Image Steganography

CE903 Group Project

Image Steganography

- Steganography is the study of hiding information within objects in a way that deceives the viewer into thinking there is no information hidden.
- Keeping information concealed so that only the intended recipient can see it.

Types of steganography techniques

1) Text files
2) Image or picture files
3) Audio files
4) Video files

Deep Learning to predict stego image

- A deep learning model can be used to read, detect and segment the images.
- In this project, we have tried couple of CNN techniques.
- CNN is a deep learning algorithm that takes input data, usually images, assign weights or significance to different features in the object to be able to differentiate them.
- The best model to work on the medical data set is the U-Net Architecture

U-Net

U-Net architecture is designed for semantic segmentation (coloured pixels).

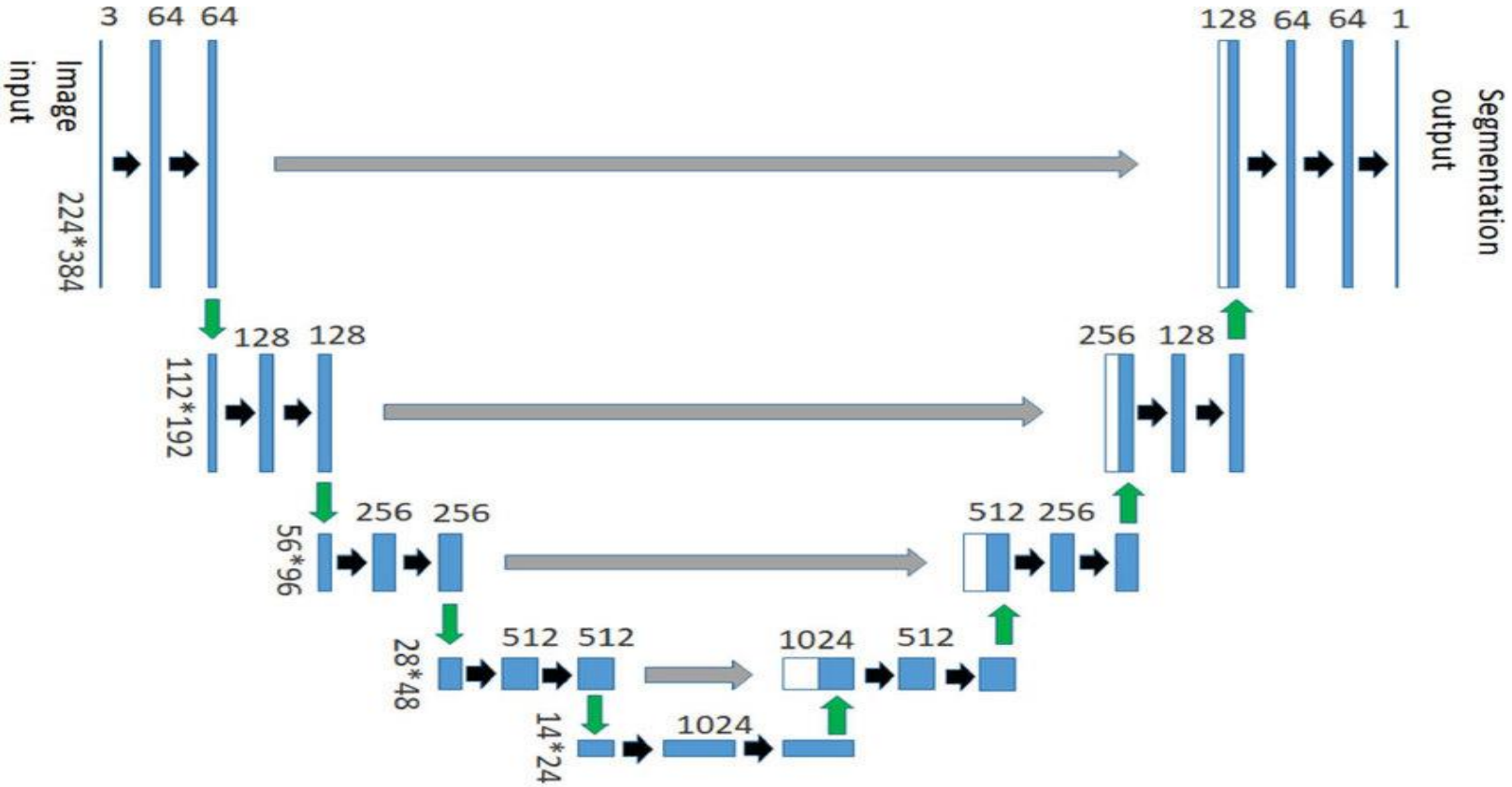
There are three types of segmentation techniques

- object detection
- semantic segmentation
- instance segmentation which is an extension of semantic segmentation

U-Net is named U because it is “U” shaped as shown in next slide

The architecture contains two paths

contraction or encoder path and expansion or decoder path.


