中国科学技术大学

本科毕业论文

题 目 基于深度卷积网络的图像去噪研究

英 文 Research on Picture Noise Reduction Based Deep Convolution Network

院 系 计算机科学与技术学院

姓 名 李京

导 师 张信明

日 期 二〇一七年五月

第二章 相关工作

2.1 图像的噪声模型

信号获取以及传输过程中，会受到外在能量所产生信号的干扰，频率、强弱变化无规律，杂乱无章。我们可以认为一个受噪声干扰的图像为原始信息与噪声信息的叠加。图像去噪的任务就是区分图像中的干扰信息，并将其去除。由于干扰源众多且不固定，噪声特性复杂且具有非常强的随机性。图像的噪声信息我们可以将其认为一个随机过程，利用概率统计的方法来分析噪声。那么绝大多数图像噪声服从绝对值为 0 的高斯分布：

$$N(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2\sigma^2}} \tag{2-1}$$

其中式中x表示噪声中的灰度值，符合高斯分布，σ为噪声的标准差。那么一个受到噪声干扰的图像可以表示为：

$$Y = R + N \tag{2-2}$$

其中 Y 为观察到的包含噪声的图像，R 为不包含噪声的图像，N 为噪声。从该模型上看，该加性噪声仅与外界因素有关，与原始图像无关。如图 2.1 所示

图 2.1 图像的噪声模型

因此，在此模型下，我们可以描述图像的去噪算法为：输入信息为包含噪声的图像 Y，那么该算法的输出则为图像的原始信息 X。

2.2 深度神经网络

深度神经网络由多个网络层堆叠而成，每一层包含多个节点。神经网络的运算在节点中完成。每个节点对于输入的每个数据，使用一组系数（亦称权重）与之结合，作为该节点的数据或者经过激活函数后作为该点的数据。如下图所示

