人工智能实验：Topic 8——自编码器

3190102060　黄嘉欣

一、自编码器

自动编码器是一种数据的压缩算法，其中数据的压缩和解压缩函数是：① 数据相关的；② 有损的；③ 从样本中自动学习的。在大部分提到自动编码器的场合，压缩和解压缩的函数是通过神经网络实现的。

① 自动编码器是数据相关的（data-specific 或 data-dependent），这意味着自动编码器只能压缩那些与训练数据类似的数据。自编码器与一般的压缩算法，如 MPEG-2，MP3 等压缩算法不同，一般的通用算法只假设了数据是“图像”或“声音”，而没有指定是哪种图像或声音。比如，使用人脸训练出来的自动编码器在压缩别的图片，如树木时的性能很差，因为它学习到的特征是与人脸相关的。

② 自动编码器是有损的，意思是解压缩的输出与原来的输入相比是退化的，MP3，JPEG 等压缩算法也是如此。这与无损压缩算法不同。

③ 自动编码器是从数据样本中自动学习的，这意味着很容易对指定类的输入训练出一种特定的编码器，而不需要完成任何新工作。

搭建一个自动编码器需要完成以下三项工作：搭建编码器，搭建解码器，设定一个损失函数，用以衡量由于压缩而损失掉的信息。编码器和解码器一般都是参数化的方程，并关于损失函数可导，典型情况是使用神经网络。编码器和解码器的参数可以通过最小化损失函数而优化，如 SGD。完成以后，自动编码器可以实现数据去噪，也可以对图像进行分割、重建等。

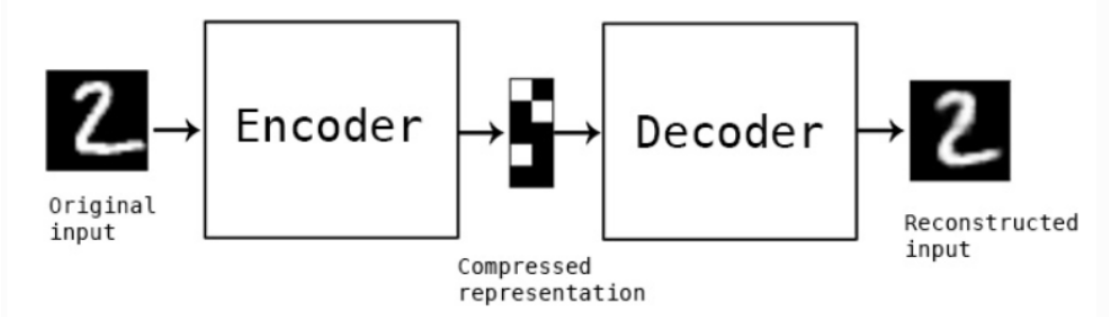


图 1.1 自动编码器

二、Keras 简介

Keras 是一个用 Python 编写的高级神经网络 API，它能够以 TensorFlow, CNTK 或

者 Theano 作为后端运行。Keras 的开发重点是支持快速的实验。能够以最小的时延把用户的想法转换为实验结果，是做好研究的关键。相对来说，Keras 允许简单而快速的原型设计（由于用户友好，高度模块化，可扩展性），同时支持卷积神经网络和循环神经网络及两者的组合，并能在 CPU 和 GPU 上无缝运行，故被广泛使用。

三、实验 8-1

① 实验题目

使用 Keras 工具包编写自动编码器进行图像去噪：把训练样本用噪声污染，利用 Keras 工具包编写一个卷积神经网络自编码器，使解码器解码出干净的照片，完成以下要求：

i) 搭建卷积神经网络自编码器，尽量取得低的相对误差；

ii) 固定 loss 为 MSE，比较至少 3 种不同优化器的训练测试结果；

iii) 固定优化器为 Adam，比较至少 3 种不同 loss 的训练测试结果；

iv) 找出相对误差小于 50%时能够获得的最大压缩比。

② 实验结果与分析

i) 搭建卷积神经网络自编码器，尽量取得低的相对误差：

```
# 编码器
x = Conv2D(32, 3, padding='valid', activation='relu', kernel_initializer='he_normal')(input_img)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(32, 3, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
encoded = MaxPooling2D(pool_size=(2, 2))(x)

# 解码器
x = Conv2DTranspose(32, 8, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(encoded)
x = Conv2D(32, 3, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
decoded = Conv2DTranspose(1, 2, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(x)
```

如上，搭建卷积神经网络自编码器，其输入维度为(28,28,1)，经过第一层卷积层后，输出维度为(26,26,32)；经过第二层池化层后，输出维度为(13,13,32)；经过第三层卷积层后，输出维度为(11,11,32)；经过第四层池化层后，输出维度为(5,5,32)。在解码器部分，经过第五层反卷积后，输出维度为(16,16,32)；经过第六层卷积层后，输出维度为(14,14,32)；经过第七层反卷积后，输出维度为(28,28,1)。