3
64
64
Image input
224*384
128
64
64
1
Segmentation output
128
128
112*192
256
128
256
256
56*96
512
256
512
512
28*48
1024
512
1024
14*24

Implementation and their results

```python
# Main Function
def main():
  while(True):
    ch = int(input(":: Welcome to Steganography ::\n"
            "1. Encode\n2. Decode\n3. Exit\n"))
    if(ch==1):
        encode()

    elif(ch==2):
        print("Decoded Word : " + decode() + "\n")

    elif(ch==3):
        break

    else:
        raise Exception("Enter correct input")

# Driver Code
if __name__ == '__main__' :


  # Calling main function
  main()
```

Main Function Output

```
:: Welcome to Steganography ::
1. Encode
2. Decode
3. Exit
```

Encoding Function

```
def encode():
  # img = input("Enter image name(with extension) : ")
  for fno in range(1,110):

    img = '/content/folder/GROUP ORIGINAL IMAGES/IMG'+str(fno)+'.PNG'
    image = Image.open(img,'r')

    myfile=open('/content/folder/IMG'+str(fno)+'.TXT',"a")

  # data = input("Enter data to be encoded : ")
    data=random.choice(MESSAGES)
    myfile.write("Message: "+data)
    if (len(data) == 0):
      raise ValueError('Data is empty')

  # pwd = input("Enter Password to be encoded : ")
    pwd=random.choice(PASSWORDS)
    myfile.write("\nPassword: "+pwd)
    myfile.close()
    if (len(pwd) == 0):
      raise ValueError('Data is empty')

    newimg = image.copy()
  #print(newimg)
    encode_img(newimg, data, pwd)

  # new_img_name = input("Enter the name of new image(with extension) : ")
    new_img_name=img.replace('.PNG','s.PNG')
    newimg.save(new_img_name, str(new_img_name.split(".")[1].upper()))
```

Encoding Result

```
:: Welcome to Steganography ::
1. Encode
2. Decode
3. Exit
1
Enter image name(with extension) : IMGs1.png
Enter data to be encoded : But I must explain to you how all this mistaken
Enter Password to be encoded : 123456789
Enter the name of new image(with extension) : IMGss1.png
```

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

  datalist = genData(data) # binary converted list
  lendata = len(datalist)# length of binary data
  imdata = iter(pix) # iterate by every pixel

  for i in range(lendata):

    # 3 pixels(r,g,b) extracting at a time
    pix = [value for value in imdata.__next__()[:3]
          + imdata.__next__()[:3]
          + imdata.__next__()[:3]]
    # Changing pixel values, odd for 1 and 0 for even
    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

