Spectral Methods for Deep Convolutional AutoEncoders

James Smith

***Abstract*—This project implements spectral pooling and upsampling techniques for deep convolutional autoencoders and compares these methods with the standard pooling and upsampling techniques for both encoding networks (max pooling) and decoding networks (nearest-neighbor and bilinear interpolation). This project finds that spectral pooling is more effective than max pooling for lossy networks but is less effective for deep networks that must pool data at low rates to be effective. Additionally, this project shows that spectral upsampling is not as effective as nearest-neighbor and bilinear interpolation methods yet remains competitive.**

I. Introduction

ARTIFICIAL neural networks (ANNs) are most commonly associated with image processing by their use in image classification; however, another highly active research is the autoencoder (AE), a neural network trained to learn its input (most commonly an image) as an output [1]. This can be useful for both image generation and compression. Deep convolutional autoencoders (CAEs) are the most successful networks for image compression while Generative Adversarial Nets (GANs) [2] are the most successful networks used in generating images. Another type of network, variational autoencoders (VAEs), [3] is also frequently used for image generation. While network structures for image compression and generation are relatively similar, this project focuses on the problem of image compression. However, it should be observed that the findings presented in this project have transformative implications towards the highly used GAN and VAE networks.

Image compression, an extension of general data compression, is an active neural network application area. Current State-of-the-Art (SotA) neural networks are not yet able to compete with leading image compression algorithms for visually appealing recovery; however, they are currently useful for compression where information preservation is more important than visual appeal [1]. Therefore, the specific scope of this project is to find CAE networks that can perform lossy image compression while preserving image information.

Autoencoding networks are normally described as having a bottleneck structure. Deep CAEs include several convolutional/deconvolutional layers coupled with pooling techniques (such as the commonly used max pooling) in order to map the image into a latent vector at the center of the bottleneck. Subsampling is done by the pooling operation and applied to reduce parameters between convolutional layers. Max pooling (as well as other "spatial" pooling methods) suffer from information loss (i.e., data is completely lost in the max operation). A different pooling method, spectral pooling [4], reduces layer parameters by removing spectral components. This project investigates the effect of both spectral pooling and its complement, spectral upsampling, for the CAE network. Experiments are implemented with CAEs on two benchmark image datasets. Specifically, this project makes the following contributions:

1) Spectral Pooling and upsampling are implemented in the (SotA) CAE models using the deep learning framework Tensorflow [5].
2) CAEs are compared using two pooling methods (max pooling and spectral pooling) as well as three upsampling methods (nearest-neighbor interpolation, bilinear interpolation, and spectral upsampling). Networks are trained for image compression and evaluated with the MNIST [6] and CIFAR-10 [7] datasets on the criterion of mean-squared error.

II. Deep Convolutional AutoEncoder

Introduced by [8], a convolutional autoencoder (CAE) is a combination of two neural networks, an encoder and decoder, purposed to extract image features and compress the features into a small latent space. Specifically, the encoder is a CNN that learns a mapping from an image to a latent vector and the decoder is a CNN that learns a mapping from the latent vector back to the original image. Compared to a classic deep ANN AE architecture [9], deep CAEs use convolutional neural networks (CNNs) to abstract image features to be encoded and then deconvolution operations to decode the latent vector.

The implementation of a deep CAE is rather intuitive. The encoder is a classic CNN composed of convolutional layers for feature extraction, pooling layers to down-sample data as a means of parameter reduction and generalization improvement [10], Rectified Linear Units (ReLUs) to prevent the vanishing gradient problem, and feedfoward multilayer perceptron (MLP) layers.

The decoder is generally made of the same building blocks as the encoder, but without pooling techniques. Pooling (especially max pooling) is an irreversible operation, with discarded data being irretrievable. Rather, a nearest neighbor method is used in the SotA deep CAE [3]. Another upsampling option that often produces better results with an additional computational cost is bilinear interpolation [11]. The purpose of both of these methods is to "reverse" the max pooling operation and interpolate into a larger dimensional space without creating undesirable artifacts associated with using convolutional or fully-connected layers. What follows is a brief review of the most widely-used subsampling and upsampling methods: max pooling, nearest-neighbor interpolation, and bilinear interpolation.

