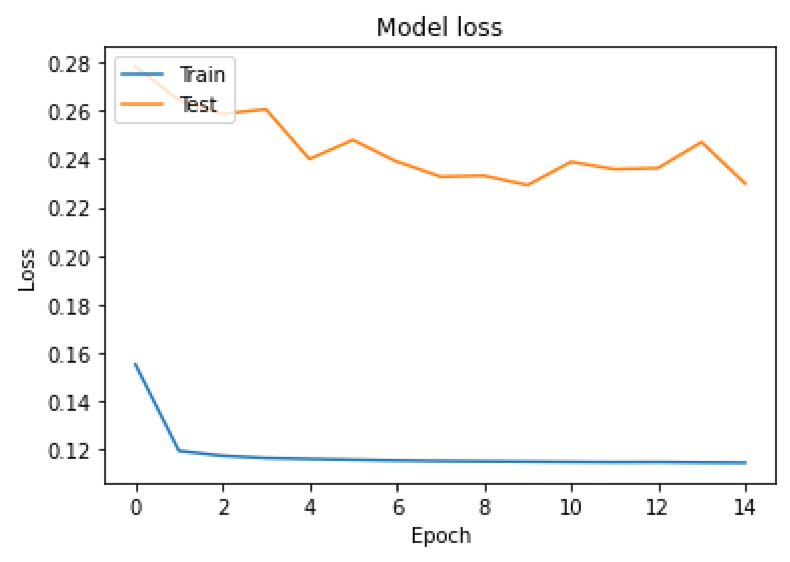
```
import os
from google.colab import drive
drive.mount('/content/drive')

path = "/content/drive/My Drive/暑期科研/"

os.chdir(path)
os.listdir(path)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&resp

Enter your authorization code:
··········
Mounted at /content/drive
['chromosome_l_x_train.npy',
 'chromosome_l_x_test.npy',
 'chromosome_l_y_test.npy',
 'chromosome_l_y_train.npy',
 'chromosome_r_y_test.npy',
 'chromosome_r_y_train.npy',
 'chromosome_r_x_test.npy',
 'chromosome_r_x_train.npy']
```

```
import glob

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


from keras.preprocessing import image
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras.layers import Input, Dense, Activation, BatchNormalization, Flatten, Conv2D
from keras.layers import MaxPooling2D, Dropout, UpSampling2D
import os

x_train_savepath = './chromosome_r_x_train.npy'
y_train_savepath = './chromosome_r_y_train.npy'

x_test_savepath = './chromosome_r_x_test.npy'
y_test_savepath = './chromosome_r_y_test.npy'
print('-------------Load Datasets-----------------')
x_train_save = np.load(x_train_savepath)
y_train = np.load(y_train_savepath)
x_test_save = np.load(x_test_savepath)
y_test = np.load(y_test_savepath)
x_train = np.reshape(x_train_save, (len(x_train_save), 150, 150,1))
x_test = np.reshape(x_test_save, (len(x_test_save), 150, 150,1))

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
# x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
print(x_train.shape)
print(x_test.shape)
#

class Autoencoder():
        def __init__(self):

                self.img_shape = (150,150,1)

                optimizer = Adam(lr=0.001)

                self.autoencoder_model = self.build_model()
                self.autoencoder_model.compile(loss='binary_crossentropy', optimizer=optimizer)
                self.autoencoder_model.summary()

        def build_model(self):
                input_layer = Input(shape=self.img_shape)

                # encoder
                h = Conv2D(64, (3, 3), activation='relu', padding='same')(input_layer)
                h = MaxPooling2D((3, 3), padding='same')(h)
                h = Conv2D(64, (5, 5), activation='relu', padding='same')(h)
                h = MaxPooling2D((5, 5), padding='same')(h)

                # decoder
                h = Conv2D(64, (5, 5), activation='relu', padding='same')(h)
                h = UpSampling2D((5, 5))(h)
                h = Conv2D(64, (3, 3), activation='relu', padding='same')(h)
                h = UpSampling2D((3, 3))(h)

                output_layer = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(h)

                return Model(input_layer, output_layer)

        def train_model(self, x_train, y_train, x_test, y_test, epochs, batch_size):
                early_stopping = EarlyStopping(monitor='val_loss',
                                                              min_delta=0,
                                                              patience=5,
                                                              verbose=1,
                                                              mode='auto')
                history = self.autoencoder_model.fit(x_train, x_train,
                                                                          batch_size=batch_size,
                                                                          epochs=epochs,
                                                                          validation_data=(x_test, x_test),
                                                                          callbacks=[early_stopping])
                plt.plot(history.history['loss'])
                plt.plot(history.history['val_loss'])
                plt.title('Model loss')
                plt.ylabel('Loss')
                plt.xlabel('Epoch')
                plt.legend(['Train', 'Test'], loc='upper left')
                plt.show()

        def eval_model(self, x_test):
                preds = self.autoencoder_model.predict(x_test)
                return preds

ae = Autoencoder()
ae = Autoencoder()
ae.train_model(x_train, y_train, x_test, y_test, epochs=30, batch_size=4)
```

```
-------------Load Datasets-----------------
(988, 150, 150, 1)
(188, 150, 150, 1)
Model: "model_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 150, 150, 1)       0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 150, 150, 64)      640
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 50, 50, 64)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 50, 50, 64)        102464
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 10, 10, 64)        102464
_________________________________________________________________
up_sampling2d_8 (UpSampling2 (None, 50, 50, 64)        0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 50, 50, 64)        36928
_________________________________________________________________
up_sampling2d_9 (UpSampling2 (None, 150, 150, 64)      0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 150, 150, 1)       577
=================================================================
Total params: 243,073
Trainable params: 243,073
Non-trainable params: 0
_________________________________________________________________
Train on 988 samples, validate on 188 samples
Epoch 1/30
988/988 [==============================] - 6s 7ms/step - loss: 0.1553 - val_loss: 0.2782
Epoch 2/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1195 - val_loss: 0.2643
Epoch 3/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1175 - val_loss: 0.2586
Epoch 4/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1166 - val_loss: 0.2606
Epoch 5/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1162 - val_loss: 0.2400
Epoch 6/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1159 - val_loss: 0.2479
Epoch 7/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1156 - val_loss: 0.2389
Epoch 8/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1154 - val_loss: 0.2327
Epoch 9/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1152 - val_loss: 0.2331
Epoch 10/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1151 - val_loss: 0.2292
Epoch 11/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1150 - val_loss: 0.2388
Epoch 12/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1149 - val_loss: 0.2358
Epoch 13/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1149 - val_loss: 0.2363
Epoch 14/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1148 - val_loss: 0.2470
Epoch 15/30
988/988 [==============================] - 6s 6ms/step - loss: 0.1146 - val_loss: 0.2298
Epoch 00015: early stopping
```



```
decoded_test = ae.eval_model(x_test)

import matplotlib.pyplot as plt
%matplotlib inline
shape = (150, 150)
fig, axes = plt.subplots(2,10,
                         figsize=(10, 2),
                         subplot_kw={
                             'xticks': [],
                             'yticks': []
                         },
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i in range(10):
    axes[0][i].imshow(np.reshape(x_test[i], shape),cmap='gray')
    axes[1][i].imshow(np.reshape(decoded_test[i], shape),cmap='gray')
plt.show()
```