深度学习原理与 TensorFlow 实践
实践指导书

智能科学与技术系

2022 年 9 月

目录

实践一 开发环境及编程入门

（一） 实践目的

熟悉Python开发环境；掌握Python程序设计基本知识与技能；熟悉TensorFlow开发环境。

（二） 实践环境

计算机、希冀平台

（三） 实践学时

4学时

（四） 实践内容与步骤

1.Python简介

Python 是一种解释型、面向对象、动态数据类型的高级程序设计语言。Python 由 Guido van Rossum 于 1989 年底发明，第一个公开发行版发行于 1991 年。Python是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。最初被设计用于编写自动化脚本(shell)，随着版本的不断更新和语言新功能的添加，越多被用于独立的、大型项目的开发。

Python 的设计具有很强的可读性，相比其他语言经常使用英文关键字，其他语言的一些标点符号，它具有比其他语言更有特色语法结构。

Python 是一种解释型语言： 这意味着开发过程中没有了编译这个环节。类似于PHP和Perl语言。

Python 是交互式语言： 这意味着，您可以在一个 Python 提示符 >>> 后直接执行代码。

Python 是面向对象语言: 这意味着Python支持面向对象的风格或代码封装在对象的编程技术。

Python 是初学者的语言：Python 对初级程序员而言，是一种伟大的语言，它支持广泛的应用程序开发，从简单的文字处理到 WWW 浏览器再到游戏。

Python发展历史

Python 是由 Guido van Rossum 在八十年代末和九十年代初，在荷兰国家数学和计算机科学研究所设计出来的。

Python 本身也是由诸多其他语言发展而来的,这包括 ABC、Modula-3、C、C++、Algol-68、SmallTalk、Unix shell 和其他的脚本语言等等。

像 Perl 语言一样，Python 源代码同样遵循 GPL(GNU General Public License)协议。

现在 Python 是由一个核心开发团队在维护，Guido van Rossum 仍然占据着至关重要的作用，指导其进展。

Python 2.7 被确定为最后一个 Python 2.x 版本，它除了支持 Python 2.x 语法外，还支持部分 Python 3.1 语法。

Python特点

1.易于学习：Python有相对较少的关键字，结构简单，和一个明确定义的语法，学习起来更加简单。

2.易于阅读：Python代码定义的更清晰。

3.易于维护：Python的成功在于它的源代码是相当容易维护的。

4.一个广泛的标准库：Python的最大的优势之一是丰富的库，跨平台的，在UNIX，Windows和Macintosh兼容很好。

5.互动模式：互动模式的支持，您可以从终端输入执行代码并获得结果的语言，互动的测试和调试代码片断。

6.可移植：基于其开放源代码的特性，Python已经被移植（也就是使其工作）到许多平台。

7.可扩展：如果你需要一段运行很快的关键代码，或者是想要编写一些不愿开放的算法，你可以使用C或C++完成那部分程序，然后从你的Python程序中调用。

8.数据库：Python提供所有主要的商业数据库的接口。

9.GUI编程：Python支持GUI可以创建和移植到许多系统调用。

10.可嵌入: 你可以将Python嵌入到C/C++程序，让你的程序的用户获得"脚本化"的能力。

在线资源

https://www.python.org/

https://www.w3schools.com/python/

https://www.w3school.com.cn/python/index.asp

https://www.runoob.com/python/python-tutorial.html

https://developers.google.com/edu/python

2.实验平台工具--Jupyter Notebook简介

什么是Jupyter Notebook？

1. 简介

Jupyter Notebook是基于网页的用于交互计算的应用程序。其可被应用于全过程计算：开发、文档编写、运行代码和展示结果。——Jupyter Notebook官方介绍

简而言之，Jupyter Notebook是以网页的形式打开，可以在网页页面中直接编写代码和运行代码，代码的运行结果也会直接在代码块下显示。如在编程过程中需要编写说明文档，可在同一个页面中直接编写，便于作及时的说明和解释。

2. 组成部分

① 网页应用

网页应用即基于网页形式的、结合了编写说明文档、数学公式、交互计算和其他富媒体形式的工具。简言之，网页应用是可以实现各种功能的工具。

② 文档

即Jupyter Notebook中所有交互计算、编写说明文档、数学公式、图片以及其他富媒体形式的输入和输出，都是以文档的形式体现的。

这些文档是保存为后缀名为.ipynb的JSON格式文件，不仅便于版本控制，也方便与他人共享。

此外，文档还可以导出为：HTML、LaTeX、PDF等格式。

3. Jupyter Notebook的主要特点

可选择语言：支持超过40种编程语言，包括Python、R、Julia、Scala等。

分享笔记本：可以使用电子邮件、Dropbox、GitHub和Jupyter Notebook Viewer与他人共享。

交互式输出：代码可以生成丰富的交互式输出，包括HTML、图像、视频、LaTeX等等。

大数据整合：通过Python、R、Scala编程语言使用Apache Spark等大数据框架工具。支持使用pandas、scikit-learn、ggplot2、TensorFlow来探索同一份数据。

主面板(Notebook Dashboard)

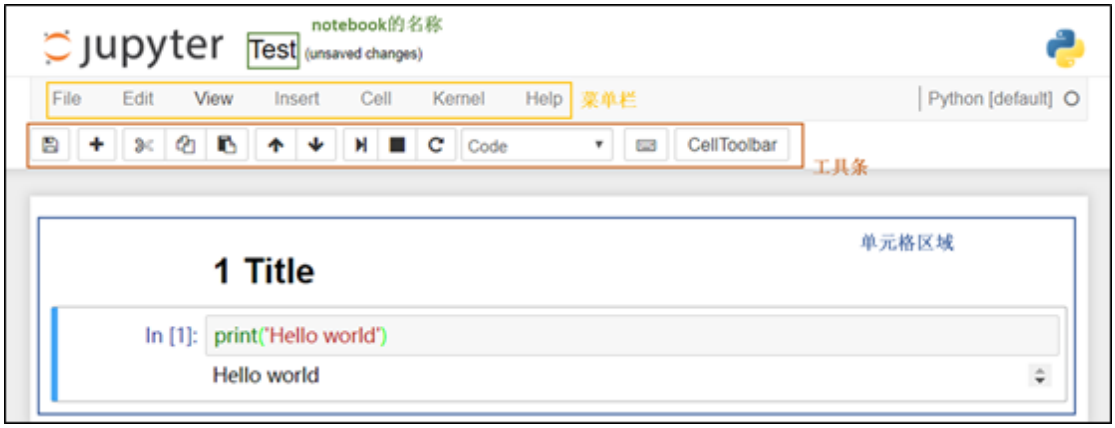
打开Notebook，可以看到主面板。在菜单栏中有Files、Running、Clusters、Conda四个选项。用到最多的是Files，我们可以在这里完成notebook的新建、重命名、复制等操作。具体功能如下：

在Running中，可以看到正在运行的notebook，我们可以选择结束正在运行的程序。

编辑界面(Notebook Editor)

