# **Steganography**

Steganography is a encryption technique for message, image, audio, video and so on.It hides multimedia information in another message, image, audio and video. Least Significant Bit (LSB) is a easy way to hide a image in the other image. We take a high bits part of image, and then put it in low bits parts of other image.

**First task - LSB:**

coverImg = cv2.imread("image/datasets\_200511\_442144\_data\_forest\_bost101.jpg",cv2.IMREAD\_COLOR)

coverImg.astype("uint8")

secretImg = cv2.imread("image/datasets\_200511\_442144\_data\_forest\_cdmc280.jpg",cv2.IMREAD\_COLOR)

secretImg.astype("uint8")

encodedImg = coverImg

width,height,channel = coverImg.shape

name1 = "CoverImg -> SecretImg"

cv2.namedWindow(name1)

cv2.imshow(name1,cv2.hconcat((coverImg,secretImg)))

encodedImg = ((coverImg >> 4) << 4) + (secretImg >> 4)

name2 = "CoverImg -> EncodedImg -> SubtractImg"

cv2.namedWindow(name2)

cv2.imshow(name2,cv2.hconcat((coverImg,encodedImg,coverImg-encodedImg)))

decode\_coverImg = (encodedImg >> 4) << 4

decode\_secretImg = (encodedImg << 4)

name3 = "Decode\_CoverImg -> Decode\_SecretImg"

cv2.namedWindow(name3)

cv2.imshow(name3,cv2.hconcat((decode\_coverImg,decode\_secretImg)))

cv2.waitKey(0)

cv2.destroyAllWindows()



**Second task - Neural Network:**

LSB is a simple way to hide a image. Neural Network (NN) is a other solution for hide a image. In this project, I refer to

<https://github.com/alexandremuzio/deep-steg/blob/master/deep_steg.ipynb?fbclid=IwAR13y18zLbA2vrOmchy927JdEYF4FRsAMw92U-FFHyEkdDkdla3hZbYqAjA>

his method and change some CNN architecture. First, I use 2d-dct to filter the high frequency data:

**for** i **in** range(0,250):

**for** j **in** range(0,3):

secretImg[i,:,:,j] = cv2.dct(np.float32(secretImg[i,:,:,j]))

coverImg[i,:,:,j] = cv2.dct(np.float32(coverImg[i,:,:,j]))

**for** row **in** range(0,len(secretImg[i,:,0,j])):

**for** col **in** range(0,len(secretImg[i,0,:,j])):

**if** row >= len(secretImg[i,:,0,j]) **and** col >= len(secretImg[i,0,:,j]):

secretImg[i, row, col, j] = 0

coverImg[i, row, col, j] = 0

secretImg[i, :, :, j] = cv2.idct(secretImg[i,:,:,j])

coverImg[i, :, :, j] = cv2.idct(coverImg[i, :, :, j])

By using 2d-dct transform to filter the high frequency, we can decrease our cnn layer and make sure the NN would focus on the low frequency part.

And for the encoder part, I use more layer for the first prep-layer, and use large filter size to make sure we have more dimension data for input image.

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep0\_3x3'**)(input\_S)

x2 = Conv2D(25, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep0\_4x4'**)(input\_S)

x3 = Conv2D(15, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep0\_5x5'**)(input\_S)

x4 = Conv2D(10, (6, 6), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep0\_6x6'**)(input\_S)

x5 = Conv2D(5, (7, 7), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep0\_7x7'**)(input\_S)

x = concatenate([x1,x2,x3, x4, x5])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep1\_3x3'**)(x)

x2 = Conv2D(10, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep1\_4x4'**)(x)

x3 = Conv2D(5, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep1\_5x5'**)(x)

x = concatenate([x1, x2, x3])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep2\_3x3'**)(x)

x2 = Conv2D(10, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep2\_4x4'**)(x)

x3 = Conv2D(5, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep2\_5x5'**)(x)

x = concatenate([x1, x2, x3])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep3\_3x3'**)(input\_C)

x2 = Conv2D(25, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep3\_4x4'**)(input\_C)

x3 = Conv2D(15, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep3\_5x5'**)(input\_C)

x4 = Conv2D(10, (6, 6), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep3\_6x6'**)(input\_C)

x5 = Conv2D(5, (7, 7), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_prep3\_7x7'**)(input\_C)

x = concatenate([x,x1,x2,x3,x4,x5])

