CIFAR-10 Reconstruction with Autoencoders

Christos Samaras
Graduate Student of Electrical and Computer Engineering Department,
Aristotle University of Thessaloniki

***Abstract*—A hybrid autoencoder is trained with CIFAR-10 train samples and statistics about skewness kurtosis, entropy, and bispectrum. Additionally a convolutional autoencoder is trained with CIFAR-10 train samples. Their reconstruction ability is compared with the reconstruction through Principal Component Analysis (PCA) with respect to average mean square error (MSE). The last one has the best performance. The convolutional autoencoder is also trained with SSIM loss function. Its results are not as good as those of the convolutional autoencoder trained with MSE.**

I. Introduction

The CIFAR-10 dataset is composed of 10 object classes with 6000 32x32 RGB images per class. An RGB image consists of three separate color channels: one for red, one for green and one for blue. Each channel stores intensity values for the respective color. These three channels can get 256 different values that range from 0 to 255. In RGB, R stands for red, G for green, and B for blue which correspond to (255, 0, 0), (0, 255, 0) and (0, 0, 255) respectively.

In an image, pixels' values change across its spatial dimensions. The rate of these changes introduces the concept of frequency into images.

An Autoencoder tries to replicate its input to its output. It is trained with unsupervised learning, thus the training samples are unlabeled. This neural network is split into two parts, the encoder and the decoder. The former learns to map the input to a latent representation (also named codings) in a space of fewer dimensions. The latter reconstructs these codings to the outputs, namely the reconstructed images. The Autoencoder does not trivially coping its input to output. Instead it tries to find effective ways to represent the data.

II. Method

A. Mathematical Background

1) Skewness: For a random variable x following a distribution, skewness denotes the asymmetry of this distribution relative to normal distribution (Fig.1). Skewness which derives from the Fisher-Pearson coefficient of skewness is defined as:

$$g_1 = \frac{m_3}{\sqrt{m_2^3}} \tag{1}$$

where m_3 is the third central moment:

$$m_3 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^3 \tag{2}$$

and m_2 is the second central moment:

$$m_2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{3}$$

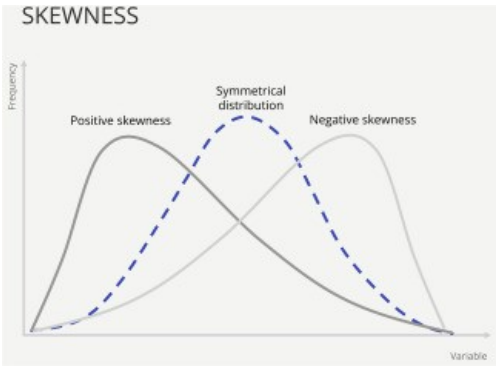

Fig. 1: Skewness

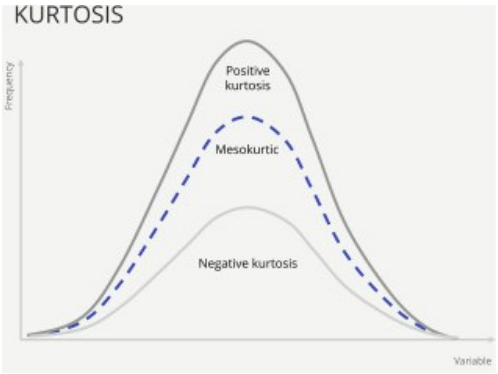

Fig. 2: Kurtosis

2) Kurtosis: For a random variable x following a distribution, kurtosis measures the sharpness or flatness of this distribution relative to normal distribution (Fig.2).

$$K = \frac{m_4}{m_2^2} \tag{4}$$

where m_4 is the fourth central moment:

$$m_4 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^4 \tag{5}$$

3) Entropy: Entropy describes the uncertainty and randomness of a distribution.