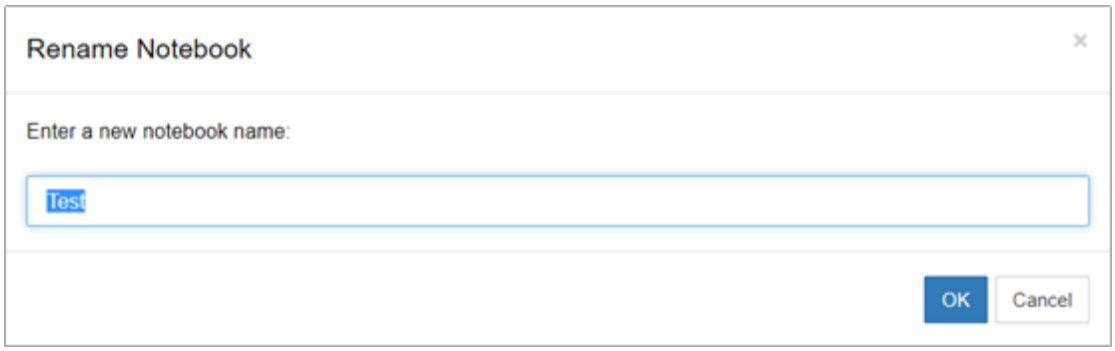
一个notebook的编辑界面主要由四部分组成：名称、菜单栏、工具条以及单元

(Cell)，如下图所示：

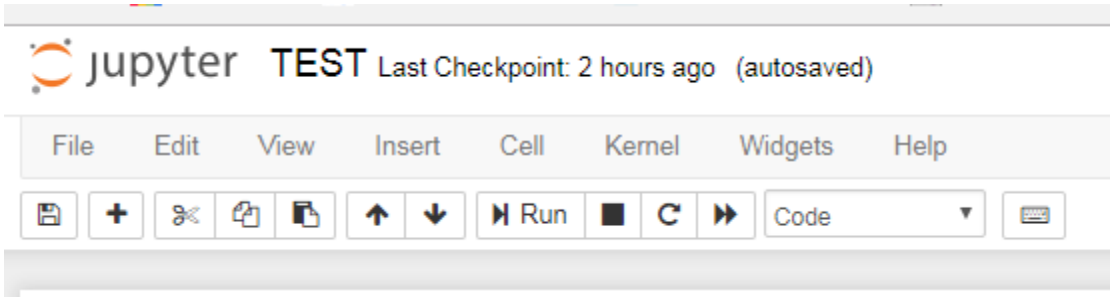


名称

在这里，我们可以修改notebook的名字，直接点击当前名称，弹出对话框进行修改：



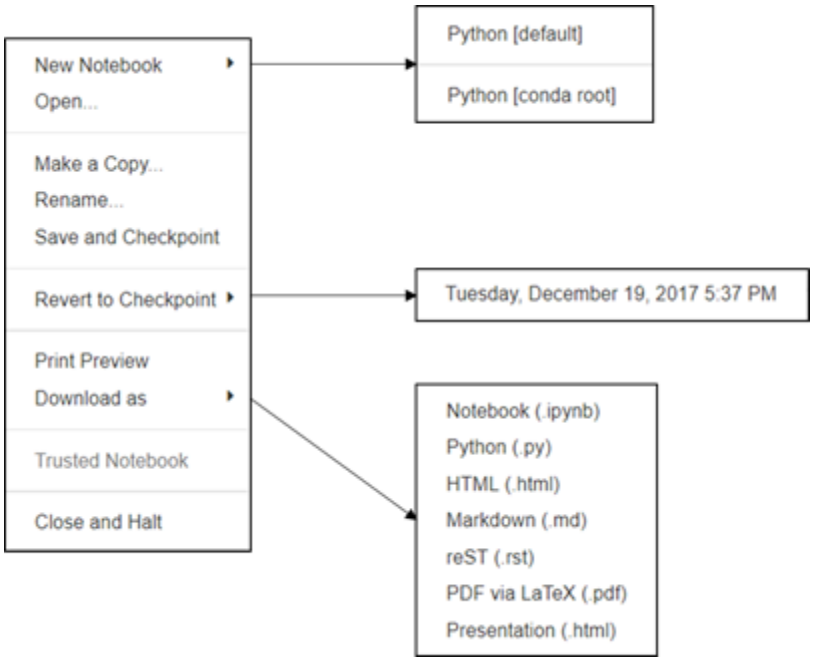
修改后：



菜单栏

菜单栏中有File、Edit、View、Insert、Cell、Kernel、Help等功能，下面逐一介绍。

File中的按钮选项如下图所示：



选项 功能

New Notebook 新建一个notebook

Open… 在新的页面中打开主面板

Make a Copy… 复制当前notebook生成一个新的notebook

Rename… notebook重命名

Save and Checkpoint 将当前notebook状态存为一个Checkpoint

Revert to Checkpoint 恢复到此前存过的Checkpoint

Print Preview 打印预览

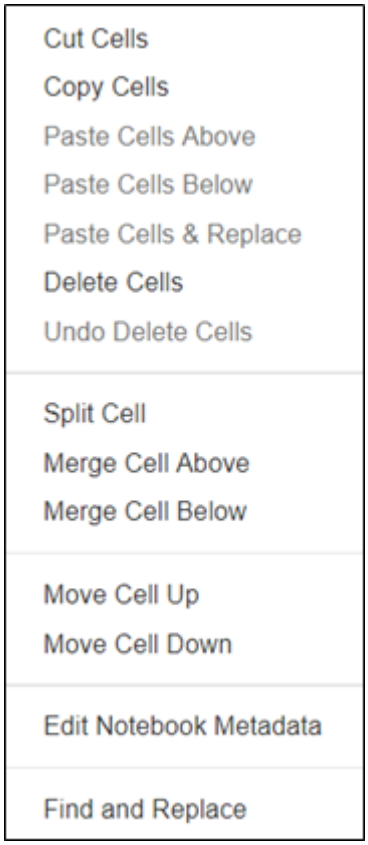
Download as 下载notebook存为某种类型的文件

Close and Halt 停止运行并退出该notebook

在这里，为notebook保存状态是比较重要的，在紧急关闭时防止数据丢失。尽管存档只有1个。另一个Download as也是个重点，

Edit

Edit中的按钮选项如下图所示：



选项 功能

Cut Cells 剪切单元

Copy Cells 复制单元

Paste Cells Above 在当前单元上方粘贴上复制的单元

Paste Cells Below 在当前单元下方粘贴上复制的单元

Paste Cells & Replace 替换当前的单元为复制的单元

Delete Cells 删除单元

Undo Delete Cells 撤回删除操作

Split Cell 从鼠标位置处拆分当前单元为两个单元

Merge Cell Above 当前单元和上方单元合并

Merge Cell Below 当前单元和下方单元合并

Move Cell Up 将当前单元上移一层

Move Cell Down 将当前单元下移一层

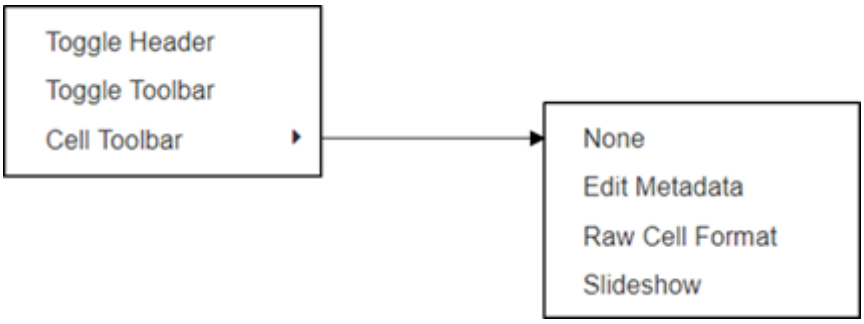
Edit Notebook Metadata 编辑notebook的元数据

Find and Replace 查找替换，支持多种替换方式：区分大小写、使用JavaScript

正则表达式、在选中单元或全部单元中替换

View

View中的按钮选项如下图所示：



选项 功能

Toggle Header 隐藏/显示Jupyter notebook的logo和名称

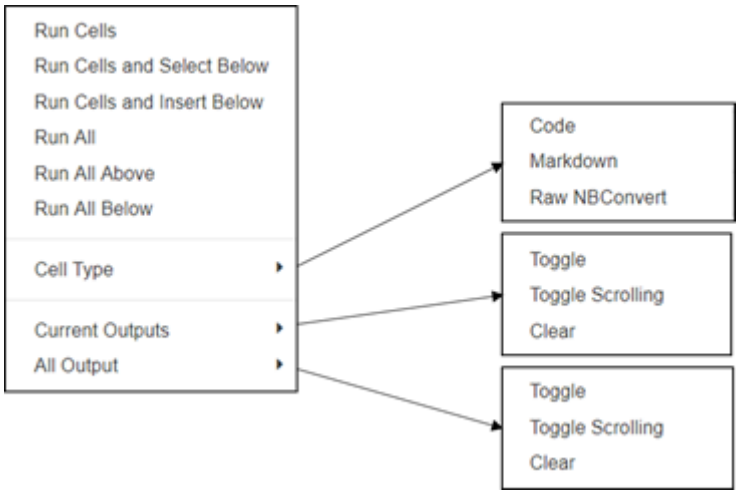
Toggle Toolbar隐藏/显示Jupyter notebook的工具条

Cell Toolbar更改单元展示式样

Insert

功能：在当前单元上方/下方插入新的单元。

Cell



选项 功能

Run Cells运行单元内代码

Run Cells and Select Below运行单元内代码并将光标移动到下一单元

Run Cells and Insert Below运行单元内代码并在下方新建一单元

Run All运行所有单元内的代码

Run All Above运行该单元（不含）上方所有单元内的代码

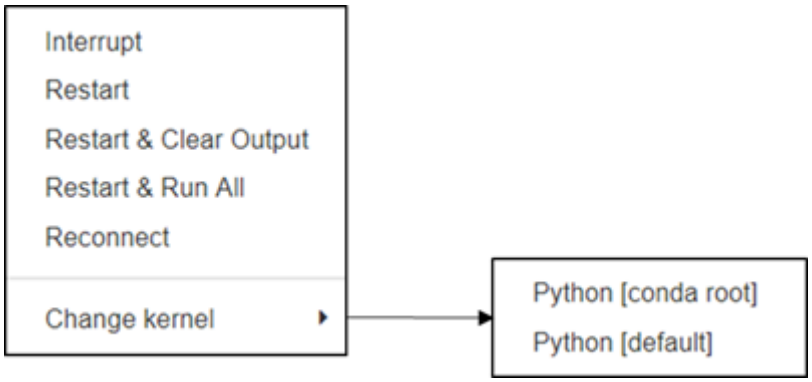
Run All Below运行该单元（含）下方所有单元内的代码

Cell Type选择单元内容的性质

Current Outputs对当前单元的输出结果进行隐藏/显示/滚动/清除

All Output对所有单元的输出结果进行隐藏/显示/滚动/清除

Kernel



选项 功能

Interrupt中断与内核连接（等同于ctrl-c）

Restart重启内核

Restart & Clear Output重启内核并清空现有输出结果

Restart & Run All重启内核并重新运行notebook中的所有代码

Reconnect重新连接到内核

Change kernel切换内核

工具条

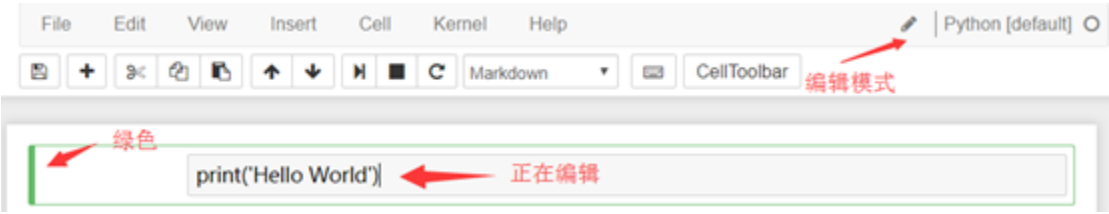
工具条中的功能基本上在菜单中都可以实现，这里是为了能更快捷的操作，将一些常用按钮放了出来。下图是对各按钮的解释。