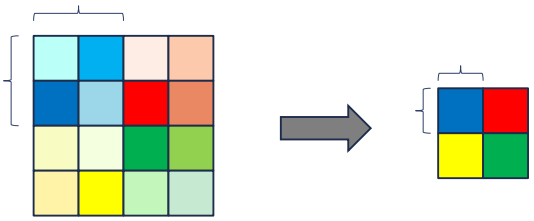


Fig. 1. Max pooling visualization

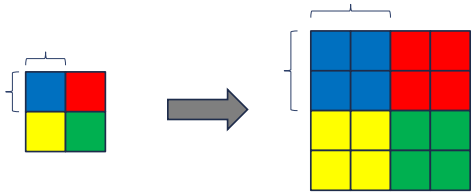


Fig. 2. Nearest-neighbor interpolation visualization

A. Max Pooling

Max pooling looks at a block of image data and passed through the maximum value found in that block. This process is demonstrated in Figure 1. The advantage of max-pooling is that it focuses on finding the most-important information in a block and filters out unimportant information. While this is intuitively useful for image classification, this method discards information needed for image recovery in the CAE image compression network.

B. Nearest-Neighbor Interpolation

Nearest-neighbor interpolation samples image data and extends each element into a higher dimensional block. This process is demonstrated in Figure 2. The advantage of nearest-neighbor is that it is a quick operation to upsample data without adding artifacts. However, parameters are reused and no significant information is gained in this process; bilinear interpolation is often used instead.

C. Bilinear Interpolation

Bilinear interpolation samples image data and determined each new element as an average of surrounding pixel values. This process is demonstrated in Figure 3. The advantage of bilinear interpolation is that it creates smooth images with additional parameters, but at the cost of extra computations compared to nearest-neighbor interpolation.

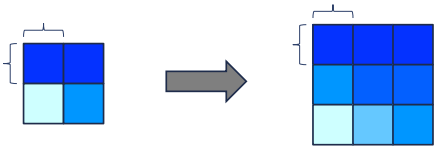


Fig. 3. Bilinear interpolation visualization

III. Spectral Pooling

Spectral pooling reduces image dimensionality by truncating the spectral representation of a image-kernel product (which is the output of a convolutional layer). Simply put, spectral pooling is a low-pass filter. Images are transformed to the spectral domain using a FFT and are then truncated to keep only the information-rich low frequency components. This process is described in Algorithm 1. For neural networks, all operations must have a backwards operation to define the gradient flow as it backpropagates throughout the network for the weight update process. The gradient backpropagation for this operation is simply the spectral pooling performed backwards, with zero-padding applied to invert the truncation step.

Algorithm 1 Spectral Pooling

Input: $X \in c \times m_1 \times m_2$ image
Output: $Y \in c \times p_1 \times p_2$
compute FFT of X
truncate X to size $c \times p_1 \times p_2$
return IFFT of X

Spectral pooling is desirable because it can be combined with the convolution theorem to achieve fast training results. The convolution theorem states that convolution can be sped considerably by being performed in the spectral domain as element–wise multiplication. This is given as:

$$\mathscr{F}\{f * g\} = \mathscr{F}\{f\} \cdot \mathscr{F}\{g\} \tag{1}$$

The transformation to the spectral domain can take advantage of the radix–2 based Cooley–Tukey Fast Fourier Transform (FFT) algorithm [12]. By using the FFT, the time complexity for the respective convolution techniques assuming a l_1 x l_2 kernel and m_1 x m_2 image are:

$$T_{\text{spatial}} = O\left(l_1 l_2 m_1 m_2\right) \tag{2}$$

$$T_{\text{spectral}} = O\left(n_1 n_2 log\left(n_1 n_2\right)\right). \tag{3}$$

where:

$$n_i = l_i + m_1 + 1. \tag{4}$$

Reducing the time complexity gives a scalable advantage as data size increases. Besides fast convolution computation,

the advantages of spectral pooling include: high parameter information retention (compression), flexibility in pooling output dimensions, and further computation savings by implementing spectral parameterization.

