A COMPREHENSIVE STUDY ON UNSUPERVISED FEATURE LEARNING USING DIFFERENT AUTOENCODERS

ABSTRACT

Unsupervised feature learning is attaining a lot of attention due to its efficiency, effectiveness and outstanding performance in deep neural architecture. This type of learning is used for learning features or attributes from unlabeled input data with an intention to discovering low-dimensional features or representations from a high-dimensional input data. Deep learning algorithms are designed to make them learn useful feature representations automatically from massive amount of unlabeled data, avoiding a lot of time-consuming engineering to label inputs. Autoencoders are one of the most effective and commonly used deep learning architectures among all the models in use. In this study, use of autoencoders in solving the problem of unsupervised feature learning is researched and some most recent algorithms are discussed with their implementation in different interesting domains.

AUTOENCODERS

An autoencoder is an unsupervised machine learning algorithm that takes an image as input and reconstructs it using fewer number of bits. While reconstructing, they create a feature space where the essential parts of the data are preserved, while non-essential (or noisy) parts are removed. Use of this algorithm forces a neural network model to learn meaningful representation of the input.

Architecturally, the simplest form of an autoencoder is a feedforward, non-recurrent neural network very similar to the multilayer perceptron (MLP) – having an input layer, an output layer and one or more hidden layers connecting them – but with the output layer having the same number of nodes as the input layer, and with the purpose of reconstructing its own inputs instead of predicting the target value Y given inputs X. The concept of autoencoders was first introduced in [8].

An autoencoder always consists of two parts, the encoder and the decoder.

1. ENCODER WITH EXAMPLE

This is the part of the network that compresses the input into a fewer number of bits. The space represented by these fewer number of bits is called the “latent-space” and the point of maximum compression is called the bottleneck. These compressed bits that represent the original input are together called an “encoding” of the input.

Let's assume we have input images of size 28x28 pixels. The images are first converted to matrices of size 28 x 28. We reshape the image to be of size 28 x 28 x 1, convert the resized image matrix to a linear array, rescale the pixel intensity values between 0 and 1, and feed this as an input to the network. The encoder transforms the 28 x 28 x 1 image to a 7 x 7 x 32 image. 7 x 7 x 32 image is a point in a 1568 (because 7 x 7 x 32 = 1568) dimensional space. This 1568-dimensional space is called the bottleneck or the latent space.

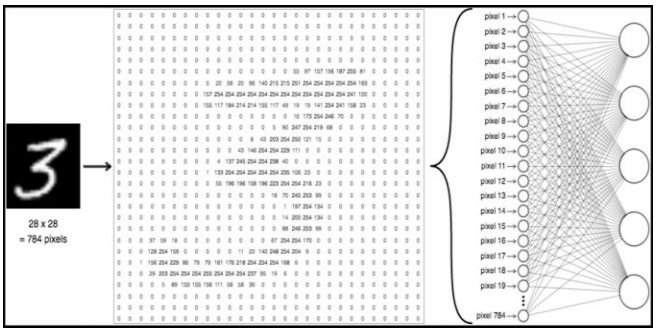


Figure 1: Input pre-processing

In figure 1, input image of hand-written digit 3 is converted to a 28x28 matrix by the pixel intensity values. There are 28x28 = 784 pixels altogether, therefore 784 nodes in the input layer where each node holds each of the pixel values from the linear array. Transformation of 28 x 28 x 1 image to a 7 x 7 x 32 image is done in the convolution and pooling layers.

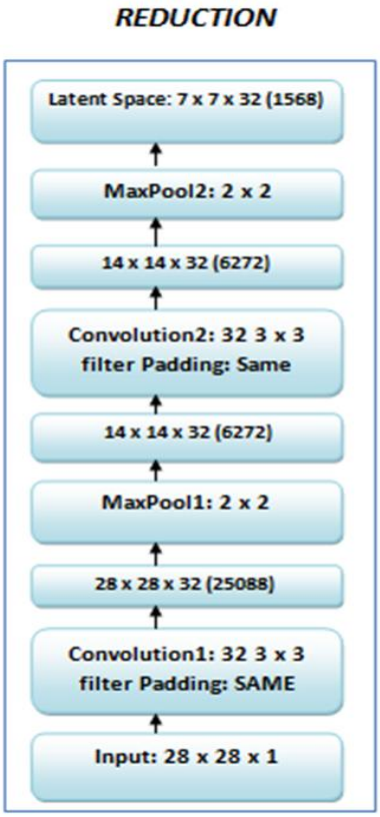


Figure 2: Encoding steps

In figure 2, the first convolution layer has 32 filters or windows of size 3x3. The primary purpose of Convolution is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. Small squares are scanned by filters which is a 3x3 matrix containing numeric values generally between -1 to +2. The filter studies successively every pixel of the image. For each of them, which we will call the "initial pixel", it multiplies the value of this pixel and values of the 8 surrounding pixels by the filter corresponding value. Then it adds the results, and the initial pixel is set to this final result value.

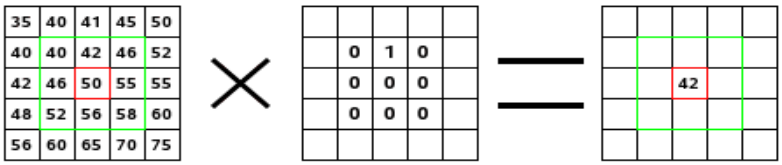


Figure 3: Convolution in action

In figure 3, each pixel is marked with its value. The initial pixel has a red border. The filter action area has a green border. The image in the middle is the filter and, on the right is the convolution result. Here is what happened: the filter reads successively, from left to right and from top to bottom, all the pixels of the filter action area. It is multiplied the value of each of them by the filter corresponding value and added results. Therefore, the initial pixel has become 42: (40*0) + (42*1) + (46*0) + (46*0) + (50*0) + (55*0) + (52*0) + (56*0) + (58*0) = 42. The filter doesn't work on the image but on a copy of the input. Each convolution result is called Activation Map or Feature Map. The more number of filters we have, the more image features get extracted and the better the network becomes at recognizing patterns in unseen images. Different values of the filter matrix will produce different Feature Maps for the same input image. In this example, for 32 filters, there are total 32 feature maps. Therefore, the dimension of transformed image is 28x28x32 after first convolution layer.

In figure 2, there is another layer called Maxpool1. Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. In case of Max Pooling, a spatial neighborhood is defined (for example, a 2×2 window) and the largest element from the rectified feature map within that window is taken.

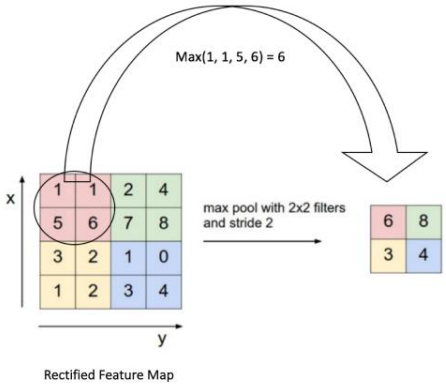


Figure 3: Max-pooling

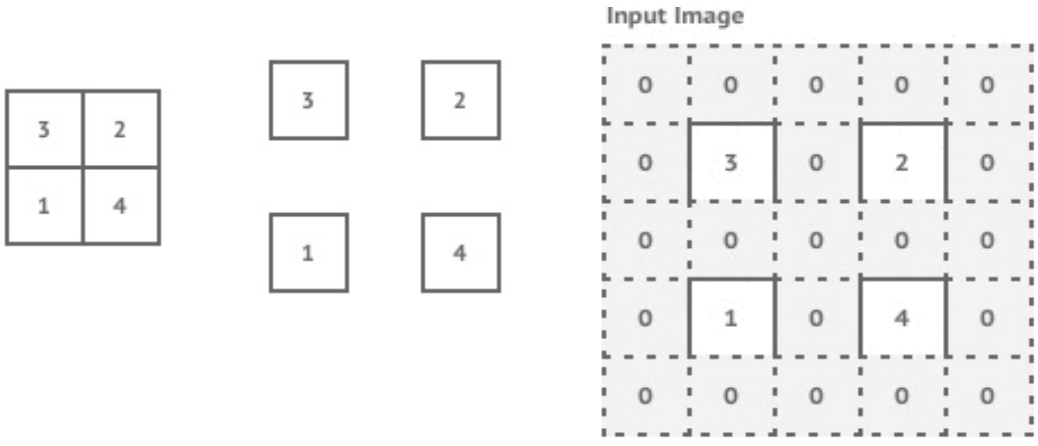
In figure 4, we slide the 2 x 2 window by 2 cells (also called 'stride') in a single feature map obtained from the convolution layer and take the maximum value in each region. As shown in figure 2, this reduces the dimensionality of each feature map. Since pooling operation is applied separately to each feature map, the dimension of the image after the first Max Pool layer is 14x14x32. Following the same procedure in layer 2 of

convolution layer and Max Pool layer the input image is finally transformed into a 7x7x32 image.

