Image Steganography on Cryptographic text using Neural Networks

MSc Research Project
Cyber Security

Aarsh Bararia
Student ID: x19215045

School of Computing
National College of Ireland

Supervisor: Prof. Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aarsh Rajesh Baraia		
Student ID:	x19215045		
Programme:	MSc in Cyber Security	**Year:**	2020-2021
Module:	MSc Research Project		
Supervisor:	Prof. Imran Khan		
Submission Due Date:	16th August, 2021		
Project Title:	Image Steganography on Cryptographic text using Neural Networks		
Word Count:	799 **Page Count:** 11		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aarsh Bararia

Student ID: x19215045

1 Introduction

This configuration manual gives a thorough report on the system configurations utilized and codes information for the following three modelling phases:

"**Image Steganography on Cryptographic text using Neural Networks"**

2 System Configurations

This section contains the Hardware and Software specifications required to perform this research.

2.1 Hardware needs

- **OS type:** Windows 10 Home Single Language
- **Internal storage:** 1TB HDD
- **Processor:** Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
- **Installed RAM:** 8.00GB (7.84 GB usable)
- **GPU:** NVIDIA GeForce GTX 1650 4.00 GB

2.2 Software Requirement

- **Anaconda Environment-Jupyter Notebook:** An open-source platform supplied by the firm Anaconda enables the installation and implementation of several applications such as the Jupyter Notebook and Python spyder, R-Studio R programming and so on. 1 The Jupyter Notebook is used for this research study to carry out the Python programming and machine learning activities.
- **Virtual Studios:** Microsoft's source code editor for Windows, Linux and MacOS is Visual Studio Code. Features include debugging support, syntax highlighting, smart code completion, snippets, and embedded Git.
- **Python Programming Language:** Python is installed in the system and a Global Environment is created to perform Deep Learning tasks and other parts of the proposed research using the Jupyter Notebook. Python version in use is version 3.8.8.

3 Research Project Advancement

The proposed research project is implemented with the help of above-mentioned Design Requirements. This research is divided into four phases first is to perform AES encryption over a text message, second part was data pre-processing of image dataset to make it useful for further parts of experiment, third part is to hide the AES encrypted message inside an image, and the last part involves designing a neural network to perform multi-image steganography.

3.1 Data Pre-processing

The database used for the proposed research is an image dataset is a collection of google images of arts, culture, places and food images. All the images were scattered across many different folders. The Data pre-processing includes collecting all the images in one folder. Dividing the main dataset in two parts Cover and secret images. Creating train and validation datasets and also resizing and making all images in equal shapes. Below is the implementation of data pre-processing:

