DeSTIN: Stacked Convolutional Autoencoder with Variable Noise

- Tejas Khot GSoC Mentor: Yuhuang Hu

Motivation:

The theoretical proposal of DeSTIN describes its ability of learning spatio-temporal features efficiently in an hierarchical fashion using an online training algorithm. DeSTIN is supposed to serve as an unsupervised pre-training algorithm that extracts optimal features from data which are then passed to a supervised learning algorithm. As of today, the temporal aspect of DeSTIN has not been implemented in its Python fork and the focus has been on learning spatial features. After conducting multiple tests with a wide range of configurations, we realize that DeSTIN in its current form is not able to train its higher layers; the features learnt by higher layers have produced low classification accuracies with SVM. The classification accuracy did not show any any improvement after pre-processing the data using ZCA whitening. Overall, the best result achieved as of my knowledge is 47% classification accuracy on test data and 55% classification accuracy on training data. This was achieved by keeping 100, 75, 50, 25 centroids in layer 1 to layer 4. We could expect a slightly better performance by increasing the number of epochs and centroids by multiple times but that is computationally inefficient.

Owing to these reasons, we propose an alternative implementation of DeSTIN using stacked convolutional autoencoders.

Denoising Autoencoder:

Without any additional constraints, conventional auto-encoders learn the identity mapping. This problem can be circumvented by using denoising autoencoders (DAEs) trying to reconstruct noisy inputs. Training involves the reconstruction of a clean input from a partially destroyed one. Input x becomes corrupted input $\bar{x}$ by adding a variable amount v of noise distributed according to the characteristics of the input image. Common choices include binomial noise (switching pixels on or off) for black and white images, or uncorrelated Gaussian noise for color images. The parameter v represents the percentage of permissible corruption. The autoencoder is trained to denoise the inputs by first finding the latent representation

$$h = f_{\theta}(\bar{x}) = \sigma(W\bar{x} + b)$$

from which to reconstruct the original input

$$y = f_{\theta'}(h) = \sigma(W' h + b')$$

Convolutional Autoencoder:

Fully connected Autoencoders(AEs) and Denoising Autoencoders(DAEs) both ignore the 2D image structure. This is not only a problem when dealing with realistically sized inputs, but also introduces redundancy in the parameters, forcing each feature to be global (i.e., to span the entire visual field). However, the trend in vision and object recognition adopted by the most successful models is to discover localized features that repeat themselves all over the input. Convolutional Autoencoders(CAEs) differs from conventional AEs as their weights are shared among all locations in the input, preserving spatial locality. The reconstruction is hence due to a linear combination of basic image patches based on the latent code.

The CAE architecture is intuitively similar to the one described above except that the weights are shared. For a mono-channel input x the latent representation of the k-*th* feature map is given by

$$h^k = \sigma(x * W^k + b^k) \tag{1}$$

where the bias is broadcasted to the whole map, σ is an activation function and * denotes the 2D convolution. A single bias per latent map is used, as we want each filter to specialize on features of the whole input (one bias per pixel would introduce too many degrees of freedom). The reconstruction is obtained using

$$y = \sigma\Big(\sum_{k \in H} h^k * \tilde{W}^k + c\Big) \tag{2}$$

where again there is one bias c per input channel. H identifies the group of latent feature maps; $\tilde{W}$ identifies the flip operation over both dimensions of the weights. The 2D convolution in equation (1) and (2) is determined by context. The convolution of an $m \times m$ matrix with an $n \times n$ matrix may in fact result in an $(m+n-1) \times (m+n-1)$ matrix (full convolution) or in an $(m-n+1) \times (m-n+1)$ (valid convolution). The cost function to minimize is the mean squared error (MSE):

$$E(\theta) = \frac{1}{2n}\Big(\sum_{i=1}^{n} (x_i - y_i)\Big)^2 \tag{3}$$

Just as for standard networks the backpropagation algorithm is applied to compute the gradient of the error function with respect to the parameters. This can be easily obtained by convolution operations using the following formula:

$$\frac{\partial E(\theta)}{\partial W^k} = x * h^k + \tilde{h}^k * \delta y \tag{4}$$

δh and δy are the deltas of the hidden states and the reconstruction, respectively.
The weights are then updated using stochastic gradient descent.

