*Methodology*

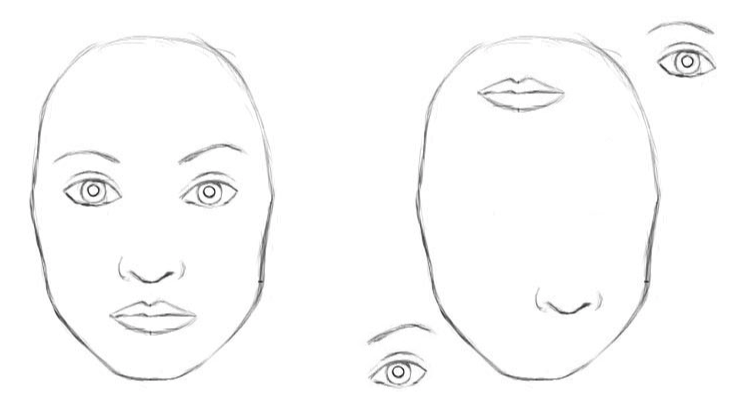
One of the major challenges in understanding images and patterns is the **limitation of traditional**

**Image Processing software** in analyzing sensitive and huge amounts of image data owing to which we will be using systems that can not only process huge amounts of data but also generate meaningful insights from them with much ease of accessibility and usage. Hence comes Deep Learning. The addition of noise and other astrophysical back-grounds makes this an even more intimidating deployable system.

When dealing with images, **the most intimidating choice is the use of Convolutional Neural Networks**. So the whole problem statement can be categorized as a classification problem with three classes achieved by using **CNN as a classifier**. Although that may seem a pretty good approach, there are **lots of improvement measures that we can employ**.

Convolutional Neural Networks in their very basic mathematical form **calculate the overlapping of the image function with the filter function**, in other words, it produces a third function expressing how the shape of one function affects the second function. Putting simply, a Convolutional Neural Network **matches** certain **filters** against the **input image**, computes the degree of overlap and concludes whether the content of the filter is present in the image. So a CNN makes predictions by looking at an image and then checking to see if certain **components** are present in that image or not. If they are, then it classifies that image accordingly. This poses a lot of **erroneous predictions** in sensitive cosmological data. Let me quickly state some of the limitations of CNNs in treating such image data.

1. CNN is **not** **positional invariant**. For a CNN, the mere presence of some ‘**components**’ is all it takes to classify an image. **Orientational and relative** positional relationships between these components are **not very important to CNN**. For example -



Both these images are exactly the same to a CNN because they have the same set of components. That’s a huge flaw in CNNs.

1. CNN’s are **not** **spatial invariant**, which means CNN does not take into account the translational and rotational factors. If the same image is spatially transformed, CNN would fail to recognize it. Again taking a simple example to illustrate the point.



Our brain can easily recognize that it’s the same image with a different viewpoint. CNNs can’t. This is because the internal representation of the Statue of Liberty in your brain does not depend on the view angle. You have probably never seen these exact pictures of it, but you still immediately knew what it was.

1. CNN’s are **Supervised Algorithms**, and it takes a huge amount of data to train them, which is in scarcity. Owing to the above two limitations of a CNN, it highly **overfits**, even after data **augmentations**. Overfitting occurs when a function fits/replicates a set of too **limited** data points and **fails to generalize** for new data points.

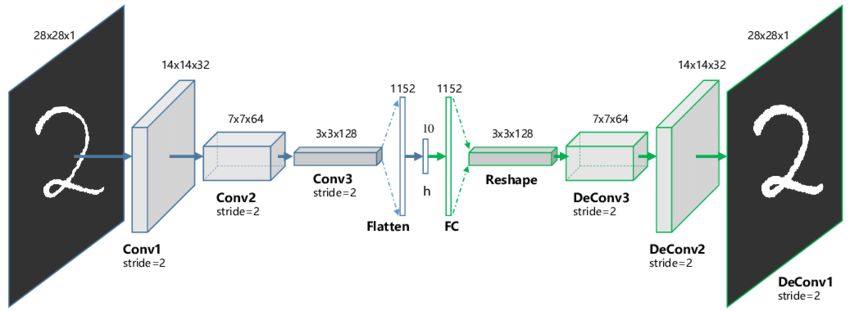
**So any CNN architecture ( even skip connections as that of a ResNet ) may not be very wise to use**.

Hence I propose a highly **scalable**, **efficient**, **unsupervised**, **positional** and **spatial** invariant use of **CONVOLUTIONAL** **AUTOENCODERS** with **DEEP CLUSTERING** for the purpose.

