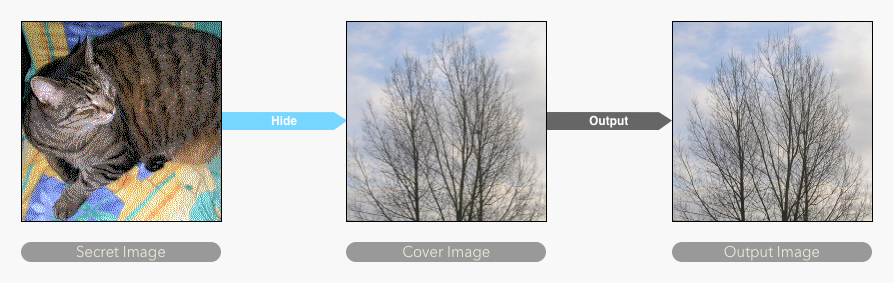
STEGANALYSIS

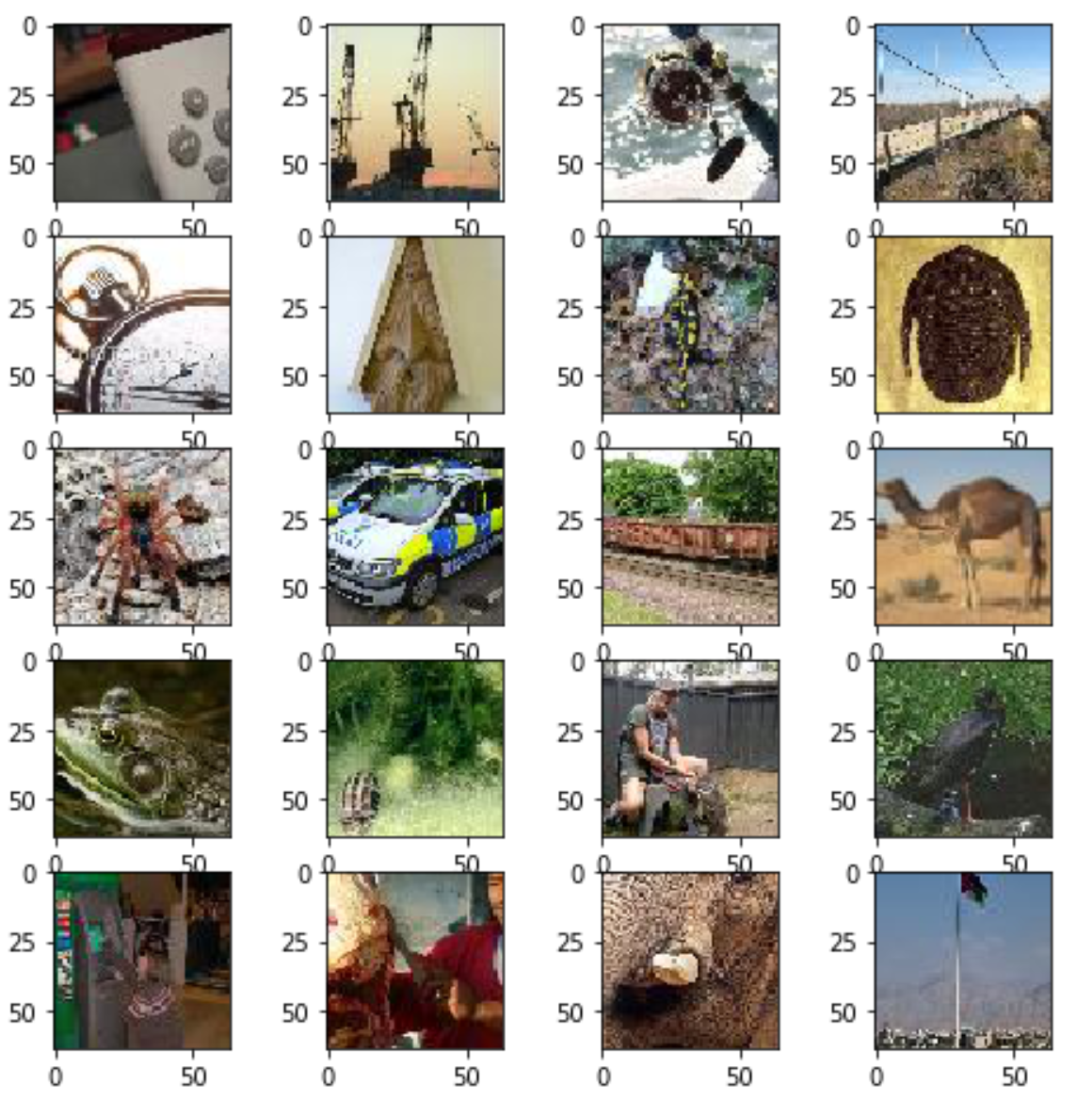
Introduction :-

***Steganography*** is the practice of concealing a secret message within another, ordinary, message. Commonly, Steganography is used to unobtrusively hide a small message within the noisy regions of a larger image. In this study, we attempt to place a full size color image within another image of the same size. Deep neural networks are simultaneously trained to create the hiding and revealing processes and are designed to specifically work as a pair. The system is trained on images drawn randomly from the ImageNet database, and works well on natural images from a wide variety of sources. Beyond demonstrating the successful application of deep learning to hiding images, we carefully examine how the result is achieved and explore extensions. Unlike many popular steganographic methods that encode the secret message within the least significant bits of the carrier image, our approach compresses and distributes the secret image's representation across all of the available bits.

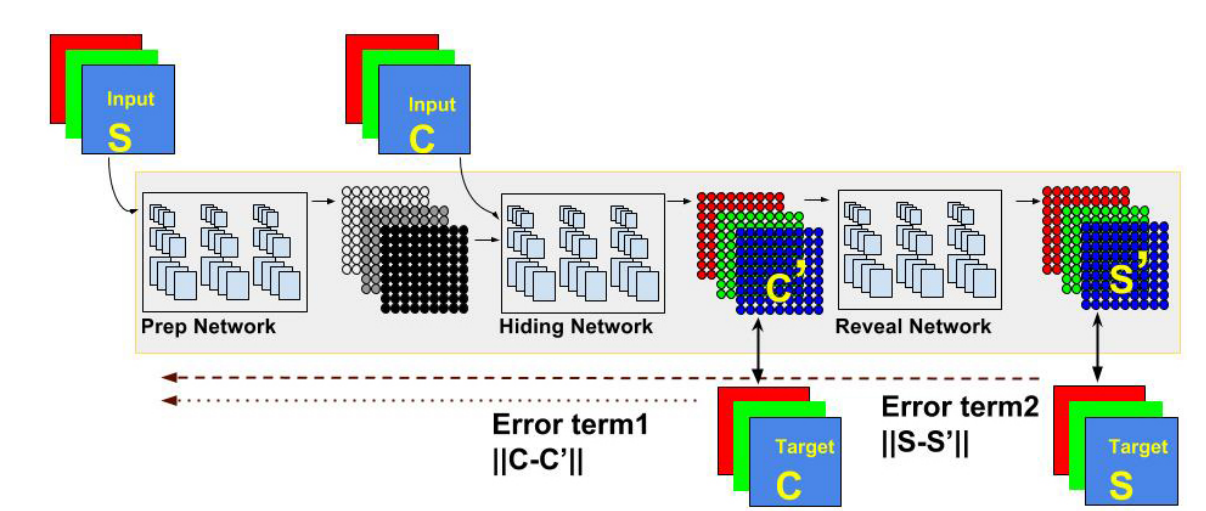


**Dataset**

**The dataset we used in this implementation is Tiny ImageNet Visual Recognition Challenge. It can be downloaded from** [**here**](https://tiny-imagenet.herokuapp.com/) **and extracted . The following figure shows some sample images of the dataset.**

****

**Model Architecture**

****

Convolutional Neural Networks have shown to learn structures that correspond to logical features. These features increase their level of abstraction as we go deeper into the network. Using a ConvNet will solve all the problems mentioned . Firstly, the convnet will have a good idea about the patterns of natural images, and will be able to make decisions on which areas are redundant, and more pixels can be hidden there. By saving space on redundant areas, the amount of hidden information can be increased. Because the architecture and the weights can be randomised, the exact way in which the network will hide the information cannot be known to anybody who doesn't have the weights.