$$H(X) = -\sum_{i=1}^{N} p(x_i) log_2(p(x_i)) \tag{6}$$

where $p(x_i)$ is the occurrence probability of x_i

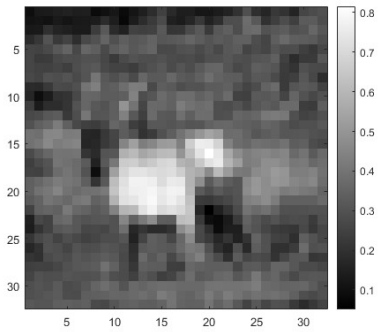

Fig. 3: Gray-scale image

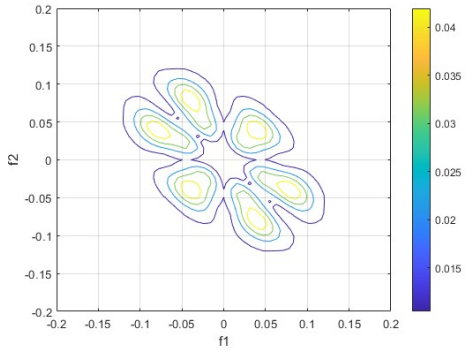

Fig. 4: Bispectrum Visualization

4) Bispectrum: In High-Order-Statistics (HOS), n^{th}-order spectrum is defined as the $(n-1)$-DFT (Discrete Fourier Transform) of the n^{th}-order cumulants. In reality, finite size of data hardens the calculation of cumulants. For this reason, Bispectrum $C_3^x(f_1, f_2)$ is calculated as:

$$C_3^x(f_1, f_2) = X(f_1)XK(f_2)X^*(f_1 + f_2) \tag{7}$$

where $X(f)$ is the Fourier Transform of $x(t)$ and $X^*(f)$ denotes the complex conjugate of X. Bispectrum reveals Quadratic Phase Coupling (QPC) phenomena. Two frequency components a and b are quadratically phase coupled when there is a component c for which $f_c = f_a + f_b$ and $\phi_c = \phi_a + \phi_b$, where f and ϕ stand for frequency and phase, respectively.

B. Features Extraction

First of all, pixels are normalized by dividing them by 255. Let's suppose that pixels of each RGB channel follow a random distribution. Skewness, kurtosis, and entropy are calculated for these distributions. Besides, RGB images are converted to gray-scale using this formula:

$$0.2989 * R + 0.5870 * G + 0.1140 * B \tag{8}$$

For the grayscale images, bispectrum is calculated using the Direct Method and *bispecd* function from HOSA Toolbox for MATLAB. Mean value, max value, and number of peaks of bispectrum magnitude are extracted. In Fig.4, bispectrum of a random sample (Fig.3) is plotted.

C. Loss Functions

1) Mean Squared Error (MSE): The MSE calculates the average squared difference between the estimated and the true value.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{9}$$

where:
y_i is the actual value for the i-th data point
$\hat{y}_i$ is the predicted value for the i-th data point

2) Structural Similarity Index Measure Loss (SSIM Loss): The structural similarity index measure (SSIM) calculates three comparison measurements between the samples: luminance, contrast, and structure.

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{10}$$

where:
μ_x is the pixel sample mean of window x
μ_y is the pixel sample mean of window y
σ_x^2 is the variance of window x
σ_y^2 is the variance of window y
σ_{xy} is the covariance of windows x and y
C_1, C_2 are constants to stabilize division

The SSIM takes values from -1 to 1, where 1 indicates perfect similarity, 0 indicates no similarity, and -1 indicates perfect dissimilarity in structure. So the SSIM loss is defined as:

$$\text{SSIM Loss} = 1 - SSIM \tag{11}$$

The SSIM loss takes values from 0 to 2, where 0 indicates perfect similarity, 1 indicates no similarity, and 2 indicates perfect dissimilarity in structure. For RBG images, SSIM is calculated for each channel, and the final SSIM is the average of these three SSIM.

In reality, the SSIM loss is implemented in Python as:

$$\text{SSIM Loss} = 1 - \text{tf.reduce\_mean(SSIM)} \tag{12}$$

so SSIM values of all samples in a batch are averaged and one single value is returned for every batch.

D. Autoencoder

To begin with, a hybrid autoencoder is built (Fig.5). Its encoder structure is inspired by SAM12 neural network from the first exercise. Thus, there is a CNN subnet and four MLPs, one for each statistical size. The output of these are concatenated and used as input to the decoder.

Also, a simple convolutional autoencoder is trained. Its encoder structure is identical to the convolutional part of encoder of hybrid autoencoder, with the only difference that the last dense layer has $Sigmoid$ as activation function instead of $Relu$. Its decoder is identical to the decoder of hybrid autoencoder. Furthermore, the reconstruction is approached just by using PCA. Their results are compared to those of the hybrid autoencoder.

