A project report on

STEGANOGRAPHY USING DEEP LEARNING

Submitted in partial fulfillment for the award of the degree of

B. TECH., COMPUTER SCIENCE AND ENGINEERING

by

SARVESH CHANDAK 20BAI1221

HARIKISHAN TAKOOR 20BAI1297

ARYAN SINGH KUSHWAHA 20BAI1288

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

November 2022

Steganography using deep learning (CNN and LSB)

ABSTRACT

Hiding data has always been significant importance to cybersecurity. Steganography is the process of hiding some type of data into other data. In this project, we aim to combine steganography techniques with deep learning in such a way that it becomes difficult for attackers to steal the hidden information. This technique has multiple applications in defense organizations intelligence agencies and other such fields where information is of utmost importance.

Current methods that hide images in other images already exist, but there are a few problems associated with these.

1. Very easy to decode.
2. The amount of information that can be hidden is generally less.
3. Existing algorithms don't use the patterns found in natural images.

To encode the images effectively such that they look like natural images, I am planning to use deep learning techniques such as Artificial Neural network along with CNN. This would only be the second layer to an already existing and famous method of steganography using LSB (Least Significant Bit) method. Basically, LSB method can hide a text into an image or an image into an image, and my deep learning CNN model can encrypt the same formed image into another image.

Keywords:
Steganography, Hiding Data, Artificial Neural Networks, CNN, LSB

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1. What is Steganography?

Steganography is the technique of hiding secret data within an ordinary, non-secret, file or message in order to avoid detection; the secret data is then extracted at its destination. The use of steganography can be combined with encryption as an extra step for hiding or protecting data.

It mainly refers to the practice of hiding a type of data into another type of data such as text in image, or image in image.

2. How useful have current Steganographic models been in the cybersecurity domain?

As discussed before in the abstract current steganographic models are prone to be decrypted easily using tools due to the technique's limitations. There are two key factors that are often in competition – the first is how obvious and easy the hidden data is to detect (whether by human perception or other forms of analysis), while the second is how much data can be hidden in a given file or piece of communication.

The higher the percentage of data someone tries to conceal, the easier it is to spot. How much data you can safely include in a given file will depend on the steganographic technique, risk level, and amount of scrutiny expected.

If data is hidden in images, it's still quite hard for the human eye to detect anomalies when 20 percent of the data has been replaced, assuming the information has been well hidden. At lower percentages, the image will look essentially the same. As more data is packed in, the quality starts to deteriorate and you may even be able to see elements of the hidden picture.

3. Introduction to CNNs

A convolutional neural network (CNN) is a network architecture for deep learning which learns directly from data, eliminating the need for manual feature extraction.

CNNs are particularly useful for finding patterns in images to recognize objects, faces, and scenes. They can also be quite effective for classifying non-image data such as audio, time series, and signal data. In steganography, CNN layers are used to learn the hierarchy of image features, a hierarchy ranging from low-level generic features to high-level specific features.

4. Steganalysis

Steganalysis is the study of detecting messages hidden using steganography. Steganography can be found through a process called steganalysis, which looks for differences in bit patterns and unusually large file sizes. Steganalysis's primary goals are to identify questionable data streams, ascertain whether or not secret messages have been encoded into them, and, if necessary, extract the concealed data.

Steganography typically starts with a number of dubious data streams, but it's unclear which of these contain concealed messages. There are several steganalysis approaches, including the ones listed below:

Strange patterns in a stego image are unbelievable. For example, some disc analysis services can filter hidden data in idle storage device divisions.

Filters can also be used to identify TCP/IP packets with obfuscated or incorrect header information. Data can be transported across the Internet using TCP/IP packets, which have reserved or unused space in their headers.

Visual detection Analyzing recurring patterns might help identify buried data or steganography tools. As the strategy is to compare the original cover image with the stego image and look for differences, these patterns can be analyzed. The term "known-carrier attack" refers to this. It is feasible for patterns to appear as signatures to a steganography program by comparing many photos. Padding or image cropping is another visual indicator of the existence of concealed data.

If an image doesn't fit into a defined size with various Stego tools, it is cropped or padded with black spaces. Additionally, the stego-image and the cover picture may have different file sizes. Another sign is a sharp rise or fall in the frequency of a particular color, or colors in a palette that develop gradually rather than at random.

The goal of this project was to create a steganographic model which could bypass such steganalysis tools.