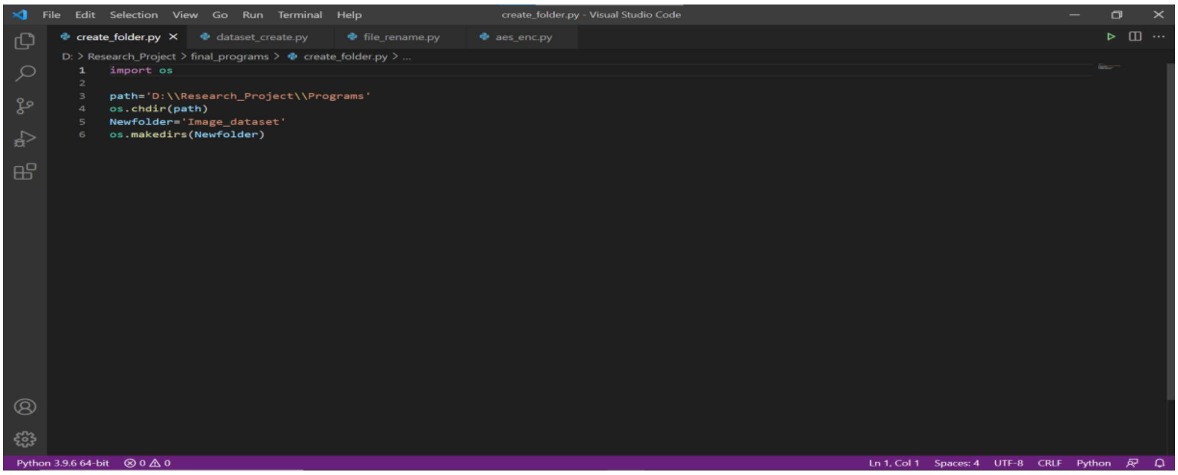


Figure 1 ***Creating new folder for dataset***

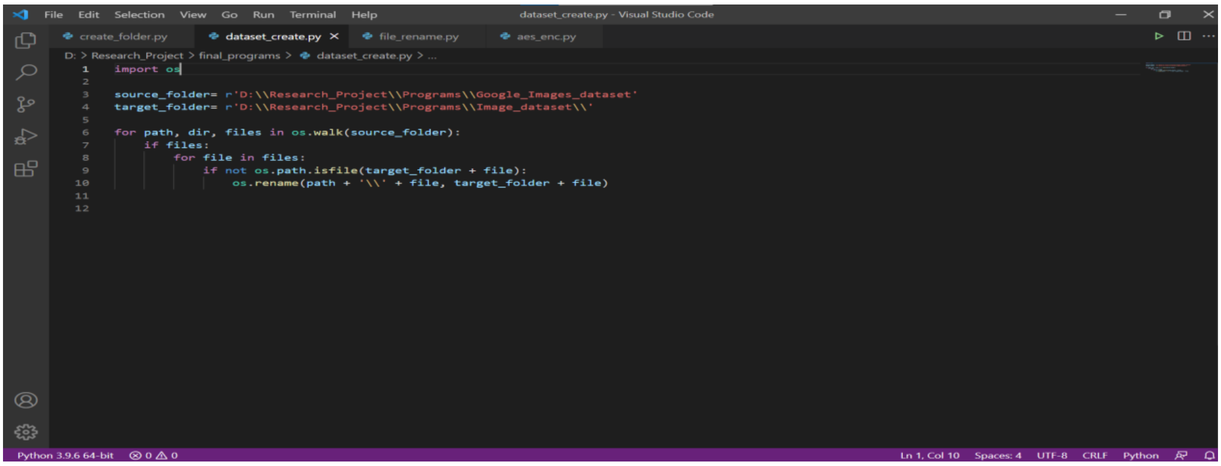


Figure 2 Collecting all the images in one folder.

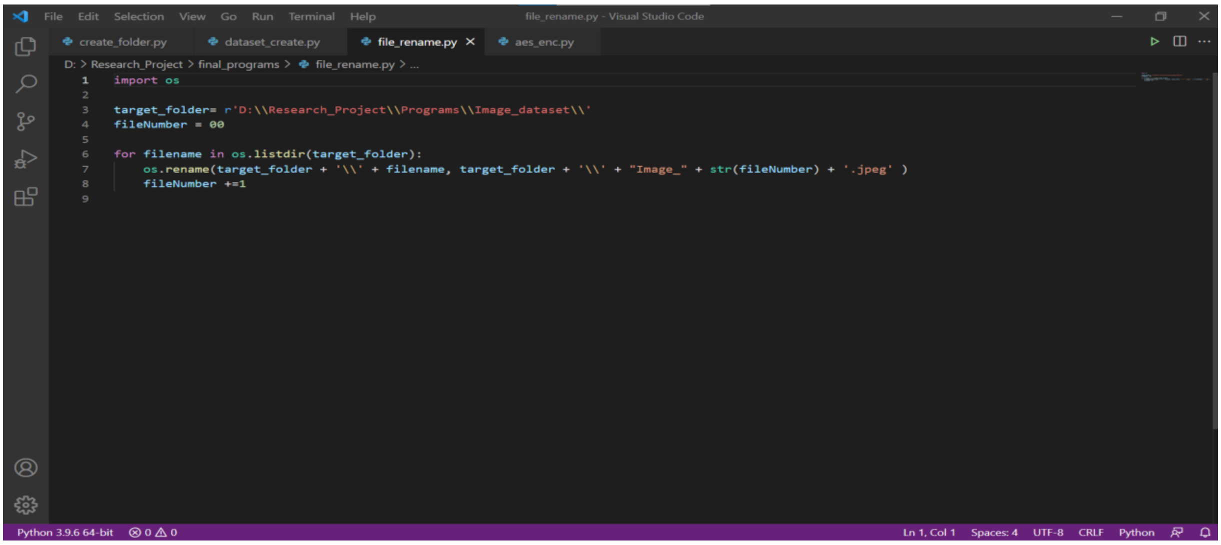
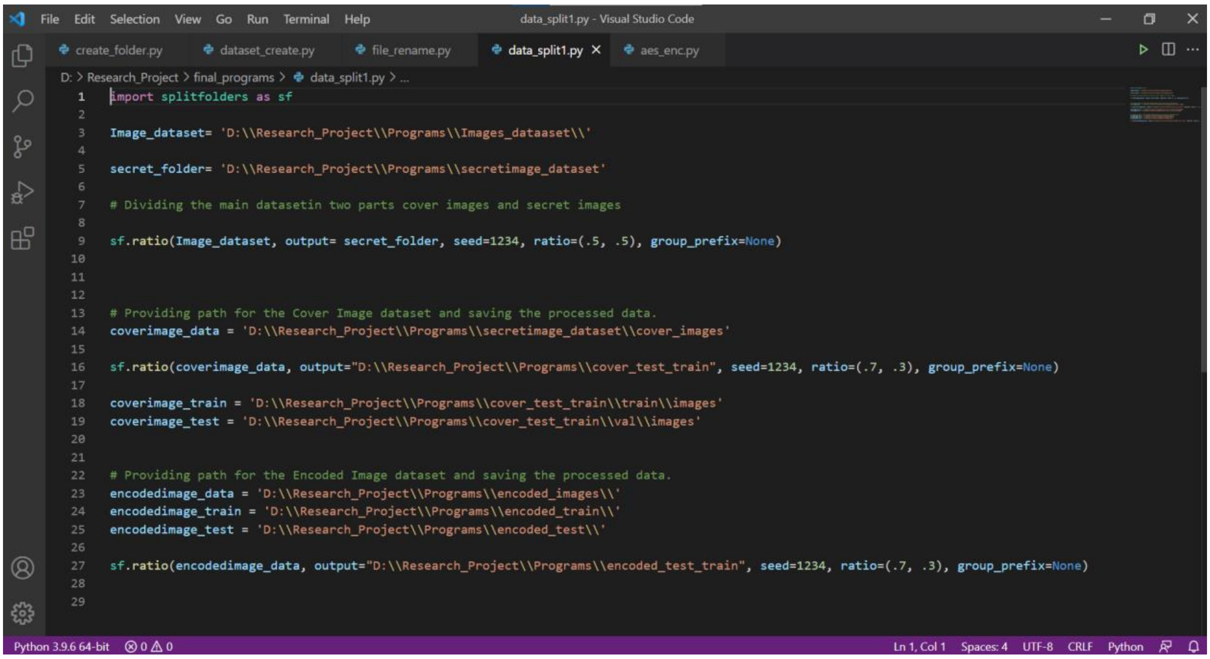