Autoencoders are **unsupervised** Neural Networks that learn how to efficiently compress and encode data and reconstruct the data back from the encoded representation to a representation that is as close to the original input as possible.

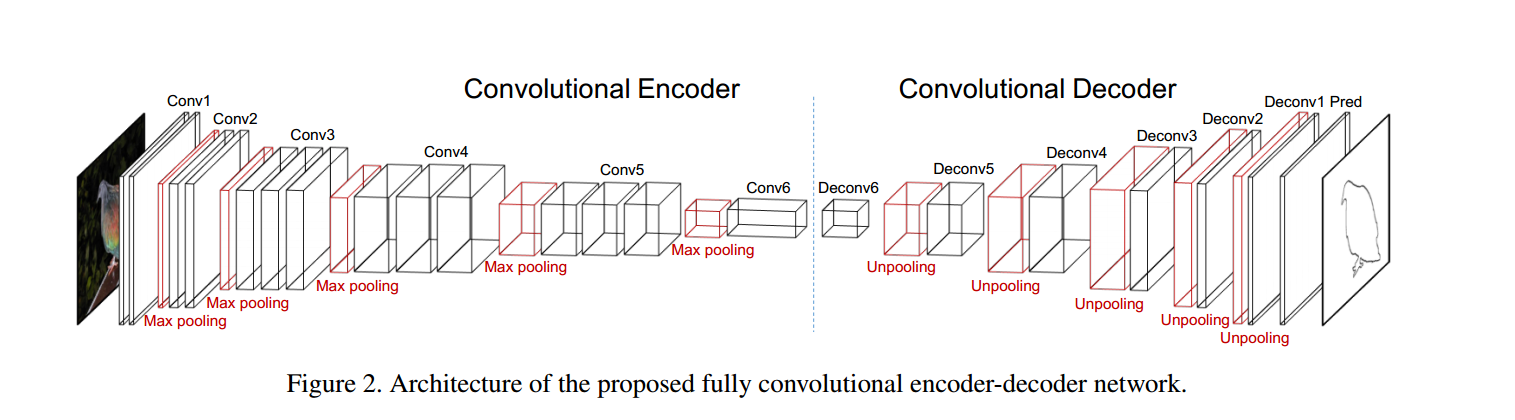
THE APPROACH

1. Since autoencoders compress data into encoded vectors, they do not only rely on the presence of certain ‘components’, they also take note of the **positional and rotational transformations** that can possibly happen with the image data. For this particular use case, we would be using Convolutional Autoencoders for generating the encoded representation, also known as the latent vector. We would be **using the limited data that we have** ( not simulated ones ) to **train** the autoencoder. The purpose of using a Convolutional Autoencoder will ensure that **spatial correlations are preserved while diminishing the dimension of the image to the latent vector**.



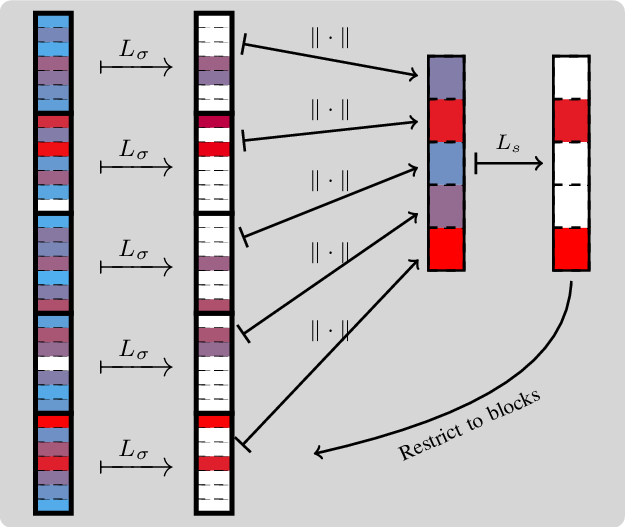
1. Once our Autoencoder is trained, we would be passing the simulated images generated from PyAutoLens into the Encoder System and observe the **Decoder Loss**. That would also give us a **benchmarking** result of how the **computer-generated images cohere with the actual ones**. Once the Decoder Loss is within limits, we would train the rest of the Autoencoder **again** by inputting the most relevant set of images ( **actual ones and generated ones or just the generated ones** ).

Once we are finished with the training of the autoencoder, we disconnect the decoder, making a standalone system of just the encoder with the purpose of generating the latent vector of any given image.