取训练集的前 10000 个数据进行训练，采用 Adam 优化器和 MSE 损失函数，初始学习率为10^{-4}，得到的损失值、相对误差（前 10 幅图片）和最终的数字图片分别如图 2.1.1-2.1.3 所示：

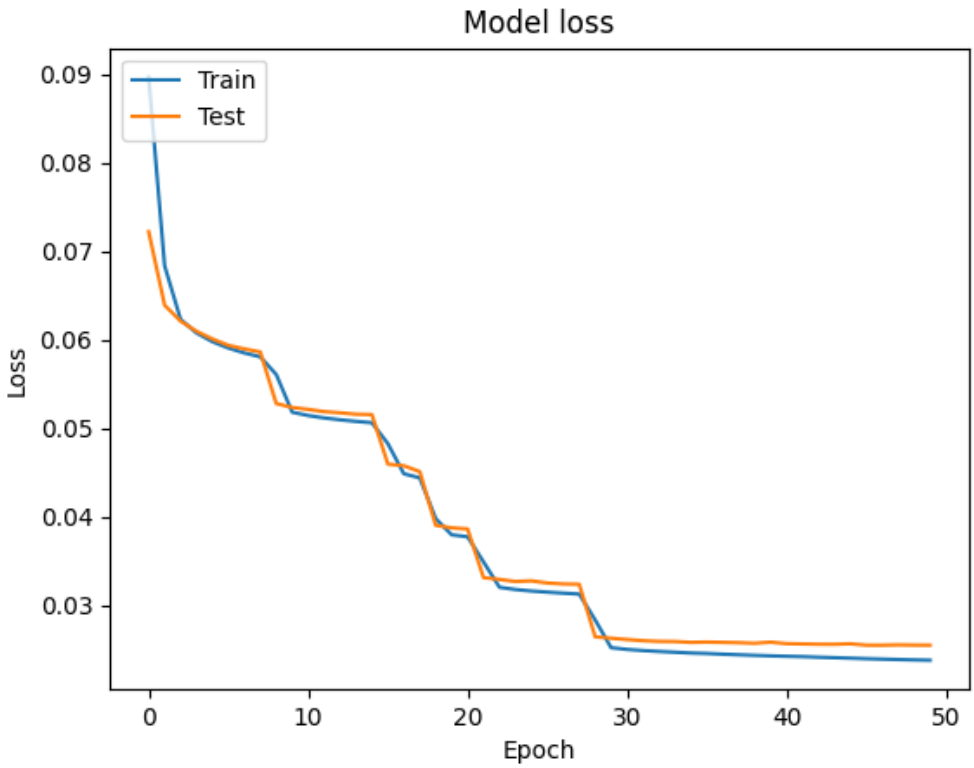


图 2.1.1 损失值随 Epoch 变化

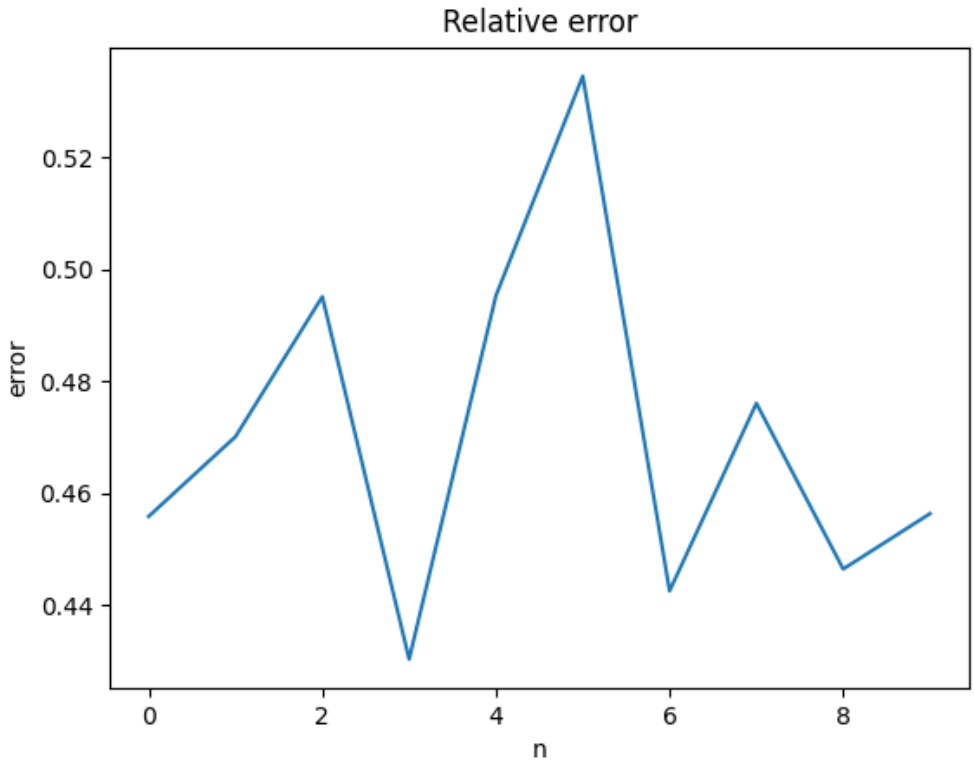


图 2.1.2 相对误差（前十幅图片）

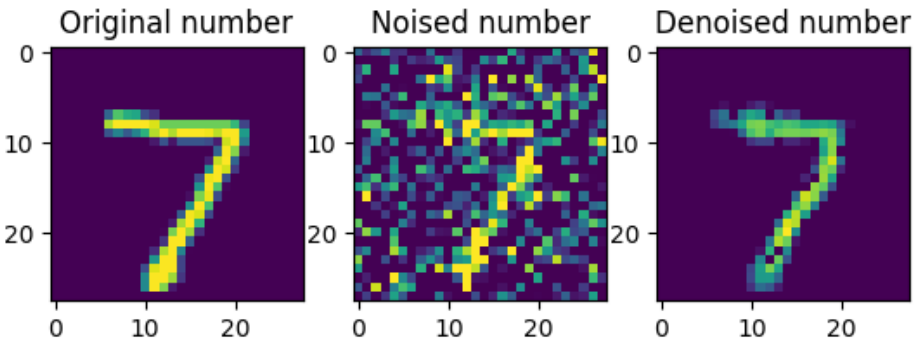


图 2.1.3 输出图片

通过计算，此时的相对误差为 0.470292，且输出的数字可以较好的得到还原。

```
The average relative error is 0.470292
```