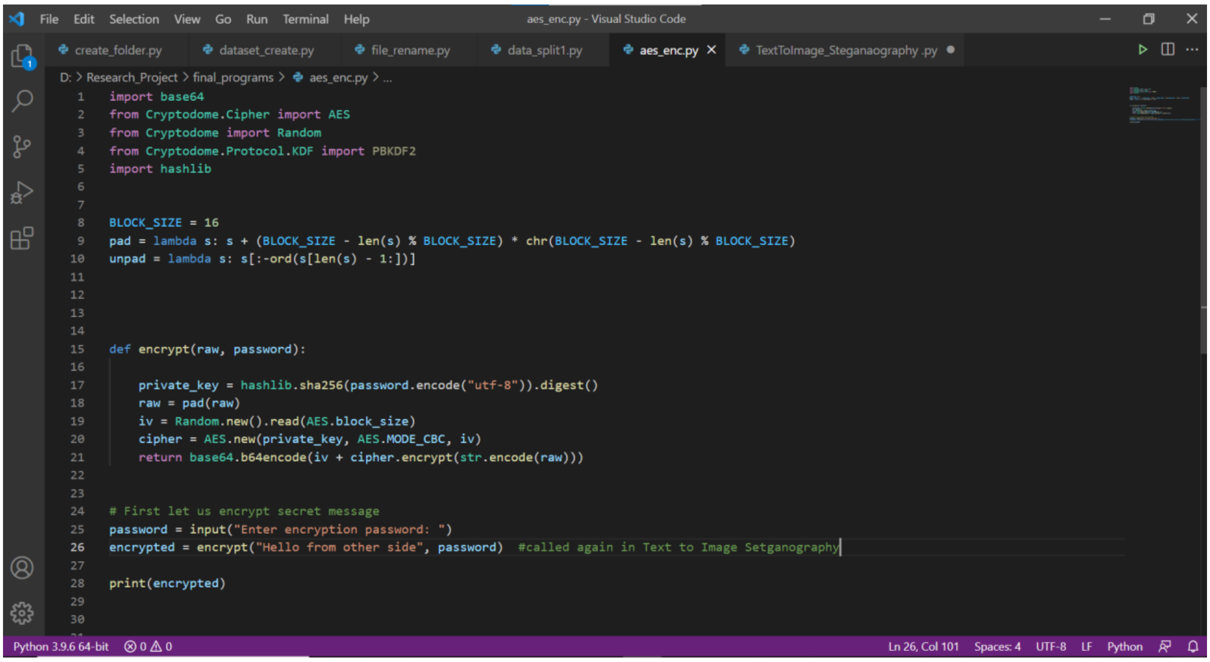
Figure 3 Renaming all images



```python
import splitfolders as sf

Image_dataset= 'D:\\Research_Project\\Programs\\Images_dataaset\\'

secret_folder= 'D:\\Research_Project\\Programs\\secretimage_dataset'

# Dividing the main datasetin two parts cover images and secret images

sf.ratio(Image_dataset, output= secret_folder, seed=1234, ratio=(.5, .5), group_prefix=None)



# Providing path for the Cover Image dataset and saving the processed data.
coverimage_data = 'D:\\Research_Project\\Programs\\secretimage_dataset\\cover_images'

sf.ratio(coverimage_data, output="D:\\Research_Project\\Programs\\cover_test_train", seed=1234, ratio=(.7, .3), group_prefix=None)

coverimage_train = 'D:\\Research_Project\\Programs\\cover_test_train\\train\\images'
coverimage_test = 'D:\\Research_Project\\Programs\\cover_test_train\\val\\images'


# Providing path for the Encoded Image dataset and saving the processed data.
encodedimage_data = 'D:\\Research_Project\\Programs\\encoded_images\\'
encodedimage_train = 'D:\\Research_Project\\Programs\\encoded_train\\'
encodedimage_test = 'D:\\Research_Project\\Programs\\encoded_test\\'

sf.ratio(encodedimage_data, output="D:\\Research_Project\\Programs\\encoded_test_train", seed=1234, ratio=(.7, .3), group_prefix=None)
```