Stacked Convolutional Autoencoders (SCAE):

Several AEs can be stacked to form a deep hierarchy. Each layer receives its input from the latent representation of the layer below. As for deep belief networks, unsupervised pre-training can be done in greedy, layer-wise fashion. Afterwards the weights can be fine-tuned using backpropagation, or the top level activations can be used as feature vectors for SVMs or other classifiers. Analogously, a CAE stack (SCAE) can be used to initialize a CNN with identical topology prior to a supervised training stage.

Max-Pooling:

For hierarchical networks in general and CNNs in particular, a max-pooling layer is often introduced to obtain translation-invariant representations. Max-pooling down-samples the latent representation by a constant factor, usually taking the maximum value over non overlapping sub-regions. This helps improving filter selectivity, as the activation of each neuron in the latent representation is determined by the "match" between the feature and the input field over the region of interest. Max-pooling was originally intended for fully-supervised feedforward architectures only.

Here we introduce a max-pooling layer that introduces sparsity over the hidden representation by erasing all non-maximal values in non overlapping subregions. This forces the feature detectors to become more broadly applicable, avoiding trivial solutions such as having only one weight "on" (identity function). During the reconstruction phase, such a sparse latent code decreases the average number of filters contributing to the decoding of each pixel, forcing filters to be more general.

Architecture:

We try to keep the architecture similar in spirit to DeSTIN by having a 4 layer hierarchy. Each layer is a convolutional autoencoder having a ReLU activation for encoding convolution and a sigmoid activation for decoding convolution. There is also a same sized max pooling layer in between encoding and decoding to introduce sparsity. There is a downsampling max pooling layer between every two layers i.e. the output from encoder of layer n is max pooled and downsampled to smaller dimensions before being fed to the encoder of layer $n + 1$. Every layer performs *same* convolutions i.e. the dimensions of all feature maps are same as the input dimensions unlike valid(smaller) and full(larger) convolutions. The input at each layer is subjected to variable noise(corruption level) that changes every epoch to eventually settle for the optimum value. We minimize the cost function that is the mean squared error in reconstruction as indicated in equation (3) above and use L2 regularization on the parameters. In our experiments, we have the following configuration:

Layer 0 :	input size (32,32), filter size (4,4)
Max_pool_0:	pool_size (4,4)
Layer 1 :	input size (8,8), filter size (2,2)
Max_pool_1:	pool_size (2,2)
Layer 2 :	input size (4,4), filter size (2,2)
Max_pool_2:	pool_size (2,2)
Layer 3 :	input size (2,2), filter size (2,2)
Max_pool_3:	pool_size (2,2)
Layer 4 :	input size (1,1), filter size (1,1)

number of filters is a hyperparameter that is being varied throughout current experiments

Training scheme:

In order to keep up with the spirit of DeSTIN, the training is done with a top-down approach in an online manner. We do not use greedy layer wise training but instead train each layer of the hierarchy every time an input is fed. To emphasize on improving the reconstruction, each layer is trained 8 times per epoch i.e. an input traverses top down 8 times each epoch. This training pattern distinguishes us from the conventional autoencoder training methods using greedy layer wise training.

Testing Architecture:

We use a LeNet like Convolutional Neural Network for performing supervised learning. The architecture is as follows: We retain all 5 layers of the unsupervised learning hierarchy. The output of the last layer is flattened and fed to a ReLU convolutional layer followed by a Softmax layer for classification. Because of the hierarchical nature of our unsupervised model, the top node has dimensions of (1,1) which means, the number of parameters fed to the ReLU convolution layer is same as the number of filters used while pre-training the last layer. This offers immense flexibility for reduction of parameters to be used for supervised training based on computational resources. We use Stochastic Gradient Descent for fine tuning.

Data:

We are currently testing on the CIFAR-10 dataset using as input both raw images(normalized) and images after using ZCA whitening.

Code:

The source code along with outputs and test results can be found in this GitHub repository: https://github.com/Tejas-Khot/ConvAE-DeSTIN

Test Results:

In the few tests performed till now, the highest accuracy obtained is ~ 64% using 128 filters in every layer and non-whitened data. More tests are in progress and results will be updated on the repository immediately on completion.

Future Directions:

This model captures local features in images and it is possible to, say, point at a node and see what features it has learnt. On the downside, this model does not retain the context of obtaining features as well as the model recently proposed by Dr. Ben Goertzel, at least in theory. Furthermore, we are yet to figure out how to incorporate the temporal aspect in this hierarchy because without it, it’s just DeSIN without the T.