ii) 固定 loss 为 MSE，比较至少 3 种不同优化器的训练测试结果：

如图 2.2.1，在网络结构不变的情况下，使用 Adagrad 优化器，其学习率设为 0.01，可得训练测试结果如下：

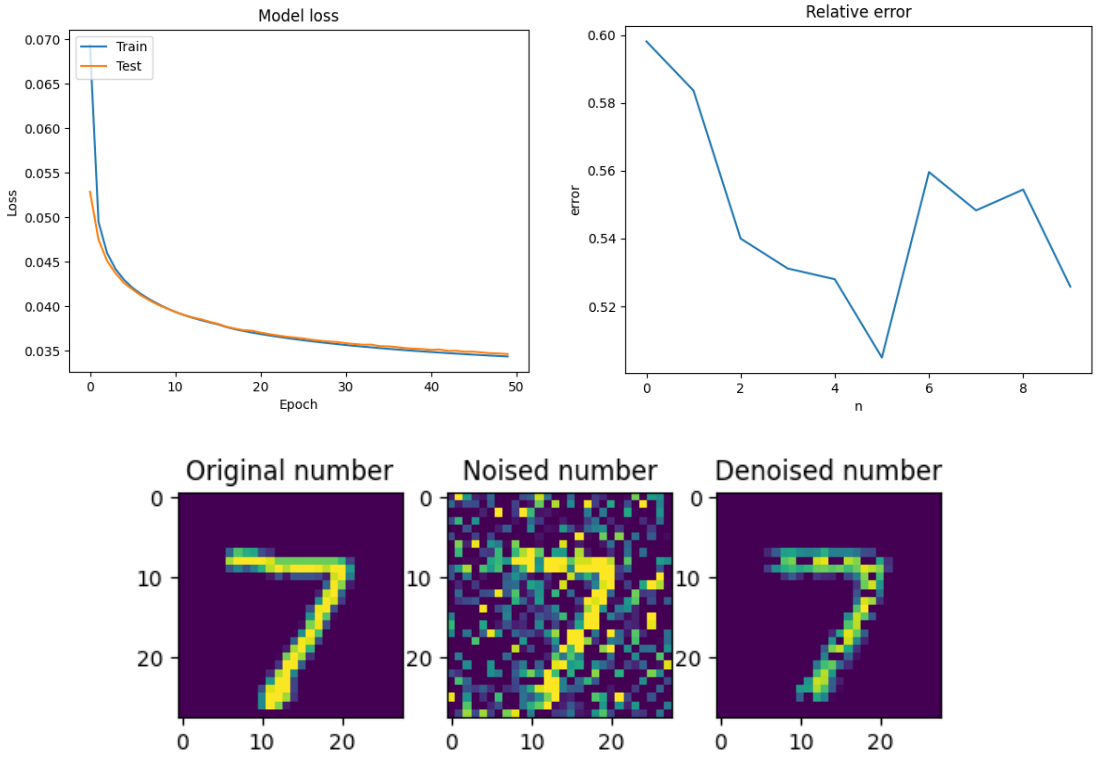


图 2.2.1 Adagrad 优化器测试结果

此时的平均相对误差为 0.547403。

如图 2.2.2，为使用 SGD 优化器、学习率为 0.01 时的训练测试结果：

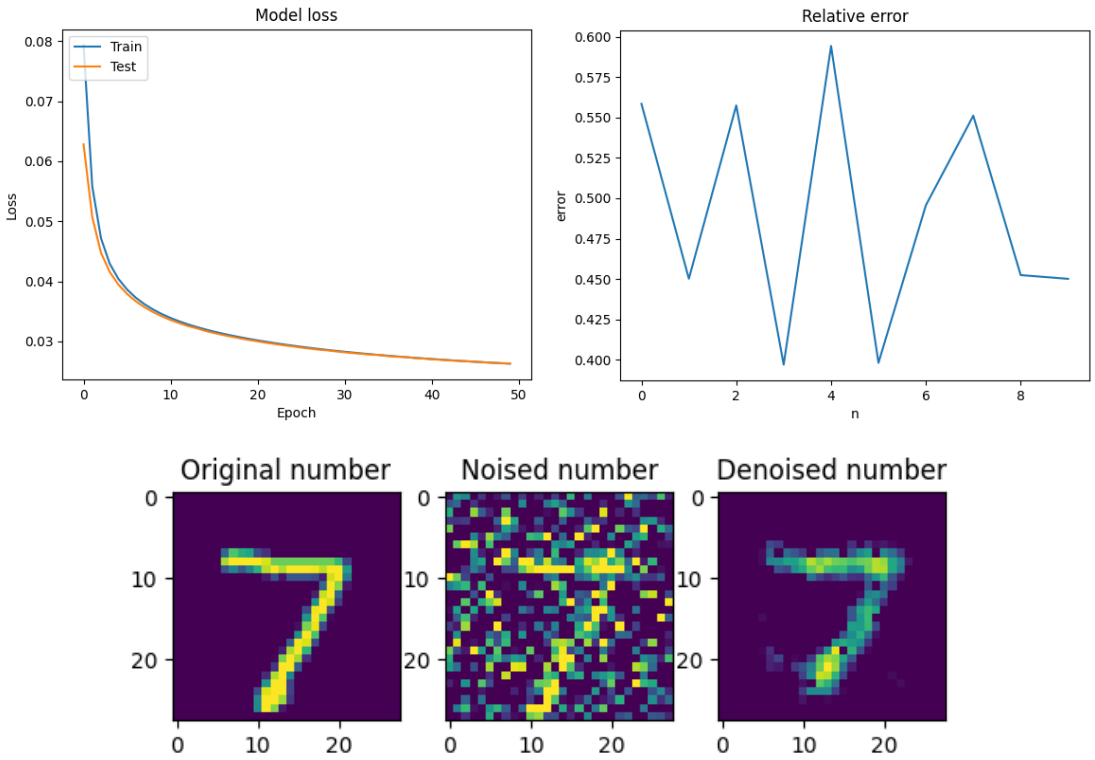