*# Hiding network*

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid0\_3x3'**)(x)

x2 = Conv2D(25, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid0\_4x4'**)(x)

x3 = Conv2D(15, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid0\_5x5'**)(x)

x4 = Conv2D(10, (6, 6), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid0\_6x6'**)(x)

x5 = Conv2D(5, (7, 7), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid0\_7x7'**)(x)

x = concatenate([x1,x2,x3, x4, x5])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid1\_3x3'**)(x)

x2 = Conv2D(10, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid1\_4x4'**)(x)

x3 = Conv2D(5, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid1\_5x5'**)(x)

x = concatenate([x1, x2, x3])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid2\_3x3'**)(x)

x2 = Conv2D(25, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid2\_4x4'**)(x)

x3 = Conv2D(15, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid2\_5x5'**)(x)

x4 = Conv2D(10, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid2\_6x6'**)(x)

x5 = Conv2D(5, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid2\_7x7'**)(x)

x = concatenate([x1,x2, x3, x4, x5])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid3\_3x3'**)(x)

x2 = Conv2D(10, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid3\_4x4'**)(x)

x3 = Conv2D(5, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_hid3\_5x5'**)(x)

x = concatenate([x1, x2, x3])

Similar with encoder, decoder part use 5 filter size to make sure we have enough keypoint data.

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev0\_3x3'**)(input\_with\_noise)

x2 = Conv2D(25, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev0\_4x4'**)(input\_with\_noise)

x3 = Conv2D(15, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev0\_5x5'**)(input\_with\_noise)

x4 = Conv2D(10, (6, 6), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev0\_6x6'**)(input\_with\_noise)

x5 = Conv2D(5, (7, 7), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev0\_7x7'**)(input\_with\_noise)

x = concatenate([x1,x2,x3, x4, x5])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev1\_3x3'**)(x)

x2 = Conv2D(10, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev1\_4x4'**)(x)

x3 = Conv2D(5, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev1\_5x5'**)(x)

x = concatenate([x1, x2, x3])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev2\_3x3'**)(x)

x2 = Conv2D(10, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev2\_4x4'**)(x)

x3 = Conv2D(5, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev2\_5x5'**)(x)

x = concatenate([x1, x2, x3])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev3\_3x3'**)(x)

x2 = Conv2D(10, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev3\_4x4'**)(x)

x3 = Conv2D(5, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev3\_5x5'**)(x)

x = concatenate([x1, x2, x3])

x1 = Conv2D(50, (3, 3), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev4\_3x3'**)(x)

x2 = Conv2D(10, (4, 4), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev4\_4x4'**)(x)

x3 = Conv2D(5, (5, 5), strides=(1, 1), padding=**'same'**, activation=**'relu'**, name=**'conv\_rev5\_5x5'**)(x)

x = concatenate([x1, x2, x3])

And finally we train our model to find a good parameter for every layer.

BATCH\_SIZE = 32

m = secretImg.shape[0]

loss\_history = []

**for** epoch **in** range(0,50):

np.random.shuffle(secretImg)

np.random.shuffle(coverImg)

t = tqdm(range(0, secretImg.shape[0], BATCH\_SIZE), mininterval=0)

ae\_loss = []

revLoss = []

**for** idx **in** t:

sbatch = secretImg[idx:min(idx + BATCH\_SIZE, m)]

cbatch = coverImg[idx:min(idx + BATCH\_SIZE, m)]

C\_prime = encoder\_model.predict([batch\_S, batch\_C])

ae\_loss.append(autoencoder\_model.train\_on\_batch(x=[sbatch, cbatch], y=np.concatenate((sbatch, cbatch), axis=3)))

revLoss.append(reveal\_model.train\_on\_batch(x=C\_prime,y=sbatch))

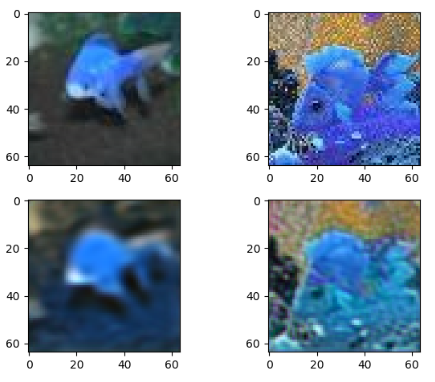
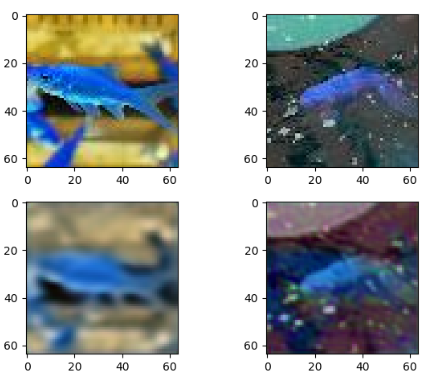
K.set\_value(autoencoder\_model.optimizer.lr, 0.001)