Spectral parameterization offers further advantages for spectral pooling in that kernels do not have to be transformed into the frequency domain. This is done by learning the filter weights in the spectral domain rather than the spatial domain. Spectral parameterization saves computation in that only two transformations must take place in each instance (the image to spectral domain and then the inverse transformation of the resulting product).

The greatest weakness of spectral pooling is the lack of proof substantiating translation invariance for this pooling technique. However, this is only important for the image classification task and may actually be an advantage for the AE problem.

A. Spectral Upsampling

In addition to implementing spectral convolution in the CAE encoder, this thesis presents spectral upsampling to be used in the decoder. This is the most important contribution given that deconvolutional layers typically struggle to invert the pooling operation as images are up-sampled. Simply put, spectral upsampling transforms inputs into the frequency domain and then uses these components to create an image of greater, predefined dimensions. This is given in Algorithm 2:

Algorithm 2 Spectral Upsampling

Input: $X \in c$ x p_1 x p_2 image
Output: $Y \in c$ x m_1 x m_2
compute FFT of X
return IFFT of spectral X with size c x m_1 x m_2

Intuitively, it can be seen that the back-propagated gradient will be calculated by applying forward spectral pooling on the gradient backwards through the layer. That is, the gradient will be transformed into the spectral domain and truncated to the number of coefficients required to down-sample into the resulting layer.

IV. Experiments

This project implements six different pooling-upsampling combinations using the two discussed pooling techniques (max-pooling and spectral pooling) and three discussed upsampling techniques (nearest neighbor interpolation, bilinear interpolation, and spectral upsampling). The project is implemented using Tensorflow and thus vanilla spectral pooling and upsampling are applied to take full advantage of Tensorflow's highly optimized methods. For each experiment, ten separate trials are run for each network and the presented results are the average of these values. The networks are trained using a batch size of 64 (that is, each iteration updates the weights based on 64 input images) for ten training epochs (the number of times each image is trained on). Each dataset is split into training, validation, and testing sets based on recommended specifications for the dataset. For each of the six networks, experiments are carried out for two different latent vector sizes. Each experiment is evaluated on training time (s) and information preservation with mean squared error (MSE) by (5):

$$MSE = \frac{\sum_{p=1}^{P} (d_p - o_p)^2}{P} \tag{5}$$

where:
d_p is the desired value at pattern p
o_p is the actual value at pattern p
P is the total number of training patterns

TABLE I
Training Results for MNIST Dataset

Pooling	Upsampling	Dim Middle	Training Time (s)	Validation Error	Testing Error
Max	NN	10	3114.58	0.2253	0.2293
Max	BL	10	3195.52	0.1683	0.1730
Max	SU	10	7947.34	0.1805	0.1822
SP	NN	10	5840.06	0.1481	0.1504
SP	BL	10	5506.64	0.1334	0.1360
SP	SU	10	9457.59	0.1734	0.1771
Max	NN	30	2684.74	0.1382	0.1413
Max	BL	30	2859.32	0.0907	0.0913
Max	SU	30	7222.52	0.1118	0.1133
SP	NN	30	5821.34	0.1021	0.1034
SP	BL	30	6972.01	0.0923	0.0937
SP	SU	30	11248.88	0.0993	0.1003

For the first experiment, images from the single-channel MNIST dataset [6] are embedded into latent vectors of size 10 and 30 (compared to 784 features in the input and output images). The network architecture used to implement these experiments is described in Figure 4. Each network is trained using the SOTA Adam optimization method [13]. Experiment results on the previously stated evaluation criterion are given in Table I with original images given in Figure 6 and produced images in Figures 6–8.

The second experiment embeds images from the three-channel CIFAR-10 dataset [7] into latent vectors of size 50 and 250 (compared to 3,072 features in the input and output images). The network architecture used to implement these experiments is described in Figure 5. Each network is trained in a similar method to experiment 1 and results are given in Table II with original images given in Figure 7 and produced images in Figures 7–10.