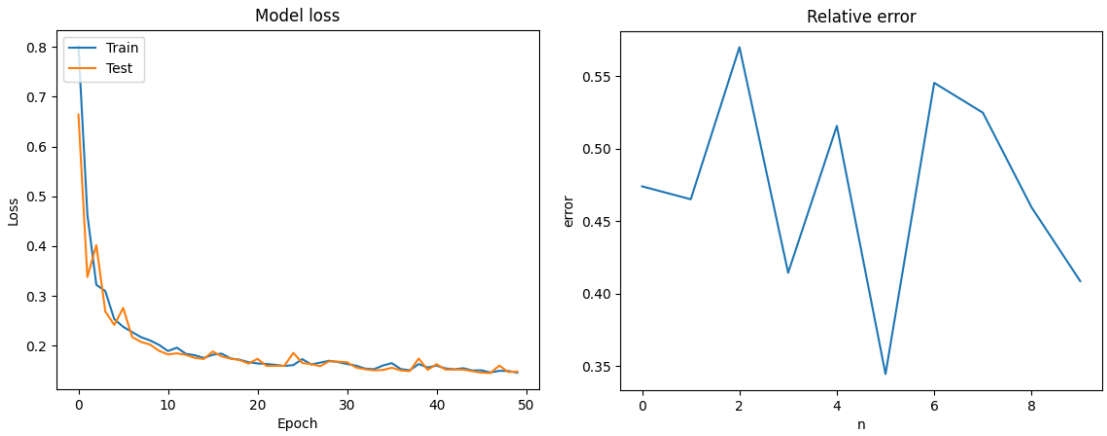
图 2.2.2 SGD 优化器测试结果

此时的平均相对误差为 0.490486。

通过比较 Adam、Adagrad 以及 SGD 优化器的输出结果，可以发现，不同优化器的性能存在差异，其中，Adam 优化器的综合性能最好，而 Adagrad 的性能稍差。相对而言， SGD 的收敛更为平滑，但最终得到的损失值与 Adam 类似；Adam 的平均相对误差比 SGD 更低。对重构后的数字而言，三种优化器都可以在一定程度上滤去噪声，恢复出比较明显的数字轮廓，符合设计要求。