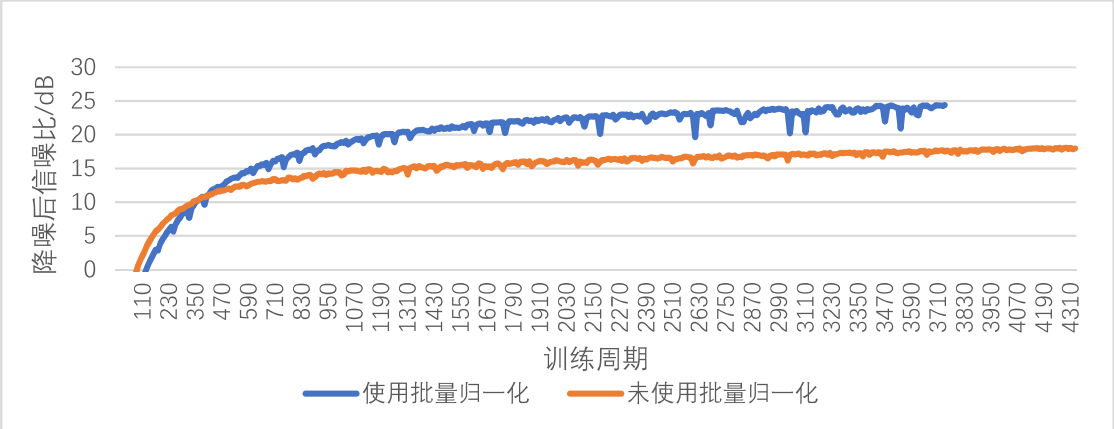


图 3.2 是否使用归一化带来影响对比

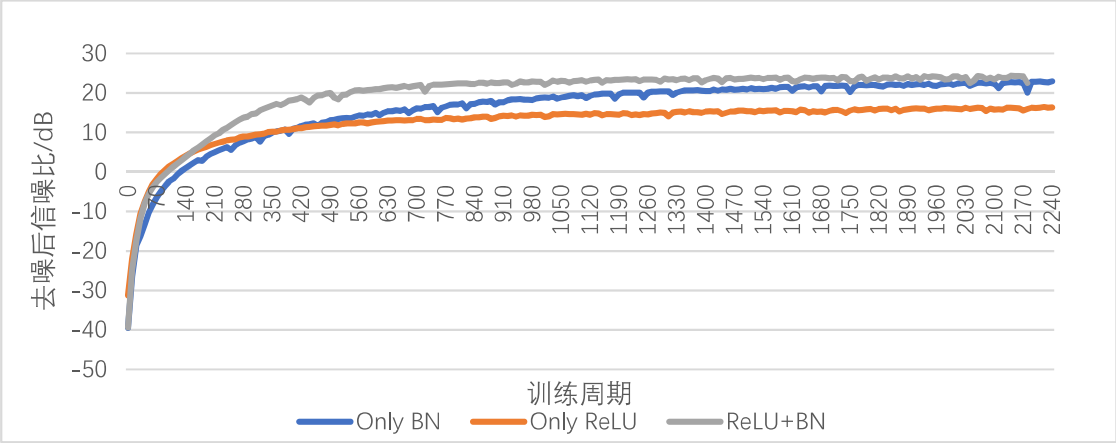


图 3.3 三种网络训练效果对比

如图 3.2 所示为在使用相同的网络结构，相同的训练参数，以及相同的梯度下降优化算法（Adam）下，训练后获得的信噪比 PSNR 的对比。图中我们可以看到，没有批量归一化的情况下，收敛速度在 500 次循环后变得十分缓慢，使得网络训练非常艰难。在没有 ReLU 激活操作的情况下，神经网络的训练速度同样不理想。当该卷积网络结合了 ReLU 激活函数的激活处理以及批量归一化处理后，它的收敛速度更快，且更稳定。

3.3 功能性拓展

以上所述卷积神经网络，仅针对特定特定尺寸且具有特定高斯噪声的图像进行噪声的区分。接下来，将对其进行拓展。

A．首先考虑尺寸问题。由 3.2 所述，神经网络每一层均由卷积操作、批量归一化操作、以及激活操作构成，并不包含全连接层。其中，批量归一化以及激活操作均对数据的尺寸没有要求。由 2.3 中卷积操作的原理，卷积操作对图像的尺

寸没有固定的要求。且该网络不包含对尺寸敏感的全连接。这就意味着，对于同样噪声类型的图像，我们不需要重新训练不同的尺寸的网络亦可进行对不同尺寸图像的去噪处理，且避免使用图像切割等方法。

对此，修改网络详见附录【2】

这里的输入层尺寸并不是固定的，其尺寸根据图像尺寸自适应。

其中，im.size 为输入图像的尺寸，是一个二元数组。最终，实现了对不同尺寸图像的去噪处理，而不仅仅针对 256×256

图 3.4 对于更大尺寸图像进行去噪

图 3.4 中，使用已有网络以及经过训练的模型对尺寸为 512×512 的图像进行去噪处理，该高斯噪声 $\sigma=16$，去噪效果与 256×256 图像的去噪效果相当，去噪 PSNR 为 32.79dB。