When comparing results, it is clear that spectral pooling technique reduces training error for the lossy network architectures. However, spectral upsampling is not necessarily as successful as the nearest-neighbor and bilinear transform methods. For the deeper networks, spectral techniques are competitive but not superior. When examining the generated MNIST images, all of the digits look readable, save the spectral upsampling. In comparison, the CIFAR-10 images are quite blurry, for reasons discussed later. In summary, results suggest that spectral upsampling in the decoder is not as effective as the other interpolation methods and spectral pooling in the encoding outperforms max pooling for lossy network compression.

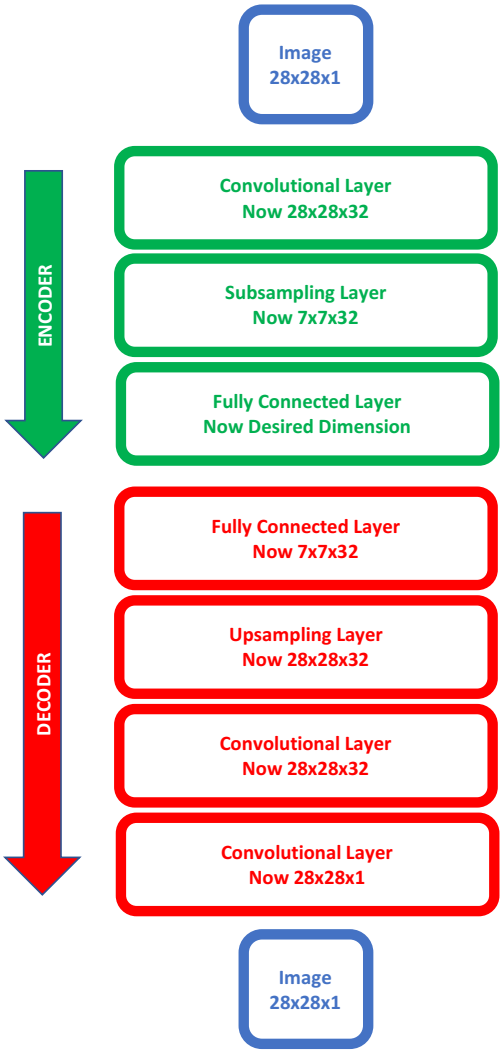


Fig. 4. CAE network structure for MNIST experiments

TABLE II
TRAINING RESULTS FOR CIFAR-10 DATASET

Pooling	Upsampling	Dim Middle	Training Time (s)	Validation Error	Testing Error
Max	NN	50	2579.30	0.5831	0.5892
Max	BL	50	2674.66	0.5803	0.5877
Max	SU	50	4169.30	0.5871	0.5935
SP	NN	50	6070.60	0.5945	0.6006
SP	BL	50	6033.94	0.5905	0.5961
SP	SU	50	7689.81	0.5963	0.6023
Max	NN	250	2613.70	0.5594	0.5625
Max	BL	250	2677.09	0.5588	0.5645
Max	SU	250	4163.23	0.5709	0.5762
SP	NN	250	6051.57	0.5680	0.5737
SP	BL	250	6020.04	0.5677	0.5783
SP	SU	250	7680.75	0.5770	0.5839