Figure 4: Splitting dataset in Cover and Secret

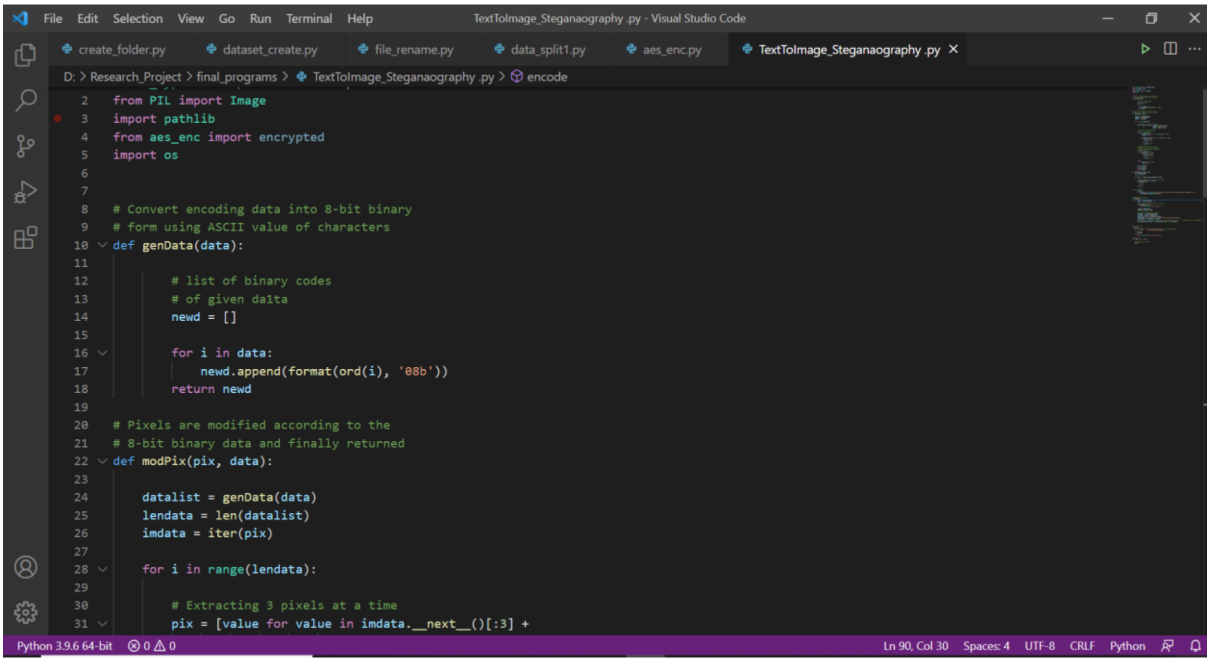
This is the second step of execution of the proposed experiment.



