**Face Autoencoder**

**Abstract**

Autoencoders using Deep Neural Networks (DNN) have recently shown outstanding performance on image compression and reconstruction that helped a lot in various applications that deal with huge amounts of unlabeled images that are easily found on the internet. We provide a simple yet powerful way to get the most out of unlabeled face images that are easily found on the internet.

**Datasets:**

UTKFace, KinFace

**Images Preprocessing:**

Using Imagemagick (Open source) to resize the images making them 100\*100 and then using Corona library (Open source) to covert jpg images into binary files that are easily loaded.

**Programming language:**

C++ (gcc compiler)

**Artificial neural networks** (**ANN**) or **connectionist systems** are computing systems that are inspired by, but not necessarily identical to, the [biological neural networks](https://en.wikipedia.org/wiki/Biological_neural_network) that constitute animal [brains](https://en.wikipedia.org/wiki/Brain). Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in [image recognition](https://en.wikipedia.org/wiki/Image_recognition), they might learn to identify images that contain cats by analyzing example images that have been manually [labeled](https://en.wikipedia.org/wiki/Labeled_data) as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

The Autoencoder neural network structure has many elements and many processes:

1- Feedforward

2- Backpropagation

3- Activation function

4- Weights

5- Number of layers

6- Number of neurons in each layer

7- Number of neurons in encoded layer

**Feedforward**

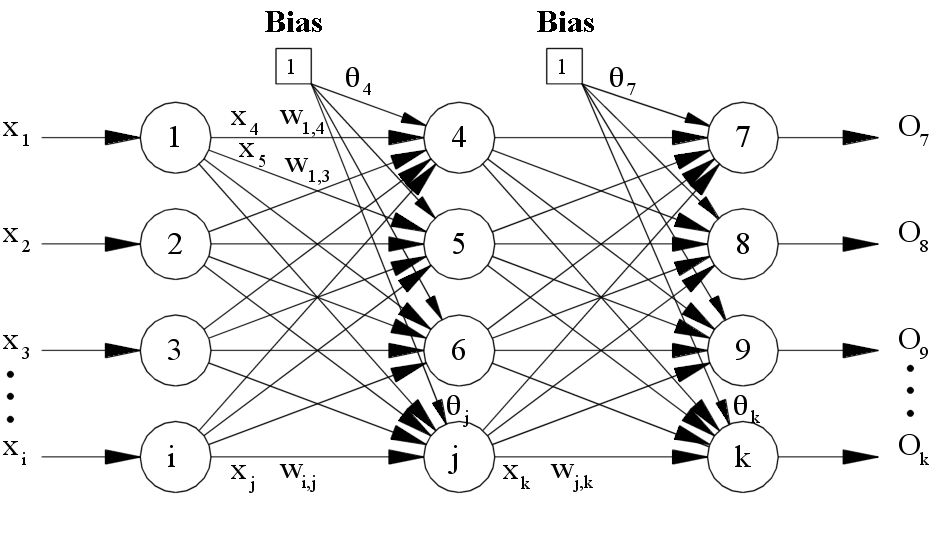
A **Feedforward neural network** is an [artificial neural network](https://en.wikipedia.org/wiki/Artificial_neural_network) wherein connections between the nodes do *not* form a cycle. As such, it is different from [recurrent neural networks](https://en.wikipedia.org/wiki/Recurrent_neural_networks).

The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes and to the output nodes. There are no cycles or loops in the network.

In feedforward we multiple each input with the weights and the result of multiplying we adding the Bias and the final result used in activation function

We use activation function **[tanh()]** and we working on region between -1 to 1 and we do this process to reach the output layer.

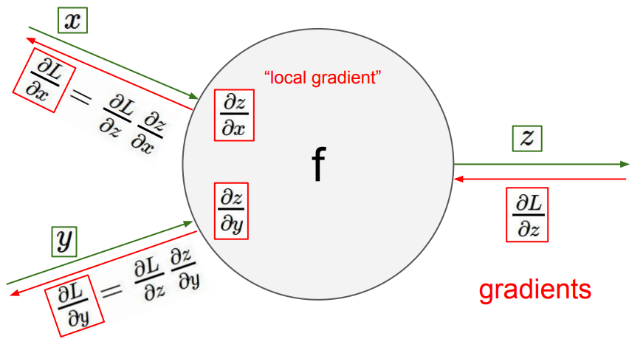
When we reach the output layer we do another process it called **Backpropagation**



**Backpropagation**

**Backpropagation** algorithms are a family of methods used to efficiently train [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network) (ANNs) following a [gradient descent](https://en.wikipedia.org/wiki/Gradient_descent) approach that exploits the [chain rule](https://en.wikipedia.org/wiki/Chain_rule). The main feature of backpropagation is its [iterative](https://en.wikipedia.org/wiki/Iterative), [recursive](https://en.wikipedia.org/wiki/Recursive) and efficient method for calculating the [weights updates](https://en.wikipedia.org/wiki/Artificial_neural_network#Components_of_an_artificial_neural_network) to improve the network until it is able to perform the task for which it is being trained It is closely related to the [Gauss–Newton algorithm](https://en.wikipedia.org/wiki/Gauss%E2%80%93Newton_algorithm).