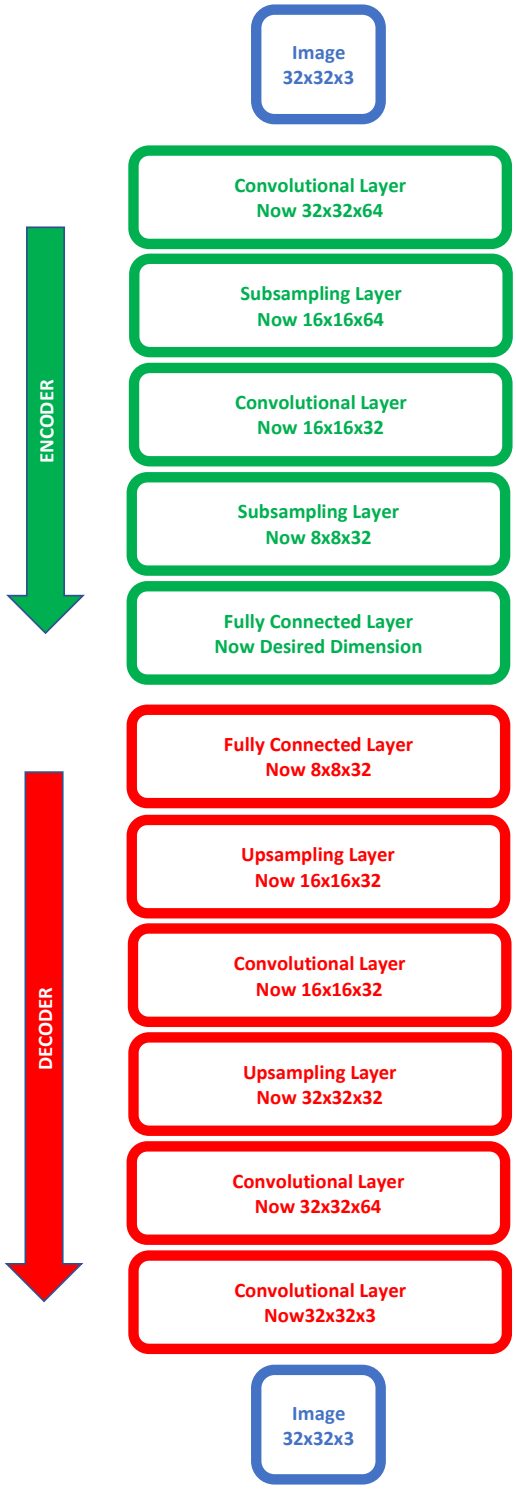


Fig. 5. CAE network structure for CIFAR-10 experiments

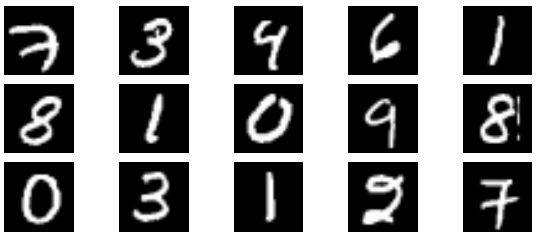


Fig. 6. Original MNIST digits

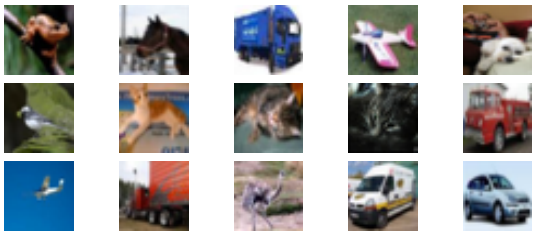


Fig. 7. Original CIFAR-10 images

Since these experiments use pixel-by-pixel mean squared error, they cannot estimate the probability distribution of the training data, resulting in blurry pictures for many cases. This is a common problem for deep CAEs. However, these proof of concept experiments show the powers of spectral pooling for both encoders and decoders, demonstrating the need for these methods to be implemented in SotA VAEs and Generative Adversarial Networks to generate images of similar quality compared to the training dataset.

V. Conclusion

This project implemented spectral pooling upsampling in deep CAEs for image compression and compared networks with SotA max pooling, nearest-neighbor interpolation, and bilinear interpolation for various latent vector dimensions on two benchmark datasets. This project found that spectral pooling outperforms max pooling for lossy networks that involve significant pooling operations, while max pooling remains superior for deep networks that pool data in small, gradual amounts. Furthermore, this project showed that spectral upsampling does not outperform SotA methods, but is competitive.