单元(Cell)

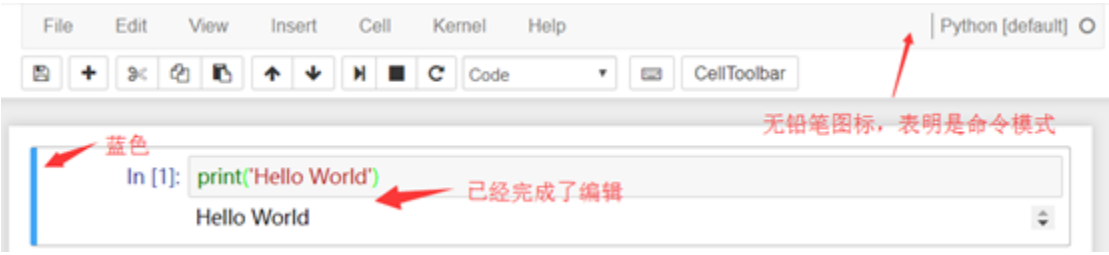
在单元中我们可以编辑文字、编写代码、绘制图片等等。

两种模式与快捷键

对于Notebook中的单元，有两种模式：命令模式(Command Mode)与编辑模式(Edit Mode)，在不同模式下我们可以进行不同的操作。



如上图，在编辑模式(Edit Mode)下，右上角出现一只铅笔的图标，单元左侧边框线呈现出绿色，点Esc键或运行单元格(ctrl-enter)切换回命令模式。



在命令模式(Command Mode)下，铅笔图标消失，单元左侧边框线呈现蓝色，按Enter键或者双击cell变为编辑状态。

Cell的四种功能

Cell有四种功能：Code、Markdown、Raw NBConvert、Heading，这四种功能可以互相切换。Code用于写代码，Markdown用于文本编辑，Raw NBConvert中的文字或代码等都不会被运行，Heading是用于设置标题的，这个功能已经包含在Markdown中了。四种功能的切换可以使用快捷键或者工具条。

Code用于写代码，三类提示符及含义如下：

提示符	含义
In[ ]	程序未运行
In[num]	程序运行后
In[*]	程序正在运行

Python基础、Matplotlib基础、Numpy基础等更多说明参见在线实验平台

实践二 数据预处理基础—以图像为例

（一） 实践目的

掌握Python语言的进行图像数据的读取与预处理；掌握基本的图像处理方法，学会图像的显示、缩放、旋转、直方图等基本操作。学会图像中值滤波和均值滤波，掌握图像锐化、模糊、去噪的基本方法。

（二） 实践环境

计算机、希冀平台

（三） 实践学时

4学时

（四） 实践内容与步骤

实验一：图像增强

图像增强主要解决由于图像的灰度级范围较小造成的对比度较低的问题，目的就是将输出图像的灰度级放大到指定的程度，使得图像中的细节看起来增加清晰。

本实验的要点包括：

掌握基本的灰度变换函数：图像反转、对数变换、线性变换

掌握直方图的基本操作：直方图均衡化

1. 图像反转

图像颜色的反转一般分为两种：一种是灰度图片的颜色反转，另一种是彩色图像的颜色反转。

灰度图像每个像素点只有一个像素值来表示，色彩范围在0-255之间，反转方法255-当前像素值。

首先需要安装OpenCV：

```
%matplotlib inline
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

将原图转换为灰度图片：

```
img = cv2.imread('./lena.jpg', 1)
height, width, deep = img.shape
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

plt.imshow(gray, cmap='gray')
plt.show()
```

反转图片中所有的像素值：

```
dst = np.zeros((height,width,1), np.uint8)
for i in range(0, height):
    for j in range(0, width):
        grayPixel = gray[i, j]
        dst[i, j] = 255-grayPixel
```

将反转后的图像保存再显示出来，可以看到灰度颜色已经反转：

```
cv2.imwrite("./lena_changed.jpg", dst)
dst = cv2.imread('./lena_changed.jpg', 1)
plt.imshow(dst)
plt.show()
```

2.彩色图像颜色反转

彩色图像的每个像素点由RGB三个元素组成，所以反转的时候需要用255分别减去b,g,r三个值。

重新读取图像并进行颜色反转：

```
img = cv2.imread('./lena.jpg', 1)
height, width, deep = img.shape
#  彩色图像颜色反转  NewR = 255-R
dst = np.zeros((height, width, deep), np.uint8)
for i in range(0, height):
    for j in range(0,width):
        (b, g, r) = img[i, j]
        dst[i, j] = (255-b,255-g,255-r)
```

将反转后的图像保存再显示出来，可以看到彩色图像颜色已经反转：