CHAPTER II
RELATED WORKS

1. Research Paper – 1: Data Encryption & Decryption Using Steganography

In this paper, video steganography methods is used to perform secure steganography communication. Video steganography is the purpose of hiding the data in video files. While in cryptography, the message has been encrypted but when to communicate with the third party, the encrypted message can be decrypted very easily and destroyed. While in steganography, it hides secret information in cover files, this is the way that the data is also hidden so that while communicateng unwanted access cannot be done easily between two parties. The concept of hiding secret data in video stream files. And the first step is video steganography, grab any video stream file, embed with cover file and it turns into stegno file with encryption & decryption and sent to the receiver. The receiver does extract the data using stegno tools from stegno file into a video file and hidden data.

2. Research Paper – 2: Information hiding in images using steganography techniques

Steganography is a technique that prevents unauthorized users to have access to the important data. This paper has reviewed some techniques of steganography and digital watermarking in both spatial and frequency domains. While other techniques such as cryptography aim to prevent adversaries from reading the secret message, steganography aims to hide the presence of the message itself. Digital watermarking is one of the most widely used application for steganography technique. Users can hide important information within an image using steganography. It can be visible or invisible. In Spatial Domain Watermarking, the widest and simplest method is Least Significant Bit (LSB), which is replacing the LSB in each pixel by information that intends to hide. Images have some unimportant regions the human visual system cannot recognize by replacing these regions with other information.

3. Research Paper - 3: Hiding a Large Amount of Data with High Security using Steganography Algorithm

Steganography is a method of information hiding and invisible communication. Its techniques hide the presence of the message itself, from an observer so there would be no knowledge of the message's existence. Here, we are hiding these messages in the form of images. Basically, the cover source of the message can be altered in "noisy areas" with many color variations, so less attention would be drawn towards that area. The given paper explains that when a 24-bit color image is used, each of the 3 RBG components can be used, so a total of 3 bits can be stored in each pixel, which would mean that in an 800*600-pixel image, a total of 1440000 bits of secret data can be contained. But since, just 3 bits from such a huge number would be a waste of the size, the newly devised algorithm hopes to insert more than one bit at each byte in one pixel of the cover-image and give better results.

CHAPTER III
EXISTING WORK

Image Steganography Using Deep Learning Techniques

Digital image steganography is the process of embedding information within a cover image in a secure, imperceptible, and recoverable way. The three main methods of digital image steganography are spatial, transform, and neural network methods. Spatial methods modify the pixel values of an image to embed information, while transform methods embed hidden information within the frequency of the image. Neural network-based methods use neural networks to perform the hiding process, which is the focus of the proposed methodology.
This research explores the use of deep convolutional neural networks (CNNs) in digital image steganography. This work extends an existing implementation that used a two-dimensional CNN to perform the preparation, hiding, and extraction phases of the steganography process. The methodology proposed in this research, however, introduced changes into the structure of the CNN and used a gain function based on several image similarity metrics to maximize the imperceptibility between a cover and steganographic image.
The performance of the proposed method was measured using some frequently utilized image metrics such as structured similarity index measurement (SSIM), mean square error (MSE), and peak signal to noise ratio (PSNR). The results showed that the steganographic images produced by the proposed methodology are imperceptible to the human eye, while still providing good recoverability. Comparing the results of the proposed methodology to the results of the original methodology revealed that our proposed network greatly improved over the base methodology in terms of SSIM and compares well to existing steganography methods.

Image Steganography Using Neural Networks

Abstract In this paper, they clarify what steganography is and what it can do. They contrast it with the related disciplines of cryptography and traffic security, present a unified terminology agreed the subject, and outline a number of approaches many of them developed to hide encrypted copyright marks or serial numbers in digital audio or video. They then present a number of attacks, some new, on such information hiding schemes. This leads to a discussion of the formidable obstacles that lie in the way of a general theory of information hiding systems. However, theoretical considerations lead to ideas of practical value, such as the use of parity checks to amplify covertness and provide public key steganography. Finally, they show that public key information hiding systems exist, and are not necessarily constrained to the case where the warden is passive. The proposed survey mainly focused on Neural Network to train the inputs and hidden layers in image steganography.