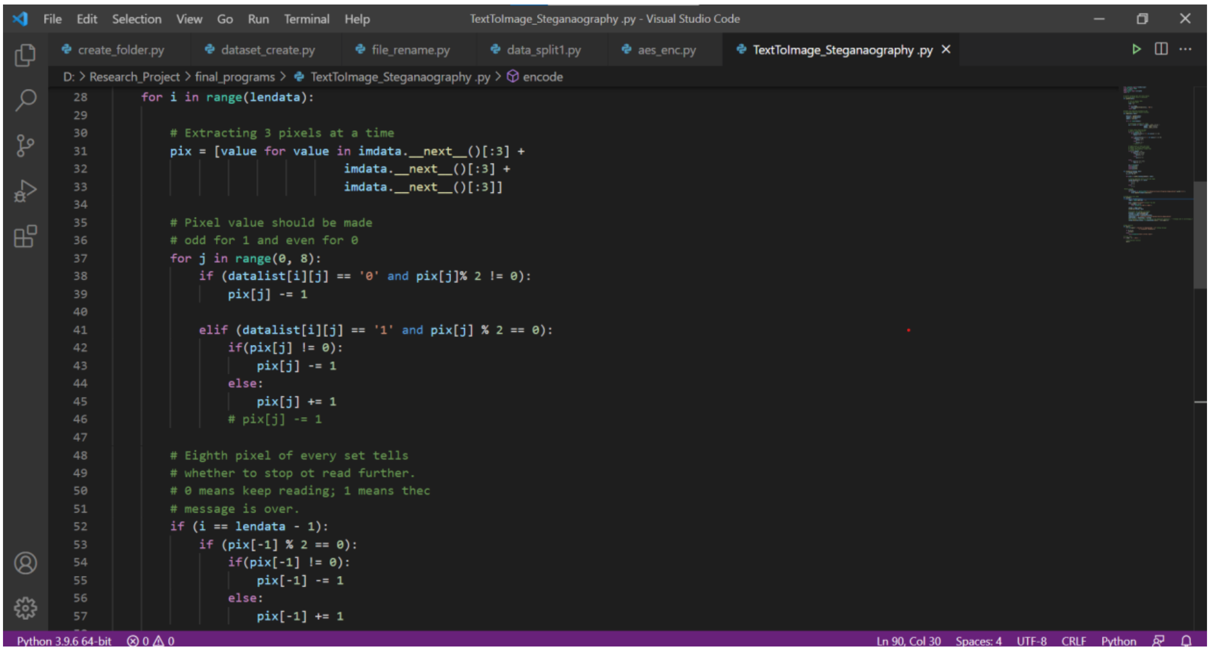
```python
import base64
from Cryptodome.Cipher import AES
from Cryptodome import Random
from Cryptodome.Protocol.KDF import PBKDF2
import hashlib


BLOCK_SIZE = 16
pad = lambda s: s + (BLOCK_SIZE - len(s) % BLOCK_SIZE) * chr(BLOCK_SIZE - len(s) % BLOCK_SIZE)
unpad = lambda s: s[:-ord(s[len(s) - 1:])]




def encrypt(raw, password):

    private_key = hashlib.sha256(password.encode("utf-8")).digest()
    raw = pad(raw)
    iv = Random.new().read(AES.block_size)
    cipher = AES.new(private_key, AES.MODE_CBC, iv)
    return base64.b64encode(iv + cipher.encrypt(str.encode(raw)))


# First let us encrypt secret message
password = input("Enter encryption password: ")
encrypted = encrypt("Hello from other side", password)  #called again in Text to Image Setganography

print(encrypted)
```

Figure 5: AES Encryption using PKDF2

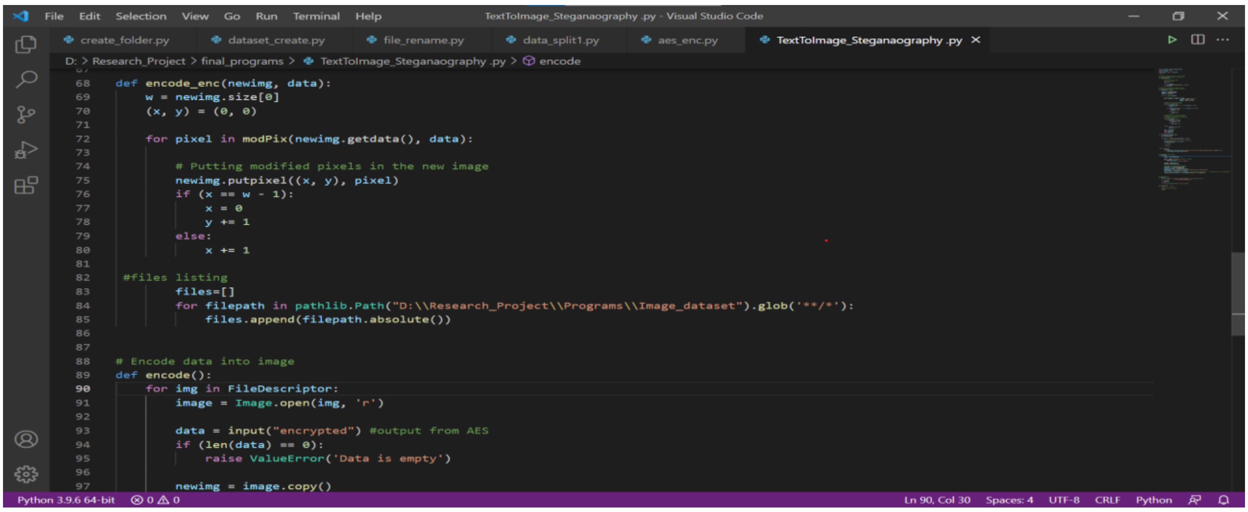
After the encryption is completed, the encrypted text generated by AES is hidden inside images using Text to Image steganography.