```
dst2 = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
plt.imshow(dst2)
plt.show()
```

3.对数变换

对数变换在图像处理中通常有以下作用：

因为对数曲线在像素值较低的区域斜率较大，像素值较高的区域斜率比较低，所以图像经过对数变换之后，在较暗的区域对比度将得到提升，因而能增

强图像暗部的细节。

本节中所使用的样图如下：

先导入所需的程序包：

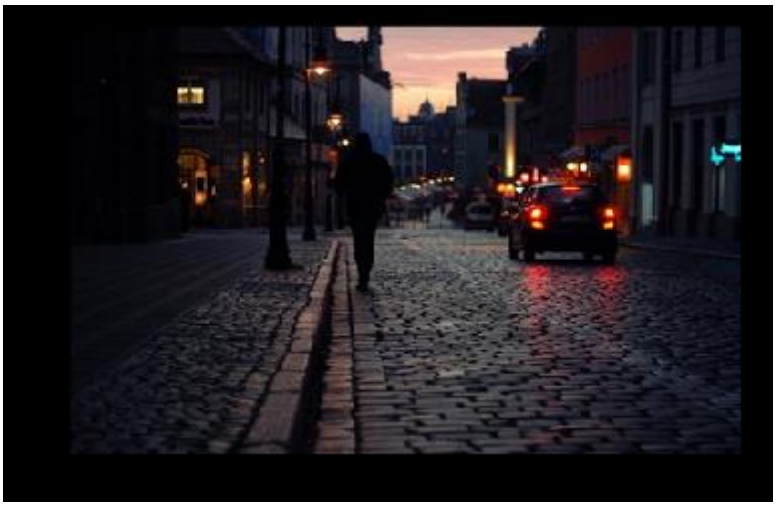
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

定义对数变化函数：

```
def log(c, img):
    output = c * np.log(1.0 + img)
    output = np.uint8(output + 0.5)
    return output
```

读取原始图像：

```
img = cv2.imread('./street.jpg')
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img2)
plt.show()
```



对图像进行对数变换：

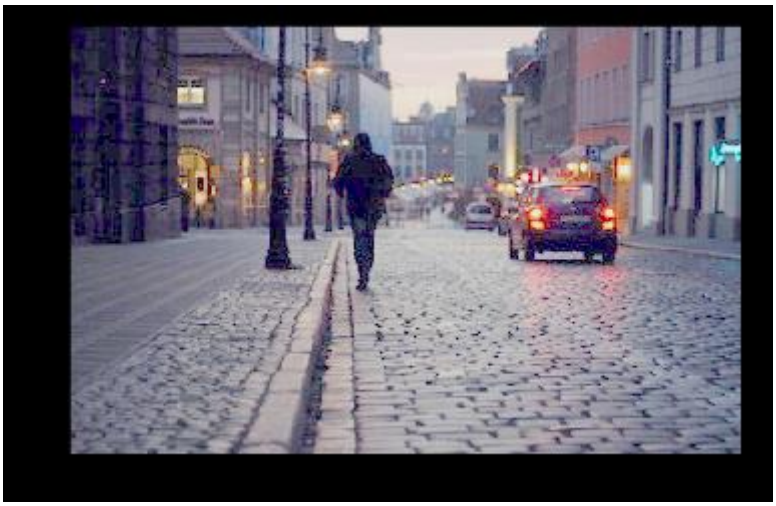
output = log(42, img)

显示结果图像：

output2 = cv2.cvtColor(output, cv2.COLOR_BGR2RGB)

plt.imshow(output2)

plt.show()



可以观察到，对数变换效果非常明显，且对于整体对比度偏低并且灰度值偏低的图像增强效果较好。

4. 灰度直方图

在讲解线性变换的方法之前先来认识一下灰度直方图，灰度直方图是图像灰度级的函数，用来描述每个灰度级在图像矩阵中的像素个数或者占有率。接下来使用程序实现直方图：

首先导入所需的程序包：

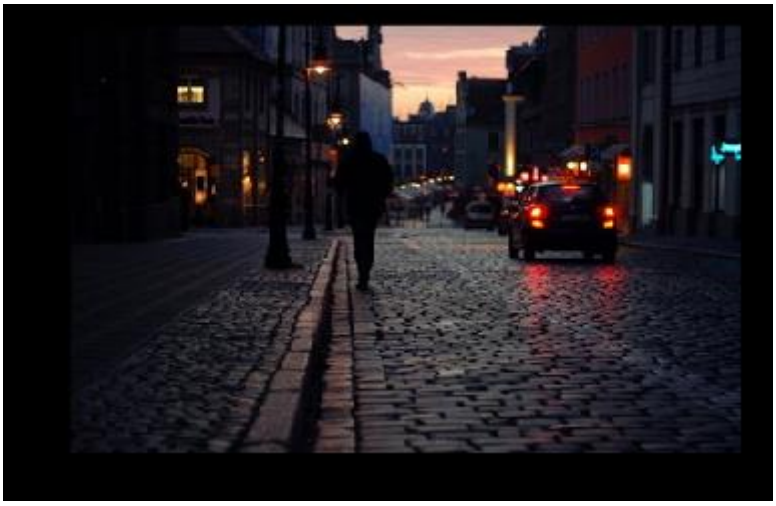
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

定义计算灰度直方图的函数：

```
def calcGrayHist(I):
    # 计算灰度直方图
    h, w = I.shape[:2]
    grayHist = np.zeros([256], np.uint64)
    for i in range(h):
        for j in range(w):
            grayHist[I[i][j]] += 1
    return grayHist
```

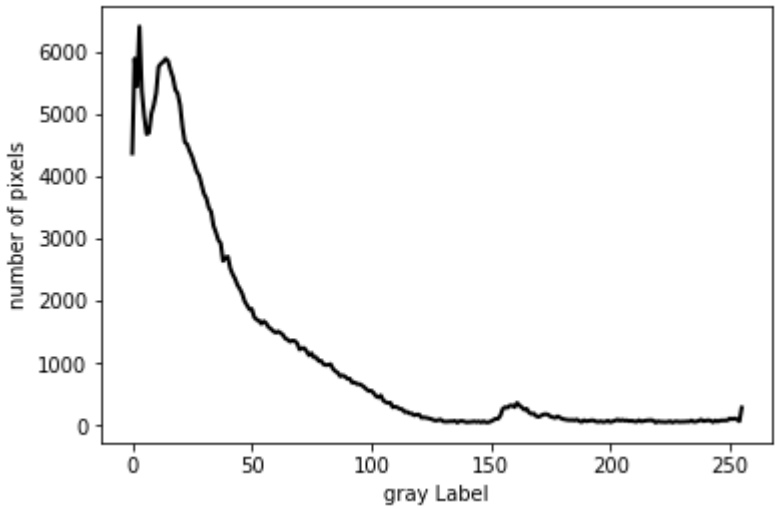
读取一张图片：

```
img = cv2.imread('./street.jpg')
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img2)
plt.show()
```



```
grayHist = calcGrayHist(img)
x = np.arange(256)
plt.plot(x, grayHist, 'r', linewidth=2, c='black')
plt.xlabel("gray Label")
```

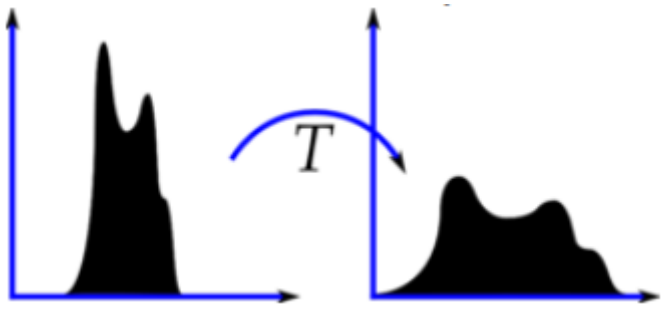
```
plt.ylabel("number of pixels")
plt.show()
```



图像的对比度是通过灰度级范围来度量的，而灰度级范围可通过观察灰度直方图得到，灰度级范围越大代表对比度越高；反之对比度越低，低对比度的图像在视觉上给人的感觉是看起来不够清晰，所以通过算法调整图像的灰度值，从而调整图像的对比度是有必要的。最简单的一种对比度增强的方法是通过灰度值的线性变换实现的。

5.直方图均衡化

直方图均衡化是图像处理领域中利用图像直方图对对比度进行调整的方法。这种方法通常用来增加许多图像的全局对比度，尤其是当图像的有用数据的对比度相当接近的时候。



5.1 灰度图像直方图均衡化

OpenCV 中的直方图均衡化函数为 cv2.equalizeHist()。这个函数的输入图片仅仅是一副灰度图像，输出结果是直方图均衡化之后的图像。

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('lena.jpg',0)

equ = cv2.equalizeHist(img)

# 并排放置原图和效果图
res = np.hstack((img,equ))
res2 = cv2.cvtColor(res, cv2.COLOR_BGR2RGB)
plt.imshow(res2)
plt.show()
```