CHAPTER IV

SIMULATION / IMPLEMENTATION RESULTS

For the CNN model, it is composed of three parts: The Preparation Network, Hiding Network (Encoder) and the Reveal Network.
The goal is to be able to encode information about the secret image S into the cover image C, generating C' that closely resembles C, while still being able to decode information from C' to generate the decoded secret image S', 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 than transforms that input into the encoded cover image C'. Finally, the Reveal Network decodes the secret image S' from C'.
For stability, we add noise before the Reveal Network (reference through some paper). Although the author of the paper didn't originally specify the architecture of the three networks.
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, we use only 2 layers with the same structure.
On running the code and training the model,

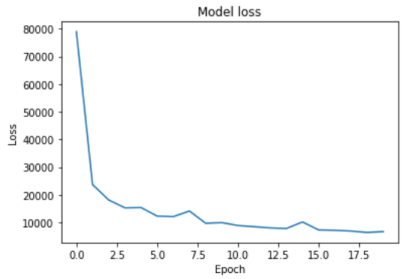


Figure1

```
:: Welcome to Steganography ::
1. Encode
2. Decode
1
Enter image name(with extension) : dragon.jpeg
Enter data to be encoded : isa
Enter the name of new image(with extension) : susnormal.jpeg
```

Figure 2

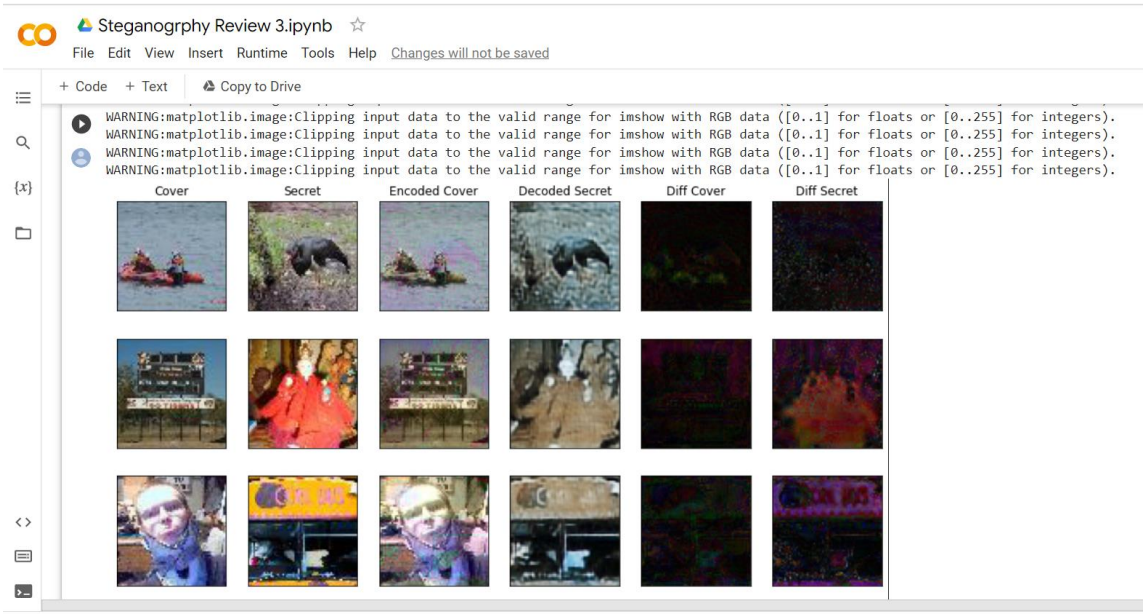


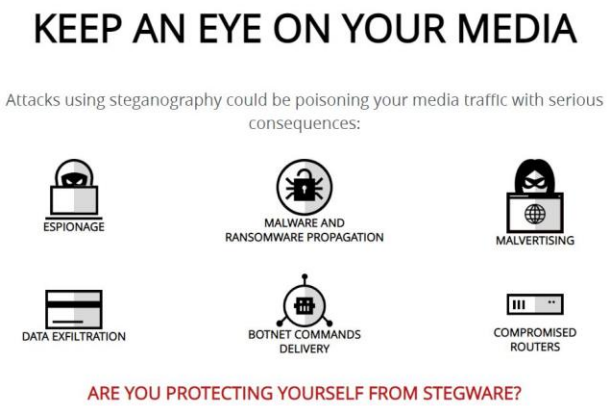
Figure 3

Sample data that has been put through the code, for cover images, which are prepared and then the secret images are encoded into them. Then the new cover image is displayed. Afterwords, the reveal network decodes the encoded image to give the secret image. The diff columns tells us the difference between the images from when they were not decoded to when they were encoded from the embedded cover image.