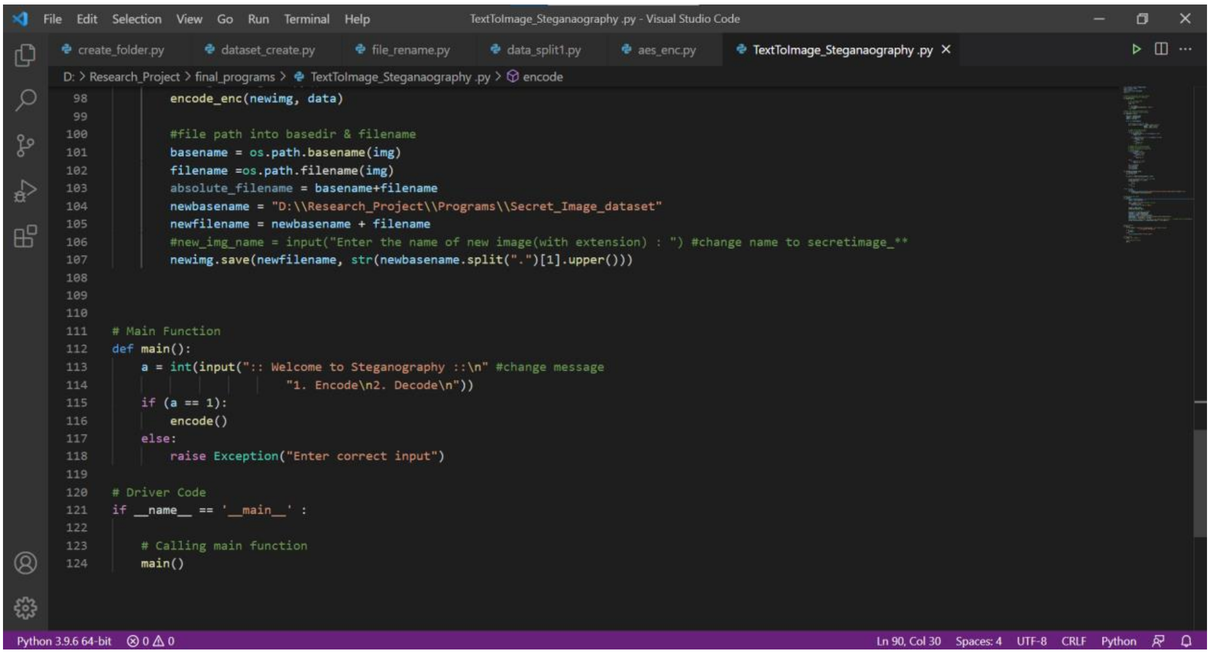
```python
from PIL import Image
import pathlib
from aes_enc import encrypted
import os


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

        # list of binary codes
        # of given da1ta
        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] +
```



```python
    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
```

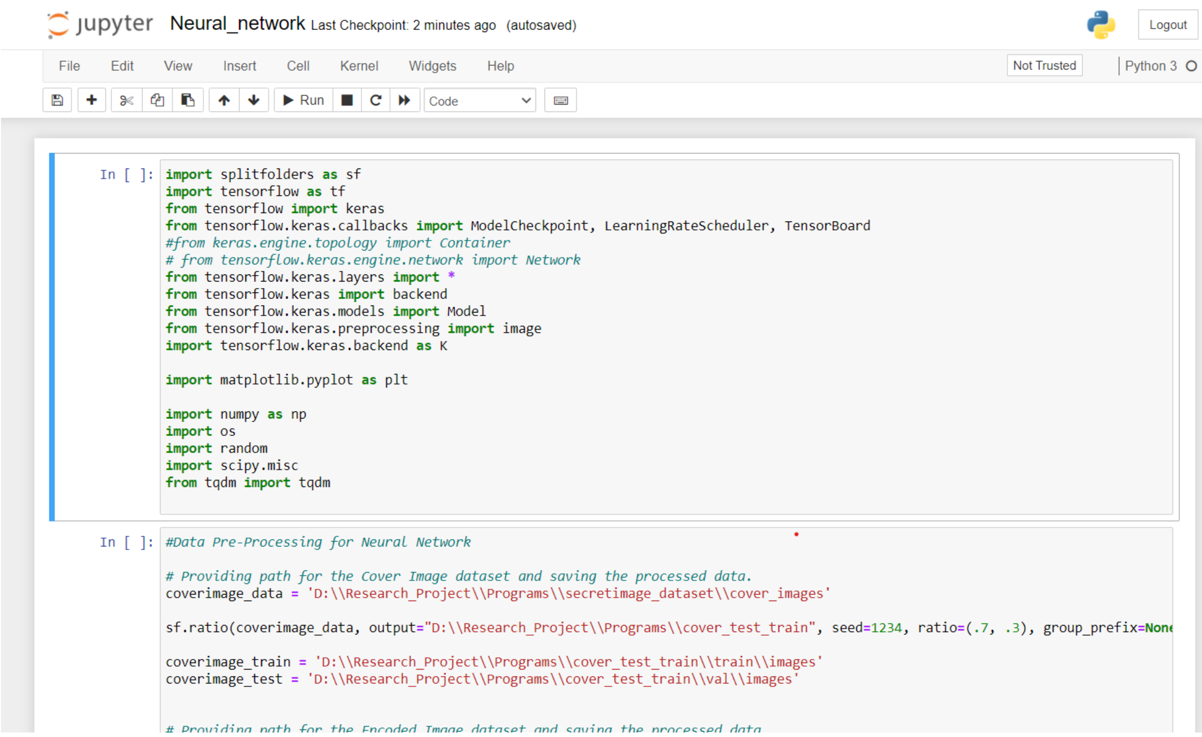


```python
def encode_enc(newimg, data):
    w = newimg.size[0]
    (x, y) = (0, 0)

    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

 #files listing
        files=[]
        for filepath in pathlib.Path("D:\\Research_Project\\Programs\\Image_dataset").glob('**/*'):
            files.append(filepath.absolute())


# Encode data into image
def encode():
    for img in FileDescriptor:
        image = Image.open(img, 'r')

        data = input("encrypted") #output from AES
        if (len(data) == 0):
            raise ValueError('Data is empty')

        newimg = image.copy()
```



```
        encode_enc(newimg, data)

        #file path into basedir & filename
        basename = os.path.basename(img)
        filename =os.path.filename(img)
        absolute_filename = basename+filename
        newbasename = "D:\\Research_Project\\Programs\\Secret_Image_dataset"
        newfilename = newbasename + filename
        #new_img_name = input("Enter the name of new image(with extension) : ") #change name to secretimage_**
        newimg.save(newfilename, str(newbasename.split(".")[1].upper()))



# Main Function
def main():
    a = int(input(":: Welcome to Steganography ::\n" #change message
                    "1. Encode\n2. Decode\n"))
    if (a == 1):
        encode()
    else:
        raise Exception("Enter correct input")

# Driver Code
if __name__ == '__main__' :

    # Calling main function
    main()
```