5.2 彩色直方图均衡化

若想对彩色直方图做均衡化，则要对颜色通道进行分解后分别均衡化，最后合成均衡化后的的彩色图像：

```
import cv2
import numpy as np
img = cv2.imread('lena.jpg',1)

#通道分解
(b,g,r) = cv2.split(img)
```

```
bH = cv2.equalizeHist(b)
gH = cv2.equalizeHist(g)
rH = cv2.equalizeHist(r)
result = cv2.merge((bH,gH,rH),)#通道合成
#  并排放置原图和效果图
res = np.hstack((img,result))
res2 = cv2.cvtColor(res, cv2.COLOR_BGR2RGB)
plt.imshow(res2)
plt.show()
```

通过对比可以看到，均衡化后的图像相比于原图，对比度有很大的提升，图像中的元素变得更加清晰。

实验二：图像平滑

图像平滑从信号处理的角度看就是去除其中的高频信息，保留低频信息。因此我们可以对图像实施低通滤波。低通滤波可以去除图像中的噪音，模糊图像(噪音是图像中变化比较大的区域，也就是高频信息)。而高通滤波能够提取图像的边缘(边缘也是高频信息集中的区域)。

根据滤波器的不同又可以分为均值滤波，高斯加权滤波，中值滤波， 双边滤波。

本实验的要点包括：掌握基本图像平滑方法

1. 均值滤波

平均滤波是将一个m*n(m, n为奇数)大小的kernel放在图像上，中间像素的值用

kernel覆盖区域的像素平均值替代。平均滤波对高斯噪声的表现比较好。

OpenCV提供了cv2.blur()函数用于处理平均滤波，使用方法如下：

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import cv2

img = cv2.imread('./lena.jpg')
output = cv2.blur(img,(10,10))
```

原图和效果图比较如下：

```
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img2)
plt.show()
output2 = cv2.cvtColor(output, cv2.COLOR_BGR2RGB)
plt.imshow(output2)
plt.show()
```

可以明显的看到，图像相比原图变得模糊。

2. 高斯滤波

即假设某一位置的像素和其邻域像素符合高斯分布。具体的说的话,就是每

一位置的像素的权重符合高斯分布.这样的话,给定一个高斯分布,及高斯核的大小,则可以计算出周边n个像素的权重值。

OpenCV中获取高斯核心的函数为cv2.getGaussianKernel,但是这个获取的一维的高斯核.对图像来说,以3 x 3邻域而言,我们应该得到一个3 x 3的权重矩阵.可以如下得到:

```
kernal_x = cv2.getGaussianKernel(3,-1)
kernal_y = cv2.getGaussianKernel(3,-1)
kernal_filter = np.dot(kernal_x,kernal_y.T)
print(kernal_filter)
[[ 0.0625    0.125     0.0625]
 [ 0.125     0.25      0.125 ]
 [ 0.0625    0.125     0.0625]]
```

则中间元素的亮度值经高斯转换后为0.0625 x p(0,0) + 0.125 x p(0,1) + ... + 0.0625 x p(2,2)，可以看到权重矩阵相加等于1。 这里,我们举例用了3 x 3的高斯核,实际上并不限定高斯核一定要是正方形。

接下来查看一下OpenCV提供的高斯滤波函数的定义：

```
GaussianBlur(
    InputArray src,
    OutputArray dst,
    Size ksize,
    double sigmaX,
    double singmaY = 0,
    int borderType = BORDER_DEFAULT
)
```

其参数sigmaX，sigmaY即x，y方向上的高斯分布的标准差。这样就可以求得不同方向上的高斯矩阵，再做矩阵乘法，即得到m x n的权重矩阵。进而求得高斯转换后的图像。

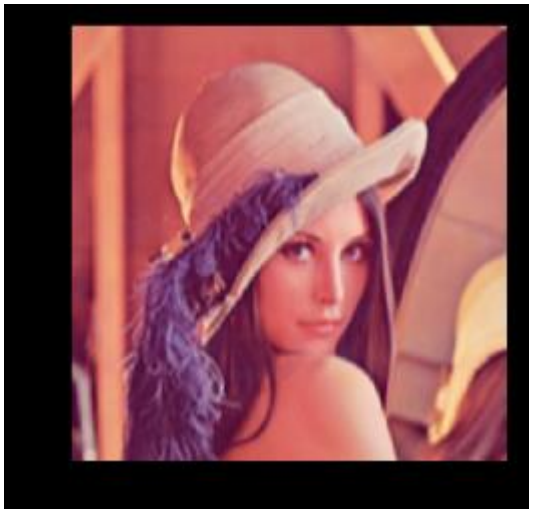
我们知道高斯分布(也叫正态分布)的特点为，标准差越大，分布越分散，标准差越小，分布越集中。所以调大GaussianBlur()中的sigmaX，sigmaY将使得图像中的每个像素更多地参考周边像素，即更为平滑或者说模糊。

```
%matplotlib inline
import cv2
import numpy as np
import matplotlib.pyplot as plt

imgname = "./lena.jpg"
img = cv2.imread(imgname)

img2 = img.copy()
img2 = cv2.GaussianBlur(img,(5,7),1)
img3 = cv2.GaussianBlur(img,(5,7),100)
```

原图和效果图比较如下：

```
output = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
output2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
output3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)
plt.imshow(output)
plt.show()
plt.imshow(output2)
plt.show()
plt.imshow(output3)
plt.show()
```





通过比较可以感受到，高斯滤波效果较为柔和，标准差越大，图像就越模糊。

3. 中值滤波

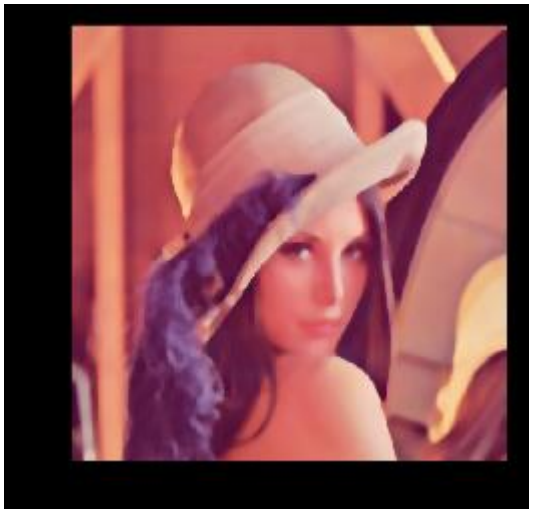
即把像素值变为邻域像素值的中位数。OpenCV提供的中值滤波函数的定义如下：

```
medianBlur(
    InputArray src,
    OutputArray dst,
    int ksize
)
```

注意，ksize的大小必须为奇数。