2. DECODER WITH EXAMPLE

This is the part of the network that reconstructs the input image using the encoding of the image. The decoder does the exact opposite of an encoder; it transforms this 7x7x32 = 1568-dimensional vector back to a 28x28x1 image. We call this output image a "reconstruction" of the original image. In figure 5, there are two Conv2d and two Conv2d_transpose layers in the decoder architecture. Conv2d_transpose layer works as the transpose convolution to use a transformation going in the opposite direction of a normal convolution, to project feature maps to a higher-dimensional space.

As example, we can map the result of the latent space in encoder from a 4-dimensional space to a 16-dimensional space, while keeping the connectivity pattern of the convolution. In our example, after applying 2 convolution layers in encoder our image resolution is reduced to ¼ th of its original size. To convert it back to the resolution of original input image, transpose convolution layers are used. One way to obtain this is to use convolution layers with ½ stride, which involves moving the filter window across by half a pixel each time. This results in an output image that has double the resolution of the input. In reality we can't actually move our kernel along by half a pixel, so instead we pad the existing pixels with zero-valued pixels and carry out a normal stride 1 convolution (moving the filter window across by one pixel each time) as shown in an example in figure 6.



(a) **(b)**

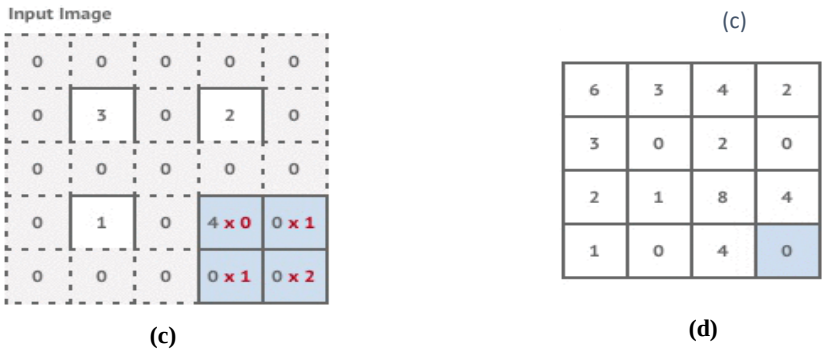


Figure 6: (a): A 2x2 image resulted from convolution. (b)-(c): zero-padding to transform it to a 5x5 matrix. (d): Apply a 2x2 matrix as the filter to obtain 4x4 output image.

In figure 6. (d), a filter of $\begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$ slides through the zero-padded image and is multiplied the value of each of the region by the filter corresponding value and added results. In the above example, the last pixel value of the resulting feature map is obtained by following, 4x0 + 0x1 + 0x1 + 0x2 = 0. Applying this same technique in our autoencoder example, our 7x7x32 image is converted to 14x14x32 after first Conv2d_transpose layer and then 14x14x32 image is converted to 28x28x32 image after the second Conv2d_transpose layer. Finally, a Conv2D convolution layer is applied with 1 3x3 filter to obtain transform this 28x28x32 image to 28x28x1 image.

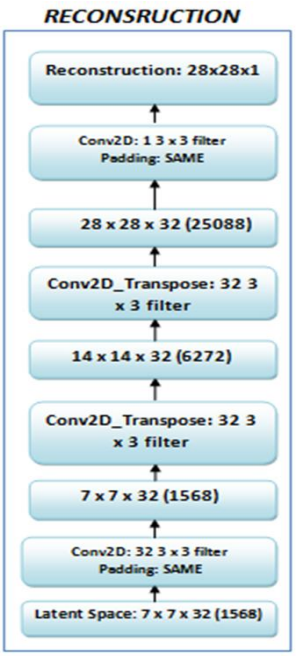
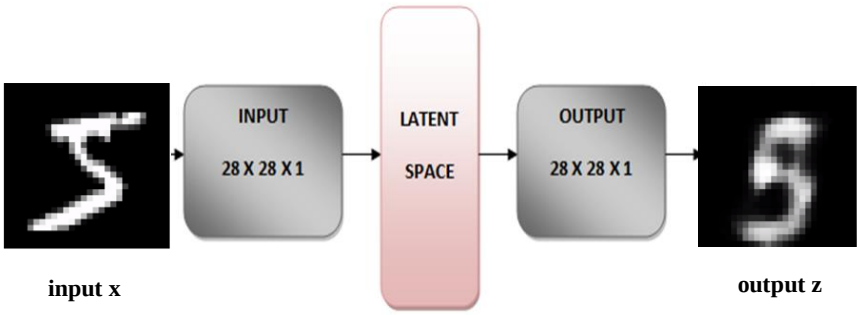


Figure 5: Decoding of input image



input x **output z**

Figure 7: Architecture of Autoencoder

In figure 7, the entire process of an autoencoder is shown. Input image x is passed through the autoencoder network and the reconstructed image z is returned as the output. In the latent space, the whole process of encoding and decoding is done.

OPTIMIZATION

Calculating loss function

After each iteration, the reconstructed output image is compared to the original input image by doing the **pixel-wise subtraction**. Total difference gives the Reconstruction Loss with respect to the input image. The parameters and specially the **values of the filters** of each layer of this model are optimized to minimize the average reconstruction error. This iteration and optimization is carried out by stochastic gradient descent algorithm.

$$\theta^* , \theta'^* = argmin_{\theta^*,\theta'} \frac{1}{n} \sum_{i=1}^{n} L\left(x^i, z^i\right) ,$$

$$\text{where } L(x,z) = ||x - z||^2 \qquad (1)$$

FEATURE REPRESENTATION USING AUTOENCODER

An autoencoder neural network tries to learn an approximation function $h_w(x) \approx x'$, where x' is the reconstructed output that is similar to original input x. Then the interesting structure about the data will be discovered in the hidden or semantic layers. For example, suppose the inputs x are the pixel intensity values from a 10x10 image i.e. 100 pixels, and there are 20 hidden nodes in semantic layer and the outputs layer also have 100 pixels. Since there are only 20 hidden nodes, the network is forced to learn an encoded representation of the input. In addition to that, if some of input features are correlated to each other, then autoencoder is able to discover some of those correlations.

While encoding, each filter is responsible for extracting different features from the input image. Figure 8 is the visualization of the features extracted by each of the 32 filters used in the Encoder in our example.