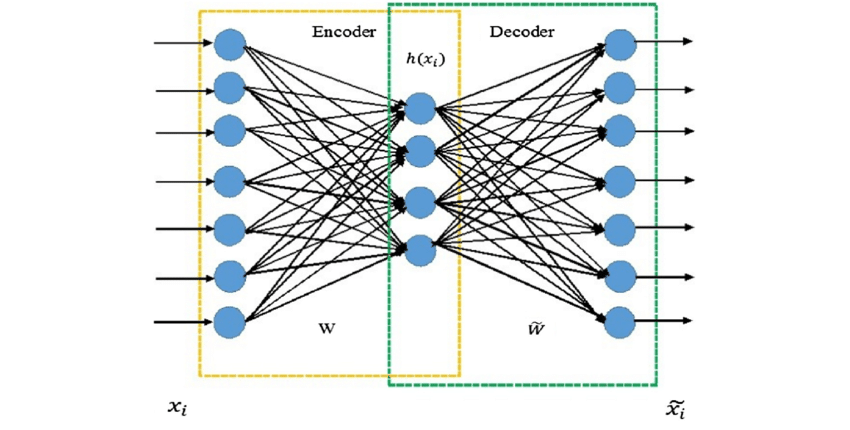
Backpropagation requires the derivatives of activation functions to be known at network design time. [Automatic differentiation](https://en.wikipedia.org/wiki/Automatic_differentiation) is a technique that can automatically and analytically provide the derivatives to the training algorithm. In the context of learning, backpropagation is commonly used by the [gradient descent](https://en.wikipedia.org/wiki/Gradient_descent) optimization algorithm to adjust the weight of neurons by calculating the [gradient](https://en.wikipedia.org/wiki/Gradient) of the [loss function](https://en.wikipedia.org/wiki/Loss_function); backpropagation computes the gradient, whereas gradient descent uses the gradients for training the model.

****

We use the normal backpropagation and the learning rate is 0.05 and it minimize to reach 0.00001.

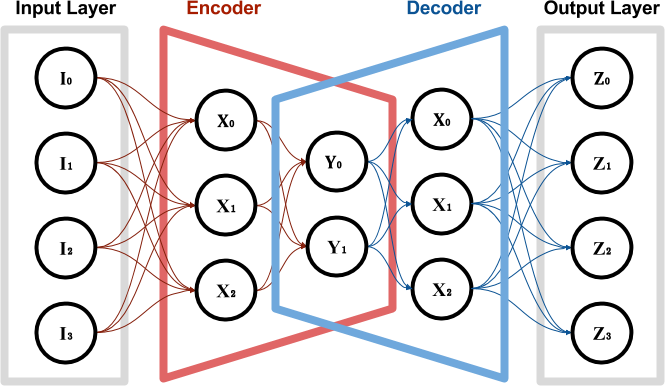
# “Big things often have small beginnings”

Starting a deep neural network is not always a good idea, deep neural networks are hard control and they are not always trust worthy. The idea behind this experiment is to have a variable sized neural network that can be resized, expanded and retrained.

Starting with a simple vanilla autoencoder (three-layer) that is fast and easy to train; we used a (10000 200 10000) network and trained this network as much as we could cause this is the mother of all networks and from it we are going to derive our deep neural networks.  


The next step is to start going deep, the stacked autoencoder is one of the best ways to go deep from a simple network. We take the output of the latent layer as an input to another small autoencoder after saving the weights of the autoencoder, then we train our small autoencoder with a smaller latent layer, after that we save the weights of the small autoencoder and combine these weights to form a our first deep neural network.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10000 | 200 | 100 | 200 | 10000 |