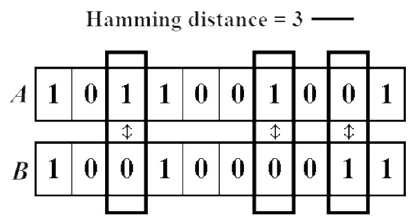
1. Now, since we have our encoder ready, we now want to **embed the encodings in an N-dimensional space where N will be determined by the degree of the latent vector so produced**. But we will just not embed the points in the space as it is. We want to embed the points in such a way that **images having close resemblance will be mapped closely in the N-dimensional space, and images not much resemblance will be mapped far from each other** ( this will ensure high clustering score and prevent misclassification of the images ).

Now how can we actually do the embeddings? On one hand we can use **Cosine Similarity** as a measure of similarity between two latent vectors. On another hand, we would be using an **intermediate Sigmoid function that will convert the latent vector into discrete values of 0 and 1**. This binarization of the latent vectors will help us measure the similarity by using **Hamming Distance** Methods.



*We use the Sigmoid Function to binarise the whole vector to values 0 and 1.*

*The whole latent vector (on the leftmost side) is binarised to only 0 and 1 ( white being 1 and red being 0 ).*

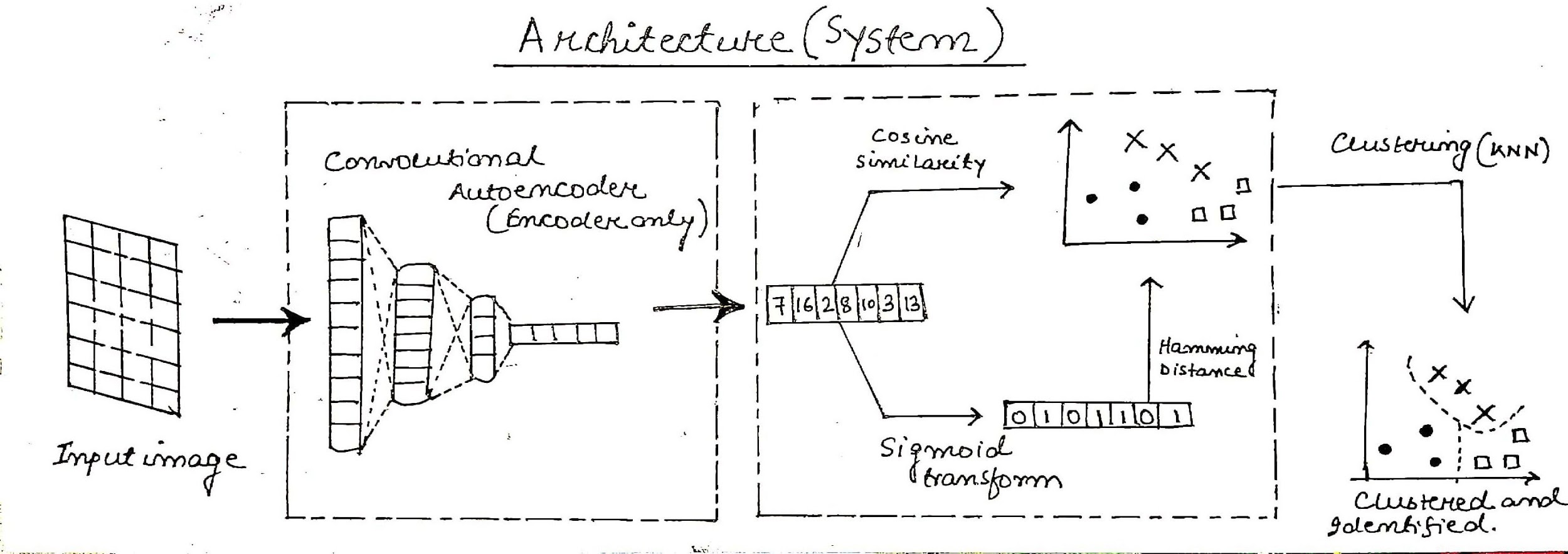


*After binarization, we use the Hamming Distance as a metric to find a degree of similarity between the two vectors.*

1. Once we have similarity measures, we would embed them into our **N-Dimensional vector space and leave an Unsupervised Clustering algorithm to cluster each data point into one of the three categories of substructure ( vortex, spherical and none )**.

The validation of the whole system could be done using images from the validation set, which may be generated randomly by the PyAutoLens package.

The whole architecture is described below -



The whole system would be unsupervised. **Python** will be used as the Programming Language for codifying the entire architecture. The whole system can be used as a **standalone system for automatically clustering each new image ( either generated or captured )**.