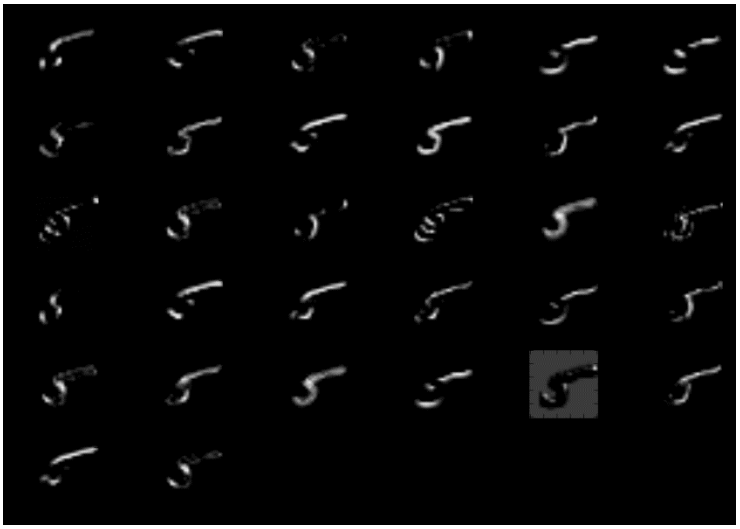


Figure 8: Features learned by 32 filters in Encoder

As shown, some filters have learnt to recognize edges, curves, etc. from the input image. This is the output from the latent space of the Encoder.

Based on the features learned by the Encoder, the Decoder tries to reconstruct the original input image, however with limited information gained from the Encoder. In other words, the decoder is forced to create the output image from the compressed version of the original image. We are interested only on the features learned by the filters or the compressed version of input sample through this entire process rather than the reconstructed output. In our example, we have reduced the dimension of the input from 28x28 matrix to a 7x7 matrix and forced our model to learn some interesting features which are the most significant ones to classify an unknown image of this particular class.

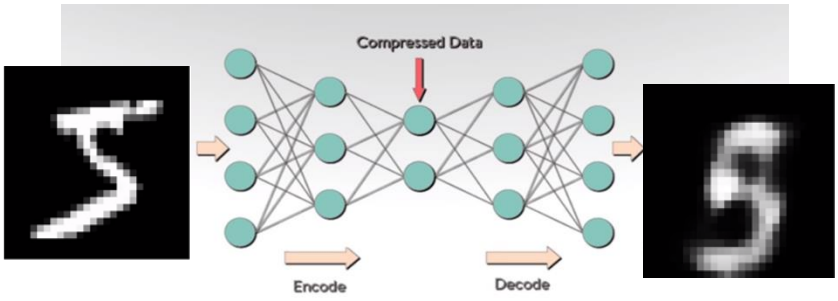


Figure 9: Feature representation in Autoencoder

In figure 9, the input image is transformed to a 2-dimensional input in the bottleneck of the latent space. On the decoder side, the original input is reconstructed using only these two features from

the resulting 2-D image. As a result, the output looks fuzzy as it has lost some of the information/features that are present in the original image.

In figure 10, different output images are produced from a 2-dimensional, 5-dimensional and 10-dimensional latent space, where it is evident that 5-D and 10-D inputs produced sharper and clear outputs compared to 2-D input.

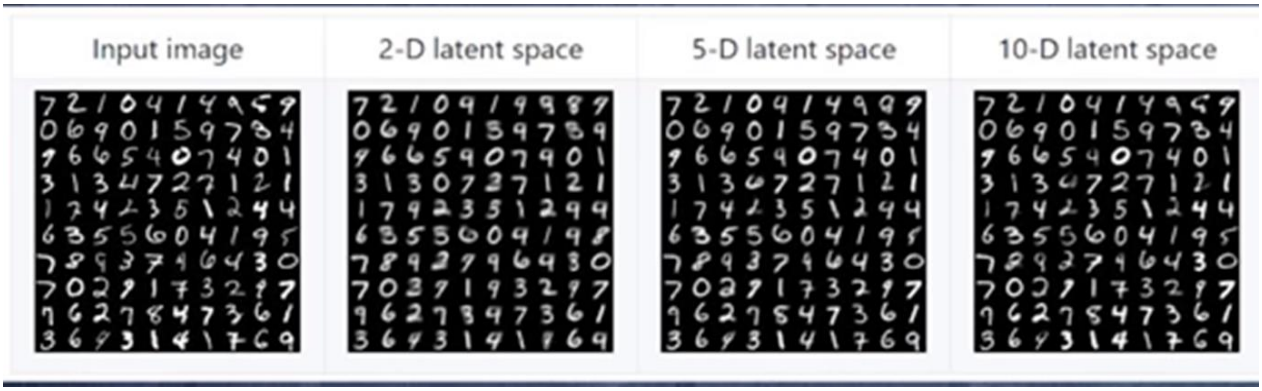


Figure 10: Reconstructed image using different feature space

EXTRACTING ROBUST FEATURES WITH DENOISING AUTOENCODERS

In [1], authors have used a modified algorithm based on autoencoder in order to extract more robust features from input images. They have referred this modified autoencoder as Denoising Autoencoder (DAE). Instead of an original input, they have trained the autoencoder model to reconstruct a clean "repaired" input from a corrupted, partially destroyed version of the original one. This is done by first corrupting the initial input x to get a partially destroyed version x'. In their experiments, they considered the following corrupting process, parameterized by the desired proportion *v* of "destruction": for each input x, a fixed number *vd* of components are chosen at random where d is the dimension of the original input, and their value is forced to 0, while the others are left untouched. All information about the chosen components is thus removed from that particular input pattern, and the autoencoder will be trained to "fill-in" these artificially introduced "blanks". The corrupted input x' is then mapped, as with the basic autoencoder, to a hidden representation $y = f(x')$ from which $z = g_{\theta'}(y)$ is reconstructed. As before, the parameters are trained

to minimize the average reconstruction error between z and original input x over a training set, i.e. to have z as close as possible to the uncorrupted input x. But the key difference is that z is now mapped from the representation of x' rather than x and thus the result of a stochastic mapping of x.

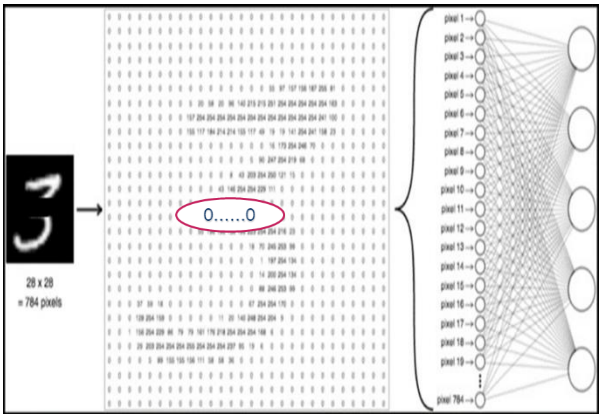


Figure 11: Passing corrupted input to autoencoder

Figure 12: Different corrupted versions for different value of *vd*

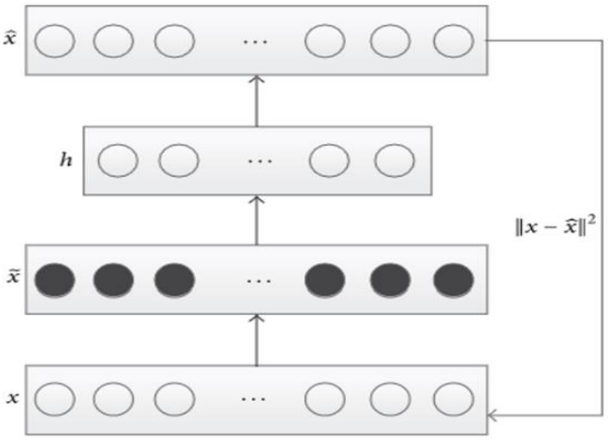


Figure 13: Architecture of Denoising AE

In figure 11, a corrupted version of original input of hand-written digit of "3" has been produced by setting some of the pixel values in the input matrix as zero. Figure 12 shows different corrupted versions of original input "5" for randomly chosen values of *vd*. Finally, figure 13 shows the architecture of the denoising autoencoder where the first layer contains the original input x, next layer converts it to a partially destructed version x' and have been passed through the hidden layers. The output layer reconstructs the x' which is as similar to original input x. The algorithm is given below.