```
%matplotlib inline
import cv2
import numpy as np
import matplotlib.pyplot as plt

imgname = "./lena.jpg"
img = cv2.imread(imgname)

dst = cv2.medianBlur(img, 1)
dst2 = cv2.medianBlur(img, 11)
```
原图和效果图对比如下：
```
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img2)
plt.show()

output2 = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
plt.imshow(output2)
plt.show()

output3 = cv2.cvtColor(dst2, cv2.COLOR_BGR2RGB)
plt.imshow(output3)
plt.show()
```





通过比较可以发现，ksize的大小越大，图像越模糊，画面呈现出油画感。

实验三：图像锐化

由于收集图像数据的器件或传输数图像的通道的存在一些质量缺陷，文物图像时间久远，或者受一些其他外界因素、动态不稳定抓取图像的影响，使得图像存在模糊和有噪声的情况，从而影响到图像识别工作的开展。这时需要开展图像锐化和边缘检测处理，加强原图像的高频部分，锐化突出图像的边缘细节，改善图像的对比度，使模糊的图像变得更清晰。

图像锐化和边缘提取技术可以消除图像中的噪声，提取图像信息中用来表征图像的一些变量，为图像识别提供基础。通常使用灰度差分法对图像的边缘、轮廓进行处理，将其凸显。

本实验的要点包括：掌握基本图像锐化方法

Prewwitt算子

Prewitt是一种图像边缘检测的微分算子，其原理是利用特定区域内像素灰度值产生的差分实现边缘检测。由于Prewitt算子采用33模板对区域内的像素值进行计算，而Robert算子的模板为22，故Prewitt算子的边缘检测结果在水平方向和垂直方向均比Robert算子更加明显。Prewitt算子适合用来识别噪声较多、灰度渐变的图像，其计算公式如下所示。

$$d_x=\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix}\quad d_y=\begin{bmatrix}-1 & -1 & -1\\ 0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix}$$

在Python中，Prewitt算子的实现过程与Roberts算子比较相似。通过Numpy定义模板，再调用OpenCV的filter2D()函数实现对图像的卷积运算，最终通过convertScaleAbs()和addWeighted()函数实现边缘提取，代码如下所示：

```
# -*- coding: utf-8 -*-
%matplotlib inline
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('./lena.jpg')
lenna_img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#Prewitt算子
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]],dtype=int)
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]],dtype=int)
x = cv2.filter2D(grayImage, cv2.CV_16S, kernelx)
```

```
y = cv2.filter2D(grayImage, cv2.CV_16S, kernely)
#转uint8
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Prewitt = cv2.addWeighted(absX,0.5,absY,0.5,0)

#显示图形
titles = [u'Original Image', u'Prewitt Filter']
images = [lenna_img, Prewitt]
for i in range(2):
    plt.subplot(1,2,i+1)
    plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])
plt.show()
```

Sobel算子

Sobel算子是一种用于边缘检测的离散微分算子，它结合了高斯平滑和微分求导。该算子用于计算图像明暗程度近似值，根据图像边缘旁边明暗程度把该区域内超过某个数的特定点记为边缘。Sobel算子在Prewitt算子的基础上增加了权重的概念，认为相邻点的距离远近对当前像素点的影响是不同的，距离越近的像素点对应当前像素的影响越大，从而实现图像锐化并突出边缘轮廓。

Sobel算子的边缘定位更准确，常用于噪声较多、灰度渐变的图像。其算法模板如公式所示，其中dx表示水平方向，dy表示垂直方向。

$$d_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad d_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Sobel算子根据像素点上下、左右邻点灰度加权差，在边缘处达到极值这一现象检测边缘。对噪声具有平滑作用，提供较为精确的边缘方向信息。因为Sobel算子结合了高斯平滑和微分求导（分化），因此结果会具有更多的抗噪性，当对精度要求不是很高时，Sobel算子是一种较为常用的边缘检测方法。

dst = Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])

src表示输入图像
dst表示输出的边缘图，其大小和通道数与输入图像相同
ddepth表示目标图像所需的深度，针对不同的输入图像，输出目标图像有不同的深度
dx表示x方向上的差分阶数，取值1或 0
dy表示y方向上的差分阶数，取值1或0
ksize表示Sobel算子的大小，其值必须是正数和奇数
scale表示缩放导数的比例常数，默认情况下没有伸缩系数
delta表示将结果存入目标图像之前，添加到结果中的可选增量值
borderType表示边框模式，更多详细信息查阅BorderTypes
注意，在进行Sobel算子处理之后，还需要调用convertScaleAbs()函数计算绝对值，并将图像转换为8位图进行显示。其算法原型如下：

dst = convertScaleAbs(src[, dst[, alpha[, beta]]])

src表示原数组
dst表示输出数组，深度为8位

alpha表示比例因子

beta表示原数组元素按比例缩放后添加的值

Sobel算子的实现代码如下所示：

```
# -*- coding: utf-8 -*-
%matplotlib inline
import cv2
import numpy as np
import matplotlib.pyplot as plt

#读取图像
img = cv2.imread('./lena.jpg')
lenna_img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

#灰度化处理图像
grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#Sobel算子
x = cv2.Sobel(grayImage, cv2.CV_16S, 1, 0) #对x求一阶导
y = cv2.Sobel(grayImage, cv2.CV_16S, 0, 1) #对y求一阶导
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#显示图形
titles = [u'Original Image', u'Sobel Filter']
images = [lenna_img, Sobel]
for i in range(2):
    plt.subplot(1,2,i+1)
    plt.imshow(images[i], 'gray')
```

