浙江大学

课程报告

课程名称：生物智能算法

学　　院：计算机科学与技术学院

专　　业：计算机科学与技术

学生姓名：康至煊

学　　号：21821222

邮　　箱：kzx1995@126.com

任课教师：袁昕

浙 江 大 学
课 程 报 告

一、论文选题：**Hiding Images in Plain Sight:Deep Steganography**

1. 论文出处

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA

2.Abstract

Steganography is the practice of concealing a secret message within another,ordinary, message. Commonly, steganography is used to unobtrusively hide a smallmessage within the noisy regions of a larger image. In this study, we attemptto place a full size color image within another image of the same size. Deepneural 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 naturalimages from a wide variety of sources. Beyond demonstrating the successfulapplication of deep learning to hiding images, we carefully examine how the resultis achieved and explore extensions. Unlike many popular steganographic methodsthat 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.

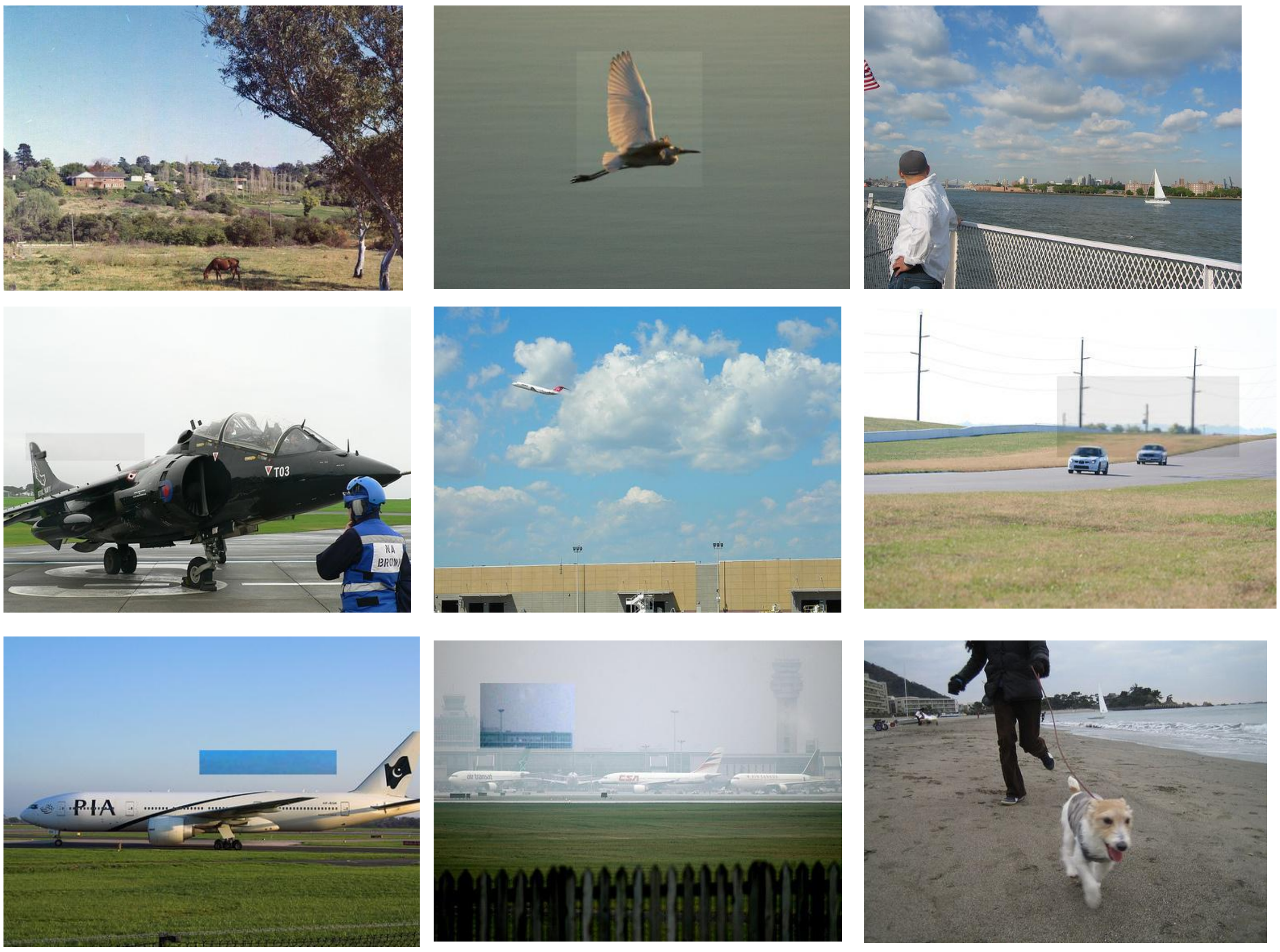
3.摘要

隐写术是一种在普通的信息中隐藏秘密信息的方法。通常，隐写术被用来在较大图像的噪声区域内隐藏一个小消息。在本研究中，我们尝试将一个全尺寸的彩色图像放置在另一个大小相同的图像中。深层神经网络同时被训练来创建隐藏和揭示过程，并被专门设计成一对。该系统对随机从 ImageNet 数据库中提取的图像进行培训，并能很好地处理来自各种来源的自然图像。除了演示如何成功地将深度学习应用于隐藏图像之外，我们还仔细研究了结果是如何实现的，并探索了扩展。与许多常用的隐写方法不同，常用方法将秘密消息编码在载波图像中最不重要的比特中，而我们的方法将需要加密的图像压缩和分发在所有可获得的 bit 中。