- Key Idea: Train AE to reconstruct a clean "repaired" input from a corrupted, partially destroyed one.

- Algorithm FeatureExtractWithDAE

    1. Pick an input x
    2. Choose a fixed number *vd* randomly for components of x, where v = proportion of destruction and d = dimension of x
    3. Pixel values of *vd* are forced to 0, while the others are left untouched
    4. All information about the chosen components is thus removed from x, and the autoencoder will be trained to "fill-in" these artificially introduced "blanks"
    5. Corrupted image x' is produced
    6. x' is passed to AE as final input
    7. Reconstructed image z is produced
    8. Reconstruction loss is minimized by minimizing the difference between $||x - z||^2$

CONTRIBUTION

Training procedure in [1] for the denoising autoencoder involves learning to recover a clean input from a corrupted version, a task known as denoising. However, their proposed algorithm not only solves the image denoising tasks, but also works as an initialization step for deep learning networks for supervised classification tasks. They have empirically evaluated their algorithm on MNIST dataset with 70,000 images of hand-written digits and concluded that training the autoencoder network on partially corrupted input or input with missing feature values have learned some interesting features of the original input that improved the classification accuracy on a supervised neural network.

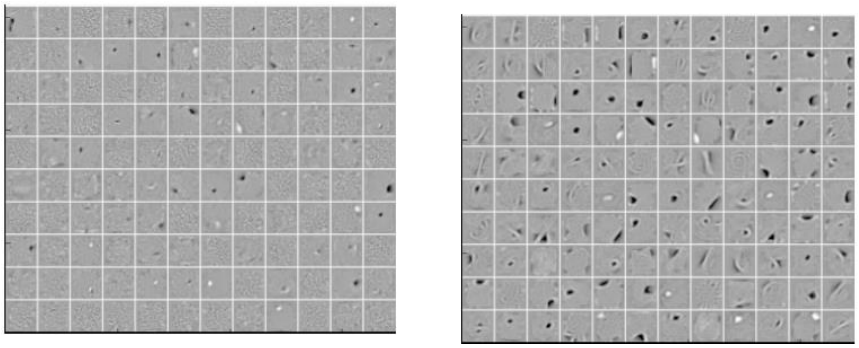


Figure 14: Feature map with (a) no destruction (b) 50% destruction

In figure 14, features extracted by a single feature map is shown for model trained on original input with no destruction compared to input with 50% of the pixel values destructed. It is evident that the later model has learned more meaningful features than the former one. Figure 15 shows features learned by a single neuron which indicates that the model trained on 50% destructed input learns the most robust feature compared to models with 0%, 10% and 20% destructed input images.

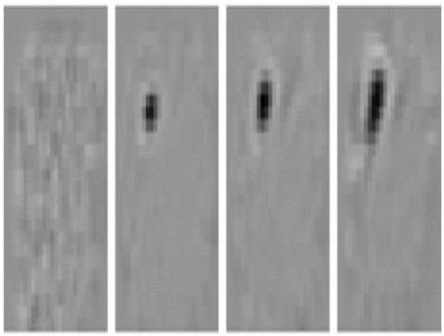


Figure 15: Single Neuron (0%, 10%, 20%, 50% destruction)

STACKED DENOSING CONVOLUTIONAL AUTO-ENCODERS FOR HIERARCHICAL FEATURE EXTRACTION

In [2], authors introduce the stacked Convolutional Auto-Encoder (CAE), which is a hierarchical unsupervised feature extractor that scales well to high-dimensional inputs. Basically, authors have merged the Denoising AE algorithm described in [1] with a convolution neural network where several Denoising autoencoders are stacked in hierarchical manner.

The **algorithm** being used in [2] is as followed. Denoising autoencoders can be stacked to form a deep network by feeding the latent representation in the latent space of the denoising autoencoder found on the layer below as input to the current layer. The unsupervised pre-training of such an architecture is done one layer at a time. Each layer is trained as a denoising autoencoder by minimizing the error in reconstructing its input (which is the output code of the previous layer). Once the first k layers are trained, the k+1-th layer can be trained using the latent representation or features obtained from the k-th layer. For the encoding and decoding part in each of the autoencoders, authors have used Convolution and Max-pooling layers described in the beginning of this paper. The final feature vector a(n) obtained from this deep architecture gives a representation of the original input in terms of higher-order features. The first layer of a stacked autoencoder tends to learn first-order features in the raw input (such as edges in an image). The second layer of a stacked autoencoder tends to learn second-order features corresponding to

patterns in the appearance of first-order features (e.g., in terms of what edges tend to occur together--for example, to form contour or corner detectors). Similarly, higher layers of the stacked autoencoder tend to learn even higher-order features. These features from the stacked autoencoder can then be used for classification problems by feeding a(n) to a softmax classifier in a supervised convolutional neural network.

To give a concrete **example** of the above algorithm, the steps for training a stacked denoising autoencoder with 2 hidden layers for classification of an image of three classes are shown below.

First, we would train an autoencoder on the raw inputs (pixel values) $x^{(k)}$ which is a corrupted/noisy version of the original input x' to learn primary features $h^{(1)(k)}$ on the raw input where k is the kth input sample (Figure 16). Next, we would feed the raw input into this trained autoencoder, obtaining the primary feature representation $h^{(1)(k)}$ for each of the inputs $x^{(k)}$. We would then use these primary features as the "raw input" to another autoencoder to learn secondary features $h^{(2)(k)}$ on these primary features (Figure 17). Following this, we would feed the primary features into the second autoencoder to obtain the secondary feature representations $h^{(2)(k)}$ for each of the primary features $h^{(1)(k)}$ (which correspond to the primary features of the corresponding inputs x(k)). We would then treat these secondary features as "raw input" to a softmax classifier (Figure 18), training it to map secondary features to digit labels for classification. Figure 19 shows the entire architecture of the stacked denoising AE (SDAE).