The model is composed of three parts: The Preparation Network, Hiding Network (Encoder) and the Reveal Network. It’s goal is to be able to encode information about the secret image S into the cover image C, generating C\_prime that closely resembles C, while still being able to decode information from C\_prime togenerate the decoded secret image S\_prime, which should resemble S as closely as possible.

The Preparation Network has the responsibility of preparing data from the secret image to be concatenated with the cover image and fed to the Hiding Network. The Hiding Network then transforms that input into the encoded cover image C\_prime. Finally, the Reveal Network decre the Reveal Network, as suggested by the paper. Although the author of the paper didn't originally specify the architecture of the three networks, we discovered aggregated layers showed good results. For both the Hiding and Reveal networks, we use 5 layers of 65 filters (50 3x3 filters, 10 4x4 filters and 5 5x5 filters). For the Preparation Network,decodes the secret image S\_prime from C\_prime. For stability, we add noise before we use only 2 layers with the same structure.

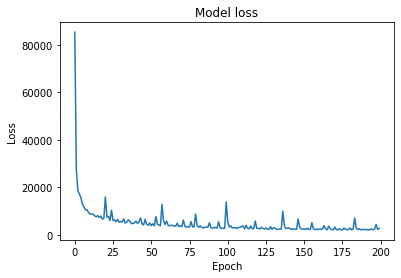
Note that the loss function for the Reveal Network is different from the loss function for the Preparation and Hiding Networks. In order to correctly implement the updates for the weights in the networks, we create stacked Keras models, one for the Preparation and Hiding Network (which share the same loss function) and one for the Reveal Network. To make sure weights are updated only once, we freeze the weights on the layers of the Reveal Network before adding it to the full model.

**Experimentation**

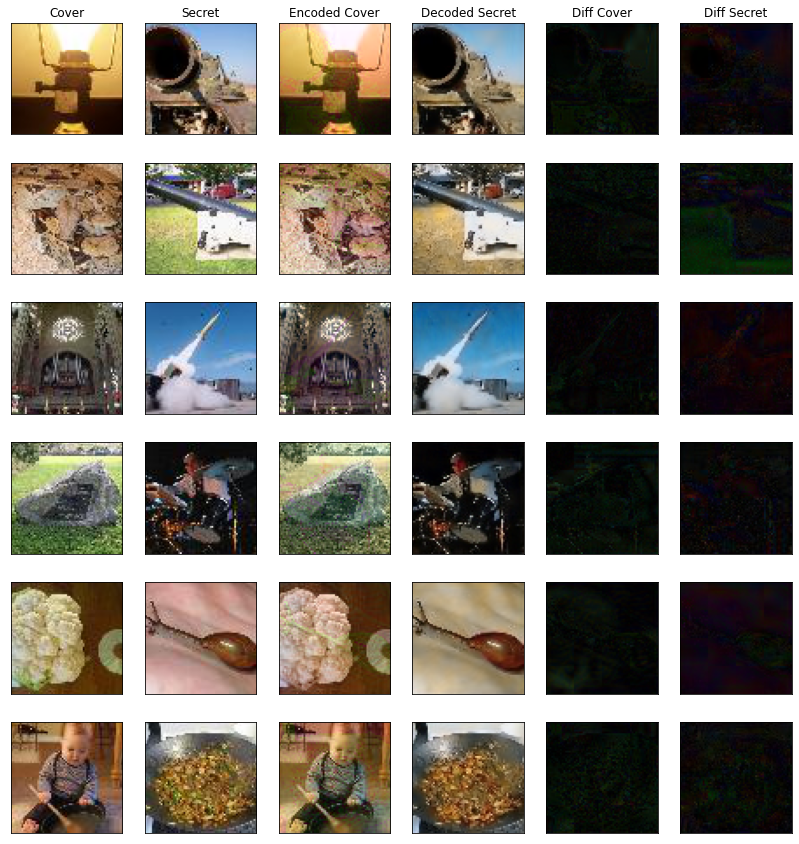
Basic Experimentation has been done by changing the activation function of the architecture to examine and learn the performance of the model . All other parameters such as convnet architecture , no. of epochs and the optimizer function are kept constant.