B．目前已进行处理的图像均为单通道灰度图。由 3.2 中网络结构，针对彩色图像不同的通道数，我们选择 c=3 或者 c=4，即可进行对彩色图像的去噪。

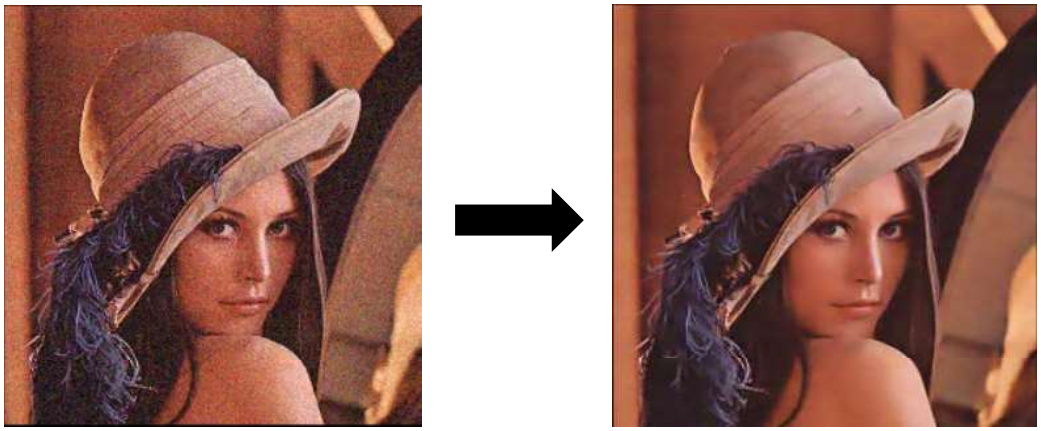


图 3.5 对彩色图进行去噪处理

图 4.1 灰度图部分测试图

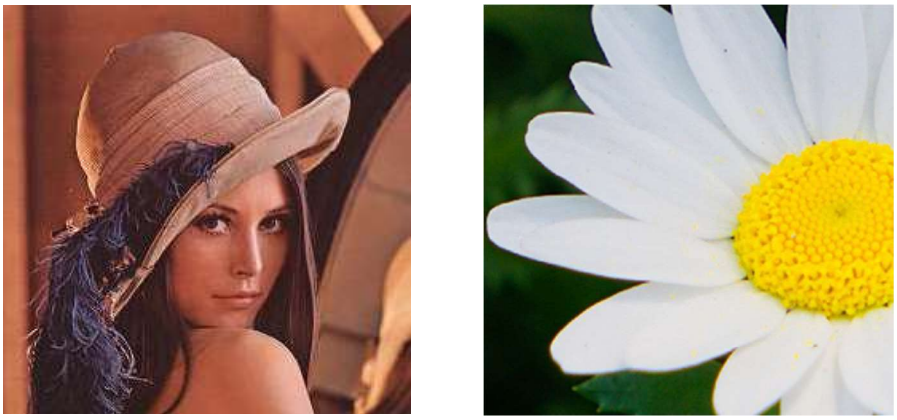


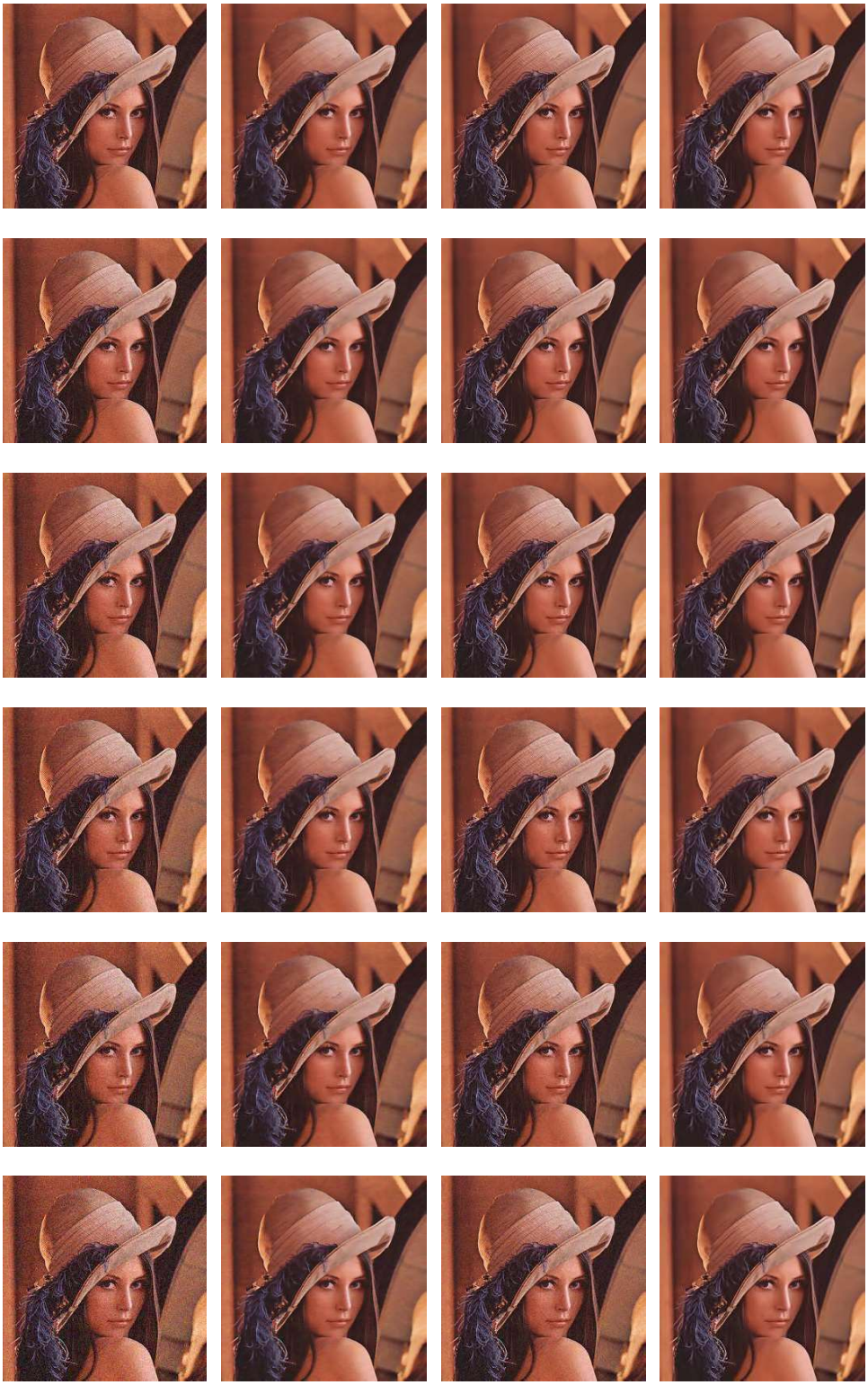