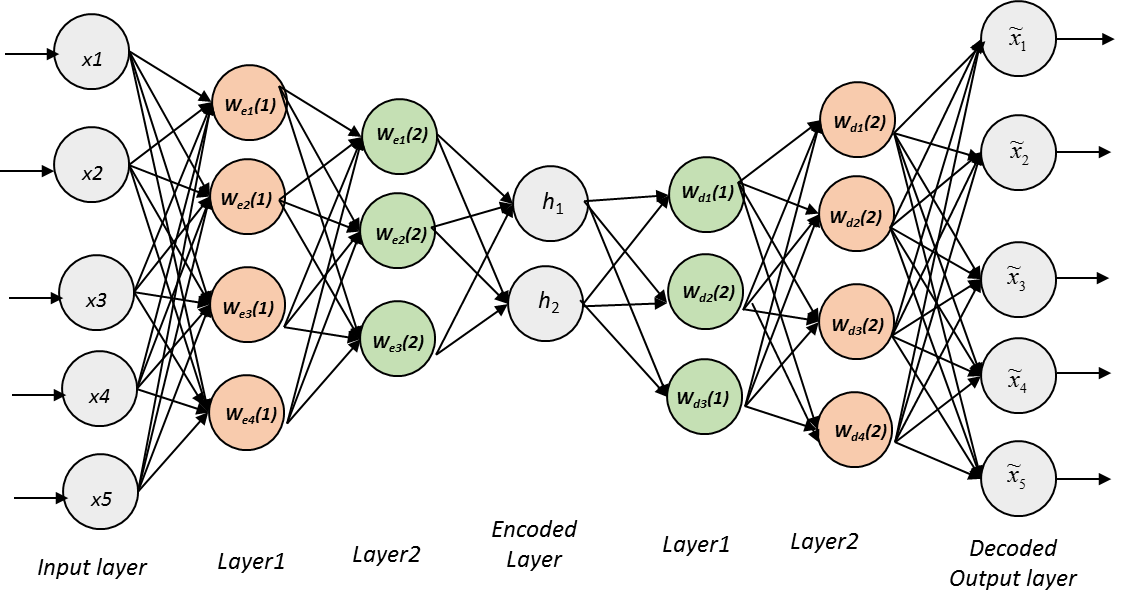


After a while of training this network the neurons started entering saturation so we decided to add more neurons and the only way to do it is to initialize the connections to these neurons to extremely small values, we added those neurons to layer one and layer three.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10000 | 400 | 100 | 400 | 10000 |

The last step is go deeper using stacked autoencoder and have the desired compression rate we need so we decided to make various sized latent layers for various applications and train them in parallel, here are the networks sizes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10000 | 400 | 100 | 24 | 100 | 400 | 10000 |
| 10000 | 400 | 100 | 40 | 100 | 400 | 10000 |
| 10000 | 400 | 100 | 50 | 100 | 400 | 10000 |
| 10000 | 400 | 100 | 70 | 100 | 400 | 10000 |
| 10000 | 400 | 100 | 84 | 100 | 400 | 10000 |



An anomaly or outlier is a data point which is significantly different from the remaining data. Hawkins defined an anomaly as an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.

***Applications:***

**Watermark remover app***:*

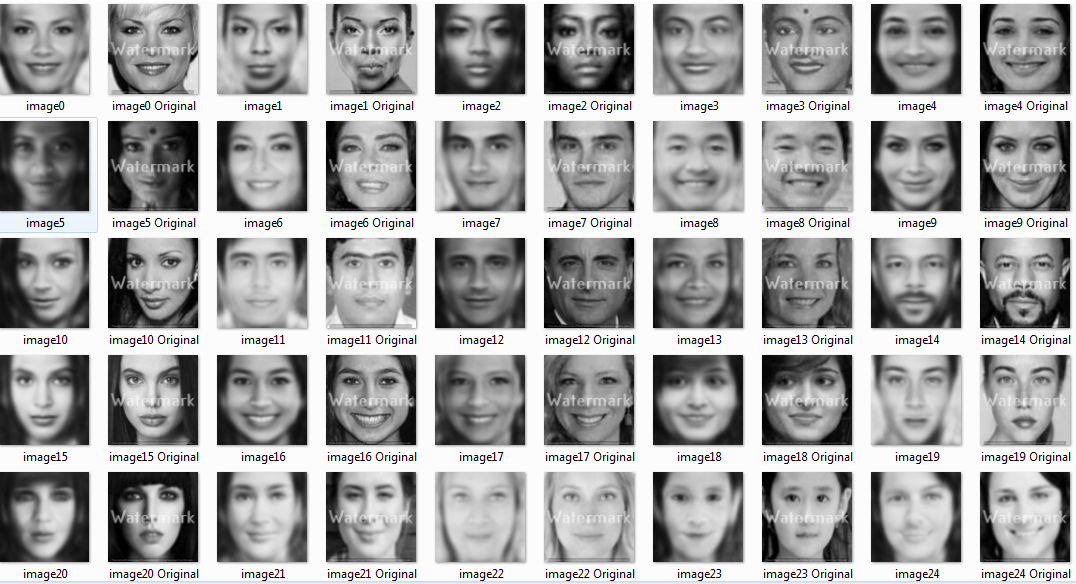
***Why this app:***

Watermark is usually placed on images or videos to protect them from being used without the owner’s permission.  However, if you come across a photo which you like to use in an article, presentation or in a project, or want to clean it up for adding it on your webpage. So we make this application.