二、论文内容：

1. 数据集：

在该篇论文中，实验所用数据集为 Tiny ImageNet Visual Recognition Challenge
部分数据展示如下：

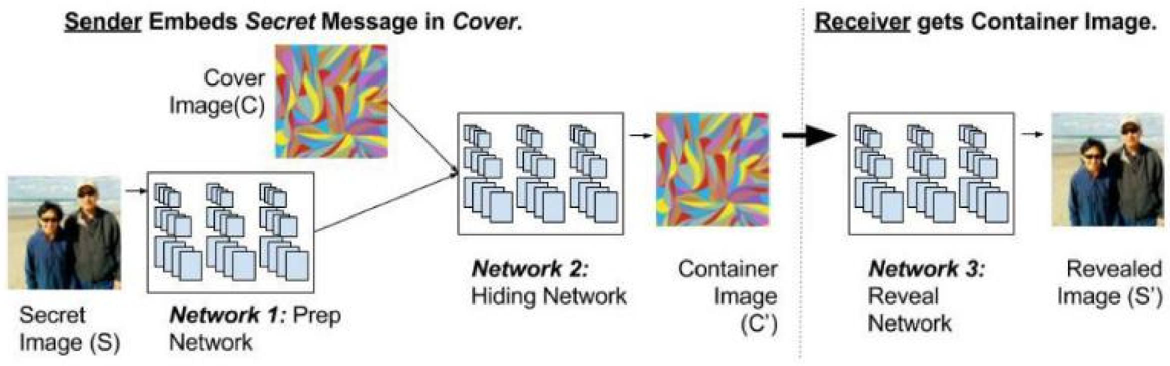


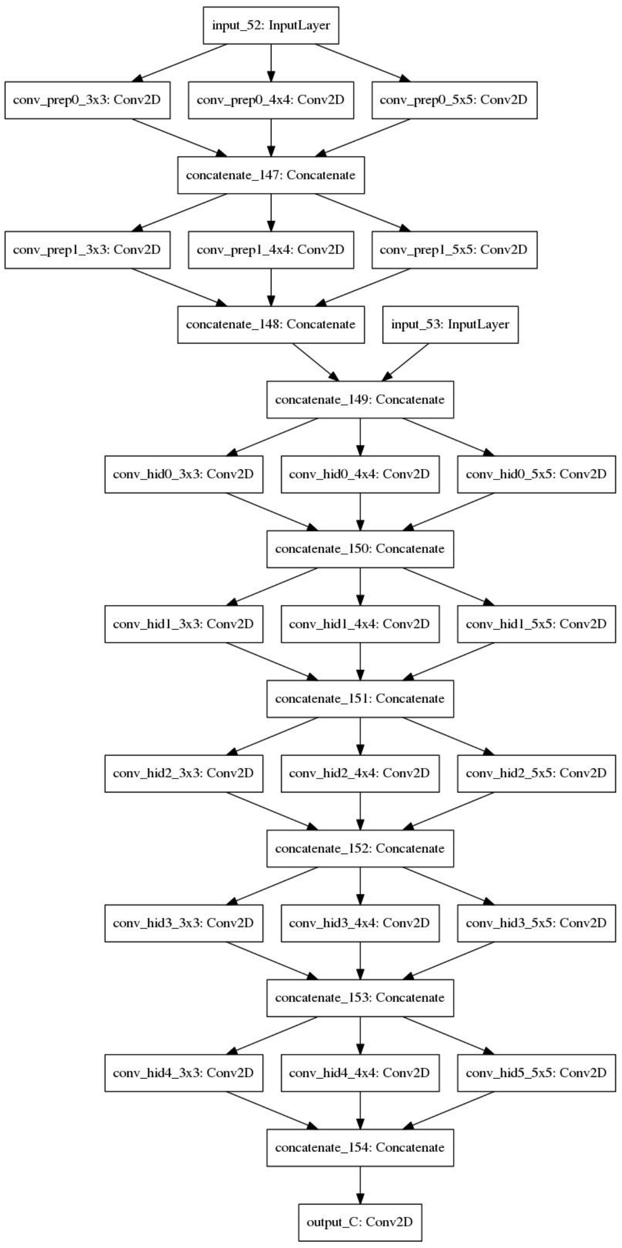
2. 论文内容简述

该模型由准备网络、隐藏网络(编码器)和显示网络三部分组成。它的目标是将秘密图像 S 进行编码并隐藏到图像 C 中，生成与 C 非常相似的 C’，同时保证能够从 C’进行解码，生成解码后的秘密图像 S’，该秘密图像 S’应该尽可能地与原秘密图像 S 相似。