图 4.1 彩色图测试部分

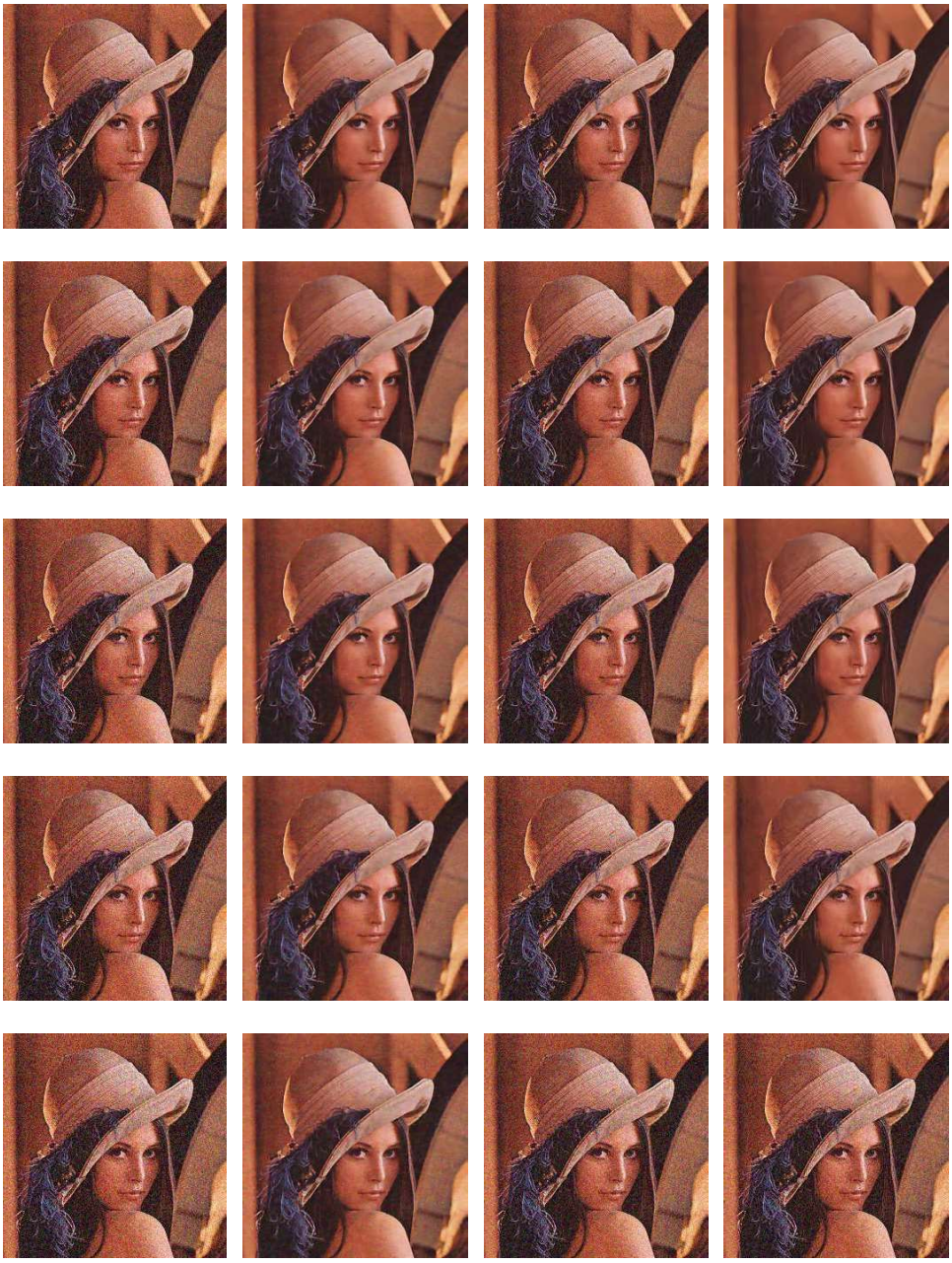
在不固定噪声水平的高斯噪声去噪测试中，使用经典的 lena 图像进行。对于 lena 图像，已经实现做好了添加了 $\sigma \in [6,25]$ 的高斯噪声的测试样例，方便进行测试的进行。这里参与对比是 BM3D 算法在指定 $\sigma = 10$ 以及 $\sigma = 20$ 情况下对以上测试样例进行去噪。