    # Eighth pixel of every set tells
    # whether to stop or 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] # returning tuples of pixels
```

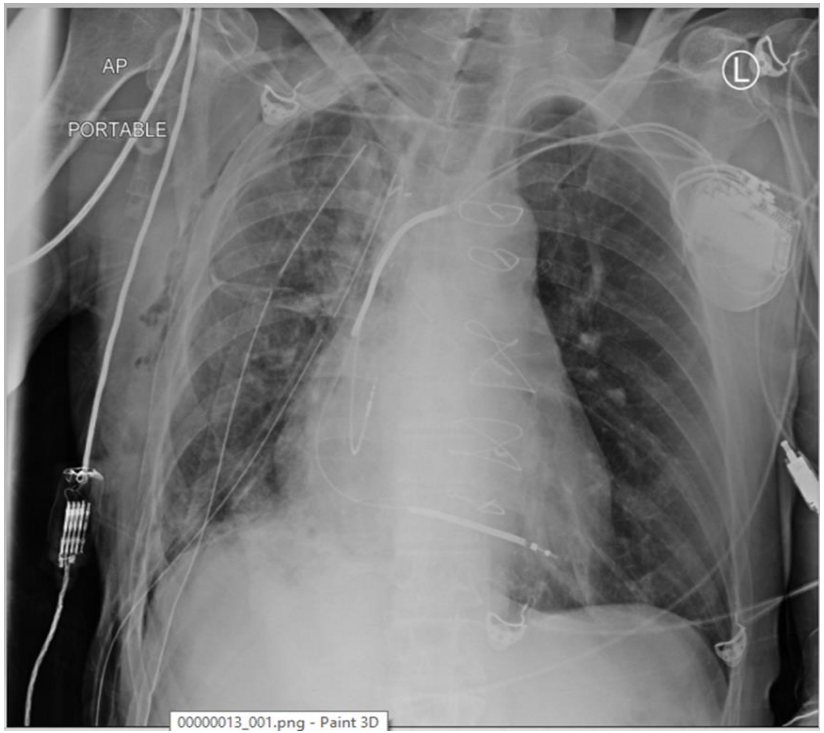
Storing Data and Password in Image

Decode Function

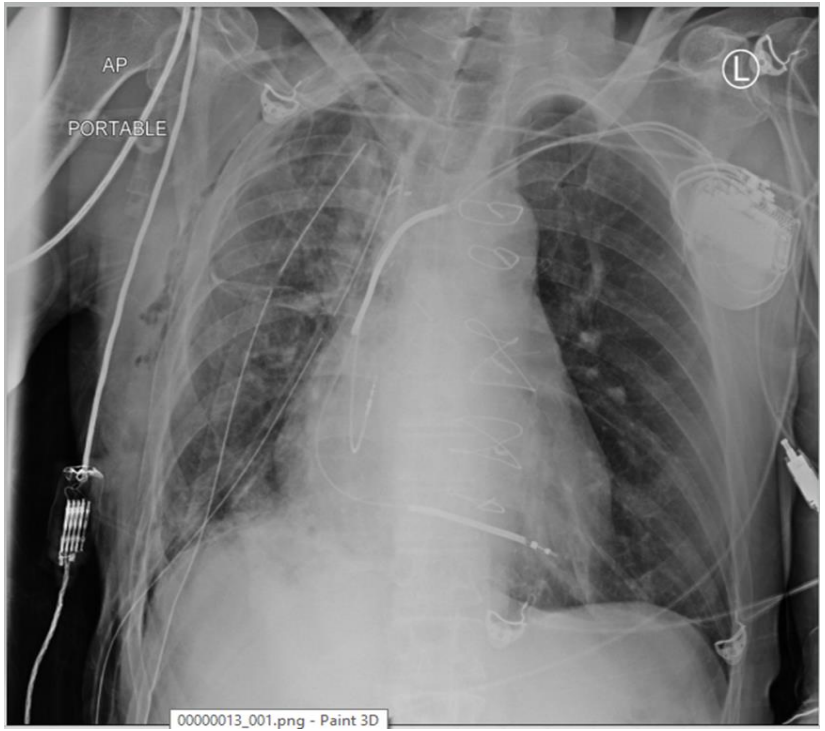
```
# Decode the data in the image
def decode():
    img = input("Enter image name(with extension) : ")
    image = Image.open(img, 'r')

    pwd = input("Enter Password : ")
    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):
            data = data.split("$")
            if(data[1] == pwd):
                return data[0]
```

Decoding Result

```
:: Welcome to Steganography ::
1. Encode
2. Decode
3. Exit
2
Enter image name(with extension) : /content/IMGss1.png
Enter Password : 123456789
Decoded Word : But I must explain to you how all this mistaken
```



Original Image



Stego Image

U-Net CNN

```
from tensorflow.keras import layers


def get_model(img_size, num_classes):
    inputs = keras.Input(shape=img_size + (3,)) #3 no of channels

    ### [First half of the network: downsampling inputs] ###

    # Entry block
    x = layers.Conv2D(32, 3, strides=2, padding="same")(inputs)  #32 is feature dimension
    x = layers.BatchNormalization()(x) #for fast learning
    x = layers.Activation("relu")(x)

    previous_block_activation = x  # Set aside residual

    # Blocks 1, 2, 3 are identical apart from the feature depth.
    for filters in [64, 128, 256]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # Project residual
        residual = layers.Conv2D(filters, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual
```

Continuing U-Net CNN

```
    ### [Second half of the network: upsampling inputs] ###

    for filters in [256, 128, 64, 32]:
        x = layers.Activation("relu")(x)
        x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.UpSampling2D(2)(x)

        # Project residual
        residual = layers.UpSampling2D(2)(previous_block_activation)
        residual = layers.Conv2D(filters, 1, padding="same")(residual)
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    # Add a per-pixel classification layer
    outputs = layers.Conv2D(num_classes, 3, activation="softmax", padding="same")(x)