***The structure of network***

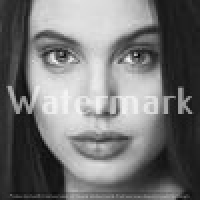
In this application we need high accuracy in the output photo, So we use a network with small percent of compression "large latent layer". The network constructed of input is a 100\*100 photo, one hidden layer 200 units and the output a 100\*100 photo.

***Results:***

******



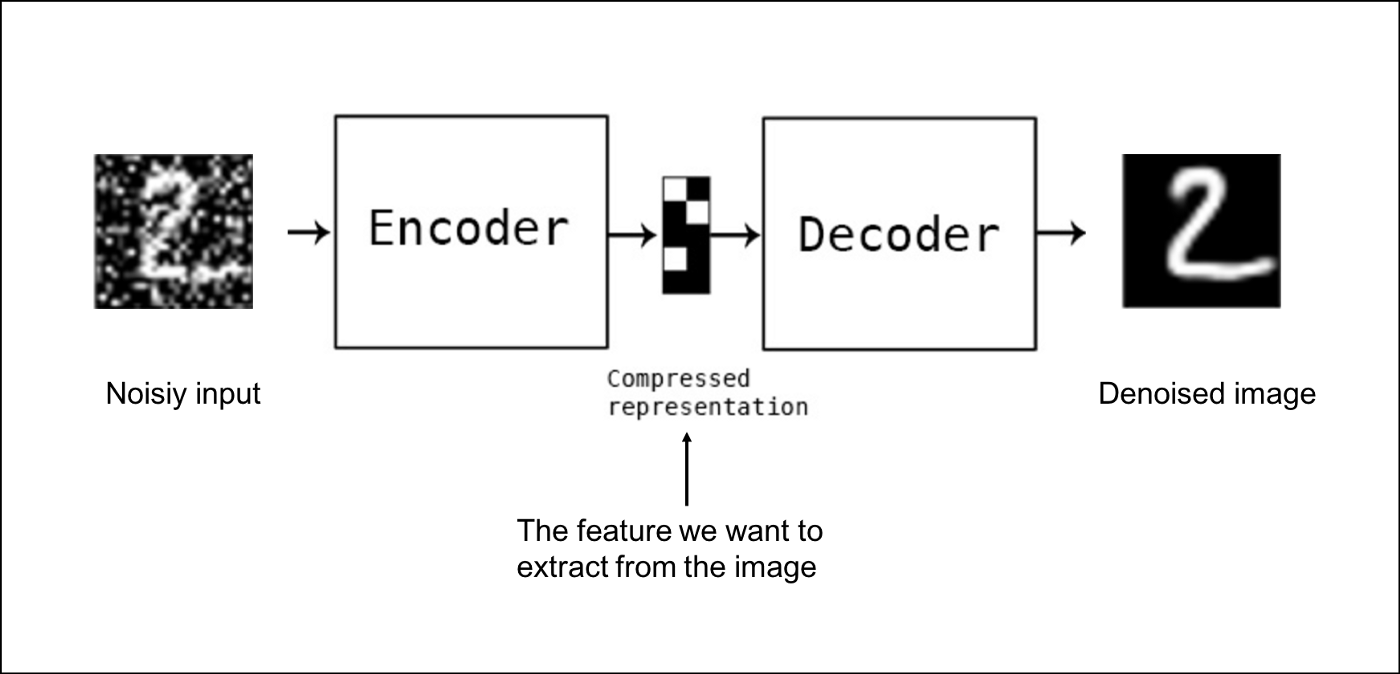






***Denoising Autoencoder :***

A denoising autoencoder (DAE) is a specific type of autoencode, learning to construct the clean input from a corrupted version.



***We make two types of noise:***

Salt and pepper noise line noise

Salt-and-pepper noise is a form of noise sometimes seen on images. It is also known as impulse noise. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels.

We used a simple autoencoder with large latent layer (200 neuron) to achieve a high quality output. The network is 10000 200 10000.

Then collect the test set put the noising mask on it. After training the network .

We developed this idea to use it in a real application. All of us has damaged photos.