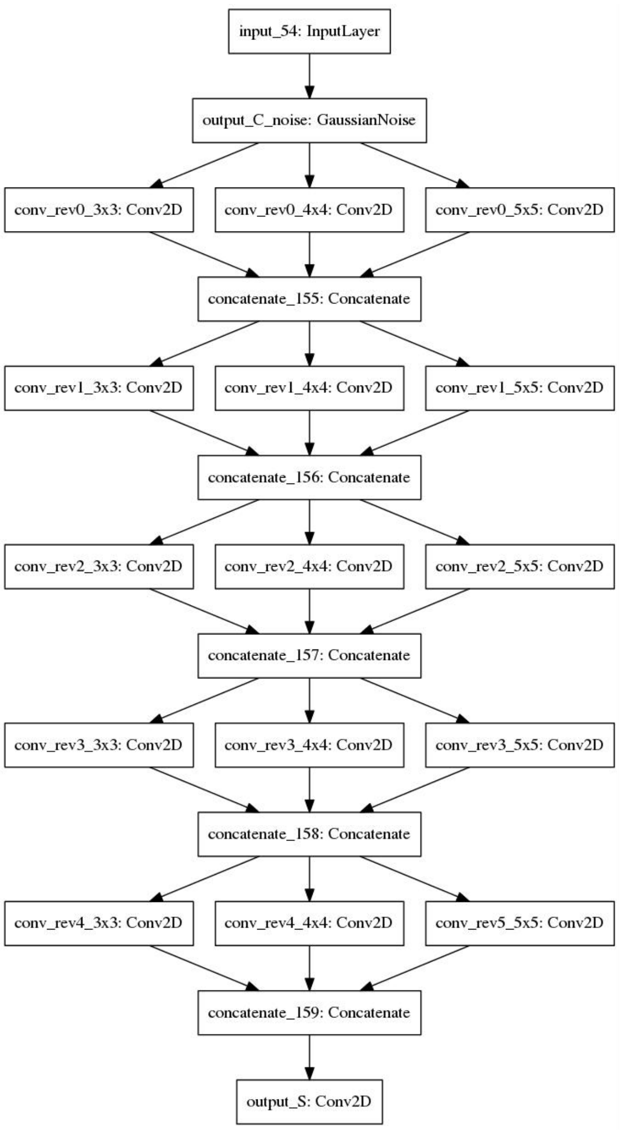
准备网络负责从秘密图像中提取信息，这些信息将与覆盖图像连接并反馈给隐藏网络。隐藏网络对输入进行编码生成覆盖图像 C’。最后，解密网络 C’进行解码得到秘密图像 S’。为了保证稳定性，我们在揭示网络之前加入了噪声。虽然本文作者并没有详细说明这三个网络的体系结构，但是通过实验可发现聚合层具有很好的效果。对于隐藏和显示网络，作者使用了 5 层 65 个过滤器(50 个 3x3 过滤器，10 个 4x4 过滤器和 5 个 5x5 过滤器)。而准备网络，只使用了两层相同的结构。