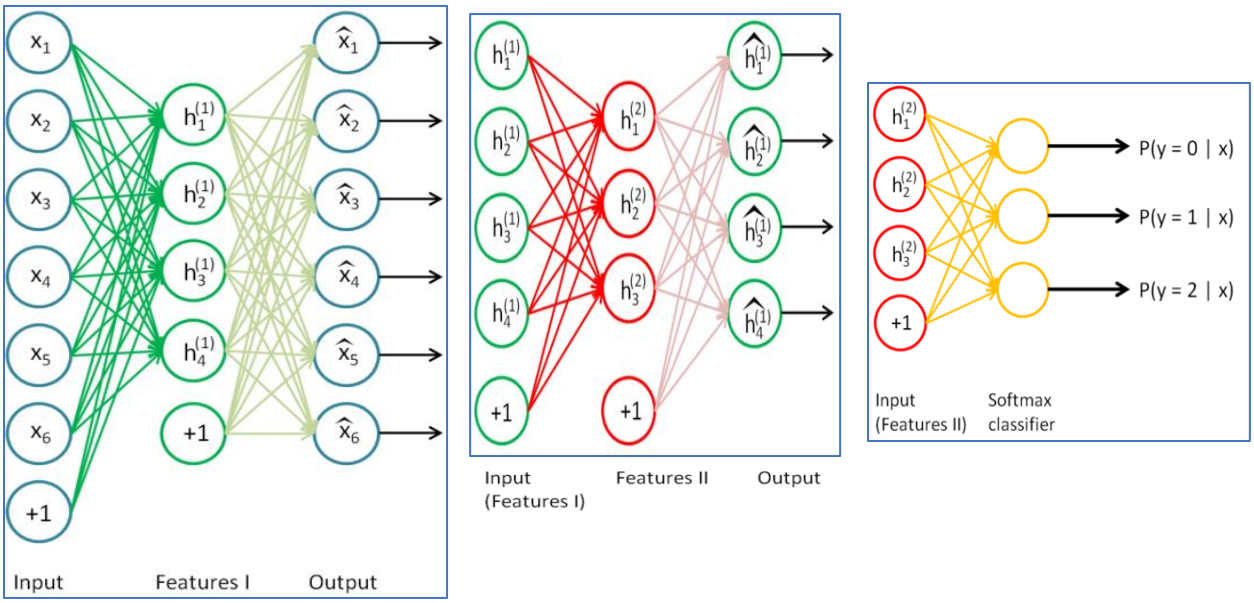


Figure 16, 17, 18: Architecture of AE with single hidden layer

CONTRIBUTION

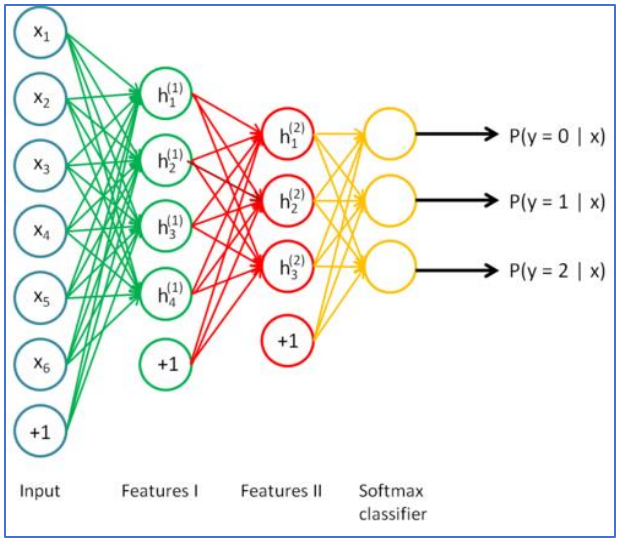


Figure 19: Stacked Denoising AE

Authors have experimented their proposed algorithm like the following, they compared 20 7 × 7 filters (learned on MNIST) of four SDAEs of the same topology but trained differently. The first is trained on original digits (a), the second on noisy inputs with 50% binomial noise (pixel on and off) added (b), the third has an additional max-pooling layer of size 2 × 2 (c), and the fourth is trained on noisy inputs (30% binomial noise) and has a max-pooling layer of size 2 × 2 (d). According to the result in figure 20, authors concluded that the impact of a max-pooling layer is striking (c), whereas adding noise (b) has almost no visual effect except on the weight magnitudes (d). Hence, as for MNIST, only a max-pooling layer guarantees convincing solutions, indicating that max-pooling is essential. They also initialized a new CNN by the learned features and other parameters from this SDAE model and found that it has improved the classification accuracy of CNNs compared to a randomly initialized CNN model.

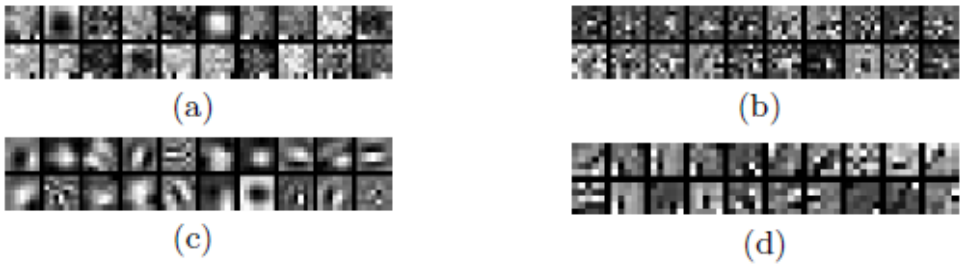


Figure 20: A randomly selected subset of the first layer's filters learned on MNIST to compare noise and pooling. (a) No max-pooling, 0% noise, (b) No max-pooling, 50% noise, (c) Max-pooling of 2x2, (d) Max-pooling of 2x2, 30% noise.

SPARSE AUTOENCODER

In standard autoencoders, there are lesser number of hidden units in the hidden layers than number of nodes in the input layer. As a result, the input is compressed while passing through the hidden layers. However, in sparse autoencoders, the no. of hidden units in each hidden layer in larger than

input nodes which forces the model to map a sparse or overcomplete representation of the input image. In [3], authors have illustrated the architecture of a sparse autoencoder. The simplest sparse autoencoder consists of a single hidden layer, z, that is connected to the input vector x by a weight matrix W forming the encoding step. The hidden layer then outputs to a reconstruction vector $\tilde{x}$, using a tied weight matrix W^T to form the decoder. The activation function is f and b is the bias term of the network explicitly set by programmers. Hence,

$$z = f(Wx + b) \text{ and } \tilde{x} = f(W^T + b').$$

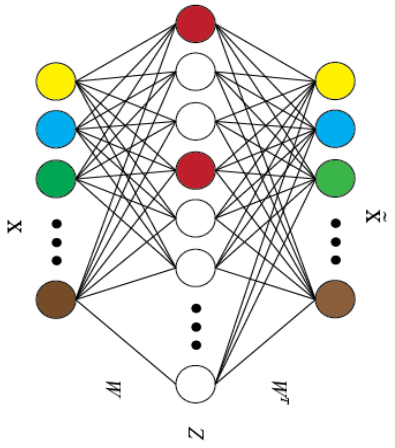


Figure 23: A sparse autoencoder network

Learning occurs via backpropagation on the reconstruction error, $\min\|x-\tilde{x}\|^2$. In figure 23, the input vector x is converted to a sparse representation on the hidden layer as z and then reconstructed as $\tilde{x}$. Now that the network is setup, the next step is to add a sparsifying component that drives the vector z towards a sparse representation. One of the ways is k-sparse autoencoder described in [3]. In this method, in the feedforward phase, after computing the hidden code $z = f(Wx + b)$, rather than reconstructing the input from all of the hidden units, the k largest hidden units are identified and set the others to zero. This identification can be done by using an activation function called ReLU on the hidden units with thresholds that are adaptively adjusted until the k larges activations are identified. That means only a few strongest hidden unit activations will extract features whereas rest of the units will not be fired. The error is then backpropagated only through the k active nodes in z. Iterative Thresholding with Inversion (ITI) is the algorithm to achieve this k largest activations which is extensively described in [3]. The advantage of using sparse autoencoder is the model learns a more global representation of the entire training set rather than learning a local representation for individual inputs (as example features that are specific to a particular input sample). Hence, it prevents models to over-fit by learning sample-specific features.

STACKED SPARSED AUTOENCODERS FOR ORGAN DETECTION IN 4D PATIENT DATA

In [4], authors have used Stacked sparse autoencoder in order to learn unsupervised multi-modal hierarchical features from a 4D Dynamic contrast-enhanced magnetic resonance imaging (DCE-MRI) dataset and used the learned features for organ detection in DCE-MRI medical image dataset. Their 4D dataset consists of a time series of 3D DCE-MRI scans from two studies of liver metastases and one study of kidney metastases each repeated at T=40 time points (each timepoint representing one scan). Dynamic Contrast Enhanced MRI (DCE-MRI) uses a continuous series of images taken before, during and after injection of a contrast agent (patient was instructed to take a single breath) so that the contrast of the successive images changes according to the blood flow and vascular permeability of the tissues can be observed.

