Image Compression with Neural Networks and SVD

David Chettrit
Stephan Seebacher
Sezer Güler
Department of Computer Science, ETH Zurich, Switzerland

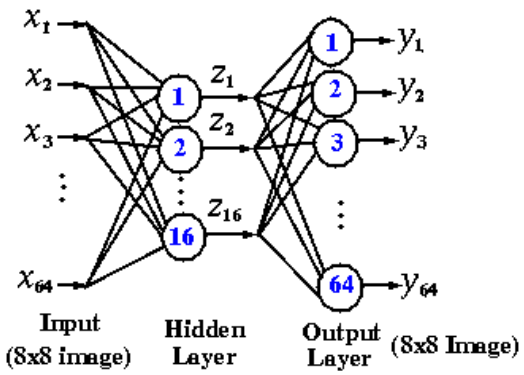

Figure 1. Neural Network Structure for compression.

***Abstract*—**

I. Introduction

In this paper we present a novel solution for image compression, which is based on neural networks.
TODO: At the end describe the structure of the paper.

II. Preliminaries

A. Neural Networks

Inspired by biological nervous systems, such as the human brain, artificial neural networks are used for information processing. In biological nervous systems, highly interconnected neurons work together to solve a specific task. Each neuron solves a subtask and communicates with other neurons. In artificial neural networks this idea is used to approximate arbitrary non-linear functions. The nodes(neurons) are connected by weighted edges and the topology is chosen depending on the problem. In the training phase, the weights of the edges are set depending on the data. Neural networks are often represented in layers. Each layer contains some of the nodes and then follows the interconnections and then the next layer.
wieso verwendet man denn gerade Neuronale netzwerke....

B. Neural Network Structure

Figure 1 shows the neural network structure we use for compression. There are three layers: input layer, hidden layer and output layer. The input layer and the hidden layer are fully interconnected as are the hidden layer and the output layer. The weights of the edges are detrermined by the training algorithm which is described in the next section. This kind of neural network is also called bottleneck-like neural network, because the hidden layer is a bottleneck.

C. Neural Network Training Algorithm

For the training 8 by 8 chunks of the image are chosen uniformly at random. The output of the bottleneck-like neural network has to be the same as the input and the weights are set accordingly. This process is iterated until the image is reconstructed faithfully. Different learning methods were tried like (....TODO) but it turned out that for the Levenberg-Marquardt method the results were best.

D. Singular Value Decomposition

The singular value decomposition(SVD) is a matrix factorization. A m×n matrix M can be written as a product of three matrices U, Σ and V^*, where U is a unitary m×m matrix, V^* is the conjugate transpose of a n×n unitary matrix V and Σ is a m×n diagonal matrix. The entries on the diagonal of Σ, which are non-negativ real numbers, are called the singular values of M. The columns of U are called left singular vectors and the rows of V^* are called right singular vectors of M.
SVD can be used for image compression the following way. The singular values are sorted in descending order and then only the largest singular values and the corresponding right and left singular vectors are used to save the image.

E. Quanitization

Quantization as shown in Figure 2 is the process of mapping a large set of values to a smaller set of values. In Figure 2 we construct 8 possible binary codes which can be represented with 3 bits. Each code represents an interval of the decimal values. For example, for the values between -1.0 and -0.75 we use the code 000. This way, we use 3 bits for a pixel that originally is represented by 8 bits. This compression method is lossy and the original image can not be reconstructed.

III. Compression

A. Compression with neural networks

We use the topology described in section II-B. This network is then trained with a set of random images. Then,

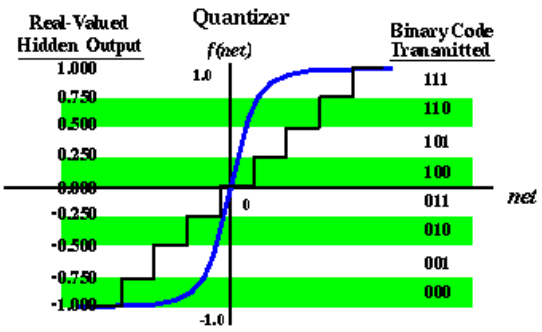

Figure 2. Neural Network Structure for compression.

Figure 3. Visual comparison of original and reconstructed image.

the image can be compressed with this network in the following way. Chunks of the image are iterated in turn and every chunk is fed into the neural network from the left hand side. If we have 8×8 chunks, we have 64 pixels and each pixel is given to one node on the input layer(which in this case has 64 nodes) of the neural network. Then the values on the hidden layer are taken as the "compressed" values. Since the hidden layer has fewer nodes than the input layer, this gives a compression. As one can see in Figure 1, the neural net used here has 16 nodes on the hidden layer. This already gives a compression ratio of 0.25.

TODO: Describe that one needs to separate neural network for compression and decompression?

B. Compression with quantization

To further improve compression, quantization as described in Section II-E is used. The output of the neural network is quantized, such that each 8 bit value is represented using 3 bits.

C. The Compression Algorithm

Overall, the compression can be summarized as follows.

1) The neural network is trained with the training set; in this case 100 images.
2) The trained network is adjusted to the actual image by selecting a few random chunks of the image. Then, the the training is done with those few chunks and the weights updated.
3) Compress 8× chunks of image in turn using neural network.
4) Compress all the resulting values with quanitization.

IV. Results

We tested the compression algorithm with 100 images which were not in the training set and get a mean error of (.....) and a compression rate of (.....). Time for everything: 907.7797
Average quadratic error: 0.0086154
Average compression rate: 0.0094089
Average inverse for sum compression/error 55.4806

and with 100 images which are all in the training set:
Time for everything: 873.8136
Average quadratic error: 0.008281
Average compression rate: 0.0092095
Average inverse for sum compression/error 57.174
Figure 3 shows an example image before any compression on the left and the reconstructed image after compression on the right.
Idea: Test with training directly on image, and compare results with pre-training.
TODO: Run program with different parameters and construct graphs to compare results.

A. Comparison to Baselines

We compare our solution with two baselines. The first baseline is a compression algorithm based on Principal Component Anlaysis(PCA) and the second algorithm is based on Clustering with Gaussian Mixture Models(GMM). In PCA, one projects high dimensional data to low dimensional data. This can be done using the eigenvalue decomposition on the covariance matrix Σ, which describes the covariance matrix as a product of three matrices U, Λ and U^T. U contains the eigenvectors of Σ and Λ contains the eigenvalues of Σ. Similar to SVD, one can sort the eigenvalues in decreasing order and then take the k largest of them and the first k columns of U and k columns of U^T to reduce the data which has to be saved.

TODO: Mention parameters used for both baselines, e.g k

Clustering is the process of assigning each value x in a data set S to some value x' of a smaller set S'. In image compression one can use this to reduce the color space. Each pixel is assigned to one of the RGB values in the smaller color set, i.e. to a cluster. Using mixture models, each pixel can be assigned to more than one cluster with a corresponding probability. In a gaussian mixture model, those values are distributed according to a normal distribution. The second baseline compresses images with

this approach.
TODO:
Compare compression rates....
Compare error....

V. Discussion

We also tried to apply SVD on the image and then compress the U and V matrices with our neural network. The training had to be adapted. This gives another improvement in the compression rate and since the neural network is applied to U and V after extracting k rows and columns of U and V, also the time performance would be better. But the decompressed U and V matrices looked very different than the original ones. This affected the reconstructed image and gave a bad mean error.

Strengths: Mean error, Compression rate.
Weakness: Time

VI. Related Work

WICHTIG: Was machen wir anders als die? Pre-Training + training on image?

References