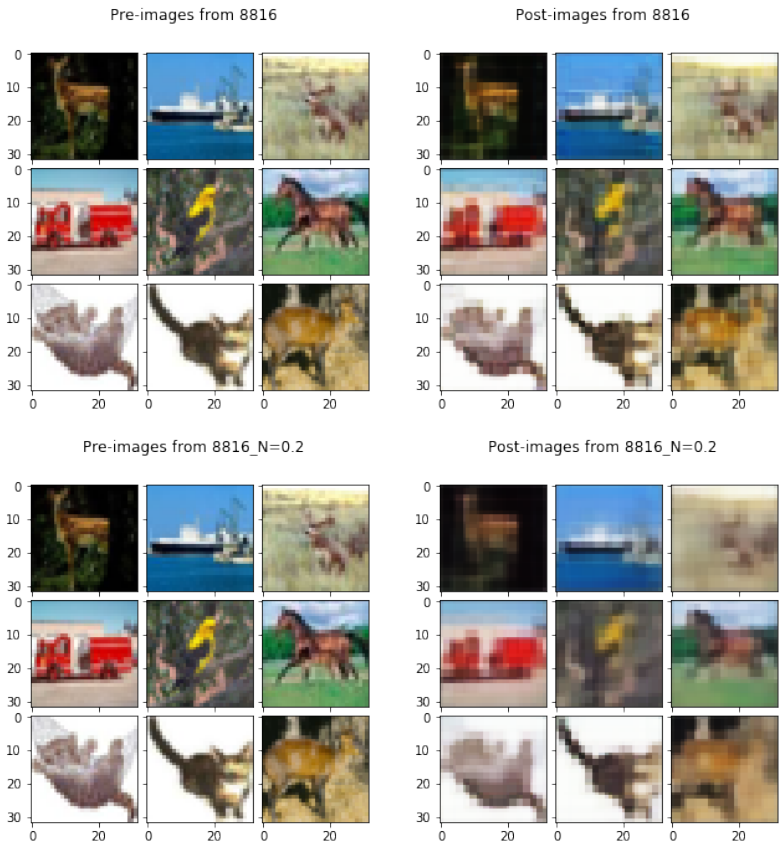

Figure 13: Pre/Post processing images for the 8816 network with and without noise during training

This figure shows pre-images and post images from the 8816 autoencoder trained with and without noise. The N=0.2 refers to the standard deviation of the Gaussian distributed noise. This is similar to above except the original images already contain a level of noise and the network is trained to return the

original image. This is a visualization of the result:

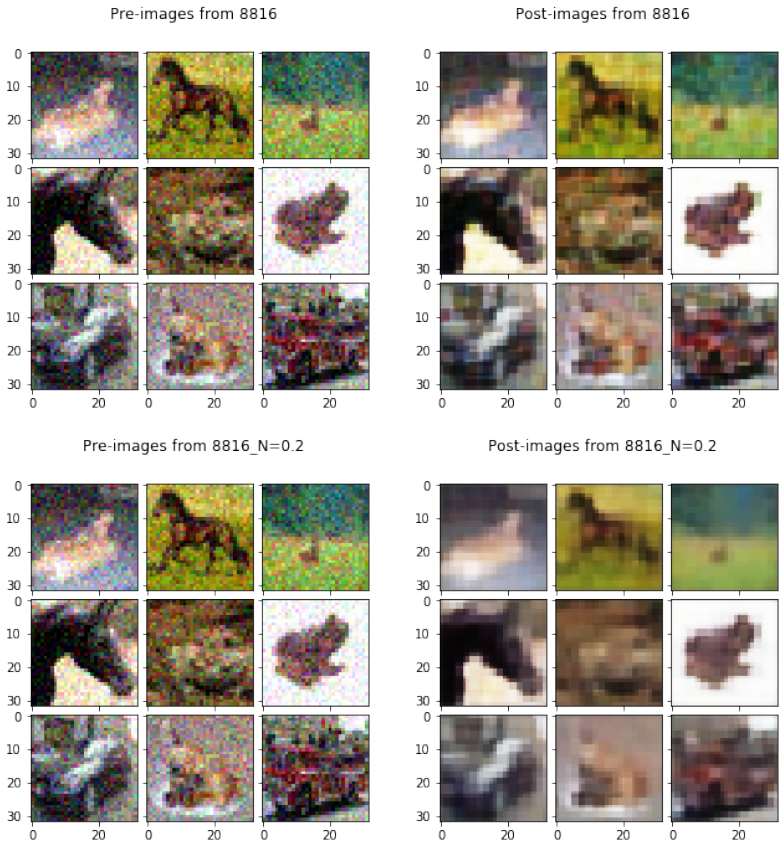

Figure 14: Pre/Post processing noisy images for the 8816 network with and without noise during training

This figure shows pre-images and post images from PCA fitted with and without noise. The N=0.2 refers to the standard deviation of the Gaussian distributed noise. This is the output when the original images already contain a level of noise on with and PCA is fitted to return the original image. This is a visualization of the result:

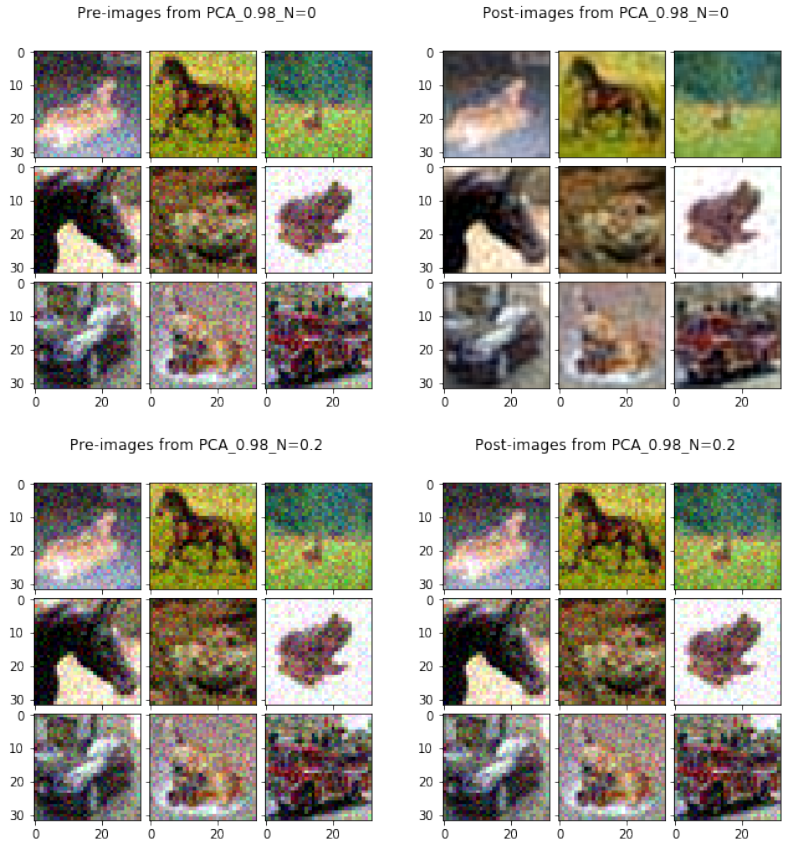

Figure 15: Pre/Post processing noisy images for the PCA network with and without noise during training

We pledge our honor that this paper comprises our own work in accordance with university regulations.
Jessica Edouard, Nebil Ibrahim, Nathan Lovett-Genovese