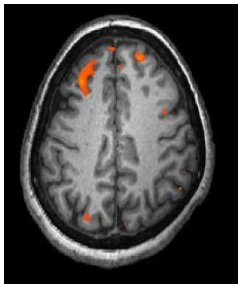


Figure 21: cluster of voxels in a 3D brain image

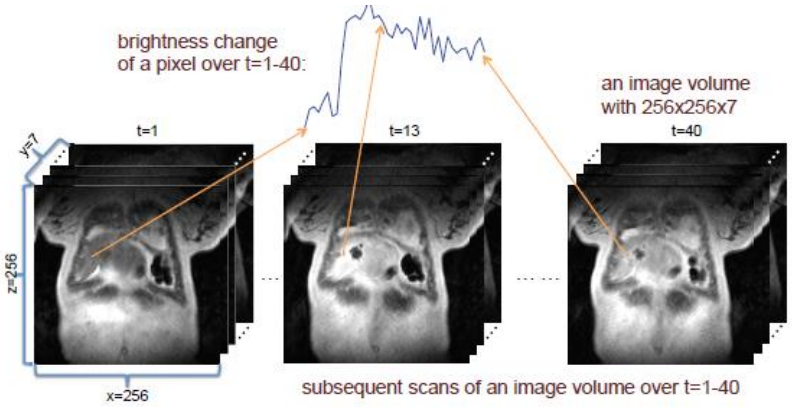


Figure 22: A 4D DCE-MRI scan of a liver patient for a time course

MRI images are 3-D images built up in units called voxels. Each voxel represents a tidy cube of tissue—a 3-D image building block analogous to the 2-D pixel of computers screens, televisions or digital cameras. Each voxel can represent a million cells. Those orange blobs in the image in figure 21 above are actually clusters of voxels—perhaps tens or hundreds of them. Figure 22 is a 4D DCE-MRI scan of a liver patient for a time course $1 \leq t \leq 40$ with volume size of $256 \times 256 \times 7$. Each pixel of an image slice in a volume gives a time series of its brightness over 40 images. The time series represents the perfusion status of the tissue in the voxel and will vary with tissue types.

PREPROCESSING OF INPUT

DCE-MRI data have both temporal and visual or spatial domains. Temporal features are learnt from the organ specific changes in intensity that occur over time, for effects of different contrast agents. Following the intensity of each 3D voxel in a set of *ny* coronal slices of matrix size *nx*×*nz* through T time-points provides a set of *nx* x *ny* x *nz* voxel which they called as "contrast uptake curves", which is like a tiny box in each scan for each patch. On the other hand, features in the spatial domain are identified by sampling ordinary 2D image "patches". For temporal feature learning, approximately 1.3×10^4 time series signals were randomly sampled from the complete set of contrast uptake curves in the training dataset, excluding the background and regions affected by breathing motion. A pixel is regarded as background or a region affected by breathing motion (contrast agent), when its image intensity falls below 10% of the maximum intensity in the image within the imaging time-course (T = 40 image volumes). Temporal features are learnt by the single-layer sparse autoencoder from the samples, where each time series is a 40-element input vector, and the N temporal features are the individual weights $wj \in R^{40\times1}$ (rows) of the weight matrix $W \in R^{N\times40}$.

TEMPORAL FEATURE LEARNING

In [4], authors have taken the following steps for temporal feature learning, (a) performed dimensionality reduction in the temporal space with a single layer sparse autoencoder network (b) did vector quantization of the features with a sigmoid activation function and finally (d) mapped the result of the vector quantization into RGB space. In figure 24, visualization of dimensionality reduction with a single layer sparse autoencoder is shown, where the size of the DCE-MRI temporal dimension is reduced from 40 to 16 elements. Different tissue types are visualized in different colors, and a liver tumor is represented as a complex pattern within liver (a), (b). It shows that ambiguities in identifying some tissue types of different organs remain, with some sub-regions of the aorta, heart, liver and spleen being represented as the same cyan color.

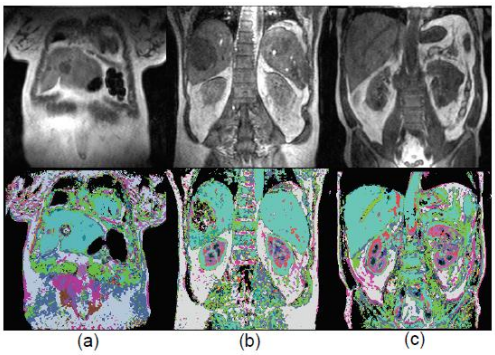


Figure 24: Visualization of temporal feature learning

Figure 25 shows 256 sparse representation of (a) temporal and (b) 8 × 8 size visual feature set learned by a feature map during unsupervised sparse feature learning.

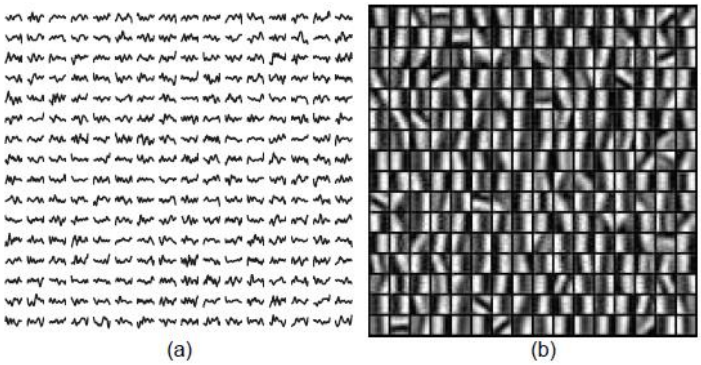


Figure 24: Features learned by feature maps

SPATIAL FEATURE LEARNING

Authors have used stacked sparse autoencoder integrated with max-pooling layers in order to teach their model to learn multiple object parts or organs from the input. As an example, a conceptual visualization of 3×3 maxpooling for 3×3 size patches in a 2D feature space using the visualization of Figure 24 (c) is shown in Figure 26 (a) that how it can capture the same feature for the patches at different locations in the kidney in Figure 24 (c), and for a 3 × 3 patch in 3D temporal feature space with 256 temporal features is visualized in Figure 26 (b). Figure 27 and figure 28 show the overall architecture of the visual feature learning networks and temporal feature extraction networks respectively. The first and second hidden layers are unsupervised feature learning

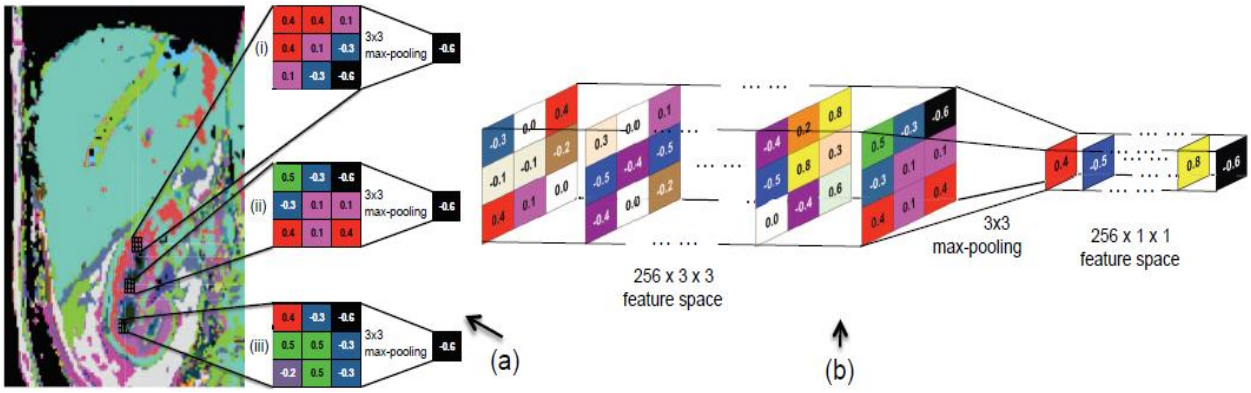


Figure 25: Architecture of Spatial feature learning

networks described earlier in Stacked AE section, and the third hidden layer is a classification network, which is trained with supervision to classify patches of different organs. The classification task is to detect multiple organs and label them as either liver, heart, kidney or spleen from the input image.