The last part of the code is to design Convolution Neural network with three networks Prep network, hiding network and Reveal network. This neural network focuses on encrypting the image output received after the text to image steganography. Neural network was designed in Jupyter Notebook.



```
import splitfolders as sf
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard
#from keras.engine.topology import Container
# from tensorflow.keras.engine.network import Network
from tensorflow.keras.layers import *
from tensorflow.keras import backend
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 tqdm
```

```
#Data Pre-Processing for Neural Network

# Providing path for the Cover Image dataset and saving the processed data.
coverimage_data = 'D:\\Research_Project\\Programs\\secretimage_dataset\\cover_images'

sf.ratio(coverimage_data, output="D:\\Research_Project\\Programs\\cover_test_train", seed=1234, ratio=(.7, .3), group_prefix=None

coverimage_train = 'D:\\Research_Project\\Programs\\cover_test_train\\train\\images'
coverimage_test = 'D:\\Research_Project\\Programs\\cover_test_train\\val\\images'
```

Figure 6: import all libraries

```
In [ ]: #Data Pre-Processing for Neural Network

        # Providing path for the Cover Image dataset and saving the processed data.
        coverimage_data = 'D:\\Research_Project\\Programs\\secretimage_dataset\\cover_images'

        sf.ratio(coverimage_data, output="D:\\Research_Project\\Programs\\cover_test_train", seed=1234, ratio=(.7, .3), group_prefix=None

        coverimage_train = 'D:\\Research_Project\\Programs\\cover_test_train\\train\\images'
        coverimage_test = 'D:\\Research_Project\\Programs\\cover_test_train\\val\\images'


        # Providing path for the Encoded Image dataset and saving the processed data.
        encodedimage_data = 'D:\\Research_Project\\Programs\\encoded_images\\'
        encodedimage_train = 'D:\\Research_Project\\Programs\\encoded_train\\'
        encodedimage_test = 'D:\\Research_Project\\Programs\\encoded_test\\'

        sf.ratio(encodedimage_data, output="D:\\Research_Project\\Programs\\encoded_test_train", seed=1234, ratio=(.7, .3), group_prefix=

        DATA_DIR = "D:\\Research_Project\\Programs\\Image_dataset\\"
        TRAIN_DIR = os.path.join(DATA_DIR, "train")
        TEST_DIR = os.path.join(DATA_DIR, "test")

        IMG_SHAPE = (64, 64)

In [ ]: #Part of Pre-processing

In [ ]: def load_dataset_small(num_images_per_class_train=10, num_images_test=500):
            """Loads training and test datasets, from Tiny ImageNet Visual Recognition Challenge.

            Arguments:
```