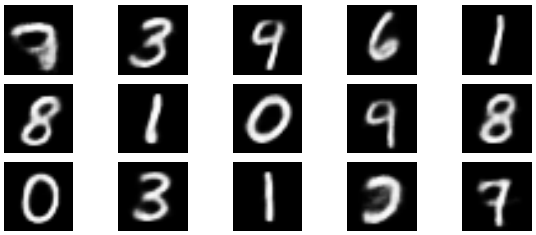
References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

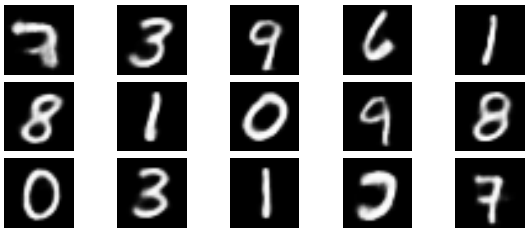
[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *arXiv:1406.2661 [cs, stat]*, Jun. 2014, arXiv: 1406.2661. [Online]. Available: http://arxiv.org/abs/1406.2661

[3] X. Hou, L. Shen, K. Sun, and G. Qiu, "Deep Feature Consistent Variational Autoencoder," *arXiv:1610.00291 [cs]*, Oct. 2016, arXiv: 1610.00291. [Online]. Available: http://arxiv.org/abs/1610.00291

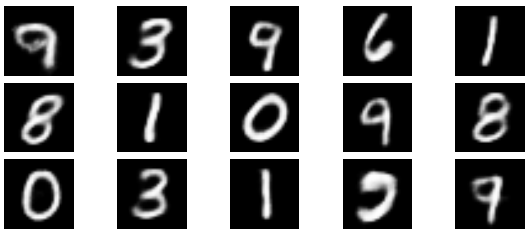
[4] O. Rippel, J. Snoek, and R. P. Adams, "Spectral Representations for Convolutional Neural Networks," *arXiv:1506.03767 [cs, stat]*, Jun. 2015, arXiv: 1506.03767. [Online]. Available: http://arxiv.org/abs/1506.03767



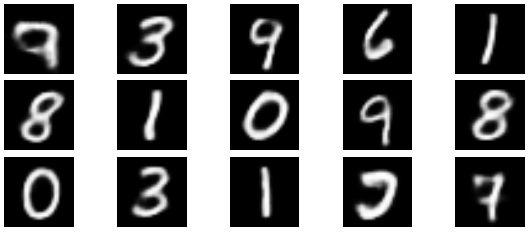
(a) Max–NN



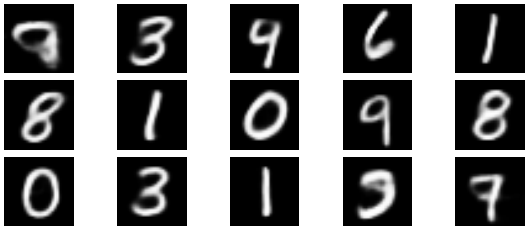
(b) Max–BL



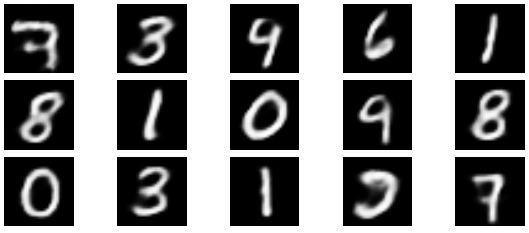
(c) Max–SU



(d) SP–NN

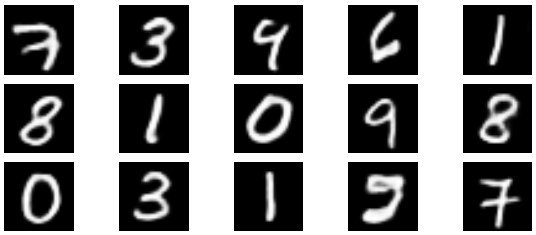


(e) SP–BL

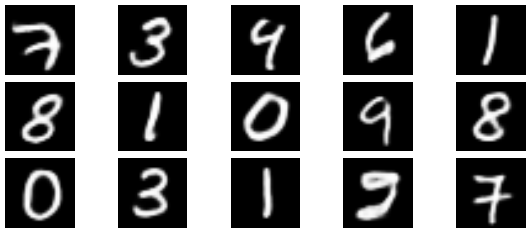


(f) SP–SU

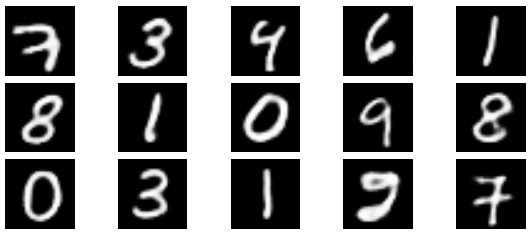
Fig. 8. MNIST digits using middle dimension 10