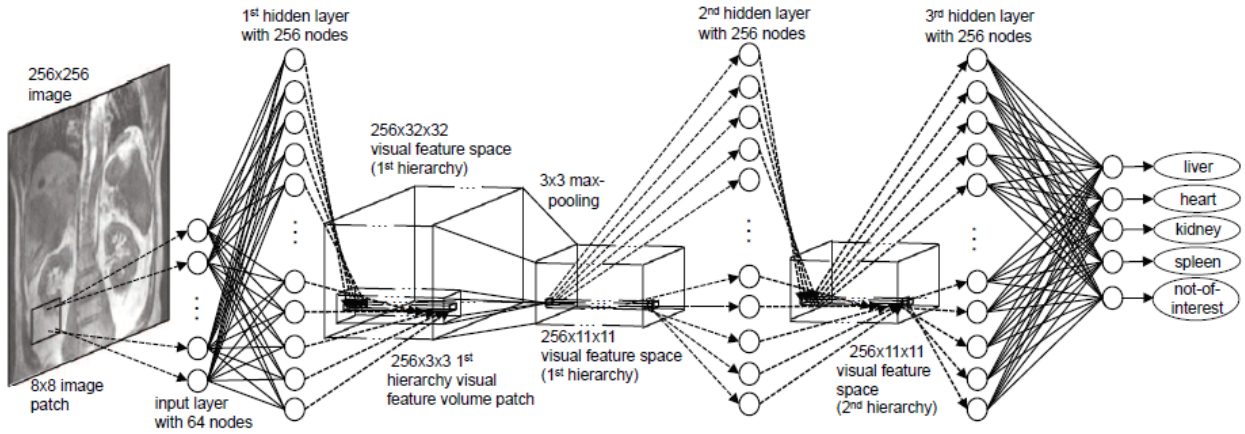


Figure 26: Architecture of the visual feature learning networks

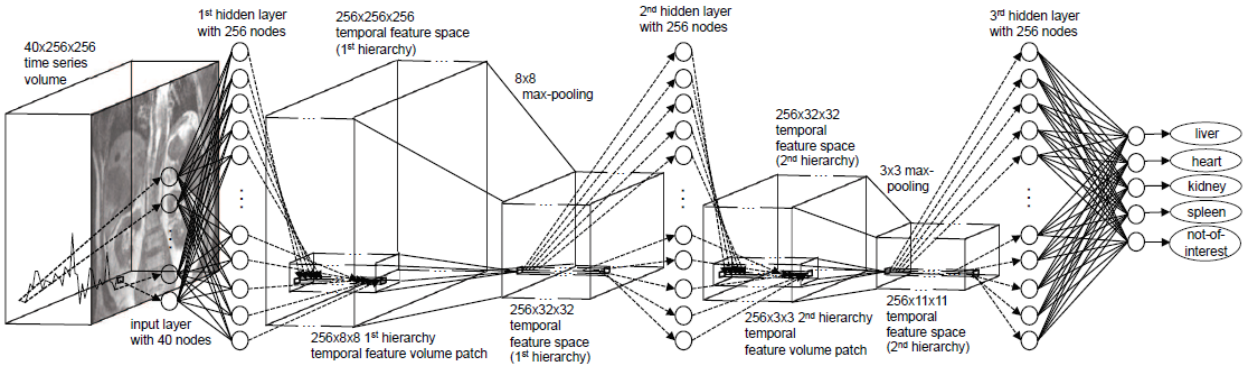


Figure 27: Architecture of the temporal feature learning networks

CONTRIBUTION

With the use of stacked sparse autoencoders, authors have performed unsupervised hierarchical feature learning, where organ classes are learned without detailed human input, and only a "roughly" labelled dataset was required to train the classifier for multiple organ detection from 4D temporal and spatial domain-based input scans.

RECURSIVE AUTOENCODER (RAE)

RAE network is a combination of the AE network and a recursive structure, which is to learn high-level features of the input as a representation. In [6], recursive neural network is used to parse different features from images of natural scenes. According to authors, building region can be

recursively split into smaller regions depicting parts such as roofs and windows. Then we can merge smaller parts into a larger region according to the rule that the parts of the same class must firstly be merged and the ones of different classes should be merged later on. Finally, by the rule, a whole building merged by parts will be formed, standing for the original one, i.e. the entire building region. The recursive representation represents rationally the entire region and the dimension of the representation will not increase. In figure (29), an illustration of recursive neural network architecture is shown which parses images and natural language sentences. Segment features and word indices (orange) are first mapped into semantic feature space (blue) and then recursively merged by the same neural network until they represent the entire image or sentence. Both mappings and merging are learned through training. Based on this concept, RAE algorithm is established in [5]. Unsupervised RAE is a tree structure. For example, taking a neighborhood region 3x3, the tree structure of unsupervised RAE is shown in Figure. 30. The structure is determined by the reconstruction error. The pixels in the neighborhood region are denoted as (x1, x2…x9), x(i) $\boldsymbol{\epsilon}$ $\mathfrak{R}^d$. The form of branching triplets of the parent with children in the binary tree is represented as (p → c1c2, where p denotes the parent and, c1, c2 are the children. For reconstructing the tree shown in figure 30, the representation of the tree form is: ((y1 → x1x2), (y2 → x3y1), ….. (y8 → x9y7)). To apply the same AE network to each pair of children recursively, the representations of hidden y(i) can keep the same dimensionality as x(i). The parent node P (e.g. y1) is computed from its children (c1; c2) : (x1: x2) like this, p = $f(W_1[c_1:c_2] + b_1)$ where the parameter matrix W_1 $\boldsymbol{\epsilon}$ $\mathfrak{R}^{dx2d}$ is multiplied by the concatenated vector of the two children $[c_1:c_2]$. After adding a bias term b_1, an element-wise activation function such as tanh or sigmoid is applied to the resulting vector. Then the AE network is adopted to reconstruct the children in the reconstruction layer, which can assess

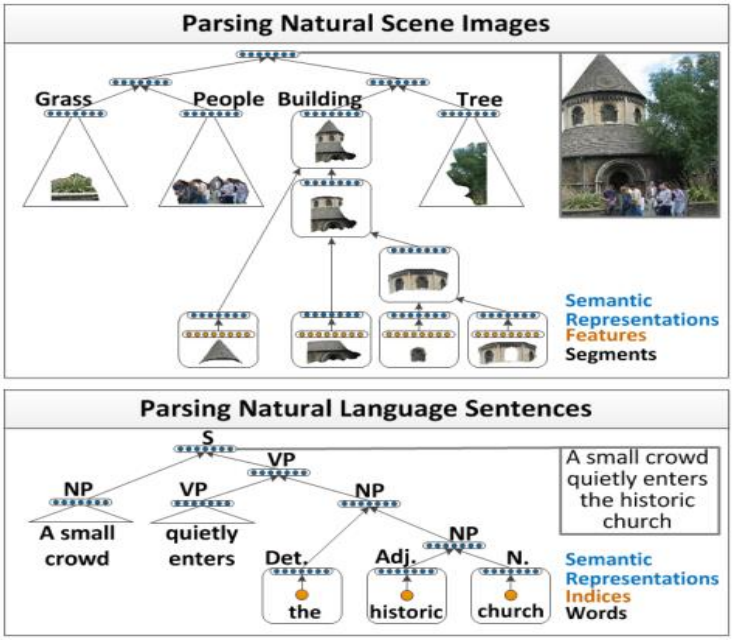


Figure 28: Illustration of recursive neural network architecture

how good the learned d-dimensional vector represents its children. The reconstructed children are obtained by, [c1*:c2*] = $W_2P_1 + b_2$. In the AE network, the objective function for training is to minimize the reconstruction error of the input pairs. For each pair, the Euclidean distance between the original input and its reconstruction is computed to measure the reconstruction error (3):

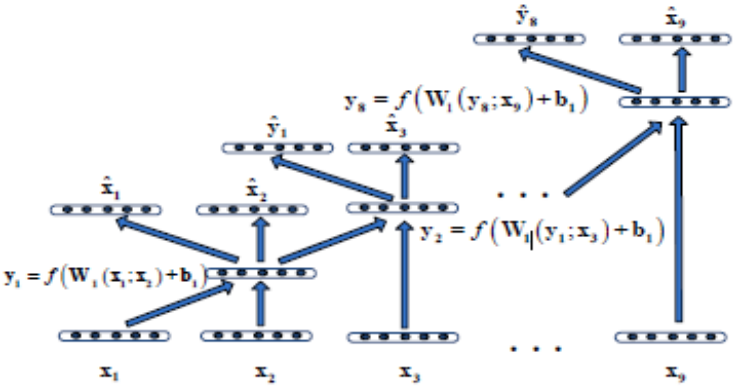


Figure 39: The structure of unsupervised recursive autoencoders

$E([c1:c2]) = ½ \| [c_1:c_2] - [c1*:c2*] \|^2$. The above process depicts how good a d-dimensional parent vector representation p is learned from two d-dimensional children (c1; c2). The learned parent vector p (e.g. y1) is then treated as a child and combined with its nearest child (e.g. x3) to learn new features (i.e. y2) by AE network. By recursively learning features with AE network, a top parent node of the tree occurs, which is the representation of the investigated pixel in the spatial neighborhood. The process repeats until the last pair of children are merged into the top node, and the tree structure is built.