Results:

We used a webiste to check how well conceiled our information is after using steganography.

Source: https://www.mcafee.com/enterprise/en-us/downloads/free-tools/steganography.html



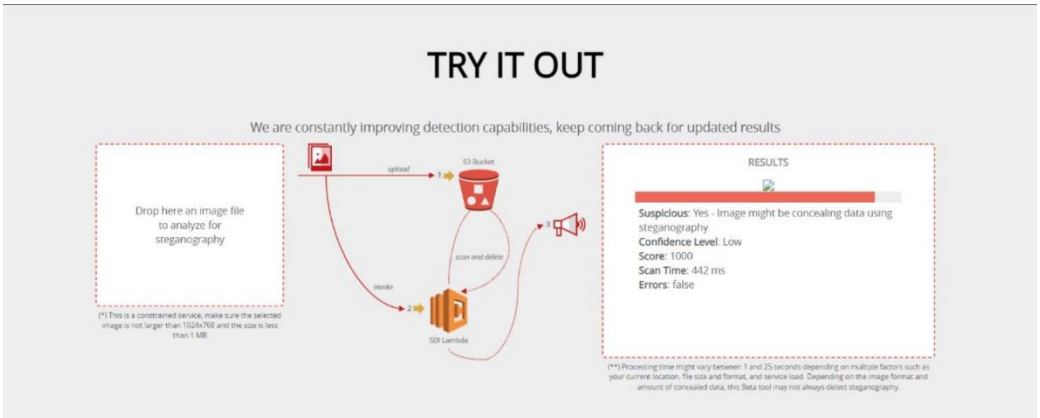


Figure 4

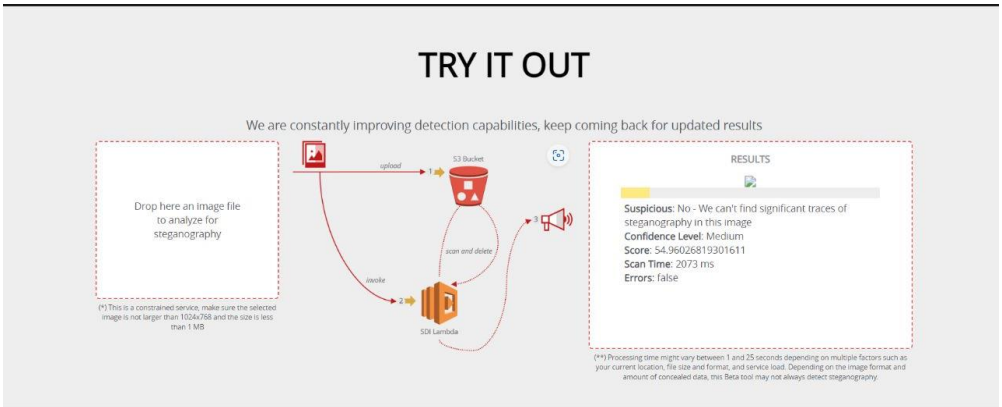


Figure 5

We can see how the deep learning model hides the data better.

CHAPTER V

CONCLUSION AND FUTURE WORKS

5.1 Conclusion

We were able to make the appropriate multi-layer steganographic model using LSB as a medium to hide text/image data into another image, and then use Convolution Neural Networks to hide the produced image into another, also creating a revealing network to be able to decode the data.

5.2 Future works

With this project, we believe we can further improve the deep learning model as with the evolution in Machine and deep learning techniques, we can improve on the model to be able to provide more security. we have also seen a lot of upcoming projects focused on using GAN for steganography as well, hence there is always room for improvement.

References

1. https://analyticsindiamag.com/ai-cryptic-messages-steganography-speech/
2. https://medium.com/buzzrobot/hiding-images-using-ai-deep-steganography-b7726bd58b06
3. https://youtu.be/8Rcr7_Khldk
4. https://youtu.be/Sb-Df9QW0sM
5. https://youtu.be/_o8nW3zF5r4
6. https://youtu.be/bsyaTUkgg_I
7. https://ieeexplore.ieee.org/document/9120935
8. https://www.researchgate.net/publication/259893801_Information_Hiding_in_Images_Using_Steganography_Techniques
9. http://docsdrive.com/pdfs/sciencepublications/jcssp/2007/223-232.pdf
10. https://www.ijert.org/image-steganography-using-neural-networks