    # Define the model
    model = keras.Model(inputs, outputs)
    return model


# Free up RAM in case the model definition cells were run multiple times
keras.backend.clear_session()

# Build model
model = get_model(img_size, num_classes)
model.summary()
```

U-Net CNN Result

```
# Configure the model for training.
# We use the "sparse" version of categorical_crossentropy
# because our target data is integers.
model.compile(optimizer="rmsprop", loss="CategoricalCrossentropy")

epochs = 5
model.fit(train_gen, epochs=epochs, validation_data=val_gen)
```

```
Epoch 1/5
18/18 [==============================] - 32s 1s/step - loss: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 2/5
18/18 [==============================] - 23s 1s/step - loss: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 3/5
18/18 [==============================] - 23s 1s/step - loss: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 4/5
18/18 [==============================] - 23s 1s/step - loss: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 5/5
18/18 [==============================] - 23s 1s/step - loss: 0.0000e+00 - val_loss: 0.0000e+00

<keras.callbacks.History at 0x7fba11eb0b50>
```

Checking PSNR, RMSE, MSE, Similarity of cosine and Dissimilarity of cosine

Number of msg Characters, PSNR, RMSE, MSE

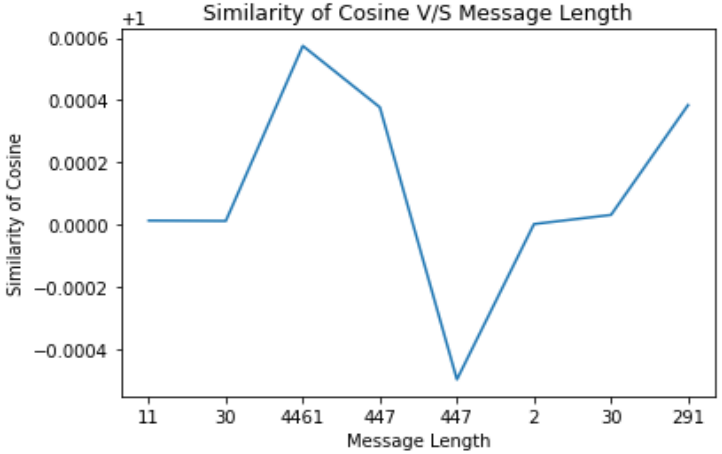
```
Original Image File Name: IMG86.PNG
Stego Image File Name: IMG86s.PNG
Message: Lorem ipsum dolor sit amet, consectetur adipisicing elit,

Password: 123456
Number of Characters in message:  447
similarity of cosine:  1.0004380440119114
dissimilarity of cosine:  -0.0004380440119113871
MSE:  0.0016367245555127428
PSNR:  75.95691835126706
RMSE:  0.00023175782697626882
```

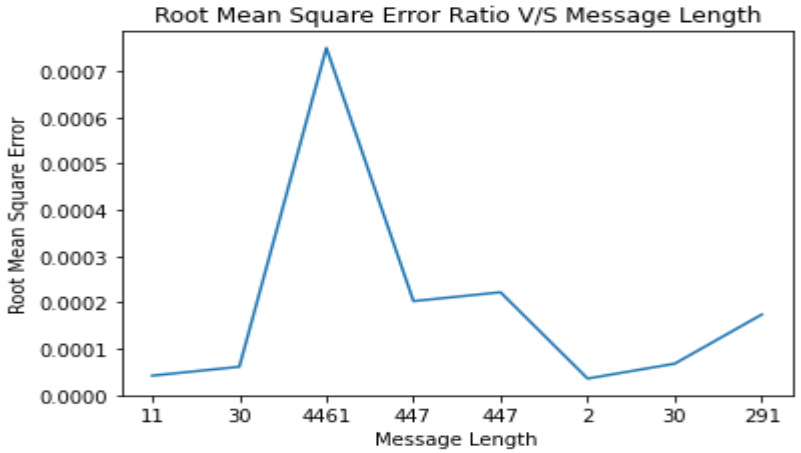
```
Original Image File Name: IMG16.PNG
Stego Image File Name: IMG16s.PNG
Message: steganography is cool and funsteganography is cool and funsteganog