```
        plt.title(titles[i])
        plt.xticks([])
        plt.yticks([])
plt.show()
```

本次实验作业：

针对这幅图像：

利用自己编写的均值滤波模板，实现均值滤波；

利用自己编写的Sobel算子，实现图像锐化。

要求：自己实现相应的函数

提示：按照图像的行和列，进行两层循环嵌套，在每个像素点位置，利用相应的模板或算子，进行相应的卷积计算

实践三 TensorFlow深度学习基本原理入门

（一） 实践目的

掌握TensorFlow基础概念；学习深度学习基本原理、Logistic回归、浅层神经网络、多层神经网络等概念。

（二） 实践环境

计算机、希冀平台

（三） 实践学时

4学时

（四） 实践内容与步骤

什么是TensorFlow?

TensorFlow是Google Brain的第二代机器学习系统，已经开源。TensorFlow受到了AI开发社区的广泛欢迎，是Github上最受欢迎的深度学习框架之一，也是整个社区上fork最多的项目。目前，TensorFlow已经被下载了超过790万次。Tensorflow是一个采用数据流图（data flow graphs），用于数值计算的开源软件库，常被应用于各种感知、语言理解、语音识别、图像识别等多项机器深度学习领域。

TensorFlow使用计算图表来执行其所有的计算。计算被表示为tf.Graph对象的一个实例，而其中的数据被表示为tf.Tensor对象，并使用tf.Operation对象对这样的张量对象进行操作。然后再使用tf.Session对象的会话中执行该图表。

图中节点（Nodes）一般表示施加的数学操作，或者表示数据输入（feed in）的起点/输出（push out）的终点，或者是读取/写入持久变量（persistent variable）的终点

图中线/边（edges）则表示在节点间相互联系的多维数据组，即张量（tensor）。

Tensorflow特点

1、真正的可移植性

引入各种计算设备的支持包括CPU/GPU/TPU，以及能够很好地运行在移动端，如安卓设备、ios、树莓派等等

2、多语言支持

Tensorflow 有一个合理的c++使用界面，也有一个易用的python使用界面来构建和执行你的graphs，你可以直接写python/c++程序

3、高度的灵活性与效率

TensorFlow是一个采用数据流图（data flow graphs），用于数值计算的开源软件库能够灵活进行组装图，执行图。随着开发的进展，Tensorflow的效率不断在提高

4、支持TensorFlow 由谷歌提供支持，谷歌投入了大量精力开发TensorFlow，它希望 TensorFlow 成为机器学习研究人员和开发人员的通用语言。

TensorFlow的核心程序由2个部分组成：

构建计算图：Building the computational graph

运行计算图：Running the computational graph

一些TensorFlow的基础概念：

张量：TensorFlow核心的基本数据单元。 张量是一个多维数组。

图：默认已经注册，一组表示 tf.Operation计算单位的对象和tf.Tensor表示操作之间流动的数据单元的对象。

计算图：由一系列的tensorflow的操作组成，并且这些操作编配成计算图的节点。

会话：TensorFlow的运行模型，通过会话来存储执行计算图。

变量：当训练模型时，用变量来存储和更新参数。建模时它们需要被明确地初始化，当创建一个变量时，需要将一个张量作为初始值传入构造函数Variable()。变量也是一种OP，是一种特殊的张量，能够进行存储持久化，它的值就是张量。是在模型训练中，允许修改的量。相同的输入，通过修改变量得

到不同输出值。

占位符：在计算图中，能够接受额外输入，通常情况下，提供的值晚于定义。意义在于，在程序执行的时候,不确定输入的是什么，提前“占个坑”。

传递字典：参数传递具体的值到run方法的占位符来进行多个输入。

名字作用域：管理变量的集合，调用范围内创建的所有计算图形元素都有名称。

TensorFlow，张量的流动，我们可以把TensorFlow当成是一个工厂，名字叫做“张量工厂”。

一个工厂通过设备，生产原料，能源等，生产出产品。那么在我们的“张量工厂”中，我们的生产原料就是“张量”，我们的设备就是“操作op”，合理组装这些设备就成为了“计算图”，这些生产原料和设备，构成了我们的工厂即“图”。那么有了生产原料和工厂，之后我们就可以进行生产，只需要老板动一动手指按下启动按钮，那么整个工厂，就会运行起来。

张量工厂的组成：

生产原料：张量tensor。

生产设备：操作operation(op)，接收仓库的tensor，清洗数据tensor，处理数据tensor 。

工厂：图graph，由“生产原料”和“生产设备”组成，整个程序。

流水线：计算图computational graph，有一系列的设备op组成，加工仓库的原料和中间产品tensor

董事会：会话session，董事会在会议上决定共产是否启动启动（run）工厂graph生产产品tensor。

仓库：变量variable，仓库内的原料tensor或者生产出的中间产品tensor，但只能是一种类型的（维度），不能原来存放的是HCl，现在存放NaCO3（因为维度不同），仓库中的tensor在进入流水线之前，必须先拆掉包装（必须先初始化），然后才能被设备使用。

通用原料容器：占位符placeholders，同一条生产线，今天我可能生产花生

酱，明天生产草莓酱，我不确定到底会生产什么，所以我单独设置这样的原料容器，提前占个位置，这样无论生产什么我都不需要专门使用一个容器，只需要防止一个通用容器。

仓库大门：传递字典feed_dict，通过这个大门，我们向仓库中存入tensor。

生产车间：名字作用域name_scope，就像是，清洗车间、组装车间等，每个车间都包含生产设备op。

基本使用--使用 TensorFlow, 你必须明白 TensorFlow:

使用图 (graph) 来表示计算任务.在被称之为 会话 (Session) 的上下文 (context) 中执行图.

使用 tensor 表示数据.

通过 变量 (Variable) 维护状态.

使用 feed 和 fetch 可以为任意的操作(arbitrary operation) 赋值或者从其中获取数据.

综述

TensorFlow 是一个编程系统, 使用图来表示计算任务. 图中的节点被称之为 op (operation 的缩写). 一个 op 获得 0 个或多个 Tensor, 执行计算, 产生 0 个或多个 Tensor. 每个 Tensor 是一个类型化的多维数组. 例如, 你可以将一小组图像集表示为一个四维浮点数数组, 这四个维度分别是 [batch, height, width, channels].

一个 TensorFlow 图描述了计算的过程. 为了进行计算, 图必须在 会话 里被启动. 会话 将图的 op 分发到诸如 CPU 或 GPU 之类的 设备 上, 同时提供执行 op 的方法. 这些方法执行后, 将产生的 tensor 返回. 在 Python 语言中, 返回的 tensor 是 numpy ndarray 对象; 在 C 和 C++ 语言中, 返回的 tensor 是 tensorflow::Tensor 实例.

计算图

TensorFlow 程序通常被组织成一个构建阶段和一个执行阶段. 在构建阶段,

op 的执行步骤 被描述成一个图. 在执行阶段, 使用会话执行执行图中的 op.

例如, 通常在构建阶段创建一个图来表示和训练神经网络, 然后在执行阶段反复执行图中的训练 op.

TensorFlow 支持 C, C++, Python 编程语言. 目前, TensorFlow 的 Python 库更加易用, 它提供了大量的辅助函数来简化构建图的工作, 这些函数尚未被 C 和 C++ 库支持.

三种语言的会话库 (session libraries) 是一致的.

构建图

构建图的第一步, 是创建源 op (source op). 源 op 不需要任何输入, 例如常量 (Constant). 源 op 的输出被传递给其它 op 做运算.

Python 库中, op 构造器的返回值代表被构造出的 op 的输出, 这些返回值可以传递给其它 op 构造器作为输入.

TensorFlow Python 库有一个默认图 (default graph), op 构造器可以为其增加节点. 这个默认图对 许多程序来说已经足够用了.

```
import tensorflow as tf

# 创建一个常量 op, 产生一个 1x2 矩阵. 这个 op 被作为一个节点
# 加到默认图中.
#
# 构造器的返回值代表该常量 op 的返回值.
matrix1 = tf.constant([[3., 3.]])

# 创建另外一个常量 op, 产生一个 2x1 矩阵.
matrix2 = tf.constant([[2.],[2.]])

# 创建一个矩阵乘法 matmul op , 把 'matrix1' 和 'matrix2' 作为输入.
```

```
# 返回值 'product' 代表矩阵乘法的结果.
product = tf.matmul(matrix1, matrix2)
```

默认图现在有三个节点，两个 constant() op，和一个matmul() op. 为了真正进行矩阵相乘运算，并得到矩阵乘法的 结果，你必须在会话里启动这个图.

在一个会话中启动图

构造阶段完成后，才能启动图. 启动图的第一步是创建一个 Session 对象，如果无任何创建参数，会话构造器将启动默认图.

```
# 启动默认图.
sess = tf.Session()

# 调用 sess 的 'run()' 方法来执行矩阵乘法 op，传入 'product' 作为该方法的参数.
# 上面提到，'product' 代表了矩阵乘法 op 的输出，传入它是向方法表明，我们希望取回
# 矩阵乘法 op 的输出.
#
# 整个执行过程是自动化的，会话负责传递 op 所需的全部输入. op 通常是并发执行的.
#
# 函数调用 'run(product)' 触发了图中三个 op (两个常量 op 和一个矩阵乘法 op) 的执行.
#
# 返回值 'result' 是一个 numpy `ndarray` 对象.
result = sess.run(product)
print (result)
# ==> [[ 12.]]
```

```
# 任务完成, 关闭会话.
sess.close()
```

Session 对象在使用完后需要关闭以释放资源. 除了显式调用 close 外, 也可以使用 "with" 代码块 来自动完成关闭动作.

```
with tf.Session() as sess: result = sess.run([product]) print result
```

在实现上, TensorFlow 将图形定义转换成分布式执行的操作, 以充分利用可用的计算资源(如 CPU 或 GPU). 一般你不需要显式指定使用 CPU 还是 GPU, TensorFlow 能自动检测. 如果检测到 GPU, TensorFlow 会尽可能地利用找到的第一个 GPU 来执行操作.

如果机器上有超过一个可用的 GPU, 除第一个外的其它 GPU 默认是不参与计算的. 为了让 TensorFlow 使用这些 GPU, 你必须将 op 明确指派给它们执行. with...Device 语句用来指派特定的 CPU 或 GPU 执行操作:

```
with tf.Session() as sess:
  with tf.device("/gpu:1"):
    matrix1 = tf.constant([[3., 3.]])
    matrix2 = tf.constant([[2.],[2.]])
    product = tf.matmul(matrix1, matrix2)
    ...
```

设备用字符串进行标识. 目前支持的设备包括:

"/cpu:0": 机器的 CPU. "/gpu:0": 机器的第一个 GPU, 如果有的话. "/gpu:1": 机器的第二个 GPU, 以此类推.

交互式使用

文档中的 Python 示例使用一个会话 Session 来 启动图, 并调用 Session.run() 方法执行操作.

为了便于使用诸如 IPython 之类的 Python 交互环境, 可以使用 InteractiveSession 代替 Session 类, 使用 Tensor.eval() 和 Operation.run() 方法代替 Session.run(). 这样可以避免使用一个变量来持有会话.