Appendix

CODE:

```
# import the image net data set
!wget http://cs231n.stanford.edu/tiny-imagenet-200.zip


# Unzipping of our data set & Delete the Zip File

!unzip /content/tiny-imagenet-200.zip
!rm /content/tiny-imagenet-200.zip
!pip install tensorflow==2.7.0
### Imports ###
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateSchedu
ler, TensorBoard
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing import image
import tensorflow.keras.backend as K

import matplotlib.pyplot as plt

import numpy as np
import os
import random
import scipy.misc
from tqdm import *
### Constants ###
DATA_DIR = "/content/tiny-imagenet-200"
TRAIN_DIR = os.path.join(DATA_DIR, "train")
TEST_DIR = os.path.join(DATA_DIR, "test")
IMG_SHAPE = (64, 64)
# Dataset creation
# The Dataset we used is Tiny ImageNet Visual Recognition Challenge.
# Our training set is made of a random subset of images from all 200 class
es.

def load_dataset_small(num_images_per_class_train=10, num_images_test=500)
:
    """Loads training and test datasets, from Tiny ImageNet Visual Recogit
ion Challenge.

    Arguments:
        num_images_per_class_train: number of images per class to load int
o training dataset.
        num_images_test: total number of images to load into training data
```

```
set.
    """
    X_train = []
    X_test = []

    # Create training set.
    for c in os.listdir(TRAIN_DIR):
        c_dir = os.path.join(TRAIN_DIR, c, 'images')
        c_imgs = os.listdir(c_dir)
        random.shuffle(c_imgs)
        for img_name_i in c_imgs[0:num_images_per_class_train]:
            img_i = image.load_img(os.path.join(c_dir, img_name_i))
            x = image.img_to_array(img_i)
            X_train.append(x)
    random.shuffle(X_train)

    # Create test set.
    test_dir = os.path.join(TEST_DIR, 'images')
    test_imgs = os.listdir(test_dir)
    random.shuffle(test_imgs)
    for img_name_i in test_imgs[0:num_images_test]:
        img_i = image.load_img(os.path.join(test_dir, img_name_i))
        x = image.img_to_array(img_i)
        X_test.append(x)

    # Return train and test data as numpy arrays.
    return np.array(X_train), np.array(X_test)
# Load dataset.
X_train_orig, X_test_orig = load_dataset_small()

# Normalize image vectors.
X_train = X_train_orig/255.
X_test = X_test_orig/255.

# Print statistics.
print ("Number of training examples = " + str(X_train.shape[0]))
print ("Number of test examples = " + str(X_test.shape[0]))
print ("X_train shape: " + str(X_train.shape)) # Should be (train_size, 64
, 64, 3)


# Show sample images from the training dataset
fig=plt.figure(figsize=(8, 8))
columns = 4
rows = 5
for i in range(1, columns*rows +1):
```

```
    # Randomly sample from training dataset
    img_idx = np.random.choice(X_train.shape[0])
    fig.add_subplot(rows, columns, i)
    plt.axis("off")
    plt.imshow(X_train[img_idx])
plt.show()


# We split training set into two halfs.
# First half is used for training as secret images, second half for cover
images.
# S: secret image
input_S = X_train[0:X_train.shape[0] // 2]
# C: cover image
input_C = X_train[X_train.shape[0] // 2:]


beta = 1.0

# Loss for reveal network
def rev_loss(s_true, s_pred):
    # Loss for reveal network is: beta * |S-S'|
    return beta * K.sum(K.square(s_true - s_pred))

# Loss for the full model, used for preparation and hidding networks
def full_loss(y_true, y_pred):
    # Loss for the full model is: |C-C'| + beta * |S-S'|
    s_true, c_true = y_true[...,0:3], y_true[...,3:6]
    s_pred, c_pred = y_pred[...,0:3], y_pred[...,3:6]

    s_loss = rev_loss(s_true, s_pred)
    c_loss = K.sum(K.square(c_true - c_pred))
    return s_loss + c_loss


# Returns the encoder as a Keras model, composed by Preparation and Hiding
 Networks.

def make_encoder(input_size):
    input_S = Input(shape=(input_size))
    input_C= Input(shape=(input_size))

    # Preparation Network
    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_prep0_3x3')(input_S)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_prep0_4x4')(input_S)
```

```
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_prep0_5x5')(input_S)
    x = concatenate([x3, x4, x5])