Figure 7: Data Pre-processing (part1)

```
In [ ]: #Part of Pre-processing

In [ ]: def load_dataset_small(num_images_per_class_train=10, num_images_test=500):
            """Loads training and test datasets, from Tiny ImageNet Visual Recognition Challenge.

            Arguments:
                num_images_per_class_train: number of images per class to load into training dataset.
                num_images_test: total number of images to load into training dataset.
            """
            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)
```

Figure 8: Data Pre-processing part 2

```python
# 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


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='relu', 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='relu', 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='relu', name='conv_hid0_5x5')(x)
    x = concatenate([x3, x4, x5])
```

Figure 9: Initialize losses and Prep network

```python
    # 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='relu', 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='relu', 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='relu', 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='relu', 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='relu', name='conv_hid5_5x5')(x)
    x = concatenate([x3, x4, x5])

    output_Cprime = Conv2D(3, (3, 3), strides = (1, 1), padding='same', activation='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
```

Figure 10: Initializing Hiding network

jupyter Neural_network Last Checkpoint: 9 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```python
# 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_input)

    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='relu', 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='relu', 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='relu', 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='relu', 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='relu', name='conv_rev5_5x5')(x)
    x = concatenate([x3, x4, x5])

    output_Sprime = Conv2D(3, (3, 3), strides = (1, 1), padding='same', activation='relu', name='output_S')(x)
```

Figure 11: Reveal network initialized

jupyter Neural_network Last Checkpoint: 11 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```python
    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
```

In [ ]:

```python
import wandb
wandb.init(project='stenography')
sweep_config = {
    'method': 'random', #grid, random
    'metric': {
      'name': 'rev_loss',
      'goal': 'minimize'
    },
    'parameters': {

        'lr':{
            'values':[0.001]
        },
        'activation':{
            'values':['relu']
        }
    }
}

sweep_id = wandb.sweep(sweep_config)
```

Figure 12: Initializing hyper-parameters

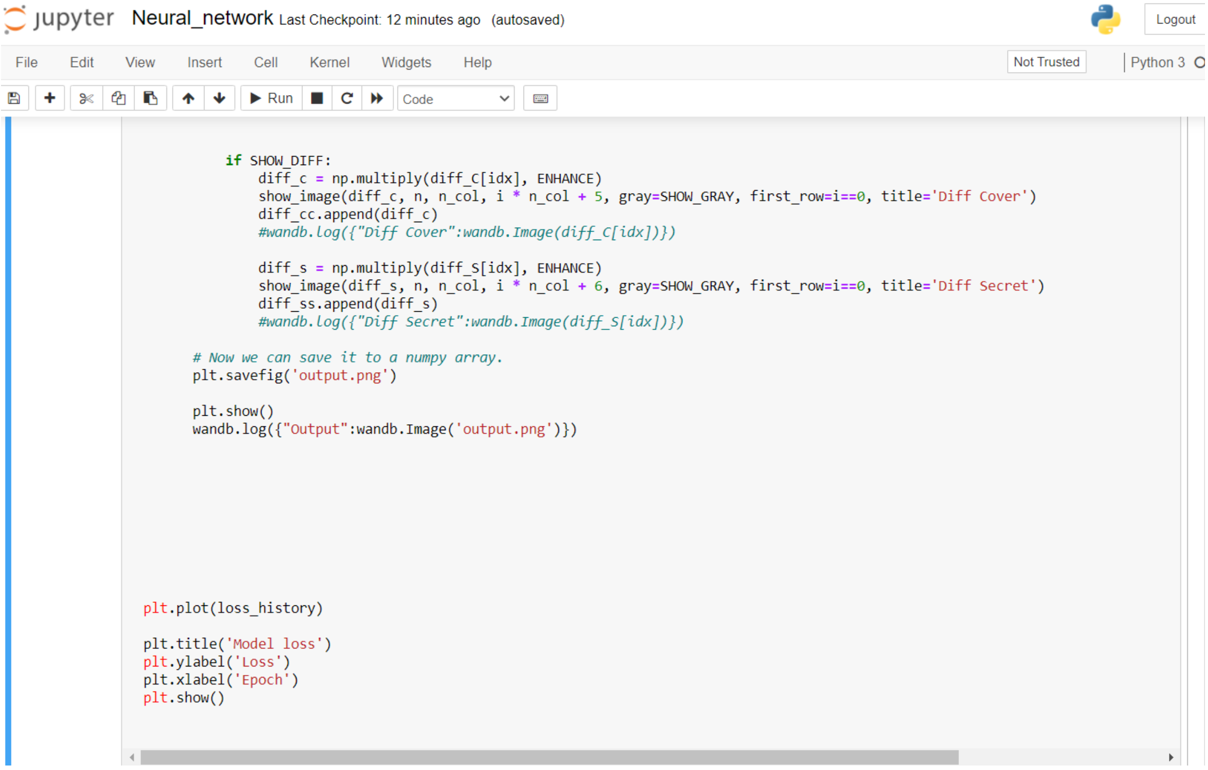


Figure 13: Making final plots