**1. Relu Activation**

**Loss vs Epochs**

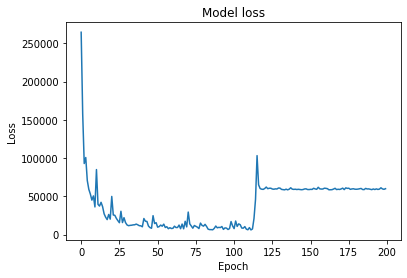
****

**Output :**

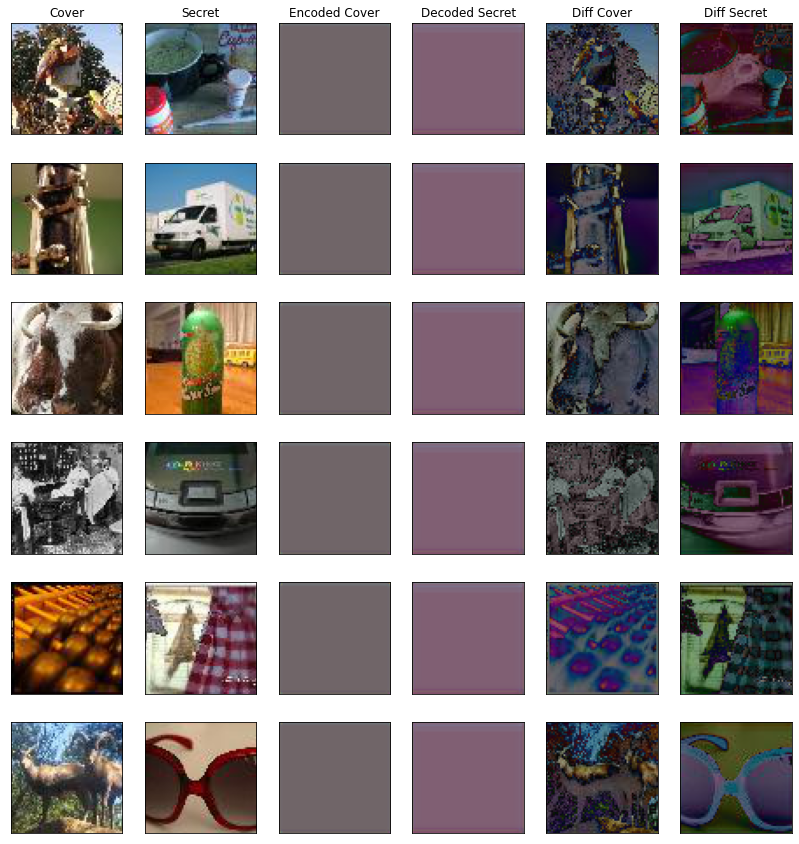
****

**2. Tanh:**

**Loss vs Epochs**

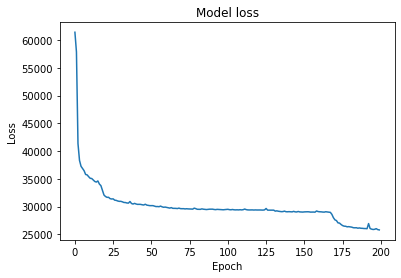


**Output**

****

**3. Sigmoid**

**Loss vs epochs**

****

**Output**

****

**Conclusion:**

*By Comparing the output result and the loss curve for each of the mentioned Activation functions , we can safely conclude that Relu activation perform much better than the other activation function and the model learns much faster for same no. of epochs . Sigmoid Shows a much good result than tanh which was a bit surprising but after seeing a bunch of papers on it , I found that though tanh provide much nonlinearity to the model it is sometimes unstable for a neural network and could lead to vanishing gradient or unpredictable loss value which could be seen here by a sudden rise in loss value at epoch =120.*