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_prep1_3x3')(x)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_prep1_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_prep1_5x5')(x)
    x = concatenate([x3, x4, x5])

    x = concatenate([input_C, x])

    # Hiding network
    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_hid0_3x3')(x)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_hid0_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_hid0_5x5')(x)
    x = concatenate([x3, x4, x5])

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_hid1_3x3')(x)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_hid1_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_hid1_5x5')(x)
    x = concatenate([x3, x4, x5])

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_hid2_3x3')(x)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_hid2_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_hid2_5x5')(x)
    x = concatenate([x3, x4, x5])

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_hid3_3x3')(x)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_hid3_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_hid3_5x5')(x)
```

```
    x = concatenate([x3, x4, x5])

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_hid4_3x3')(x)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_hid4_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_hid5_5x5')(x)
    x = concatenate([x3, x4, x5])

    output_Cprime = Conv2D(3, (3, 3), strides = (1, 1), padding='same', ac
tivation='relu', name='output_C')(x)

    return Model(inputs=[input_S, input_C],
                 outputs=output_Cprime,
                 name = 'Encoder')


# Returns the decoder as a Keras model, composed by the Reveal Network
def make_decoder(input_size, fixed=False):

    # Reveal network
    reveal_input = Input(shape=(input_size))

    # Adding Gaussian noise with 0.01 standard deviation.
    input_with_noise = GaussianNoise(0.01, name='output_C_noise')(reveal_i
nput)

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_rev0_3x3')(input_with_noise)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_rev0_4x4')(input_with_noise)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_rev0_5x5')(input_with_noise)
    x = concatenate([x3, x4, x5])

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_rev1_3x3')(x)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_rev1_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_rev1_5x5')(x)
    x = concatenate([x3, x4, x5])

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_rev2_3x3')(x)
```

```
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_rev2_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_rev2_5x5')(x)
    x = concatenate([x3, x4, x5])

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_rev3_3x3')(x)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_rev3_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_rev3_5x5')(x)
    x = concatenate([x3, x4, x5])

    x3 = Conv2D(50, (3, 3), strides = (1, 1), padding='same', activation='
relu', name='conv_rev4_3x3')(x)
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='
relu', name='conv_rev4_4x4')(x)
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='r
elu', name='conv_rev5_5x5')(x)
    x = concatenate([x3, x4, x5])

    output_Sprime = Conv2D(3, (3, 3), strides = (1, 1), padding='same', ac
tivation='tanh', name='output_S')(x)

    if not fixed:
        return Model(inputs=reveal_input,
                     outputs=output_Sprime,
                     name = 'Decoder')
    else:
        return Network(inputs=reveal_input,
                       outputs=output_Sprime,
                       name = 'DecoderFixed')


# Full model.
def make_model(input_size):
    input_S = Input(shape=(input_size))
    input_C = Input(shape=(input_size))

    encoder = make_encoder(input_size)

    decoder = make_decoder(input_size)
    decoder.compile(optimizer='adam', loss=rev_loss)
    decoder.trainable = False
```

```
    output_Cprime = encoder([input_S, input_C])
    output_Sprime = decoder(output_Cprime)

    autoencoder = Model(inputs=[input_S, input_C],
                        outputs=concatenate([output_Sprime, output_Cprime]
))
    autoencoder.compile(optimizer='adam', loss=full_loss)