Password: thomas
Number of Characters in message:  316
similarity of cosine:  0.9996073304526766
dissimilarity of cosine:  0.00039266954732342096
MSE:  0.0008934958380300009
PSNR:  78.619878269975
RMSE:  0.0001833242064347572
```

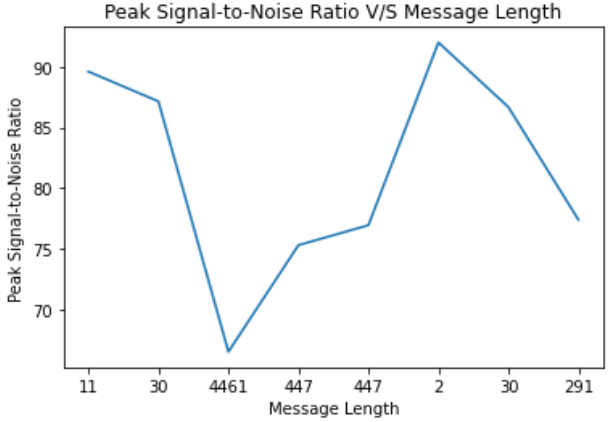
```
Original Image File Name: IMG37.PNG
Stego Image File Name: IMG37s.PNG
Message: steganography is cool and fun

Password: password
Number of Characters in message:  30
similarity of cosine:  0.9999532766640145
dissimilarity of cosine:  4.672333598554346e-05
MSE:  0.00012601313010430487
PSNR:  87.12664561506259
RMSE:  6.609028190260555e-05
```

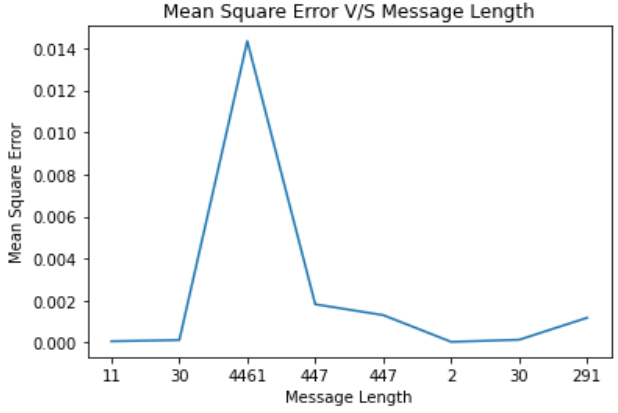
```
Original Image File Name: IMG93.PNG
Stego Image File Name: IMG93s.PNG
Message: Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed

Password: thomas
Number of Characters in message:  4470
similarity of cosine:  1.0021343380809138
dissimilarity of cosine:  -0.002134338080913789
MSE:  0.015255162888053253
PSNR:  66.12463168131626
RMSE:  0.0005918264696683497
```

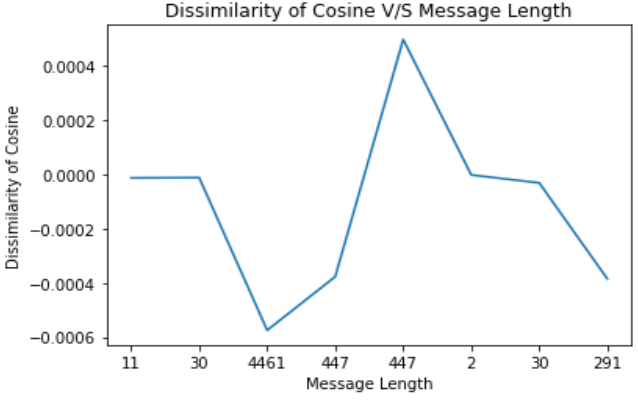
Graphical Representation of Validity Check


Similarity of Cosine V/S Message Length
+1
0.0006
0.0004
0.0002
0.0000
−0.0002
−0.0004
Similarity of Cosine
11
30
4461
447
447
2
30
291
Message Length


Root Mean Square Error Ratio V/S Message Length
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
Root Mean Square Error
11
30
4461
447
447
2
30
291
Message Length


Peak Signal-to-Noise Ratio V/S Message Length
90
85
80
75
70
Peak Signal-to-Noise Ratio
11
30
4461
447
447
2
30
291
Message Length


Mean Square Error V/S Message Length
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Mean Square Error
11
30
4461
447
447
2
30
291
Message Length


Dissimilarity of Cosine V/S Message Length
0.0004
0.0002
0.0000
−0.0002
−0.0004
−0.0006
Dissimilarity of Cosine
11
30
4461
447
447
2
30
291
Message Length

Thank You

Any Questions?