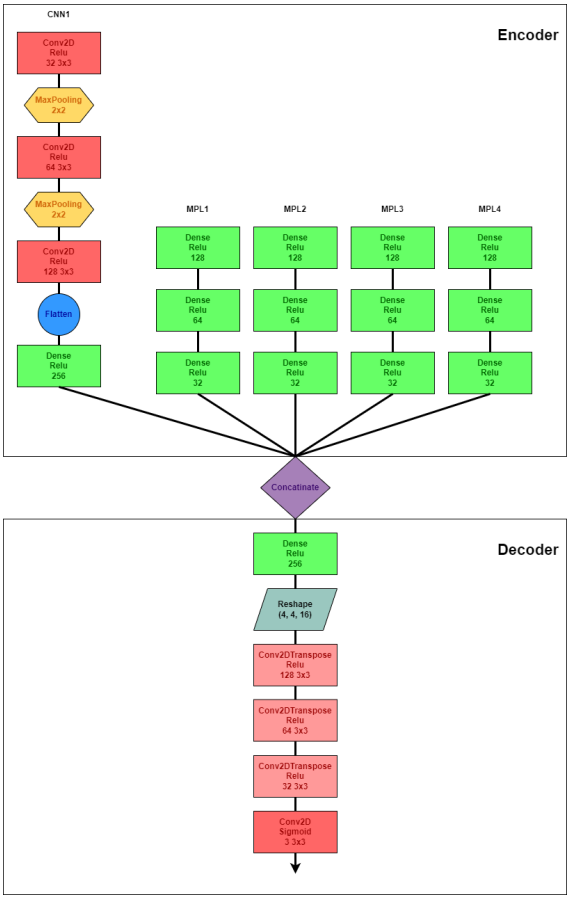

Fig. 5: Structure of Hybrid Autoencoder

A stride of $(2, 2)$ is used in both decoders' convolutioal transpose layers so that both the height and width are upsamled x2 times. Epochs are set equal to 50, batch size equal to 128. PCA is used so 90% of the features will finally remain.

All models has MSE loss function. The convolutional autoencoder is also trained using SSIM loss function (Conv AE SSIM) and compared with MSE (Conv AE MSE).

Finally, all models are trained and tested with the whole train and test sets.

III. Conclusions

In Fig.6 reconstructed images of each model are compared to the original ones. As seen, the convolutional autoencoder is slightly better than the hybrid autoencoder. This can also be confirmed by the fact that loss of hybrid autoencoder is slightly higher that the loss of convolutional autoencoder after 50 epochs of training (Fig.7).

The reconstruction with PCA seems to be the best. This is probably due to the fact that 90% of features still remain after dimensionality reduction.

TABLE I: Training Time of Reconstruction Models

Recontruction Model	Training Time (sec)
Hybrid Autoencoder	3169.55
Convolutional Autoencoder MSE	3253.50
Convolutional Autoencoder SSIM	4225.83
PCA reconstruction	113.38

