

问题回答：

初步要求：

1:

常见图像格式：

图 像	BMP	JPEG	PNG
损 失	无	有	无
特 点	由文件头、位图信息头和像素数据组成	由图像压缩、色彩空间、基线或渐进组成	由图像压缩、调色板和直接颜色、透明度组成

next :常用的图像处理编程语言 有：

Python:

Python其中丰富的库（*PIL*、*scikit-image*等）使其在该领域受到广泛应用

MATLAB:

典中典，然学不会

JAVA:

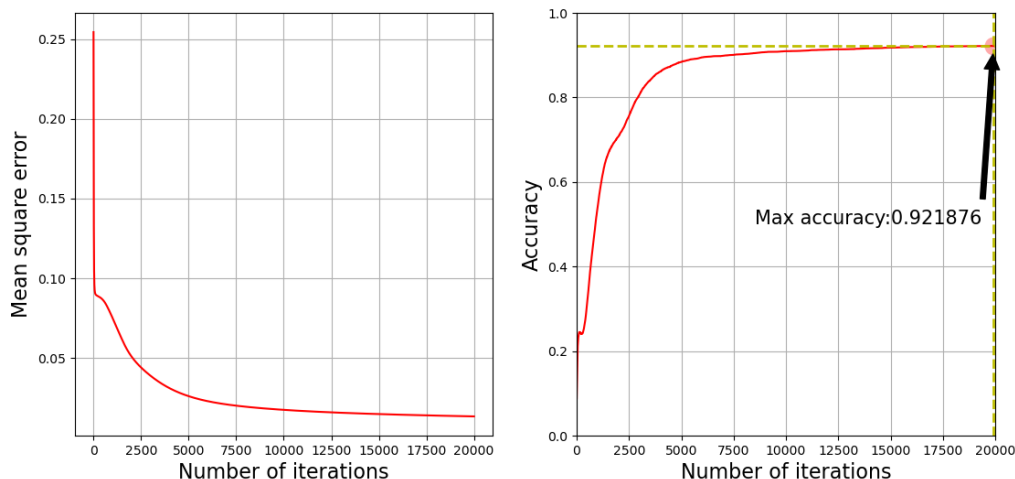
JAVA同Python一样也有应用于图像处理的库 eg: *JAI*、*ImageJ*

next : OpenCV :

OpenCV是一个支持多种语言的计算机视觉库（包括 *Python*），除了图像相关的操作：读取，预处理之类的，他其中如 *pytorch* 般也有些机器学习，深度学习模块：SVM、随机森林等

记得之前参加 *Robocon* 夏令营的时候学长教的机器人视觉就是 *OpenCV* 实现的，功能强大，资料丰富

2: *done!*（好耶）



3: 夺命三连问:

源代码使用 *PIL* 库读取数据，具体就是，用 *Image.open()* 函数打开文件，用 *plt.imread()* 函数把图片读取为数组形式

首先把文件路径存储在 *data_dir* 里面

使用 *listdir()* 存了所有文件的列表

读取为数组格式后使用 *reshape()* 函数将图片变成一维向量（784维）

训练就是BP算法：设计神经网络、向前向后传播然后更新输出层和隐含层的参数

神经网络结构：输入层有784个神经元，隐含层只有15个，输出层有10个，激活函数是sigmoid

进阶要求:

1:

事实上在完成task2任务时我一边阅读理解代码一边学习b站的视频并尝试使用pytorch中成熟完备的各种神经网络API复现task2的代码作为对比学习。由此，我得以深刻理解使用API时神经网络中的一切到底在如何运作，而加深网络结构的任务我就在我的对比学习代码中完成了（torch.py）。以下是结果：

```
Epoch [0/10], Step [000/938], Loss: 0.0100
Epoch [8/10], Step [700/938], Loss: 0.0102
Epoch [8/10], Step [800/938], Loss: 0.0121
Epoch [8/10], Step [900/938], Loss: 0.1192
Epoch [9/10], Step [100/938], Loss: 0.0447
Epoch [9/10], Step [200/938], Loss: 0.0470
Epoch [9/10], Step [300/938], Loss: 0.0159
Epoch [9/10], Step [400/938], Loss: 0.0380
Epoch [9/10], Step [500/938], Loss: 0.0215
Epoch [9/10], Step [600/938], Loss: 0.0256
Epoch [9/10], Step [700/938], Loss: 0.0260
Epoch [9/10], Step [800/938], Loss: 0.0136
Epoch [9/10], Step [900/938], Loss: 0.0240
Epoch [10/10], Step [100/938], Loss: 0.0044
Epoch [10/10], Step [200/938], Loss: 0.0250
Epoch [10/10], Step [300/938], Loss: 0.0225
Epoch [10/10], Step [400/938], Loss: 0.0834
Epoch [10/10], Step [500/938], Loss: 0.0446
Epoch [10/10], Step [600/938], Loss: 0.0316
Epoch [10/10], Step [700/938], Loss: 0.0299
Epoch [10/10], Step [800/938], Loss: 0.0010
Epoch [10/10], Step [900/938], Loss: 0.0936
Test Accuracy: 97.69%
```

2:

CNN 卷积神经网络 其本质可以抽象成卷积层+池化层+全连接BP

卷积的存在使其在处理图像方面具有天然的优势

task2——CNN结构位于 /机器学习task2/CNN.py中，其结果如下：（99.07%）



+ 代码 + 文本

连接 ▾



{x}



```
Epoch [8/10], Step [600/938], Loss: 0.0039
Epoch [8/10], Step [700/938], Loss: 0.0004
Epoch [8/10], Step [800/938], Loss: 0.0517
Epoch [8/10], Step [900/938], Loss: 0.0013
Epoch [9/10], Step [100/938], Loss: 0.0272
Epoch [9/10], Step [200/938], Loss: 0.0001
Epoch [9/10], Step [300/938], Loss: 0.0026
Epoch [9/10], Step [400/938], Loss: 0.0011
Epoch [9/10], Step [500/938], Loss: 0.0001
Epoch [9/10], Step [600/938], Loss: 0.0013
Epoch [9/10], Step [700/938], Loss: 0.0061
Epoch [9/10], Step [800/938], Loss: 0.0007
Epoch [9/10], Step [900/938], Loss: 0.0305
Epoch [10/10], Step [100/938], Loss: 0.0005
Epoch [10/10], Step [200/938], Loss: 0.0345
Epoch [10/10], Step [300/938], Loss: 0.0316
Epoch [10/10], Step [400/938], Loss: 0.0000
Epoch [10/10], Step [500/938], Loss: 0.0013
Epoch [10/10], Step [600/938], Loss: 0.0004
Epoch [10/10], Step [700/938], Loss: 0.0002
Epoch [10/10], Step [800/938], Loss: 0.0194
Epoch [10/10], Step [900/938], Loss: 0.0007
Test Accuracy: 99.07%
```

真是不由得感叹 *pytorch* 库的存在让代码任务轻松了多少