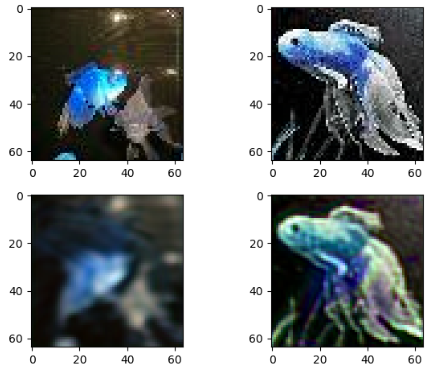
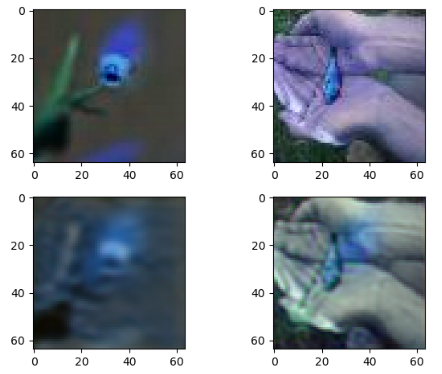
K.set\_value(reveal\_model.optimizer.lr, 0.001)

t.set\_description(**'Epoch {} | Batch: {:3} of {}. Loss AE {:10.2f} | Loss Rev {:10.2f}'**.format(epoch + 1, idx, m,np.mean(ae\_loss), np.mean(revLoss)))

loss\_history.append(np.mean(ae\_loss))

Because of timing limit, I just use 50 iteration to check my NN architecture could get a good result.





Above image is :

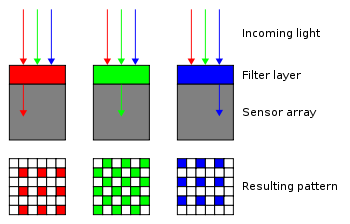
image(1,1) - secertImage , image(1,2) - coverImage

image(2,1) - decodeSecretImage, image(2,2) - decodeCoverImage

It can observed that although we just use 50 iteration for training our model, we can get a certain degree results for our output image. If we train our model more times, we can get better result for our output image.

**Optional Task - Hide two secret image - LSB:**

If we want to use 1 cover image to hide 2 secret image by LSB methods, we need to decrease secret image data first, and then put it in the cover image. Bayer is a color filter array for RGB image. It can tranfrom 3 channel RGB image into 1 channel bayer image.



Therefore, we decrease our secret image data first:

**for** row **in** range(0,height):

count = 1

**for** col **in** range(0,width):

**if** row%2 == 0: *## RGRGRGRG*

**if** count == 1:

secretImg1\_Bayer[row, col] = secretImg1[row, col, 0]

secretImg2\_Bayer[row, col] = secretImg2[row, col, 0]

count = count + 1

**elif** count == 2:

secretImg1\_Bayer[row, col] = secretImg1[row, col, 1]

secretImg2\_Bayer[row, col] = secretImg2[row, col, 1]

count = 1

**elif** row%2 == 1: *## GBGBGBGBGB*

**if** count == 1:

secretImg1\_Bayer[row, col] = secretImg1[row, col, 1]

secretImg2\_Bayer[row, col] = secretImg2[row, col, 1]

count = count + 1

**elif** count == 2:

secretImg1\_Bayer[row, col] = secretImg1[row, col, 2]

secretImg2\_Bayer[row, col] = secretImg2[row, col, 2]

count = 1

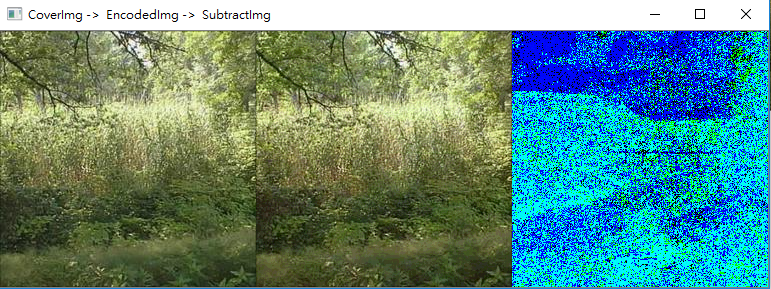
And then put it in the lower bits of cover image.