iii) 固定优化器为 Adam，比较至少 3 种不同 loss 的训练测试结果：

将 loss 设置为 binary_crossentropy，得到训练测试结果如图 2.3.1 所示：



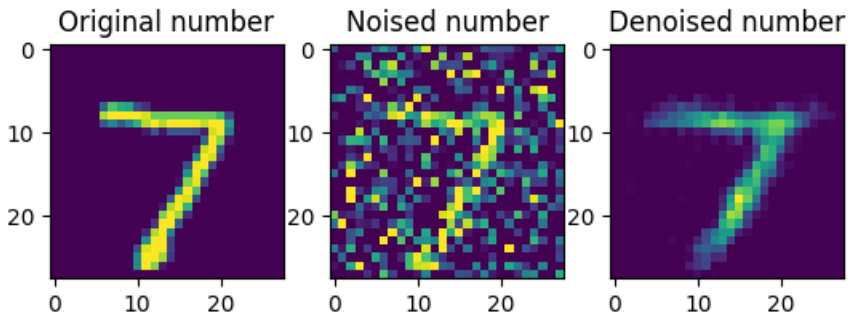


图 2.3.1 “binary_crossentropy”测试结果

输出结果的平均相对误差为 0.472083。

同理，将 loss 修改为 mean_absolute_error 类型，得到输出结果如下：

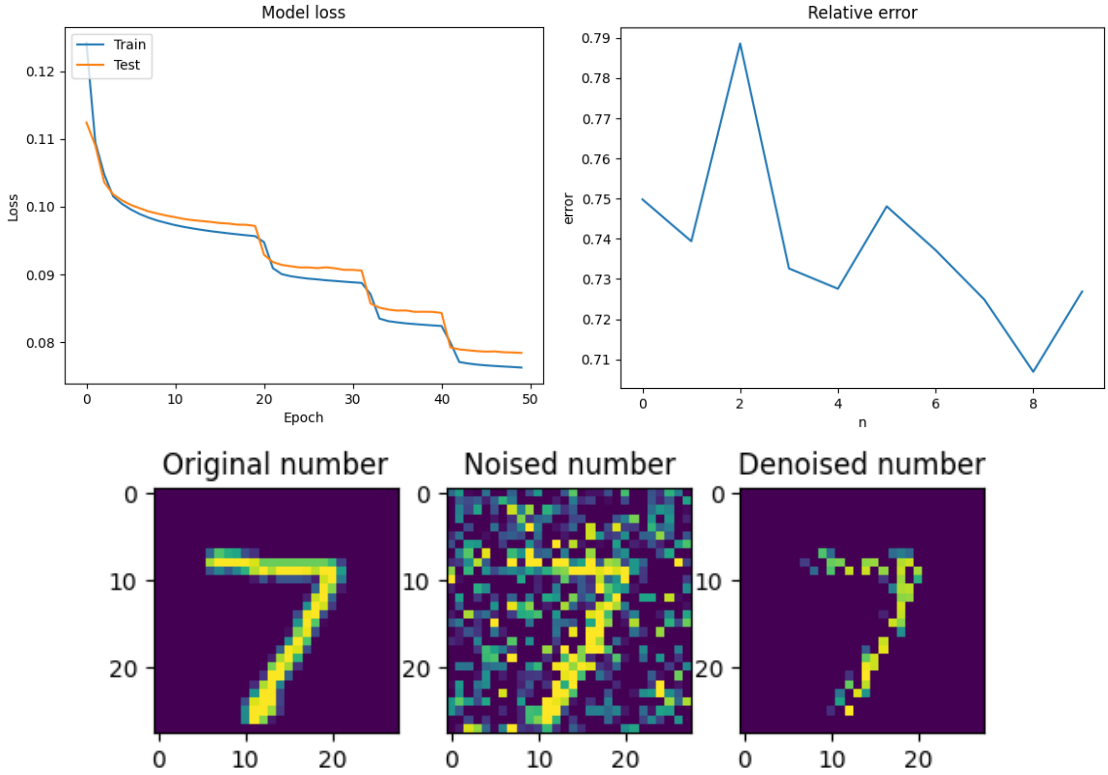


图 2.3.2 “mean_absolute_error”测试结果

此时的平均相对误差为 0.738198。

比较 loss 分别为均方误差、二值交叉熵和平均绝对误差时的测试结果，可以发现不同类型的 loss 会对网络的收敛和最终输出产生较大影响。相对而言，均方误差与二值交叉熵的相对误差类似，但二值交叉熵的收敛更为平滑，且迭代 50 次以后的损失值更低；平均绝对误差的效果最差，其损失值和相对误差都明显高于另外两种 loss。从输出图像来看，平均绝对误差的数字较为模糊，但三种误差类型的输出都大致可以辨别出数字“7”，满足设计要求。