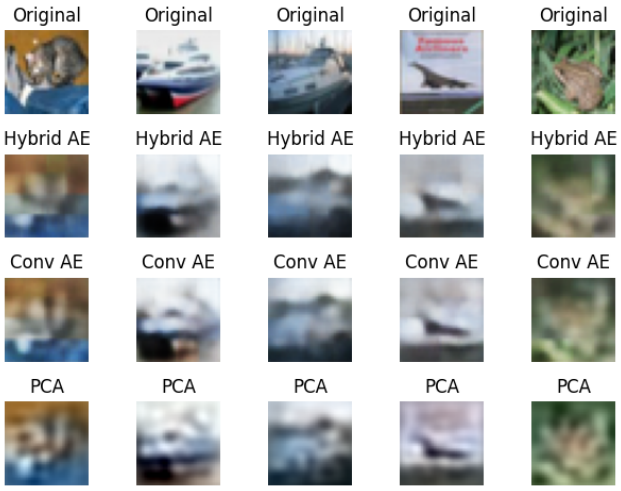

Fig. 6: Reconstructions Comparison

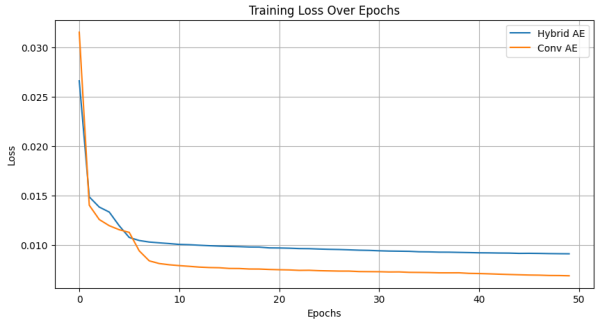

Fig. 7: Autoencoders Loss over each epoch

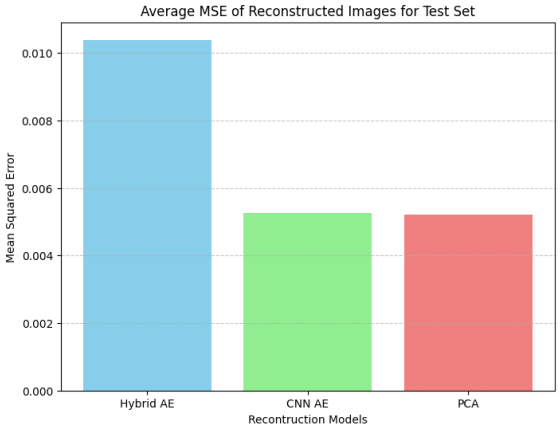

Fig. 8: Average MSE of test set reconstruction

TABLE II: Average MSE of test set reconstruction

Recontruction Model	Hybrid AE	Conv AE	PCA
Average MSE	0.0104	0.0053	0.0052

Each of the three models is tested on the test set, and the average Mean Square Error is calculated (Fig. 8). PCA is slightly better than Conv AE. As mentioned before, the reason for this is probably the fact that 90% of the features remain after dimensionality reduction.

In contrast with the previous two exercises where statistics were helping the model to correctly classify the samples, in this case statistics do not benefit the reconstruction.

When it comes to the convolutional autoencoder using SSIM loss, its performance is not as good as that of Conv AE MSE (Fig.10). These two autoencoders are tested on the test set and the average MSE is calculated(Fig.11). The Conv AE SSIM has a higher average MSE than Conv AE MSE. This probably occurs because the CIFAR10 dataset has very low-resolution images (32x32) and each pixel represents a relatively large portion of the image. Thus, errors in pixel intensity which are captured by MSE, already imply structural differences in low-resolution images. The SSIM uses a 11x11 pixels sliding window. On CIFAR10 32x32 images, this window size might be too large relative to the image size. This could make SSIM less effective at capturing local patterns in such small images. For these reasons, Conv AE SSIM ends up behaving worse than Conv AE MSE.

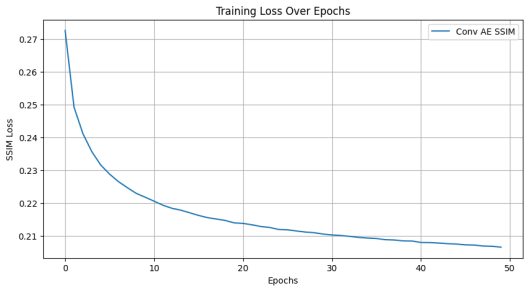

Fig. 9: SSIM Loss over each epoch of Conv SSIM training

Fig. 10: Reconstruction of Conv MSE vs Conv SSIM

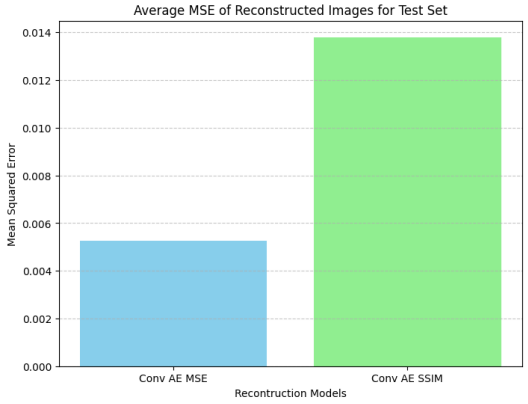

Fig. 11: Average MSE of test set reconstruction of Conv MSE vs Conv SSIM

Note 1: All models are implemented in Python with the help of TensorFlow, an open-source machine learning library.

Note 2: Statistics have been calculated in the 1st project.