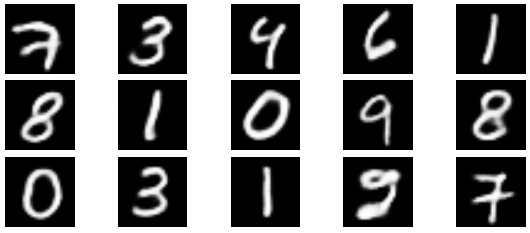
(a) Max–NN



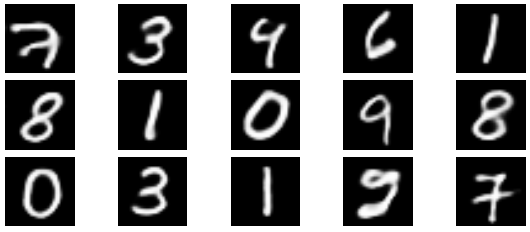
(b) Max–BL



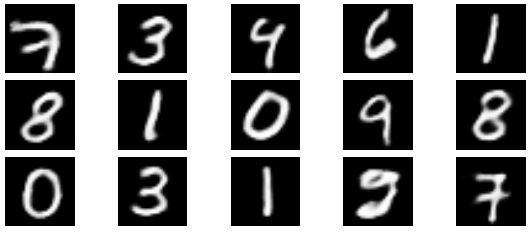
(c) Max–SU



(d) SP–NN

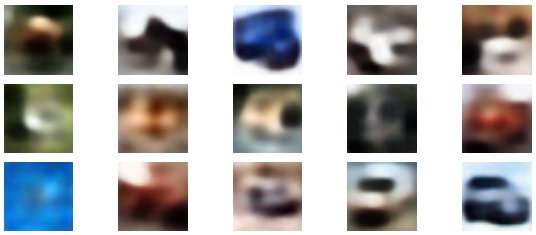


(e) SP–BL

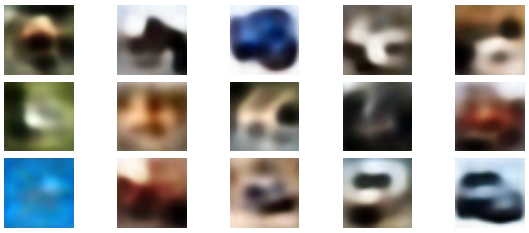


(f) SP–SU

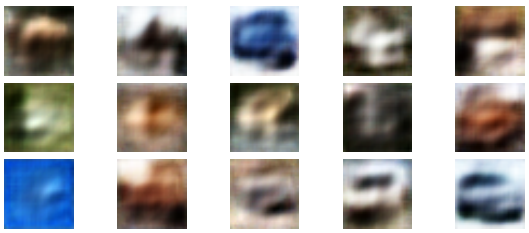
Fig. 9. MNIST digits using middle dimension 30