iv) 找出相对误差小于 50%时能够获得的最大压缩比：

由 i)中结果可知，当压缩比为 $\frac{5}{28}$ 时，输出结果的相对误差为 47.03%。修改网络结构如下，此时输入维度为(28,28,1)，而经过压缩：((28-4)/2-4)/2=4，即压缩后的维度为(4,4,32)，压缩比为 $\frac{4}{28}=\frac{1}{7}$。

```
# 编码器
x = Conv2D(32, 5, padding='valid', activation='relu', kernel_initializer='he_normal')(input_img)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(32, 5, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
encoded = MaxPooling2D(pool_size=(2, 2))(x)

# 解码器
x = Conv2DTranspose(32, 8, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(encoded)
x = Conv2D(32, 3, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
decoded = Conv2DTranspose(1, 6, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(x)
```

此时相对误差为 0.348318，结果有一定程度的改善。其输出如图 2.4.1 所示：

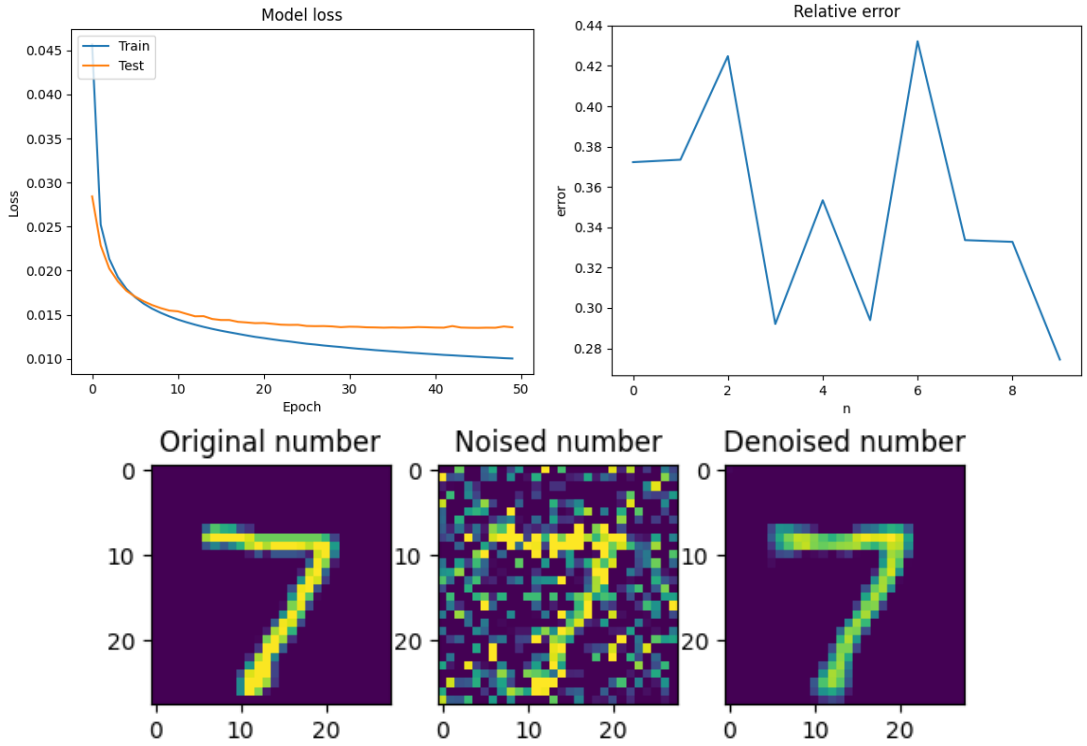


图 2.4.1 压缩比为 $\frac{1}{7}$ 时测试结果

类似的，将压缩比进一步提高到 $\frac{3}{28}$，如下：

```
# 编码器
x = Conv2D(32, 5, padding='valid', activation='relu', kernel_initializer='he_normal')(input_img)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(32, 7, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
encoded = MaxPooling2D(pool_size=(2, 2))(x)

# 解码器
x = Conv2DTranspose(32, 8, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(encoded)
x = Conv2D(32, 4, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
x = Conv2DTranspose(1, 2, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(x)
x = Conv2D(32, 5, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
decoded = Conv2DTranspose(1, 2, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(x)
```