3. 模型架构展示


Sender Embeds Secret Message in Cover.
Receiver gets Container Image.
Cover Image(C)
Secret Image (S)
Network 1: Prep Network
Network 2: Hiding Network
Container Image (C')
Network 3: Reveal Network
Revealed Image (S')



encoder_model



decoder_model

三、复现实验

1.创建数据集

```
def load_dataset_small(num_images_per_class_train=10, num_images_test=500):
    #Loads training and test datasets, from Tiny ImageNet Visual Recogition Challenge.

    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)
```

2.构建模型

```
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
```

3.编码层

```
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])

    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')
```

4. 解码层

```
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_no
    x4 = Conv2D(10, (4, 4), strides = (1, 1), padding='same', activation='relu', name='conv_rev0_4x4')(input_with_no
    x5 = Conv2D(5, (5, 5), strides = (1, 1), padding='same', activation='relu', name='conv_rev0_5x5')(input_with_noi
    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)
```

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

5.训练

```
NB_EPOCHS = 1000
BATCH_SIZE = 32

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 = []
    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])

        ae_loss.append(autoencoder_model.train_on_batch(x=[batch_S, batch_C],
                                               y=np.concatenate((batch_S, batch_C),axis=3)))
        rev_loss.append(reveal_model.train_on_batch(x=C_prime,
                                              y=batch_S))

        # Update learning rate
        K.set_value(autoencoder_model.optimizer.lr, lr_schedule(epoch))
        K.set_value(reveal_model.optimizer.lr, lr_schedule(epoch))

        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(rev_loss)))
    loss_history.append(np.mean(ae_loss))
```

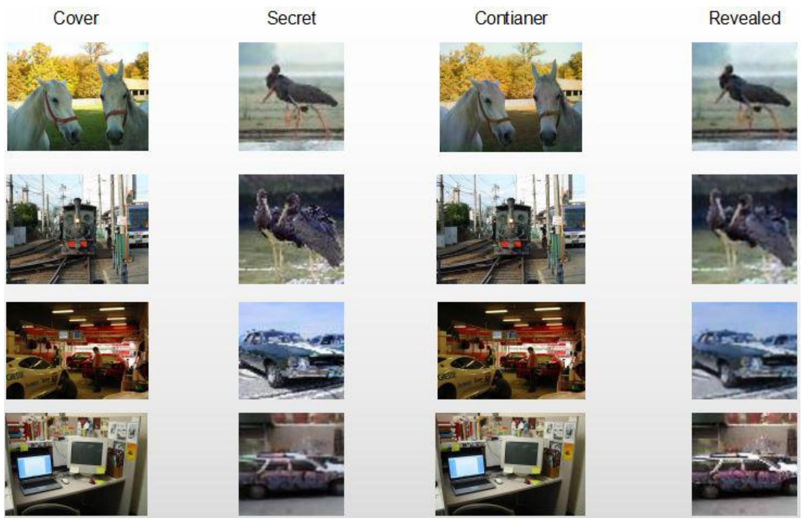
6. 保存模型

```
autoencoder_model.save_weights('model.hdf5')
```

完整代码见 github 项目。

需要注意的是，用于揭示网络的损失函数不同于用于准备和隐藏网络的损失函数。为了正确实现网络权重的更新，创建了堆叠的 Keras 模型，一个用于准备和隐藏网络(共享相同的丢失功能)，一个用于显示网络。为了确保权重只更新一次，冻结了 Reveal 网络层上的权重，然后将其添加到完整模型中。