对于以上所提出的训练以及测试任务，均使用 PyCharm 2017.1.2 进行。所使用 PC 环境为 CPU：Intel(R) Core(TM) i7-4710HQ 2.5GHz，GPU：Nvidia GTX850M。进行测试前，对固定噪声水平的去噪模型进行了 9 个小时的训练，对不固定噪声水平额去噪模型进行了 12 个小时的训练。

4.3 测试结果及分析

首先针对固定噪声水平的高斯去噪。在这次对比中，分别对本文提出的 CNN





包含噪声图　　CNN　　BM3D-10　　BM3D-20

图 4.5 对于不同噪声水平图像的去噪

从上到下噪声水平分别是$\sigma = 4,6,8,...,26$，从左到右分别为包含噪声的图像、使用该神经网络去噪后的图像、BM3D 算法设定$\sigma = 10$去噪后的图像以及 BM3D 算法设定$\sigma = 20$去噪后的图像。

图 4.5 展示了去噪网络以及 BM3D 算法对不同噪声情况下的去噪效果。有图可见，当 BM3D 算法在噪声水平设定准确时，去噪效果较好。当噪声水平大于设定噪声水平时，会出现较多的噪声残留现象。当噪声水平小于设定噪声水平时，则容易出现细节丢失的问题。而对于使用不同噪声进行过训练的去噪神经网络，

结果表现较为稳定，如表 4.3 所示。

PSNR/dB	BM3D-10	BM3D-20	CNN
σ =6	**36.97356015**	33.40870956	33.38923828
σ =8	**36.50051979**	33.34458741	33.31812452
σ =10	**35.7388998**	33.27873705	33.24743425
σ =12	**34.45901039**	33.19661753	33.14199662
σ =14	**32.62281171**	33.09027261	33.06435331
σ =16	30.68511527	**32.85377552**	32.83433966
σ =18	29.04540531	32.73299558	**34.44476086**
σ =20	27.62150262	32.31388431	**34.17393593**
σ =22	26.46328115	31.85535024	**33.85981695**
σ =24	25.53295404	31.13653877	**33.49729805**
σ =26	24.6431172	24.6431172	**33.02179337**