此时得到相对误差为 0.443126，满足要求，输出图表如图 2.4.2 所示：

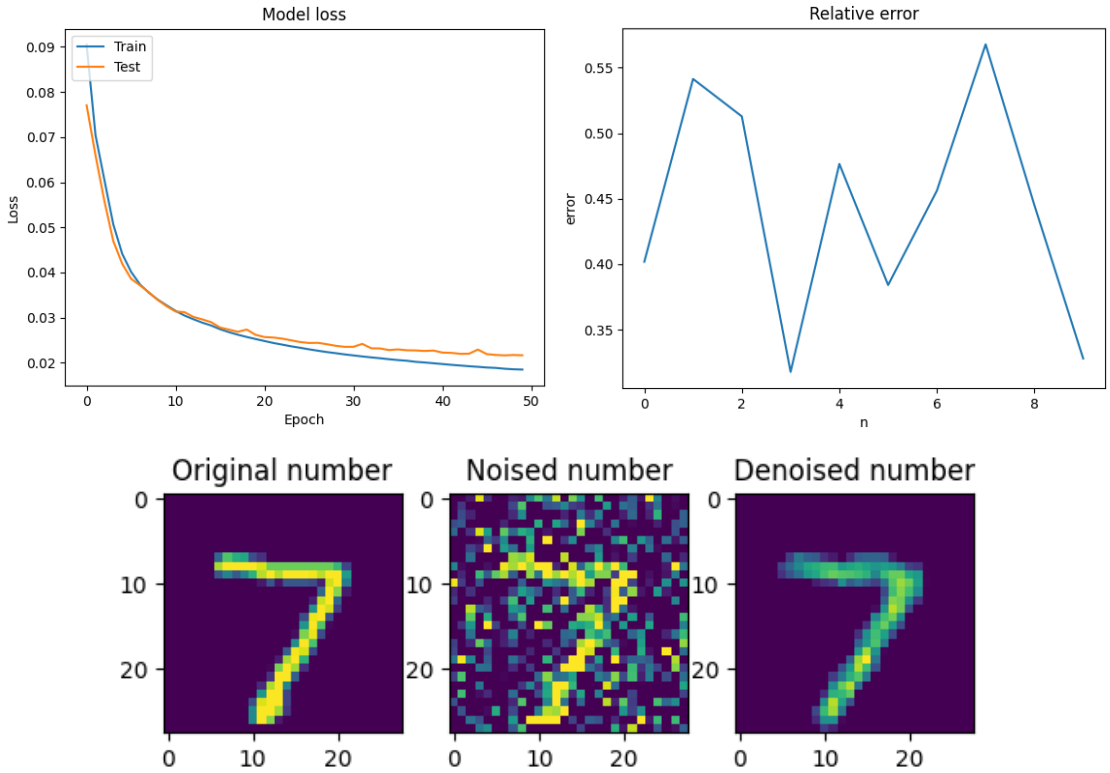


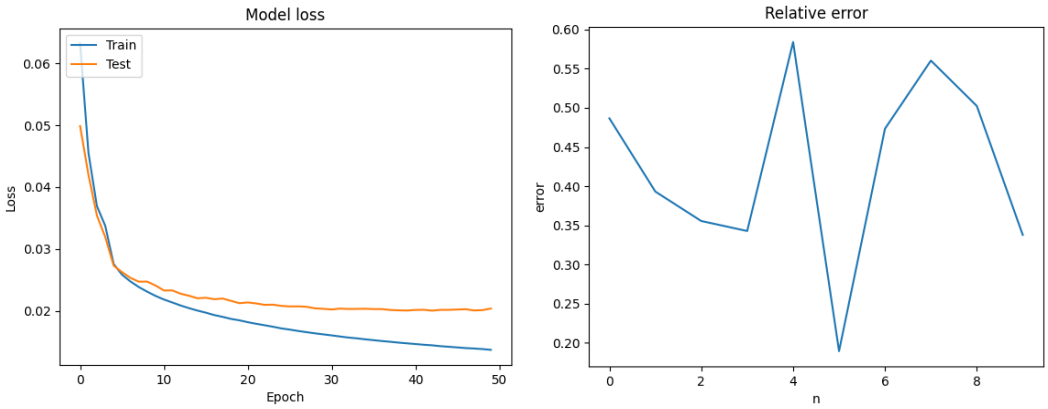
图 2.4.2 压缩比为$\frac{3}{28}$时测试结果

修改网络，进一步将输入压缩到(1,1,32)，如下：

```
# 编码器
x = Conv2D(32, 8, padding='valid', activation='relu', kernel_initializer='he_normal')(input_img)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(32, 8, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
encoded = MaxPooling2D(pool_size=(2, 2))(x)

# 解码器
x = Conv2DTranspose(32, 8, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(encoded)
x = Conv2D(32, 3, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
x = Conv2DTranspose(32, 6, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(x)
x = Conv2D(32, 3, padding='valid', activation='relu', kernel_initializer='he_normal')(x)
decoded = Conv2DTranspose(1, 2, strides=(2, 2), padding='valid', activation='relu', use_bias=False, kernel_initializer='he_normal')(x)
```

得到前 10 幅图像的平均相对误差为 0.422593，小于 50%，收敛情况和重构后的图像如图 2.4.3 所示。可见，当压缩比为$\frac{1}{28}$时，网络仍能够较好的实现图像去噪。