    return encoder, decoder, autoencoder


def lr_schedule(epoch_idx):
    if epoch_idx < 200:
        return 0.001
    elif epoch_idx < 400:
        return 0.0003
    elif epoch_idx < 600:
        return 0.0001
    else:
        return 0.00003


def train(e, lr = 0.001):

  NB_EPOCHS = e
  BATCH_SIZE = 32



  encoder_model, reveal_model, autoencoder_model = make_model(input_S.shap
e[1:])

  K.set_value(autoencoder_model.optimizer.lr,lr)
  K.set_value(reveal_model.optimizer.lr,lr)
  m = input_S.shape[0]
  loss_history = []

  for epoch in range(NB_EPOCHS):

      np.random.shuffle(input_S)
      np.random.shuffle(input_C)
      t = tqdm(range(0, input_S.shape[0], BATCH_SIZE),mininterval=0)

      ae_loss = []
      rev_loss = []
```

```
        i = 0

        for idx in t:

            batch_S = input_S[idx:min(idx + BATCH_SIZE, m)]
            batch_C = input_C[idx:min(idx + BATCH_SIZE, m)]

            C_prime = encoder_model.predict([batch_S, batch_C])
            aeLoss = autoencoder_model.train_on_batch(x=[batch_S, batch_C],
                                                      y=np.concatenate((batc
h_S, batch_C),axis=3))

            ae_loss.append(aeLoss)

            revLoss = reveal_model.train_on_batch(x=C_prime, y=batch_S)
            rev_loss.append(revLoss)


            i = idx
            t.set_description('Epoch {} | Batch: {:3} of {}. Loss AE {:10.2f
} | Loss Rev {:10.2f} | lr {}'.format(epoch + 1, i, m, np.mean(ae_loss), n
p.mean(rev_loss), lr ))


        lr = lr_schedule(epoch + 1)
        mean_ae_loss = np.mean(ae_loss)
        loss_history.append(mean_ae_loss)

    autoencoder_model.save_weights('./model_weights.hdf5')
    autoencoder_model.save("model.h5")
    return loss_history, autoencoder_model

loss_history, autoencoder_model = train(20) # train funtion takes number o
f epochs as a input as i have traied my model for 1000 epochs i have saved



# Plot loss through epochs
plt.plot(loss_history)
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()


from keras.models import load_model
```

```
# Load model

autoencoder_model.load_weights('/content/model_weights.hdf5')


# Retrieve decoded predictions.
decoded = autoencoder_model.predict([input_S, input_C])
decoded_S, decoded_C = decoded[...,0:3], decoded[...,3:6]

# Get absolute difference between the outputs and the expected values.
diff_S, diff_C = np.abs(decoded_S - input_S), np.abs(decoded_C - input_C)


def pixel_errors(input_S, input_C, decoded_S, decoded_C):
    """Calculates mean of Sum of Squared Errors per pixel for cover and se
cret images. """
    see_Spixel = np.sqrt(np.mean(np.square(255*(input_S - decoded_S))))
    see_Cpixel = np.sqrt(np.mean(np.square(255*(input_C - decoded_C))))

    return see_Spixel, see_Cpixel

def pixel_histogram(diff_S, diff_C):
    """Calculates histograms of errors for cover and secret image. """
    diff_Sflat = diff_S.flatten()
    diff_Cflat = diff_C.flatten()

    fig = plt.figure(figsize=(15, 5))
    a=fig.add_subplot(1,2,1)

    imgplot = plt.hist(255* diff_Cflat, 100, normed=1, alpha=0.75, facecol
or='red')
    a.set_title('Distribution of error in the Cover image.')
    plt.axis([0, 250, 0, 0.2])

    a=fig.add_subplot(1,2,2)
    imgplot = plt.hist(255* diff_Sflat, 100, normed=1, alpha=0.75, facSeco
lor='red')
    a.set_title('Distribution of errors in the Secret image.')
    plt.axis([0, 250, 0, 0.2])

    plt.show()
# Print pixel-wise average errors in a 256 scale.
S_error, C_error = pixel_errors(input_S, input_C, decoded_S, decoded_C)

print ("S error per pixel [0, 255]:", S_error)
```

```
print ("C error per pixel [0, 255]:", C_error)
# Configs for results display

# Show images in gray scale
SHOW_GRAY = False
# Show difference bettwen predictions and ground truth.
SHOW_DIFF = True

# Diff enhance magnitude
ENHANCE = 1

# Number of secret and cover pairs to show.
n = 6

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

def show_image(img, n_rows, n_col, idx, gray=False, first_row=False, title
=None):
    ax = plt.subplot(n_rows, n_col, idx)
    if gray:
        plt.imshow(rgb2gray(img), cmap = plt.get_cmap('gray'))
    else:
        plt.imshow(img)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    if first_row:
        plt.title(title)

plt.figure(figsize=(14, 15))
rand_indx = [random.randint(0, 1000) for x in range(n)]
# for i, idx in enumerate(range(0, n)):
for i, idx in enumerate(rand_indx):
    n_col = 6 if SHOW_DIFF else 4

    show_image(input_C[idx], n, n_col, i * n_col + 1, gray=SHOW_GRAY, firs
t_row=i==0, title='Cover')

    show_image(input_S[idx], n, n_col, i * n_col + 2, gray=SHOW_GRAY, firs
t_row=i==0, title='Secret')

    show_image(decoded_C[idx], n, n_col, i * n_col + 3, gray=SHOW_GRAY, fi
rst_row=i==0, title='Encoded Cover')

    show_image(decoded_S[idx], n, n_col, i * n_col + 4, gray=SHOW_GRAY, fi
```

```
rst_row=i==0, title='Decoded Secret')


    if SHOW_DIFF:
        show_image(np.multiply(diff_C[idx], ENHANCE), n, n_col, i * n_col
+ 5, gray=SHOW_GRAY, first_row=i==0, title='Diff Cover')

        show_image(np.multiply(diff_S[idx], ENHANCE), n, n_col, i * n_col
+ 6, gray=SHOW_GRAY, first_row=i==0, title='Diff Secret')

plt.show()


plt.figure(figsize=(14, 15))
show_image(decoded_C[5], n, n_col, i * n_col + 2, gray=SHOW_GRAY, first_ro
w=i==0, title='Secret')
# Python program implementing Image Steganography

# PIL module is used to extract
# pixels of image and modify it
from PIL import Image

# Convert encoding data into 8-bit binary
# form using ASCII value of characters
def genData(data):

    # list of binary codes
    # of given data
    newd = []

    for i in data:
      newd.append(format(ord(i), '08b'))
    return newd

# Pixels are modified according to the
# 8-bit binary data and finally returned
def modPix(pix, data):

  datalist = genData(data)
  lendata = len(datalist)
  imdata = iter(pix)

  for i in range(lendata):

    # Extracting 3 pixels at a time
    pix = [value for value in imdata.__next__()[:3] +
```

```
                imdata.__next__()[:3] +
                imdata.__next__()[:3]]

    # Pixel value should be made
    # odd for 1 and even for 0
    for j in range(0, 8):
      if (datalist[i][j] == '0' and pix[j]% 2 != 0):
        pix[j] -= 1

      elif (datalist[i][j] == '1' and pix[j] % 2 == 0):
        if(pix[j] != 0):
          pix[j] -= 1
        else:
          pix[j] += 1
        # pix[j] -= 1

    # Eighth pixel of every set tells
    # whether to stop ot read further.
    # 0 means keep reading; 1 means thec
    # message is over.
    if (i == lendata - 1):
      if (pix[-1] % 2 == 0):
        if(pix[-1] != 0):
          pix[-1] -= 1
        else:
          pix[-1] += 1

    else:
      if (pix[-1] % 2 != 0):
        pix[-1] -= 1

    pix = tuple(pix)
    yield pix[0:3]
    yield pix[3:6]
    yield pix[6:9]

def encode_enc(newimg, data):
  w = newimg.size[0]
  (x, y) = (0, 0)

  for pixel in modPix(newimg.getdata(), data):

    # Putting modified pixels in the new image
    newimg.putpixel((x, y), pixel)
    if (x == w - 1):
```

```
      x = 0
      y += 1
    else:
      x += 1

# Encode data into image
def encode():
  img = input("Enter image name(with extension) : ")
  image = Image.open(img, 'r')

  data = input("Enter data to be encoded : ")
  if (len(data) == 0):
    raise ValueError('Data is empty')

  newimg = image.copy()
  encode_enc(newimg, data)

  new_img_name = input("Enter the name of new image(with extension) : ")
  newimg.save(new_img_name, str(new_img_name.split(".")[1].upper()))

# Decode the data in the image
def decode():
  img = input("Enter image name(with extension) : ")
  image = Image.open(img, 'r')

  data = ''
  imgdata = iter(image.getdata())

  while (True):
    pixels = [value for value in imgdata.__next__()[:3] +
              imgdata.__next__()[:3] +
              imgdata.__next__()[:3]]

    # string of binary data
    binstr = ''

    for i in pixels[:8]:
      if (i % 2 == 0):
        binstr += '0'
      else:
        binstr += '1'

    data += chr(int(binstr, 2))
    if (pixels[-1] % 2 != 0):
      return data
```

```
# Main Function
def main():
  a = int(input(":: Welcome to Steganography ::\n"
            "1. Encode\n2. Decode\n"))
  if (a == 1):
    encode()

  elif (a == 2):
    print("Decoded Word : " + decode())
  else:
    raise Exception("Enter correct input")

# Driver Code
if __name__ == '__main__' :

  # Calling main function
  main()
```