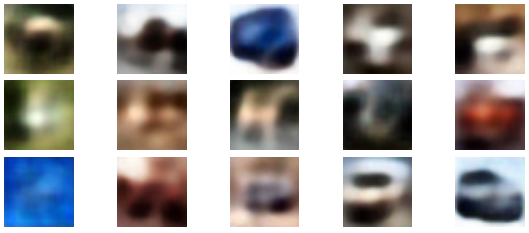
(a) Max–NN



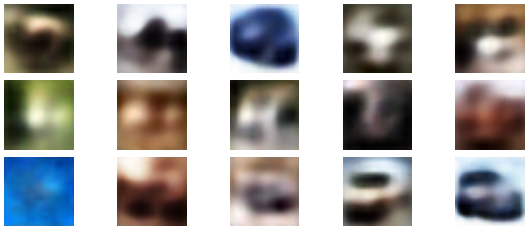
(b) Max–BL



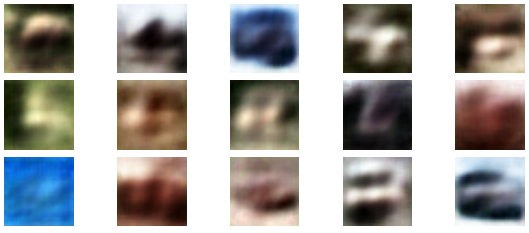
(c) Max–SU



(d) SP–NN

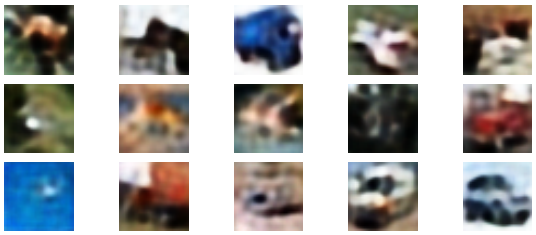


(e) SP–BL

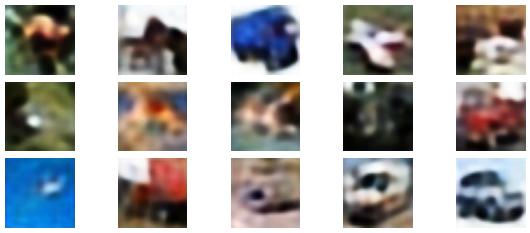


(f) SP–SU

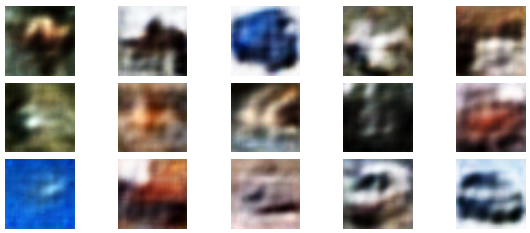
Fig. 10. CIFAR-10 images using middle dimension 50



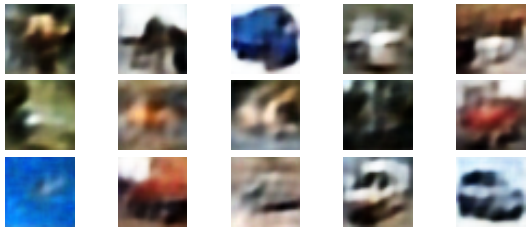
(a) Max–NN



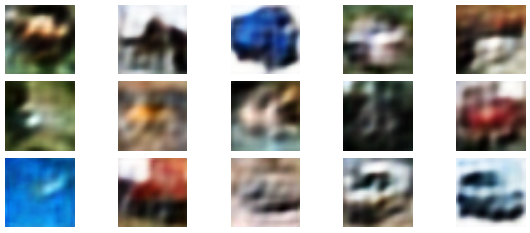
(b) Max–BL



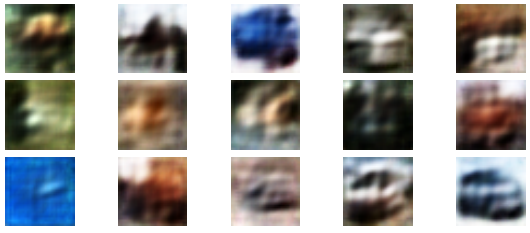
(c) Max–SU



(d) SP–NN



(e) SP–BL



(f) SP–SU

Fig. 11. CIFAR-10 images using middle dimension 250

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[7] "CIFAR-10 (Canadian Institute for Advanced Research)."

[8] J. Masci, U. Meier, D. Cirean, and J. Schmidhuber, "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction," in *Artificial Neural Networks and Machine Learning ICANN 2011*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Jun. 2011, pp. 52–59.

[9] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006. [Online]. Available: http://science.sciencemag.org/content/313/5786/504

[10] D. Scherer, A. Mller, and S. Behnke, "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition," in *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III*, ser. ICANN'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 92–101. [Online]. Available: http://dl.acm.org/citation.cfm?id=1886436.1886447

[11] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and Checkerboard Artifacts," *Distill*, vol. 1, no. 10, p. e3, Oct. 2016. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[12] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: http://www.jstor.org/stable/2003354

[13] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Dec. 2014, arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980