coverImg[:,:,0] = ((np.uint8(coverImg[:,:,0]) >> 4) << 4) + (np.uint8(secretImg1\_Bayer) >> 4)

coverImg[:,:,1] = ((np.uint8(coverImg[:,:,1]) >> 4) << 4) + (np.uint8(secretImg2\_Bayer) >> 4)

The following is original image and encoded image:

****



Finally, if we want to decode the encoded image to get the secret image, we need to do the demorsaic for the bayer image.There are lots of method for debayer a image. And we just use pure methods to debayer our image, just use the neighbors of current pixel to get the whole image.

secretImg1\_Bayer = np.uint8(coverImg[:,:,0]) << 4

secretImg2\_Bayer = np.uint8(coverImg[:,:,1]) << 4

**for** row **in** range(0,height-1):

count = 1

**for** col **in** range(0,width-1):

**if** row%2 == 0: *## RGRGRGRG*

**if** count == 1:

secretImg1\_deBayer[row, col, 0] = secretImg1\_Bayer[row , col ]

secretImg1\_deBayer[row, col, 1] = secretImg1\_Bayer[row , col + 1]

secretImg1\_deBayer[row, col, 2] = secretImg1\_Bayer[row + 1, col + 1]

secretImg2\_deBayer[row, col, 0] = secretImg2\_Bayer[row , col ]

secretImg2\_deBayer[row, col, 1] = secretImg2\_Bayer[row , col + 1]

secretImg2\_deBayer[row, col, 2] = secretImg2\_Bayer[row + 1, col + 1]

count = count + 1

**elif** count == 2:

secretImg1\_deBayer[row, col, 0] = secretImg1\_Bayer[row , col - 1]

secretImg1\_deBayer[row, col, 1] = secretImg1\_Bayer[row , col ]

secretImg1\_deBayer[row, col, 2] = secretImg1\_Bayer[row + 1, col ]

secretImg2\_deBayer[row, col, 0] = secretImg2\_Bayer[row , col - 1]

secretImg2\_deBayer[row, col, 1] = secretImg2\_Bayer[row , col ]

secretImg2\_deBayer[row, col, 2] = secretImg2\_Bayer[row + 1, col ]

count = 1

**elif** row%2 == 1: *## GBGBGBGBGB*

**if** count == 1:

secretImg1\_deBayer[row, col, 0] = secretImg1\_Bayer[row - 1 ,col ]

secretImg1\_deBayer[row, col, 1] = secretImg1\_Bayer[row , col ]

secretImg1\_deBayer[row, col, 2] = secretImg1\_Bayer[row , col + 1]

secretImg2\_deBayer[row, col, 0] = secretImg2\_Bayer[row - 1, col ]

secretImg2\_deBayer[row, col, 1] = secretImg2\_Bayer[row , col ]

secretImg2\_deBayer[row, col, 2] = secretImg2\_Bayer[row , col + 1]

count = count + 1

**elif** count == 2:

secretImg1\_deBayer[row, col, 0] = secretImg1\_Bayer[row - 1 ,col - 1]

secretImg1\_deBayer[row, col, 1] = secretImg1\_Bayer[row , col - 1]

secretImg1\_deBayer[row, col, 2] = secretImg1\_Bayer[row , col ]

secretImg2\_deBayer[row, col, 0] = secretImg2\_Bayer[row - 1, col - 1]

secretImg2\_deBayer[row, col, 1] = secretImg2\_Bayer[row , col - 1]

secretImg2\_deBayer[row, col, 2] = secretImg2\_Bayer[row , col ]

count = 1

name3 = **"Decode\_CoverImg -> Decode\_SecretImg1 -> Decode\_SecretImg2"**

decode\_coverImg = coverImg.copy()

decode\_coverImg[:,:,0] = np.uint8(decode\_coverImg[:,:,0] >> 4) << 4

decode\_coverImg[:,:,1] = np.uint8(decode\_coverImg[:,:,1] >> 4) << 4



Platform:pycharm

Tool : tensorflow-1.5, keras, opencv-python, glob, matplotlib