RECURSIVE AUTOENCODERS FOR SOCIAL REVIEW SPAM DETECTION

In [7], authors have used the same concept of RAE described above for social review spam detection. Given one sentence of the social media review, the model views the sentence as an ordered list of words, that is, a word vector. Then each word vector $x \in R^n$ is simply initialized by sampling it from a zero mean Gaussian distribution: $x \sim N(0, \sigma 2)$. These word vectors are then stacked into a word embedding matrix, where |V | is the size of the vocabulary. Assume a sentence is given, each word has an associated vocabulary index k into the embedding matrix which is used to retrieve the words vector representation. Mathematically, this algorithm a simple projection layer where a binary vector b is used which is zero in all positions except at the kth index. The sentences of social media reviews are represented as an ordered list of these vectors (x1, ...x_m). Given a list of word vectors x = (x1, ..., x_n), the tree structure is in the form of branching triplets of parents with children:($pj \rightarrow c_i c_{i+1}$), where c_i may be either an input word vector xi or a

nonterminal node in the tree. Then parent vector p_j is computed from the children (c_i, c_{i+1}) using the same formula used in [6]. The reconstruction errors of this input pair is also calculated by equation (3). This process repeat until the full tree is constructed and the reconstruction error at each nonterminal node is computed.

DEEP CONVOLUTIONAL AUTO-ENCODER WITH POOLING-UNPOOLING LAYERS

In [10], authors have used a deep convolutional autoencoder for dimensionality reduction and obtaining unsupervised clusters from unlabeled dataset. The AE network consists of convolutional, pooling, fully-connected, deconvolution, unpooling and loss layers. Apart from the pooling and unpooling layer, all the other layers work similarly as explained before. In this algorithm, a max-pooling layer pools features by taking the maximum activity within input feature maps (outputs of the previous convolutional layer) and produces (i) its output feature map with reduced size according to the size of pooling filter and (ii) supplemental switch variables (switches) which describe the position of these max-pooled features. On the other hand, the unpooling layer restores the max-pooled feature either (i) into the correct place, specified by the switches, or (ii) into some specific place within the unpooled output feature map. Figure 31 illustrates convolution – deconvolution and pooling – unpooling operations used in [10]. Apart from that they have used two loss layers as Sigmoid Cross-Entropy loss and Euclidean Loss respectively. The cross-entropy (logistic) loss layer is defined by the expression, $E = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)]$.

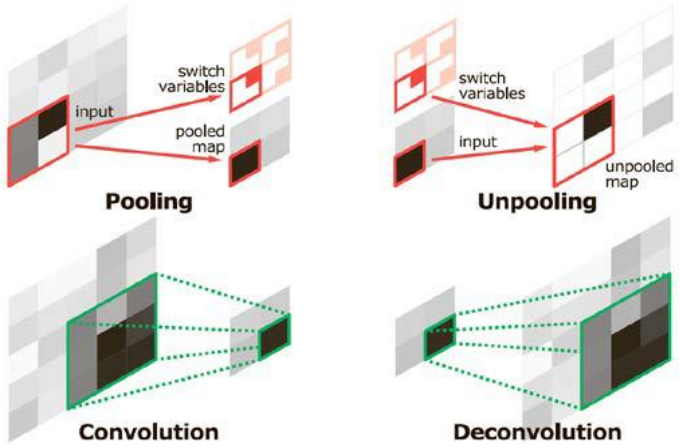


Figure 310: Deconvolution and unpooling

where N is a number of samples, y are the targets, y ϵ {0,1} and $y'_n = f(W.x_n)$ where f is a logistic function, w is the vector of weights optimized through gradient descent and x is the input vector. the Euclidean (L2) loss layer defined by expression, $E = \frac{1}{2N}\sum_{n=1}^{N}\|\hat{y}_n - y_n\|_2^2$, where $\hat{y}$ are the predictions, $\hat{y}$ ϵ {$-\infty, +\infty$} and y are the targets y ϵ {$-\infty, +\infty$}. They have used

this algorithm to reduce the dimensionality of input to 2-D and visualized the clusters for each of the ten classes for MNIST dataset ranged from 0-9.

REFERENCES

[1] Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. (2008). Extracting and composing robust features with denoising autoencoders. Proceedings of the 25th International Conference on Machine Learning - ICML 08. doi:10.1145/1390156.1390294

[2] Masci, J., Meier, U., Cireşan, D., & Schmidhuber, J. (2011). Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. Lecture Notes in Computer Science Artificial Neural Networks and Machine Learning – ICANN 2011, 52-59. doi:10.1007/978-3-642-21735-7_7

[3] Makhzani, A., & Frey, B. (2014). K-Sparse Autoencoders. ArXiv:1312.5663 [cs.LG]

[4] Shin, H., Orton, M. R., Collins, D. J., Doran, S. J., & Leach, M. O. (2013). Stacked Autoencoders for Unsupervised Feature Learning and Multiple Organ Detection in a Pilot Study Using 4D Patient Data. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8), 1930-1943. doi:10.1109/tpami.2012.277

[5] Zhang, X., Liang, Y., Li, C., Jiao, L., & Zhou, H. (2017). Recursive autoencoders based unsupervised feature learning for hyperspectral image classification. IEEE Geoscience and Remote Sensing Letters. DOI: 10.1109/LGRS.2017.2737823

[6] Wang, B., Huang, J., Zheng, H., & Wu, H. (2016). Semi-Supervised Recursive Autoencoders for Social Review Spam Detection. 2016 12th International Conference on Computational Intelligence and Security (CIS). doi:10.1109/cis.2016.0035

[7] R. Socher, C. Lin, Andrew Y. Ng, and C. D. Manning. “Parsing natural scenes and natural language with recursive neural networks,” in Proc. ICML. 2011.

[8] Coates, A., Lee, H., & Y. Ng, A. (2011). An Analysis of Single-Layer Networks in Unsupervised Feature Learning. International Conference on Artificial Intelligence and Statistics (AISTATS), 15(JMLR: W&CP 15)

[9] J. Zhao, M. Mathieu, R. Goroshin, Y. LeCun, Stacked what-where auto-encoders, in: International Conference on Learning Representations (ICLR), town, arXiv:1506.02351, San Juan, 2016.

[10] Turchenko, V., Chalmers, E., & Luczak, A. (n.d.). A Deep Convolutional Auto-Encoder with Pooling - Unpooling Layers in Caffe

[11] Nervana Systems/Neon, Convolutional autoencoder example network for MNIST data set. https://github.com/NervanaSystems/neon/blob/master/examples/conv_autoencoder.py, 2015

[12] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, Lecture Notes in Computer Sci. 8689 (2014) 818-833.

[13] G.E. Hinton, A. Krizhevsky, S.D. Wang, Transforming auto-encoders, Lecture Notes in Computer Sci. 6791 (2011) 44-51.