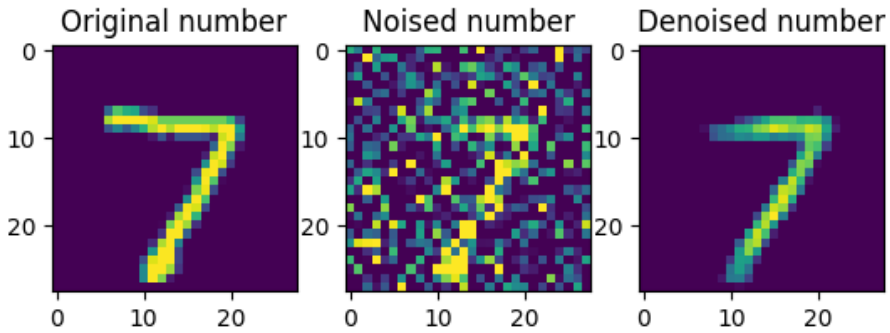


图 2.4.3 压缩比为$\frac{1}{28}$时测试结果

综上，从输出平均相对误差来看，在满足相对误差小于 50%要求的基础上，自动编码器所能够达到的最大压缩比为$\frac{1}{28}$。当然，若要求测试集中每一张图片的相对误差都不大于 50%，则此时对应的最大压缩比为$\frac{1}{7}$。

四、附录：实验 Python 代码——*Autoencoder.py*

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
import numpy as np


if __name__ == '__main__':
    ## 数据处理
    (x_train, _), (x_test, _) = mnist.load_data()
    x_train = x_train.astype('float32') / 255.
    x_test = x_test.astype('float32') / 255.
    x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
    x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

    x_train = x_train[:10000]
    x_test = x_test[0:10000]

    ## 添加噪声
    noise_factor = 0.5
    x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0,
scale=1.0, size=x_train.shape)
    x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0,
scale=1.0, size=x_test.shape)
    x_train_noisy = np.clip(x_train_noisy, 0., 1.)
```

```
    x_test_noisy = np.clip(x_test_noisy, 0., 1.)

    input_img = Input(shape=(28, 28, 1))

    # 编码器
    x = Conv2D(32, 3, padding='valid', activation='relu',
kernel_initializer ='he_normal')(input_img)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Conv2D(32, 3, padding='valid', activation='relu',
kernel_initializer='he_normal')(x)
    encoded = MaxPooling2D(pool_size=(2, 2))(x)

    # 解码器
    x = Conv2DTranspose(32, 8, strides=(2, 2), padding='valid',
activation='relu', use_bias=False,
kernel_initializer='he_normal')(encoded)
    x = Conv2D(32, 3, padding='valid', activation='relu',
kernel_initializer='he_normal')(x)
    decoded = Conv2DTranspose(1, 2, strides=(2, 2), padding='valid',
activation='relu', use_bias=False, kernel_initializer='he_normal')(x)

    learning_rate = 1e-4
    autoencoder = Model(input_img, decoded)
    autoencoder.compile(optimizer = Adam(lr = learning_rate),
loss='mean_squared_error')  # Adam
    # autoencoder.compile(optimizer = Adagrad(lr = 0.01),
loss='mean_squared_error')        # Adagrad
    # autoencoder.compile(optimizer = SGD(lr = 0.01),
loss='mean_squared_error')           # SGD

    ## 训练
    filepath="./records/Auto-{epoch:02d}-{val_loss:.2f}.hdf5"
    model_checkpoint = ModelCheckpoint(filepath,
monitor='val_loss',verbose=1, save_best_only=True, period=5)

    history = autoencoder.fit(x_train_noisy, x_train,
                    epochs=50,
                    batch_size=10,
                    shuffle=True,
                    validation_data=(x_test_noisy, x_test),
                    callbacks=[model_checkpoint])

    ## 画出训练过程
    plt.figure(0)
```

```
    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()

    ## 典型结果测试、可视化、计算误差
    Inx_N = 10
    errs = []
    results = []
    for inx in range(Inx_N):
        test_inp = x_test_noisy[inx, ]
        test_inp = np.reshape(test_inp, (1, )+test_inp.shape)
        result = autoencoder.predict(test_inp, batch_size=1)
        err = np.linalg.norm(x_test[inx].reshape(28,28)-
result.reshape(28,28))/np.linalg.norm(x_test[inx].reshape(28,28))
        results.append(result)
        errs.append(err)

    xlabel = np.linspace(0, Inx_N, Inx_N, endpoint=False)
    plt.figure(1)
    plt.plot(xlabel, errs)
    plt.title('Relative error')
    plt.ylabel('error')
    plt.xlabel('n')
    plt.show()
    print('The average relative error is %f.' %
(np.sum(errs)/len(errs)))

    plt.figure(2)
    plt.subplot(1,3,1)
    plt.imshow(x_test[0].reshape(28,28))
    plt.title('Original number')
    plt.subplot(1,3,2)
    plt.imshow(x_test_noisy[0].reshape(28,28))
    plt.title('Noised number')
    plt.subplot(1,3,3)
    plt.imshow(results[0].reshape(28,28))
    plt.title('Denoised number')
    plt.show()
```