表 4.3 不同噪声水平下去噪效果

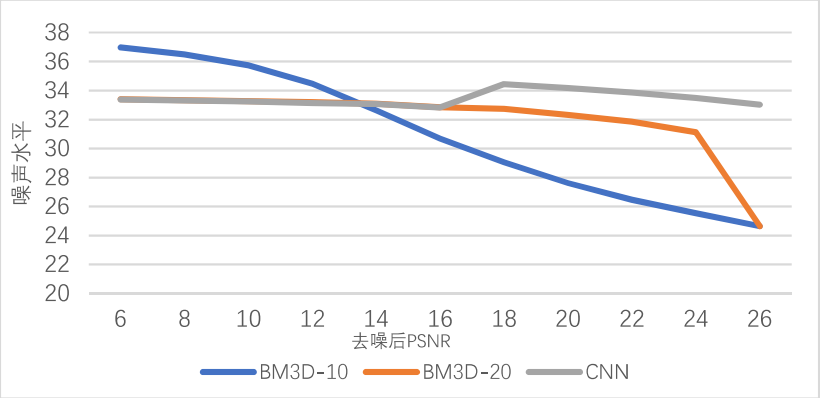


图 4.6 不同噪声水平下去噪效果

图 4.6 可以非常直观的对比集中去噪算法的去噪效果。其中，使用卷积网络进行去噪表现出了较强的稳定性。即使是一个较为简单的网络结构，在噪声水平为 $\sigma = 6$ 到 $\sigma = 26$ 中均表现出了不错的成绩。在该轮测试中，卷积网络同样表现出了较高的性能，这里不再详述。

综上，使用卷积网络进行去噪处理，可以同时获得较好的去噪效果以及优秀的性能表现，结果令人欣慰。

附录

【1】 附代码 1

```
x_image = tf.reshape(x, [-1, 256, 256, 1])

W_conv1 = weight_variable([5, 5, 1, 24])  # 第一层卷积层
b_conv1 = bias_variable([24])  # 第一层卷积层的偏置量
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

W_conv2 = weight_variable([5, 5, 24, 24])  # 第二次卷积层
b_conv2 = bias_variable([24])  # 第二层卷积层的偏置量
h_conv2 = tf.nn.relu(batchnormalize(conv2d(h_conv1, W_conv2) +
b_conv2))

W_conv3 = weight_variable([5, 5, 24, 24])  # 第三次卷积层
b_conv3 = bias_variable([24])  # 第二层卷积层的偏置量
h_conv3 = tf.nn.relu(batchnormalize(conv2d(h_conv2, W_conv3) +
b_conv3))

W_conv4 = weight_variable([5, 5, 24, 1])  # 第四次卷积层
b_conv4 = bias_variable([1])  # 第二层卷积层的偏置量
h_conv4 = conv2d(h_conv3, W_conv4) + b_conv4
y = tf.reshape(h_conv4, [-1, 65536])
```

【2】 附代码 2

```
x_image = tf.reshape(x, [-1, im.size[0], im.size[1], 1])

W_conv1 = weight_variable([5, 5, 1, 24])  # 第一层卷积层
b_conv1 = bias_variable([24])  # 第一层卷积层的偏置量
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

W_conv2 = weight_variable([5, 5, 24, 24])  # 第二次卷积层
b_conv2 = bias_variable([24])  # 第二层卷积层的偏置量
h_conv2 = tf.nn.relu(batchnormalize(conv2d(h_conv1, W_conv2) +
b_conv2))

W_conv3 = weight_variable([5, 5, 24, 24])  # 第三次卷积层
b_conv3 = bias_variable([24])  # 第二层卷积层的偏置量
h_conv3 = tf.nn.relu(batchnormalize(conv2d(h_conv2, W_conv3) +
b_conv3))
```

```
W_conv4 = weight_variable([5, 5, 24, 1])  # 第四次卷积层
b_conv4 = bias_variable([1])  # 第二层卷积层的偏置量
h_conv4 = conv2d(h_conv3, W_conv4) + b_conv4
y = tf.reshape(h_conv4, [1, -1])
```

【3】 附代码 3

```
loss = tf.reduce_sum((y_ - y)××2)
train_step = tf.train.AdamOptimizer(2e-4).minimize(loss)
```