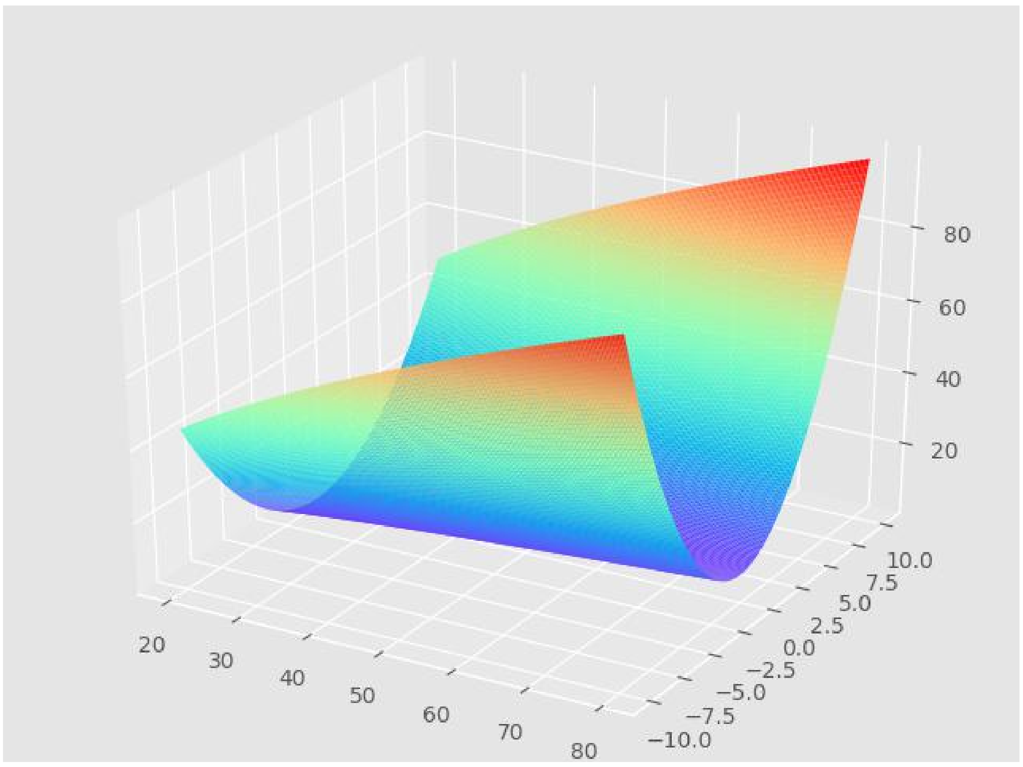
7. 复现结果



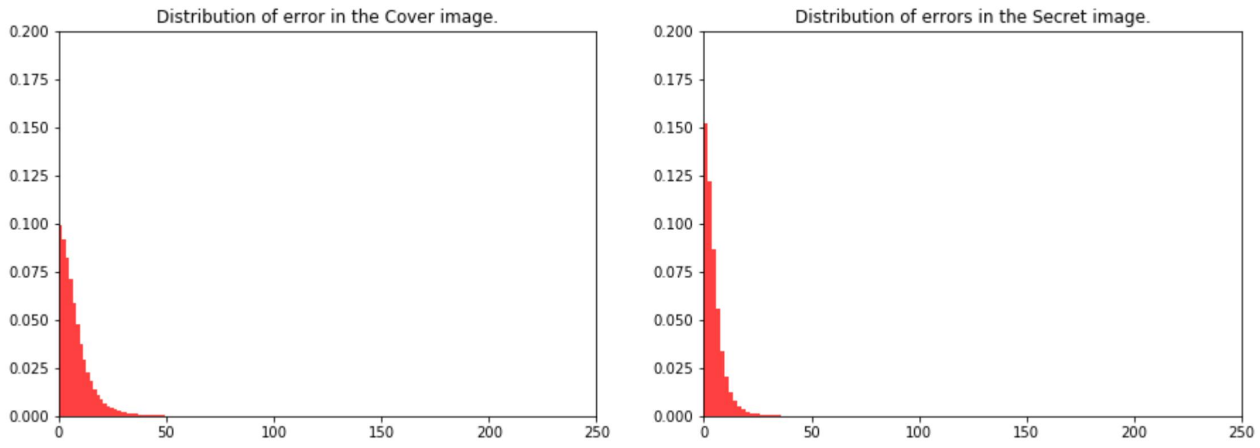
由结果可看出，做为载体的图片在隐藏进信息后色彩会有些许变化，而秘密图片在从载体图片中解析出来后，较原图有少许失真。

四、模型仿真

1. loss 下降图



2. 误差图


Distribution of error in the Cover image.
Distribution of errors in the Secret image.