```
# 进入一个交互式 TensorFlow 会话.
import tensorflow as tf
sess = tf.InteractiveSession()

x = tf.Variable([1.0, 2.0])
a = tf.constant([3.0, 3.0])

# 使用初始化器 initializer op 的 run() 方法初始化 'x'
x.initializer.run()

# 增加一个减法 sub op, 从 'x' 减去 'a'. 运行减法 op, 输出结果
sub = tf.subtract(x, a)
print (sub.eval())
# ==> [-2. -1.]
sess.close()
```

Tensor

TensorFlow 程序使用 tensor 数据结构来代表所有的数据, 计算图中, 操作间传递的数据都是 tensor. 你可以把 TensorFlow tensor 看作是一个 n 维的数组或列表. 一个 tensor 包含一个静态类型 rank, 和 一个 shape.

变量

Variables for more details. 变量维护图执行过程中的状态信息. 如何使用变量实现一个简单的计数器.

```
# 创建一个变量, 初始化为标量 0.
state = tf.Variable(0, name="counter")

# 创建一个 op, 其作用是使 state 增加 1

one = tf.constant(1)
new_value = tf.add(state, one)
update = tf.assign(state, new_value)

# 启动图后, 变量必须先经过`初始化` (init) op 初始化,
# 首先必须增加一个`初始化` op 到图中.
init_op = tf.initialize_all_variables()

# 启动图, 运行 op
with tf.Session() as sess:
  # 运行 'init' op
  sess.run(init_op)
  # 打印 'state' 的初始值
  print (sess.run(state))
  # 运行 op, 更新 'state', 并打印 'state'
  for _ in range(3):
    sess.run(update)
    print (sess.run(state))
sess.close()
# 输出:
```

```
# 0
# 1
# 2
# 3
```

代码中 assign() 操作是图所描绘的表达式的一部分，正如 add() 操作一样. 所以在调用 run() 执行表达式之前，它并不会真正执行赋值操作.

通常会将一个统计模型中的参数表示为一组变量. 例如，你可以将一个神经网络的权重作为某个变量存储在一个 tensor 中. 在训练过程中，通过重复运行训练图，更新这个 tensor.

Fetch

为了取回操作的输出内容，可以在使用 Session 对象的 run() 调用 执行图时，传入一些 tensor，这些 tensor 会帮助你取回结果. 在之前的例子里，我们只取回了单个节点 state，但是你也可以取回多个 tensor:

```
input1 = tf.constant(3.0)
input2 = tf.constant(2.0)
input3 = tf.constant(5.0)
intermed = tf.add(input2, input3)
mul = tf.multiply(input1, intermed)

with tf.Session() as sess:
    result = sess.run([mul, intermed])
    print (result)
sess.close()
# 输出:
# [array([ 21.], dtype=float32), array([ 7.], dtype=float32)]
```

Feed

上述示例在计算图中引入了 tensor, 以常量或变量的形式存储. TensorFlow 还提供了 feed 机制, 该机制 可以临时替代图中的任意操作中的 tensor 可以对图中任何操作提交补丁, 直接插入一个 tensor.

feed 使用一个 tensor 值临时替换一个操作的输出结果. 你可以提供 feed 数据作为 run() 调用的参数. feed 只在调用它的方法内有效, 方法结束, feed 就会消失. 最常见的用例是将某些特殊的操作指定为 "feed" 操作, 标记的方法是使用 tf.placeholder() 为这些操作创建占位符.

```
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.multiply(input1, input2)

with tf.Session() as sess:
    print (sess.run([output], feed_dict={input1:[7.], input2:[2.]}))
sess.close()
#  输出:
# [array([ 14.], dtype=float32)]
```

更多实验请见在线平台

实践四 TensorFlow深度学习应用

（一） 实践目的

掌握TensorFlow框架下深度学习算法的应用；掌握卷积神经网络概念；了解残差网络、循环神经网络等概念。

（二） 实践环境

计算机、希冀平台

（三） 实践学时

4学时

（四） 实践内容与步骤

1、卷积神经网络的概念

上世纪60年代，Hubel等人通过对猫视觉皮层细胞的研究，提出了感受野这个概念，到80年代，Fukushima在感受野概念的基础之上提出了神经认知机的概念，可以看作是卷积神经网络的第一个实现网络，神经认知机将一个视觉模式分解成许多子模式（特征），然后进入分层递阶式相连的特征平面进行处理，它试图将视觉系统模型化，使其能够在即使物体有位移或轻微变形的时候，也能完成识别。

卷积神经网络是多层感知机（MLP）的变种，由生物学家休博尔和维瑟尔在早期关于猫视觉皮层的研究发展而来，视觉皮层的细胞存在一个复杂的构造，这些细胞对视觉输入空间的子区域非常敏感，称之为感受野。

CNN由纽约大学的Yann Lecun于1998年提出，其本质是一个多层感知机，成功的原因在于其所采用的局部连接和权值共享的方式：一方面减少了权值的数量使得网络易于优化，另一方面降低了模型的复杂度，也就是减小了过拟合的风险。

该优点在网络的输入是图像时表现的更为明显，使得图像可以直接作为网络的输入，避免了传统识别算法中复杂的特征提取和数据重建的过程，在二维图像的处理过程中有很大的优势，如网络能够自行抽取图像的特征包括颜色、

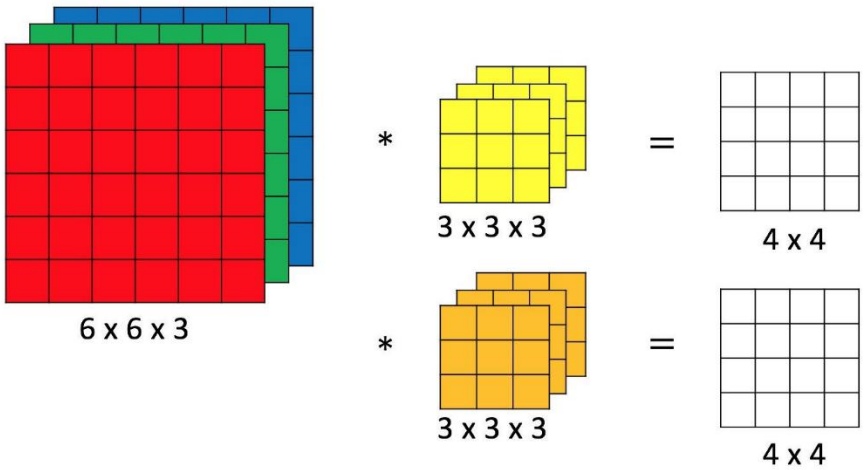
纹理、形状及图像的拓扑结构，在处理二维图像的问题上，特别是识别位移、缩放及其他形式扭曲不变性的应用上具有良好的鲁棒性和运算效率等。

2、网络结构

基础的CNN由 卷积(convolution), 激活(activation), and 池化(pooling)三种结构组成。CNN输出的结果是每幅图像的特定特征空间。当处理图像分类任务时，我们会把CNN输出的特征空间作为全连接层或全连接神经网络(fully connected neural network, FCN)的输入，用全连接层来完成从输入图像到标签集的映射，即分类。当然，整个过程最重要的工作就是如何通过训练数据迭代调整网络权重，也就是后向传播算法。目前主流的卷积神经网络(CNNs)，比如VGG, ResNet都是由简单的CNN调整，组合而来。

卷积

卷积神经网络结构中最重要的部分，过滤器（filter），如图中黄色和橙色的 3×3×3 矩阵所示。可以参考: https://cs231n.github.io/convolutional-networks/ 中的demo。



filter 可以将当前层神经网络上的一个子节点矩阵转化为下一层神经网络上的一个单位节点矩阵。单位节点矩阵制的是长和宽都是 1，但深度不限的节点矩阵。进行卷积操作，需要注意 filter 的个数 KK、filter 的尺寸 FF、卷积步长 stride 的大小 SS 以及 padding 的大小 PP。图 中 K=2K=2，F=3F=3，

S=1S=1，P=0P=0。

常用的filter尺寸有 3×3 或 5×5，即图黄色和橙色矩阵中的前两维，这个是人为设定的filter的节点矩阵深度，即图黄色和橙色矩阵中的最后一维（filter 尺寸的最后一维），是由当前层神经网络节点矩阵的深度（RGB 图像节点矩阵深度为 3）决定的；卷积层输出矩阵的深度（也称为 filter 的深度）是由该卷积层中 filter 的个数决定，该参数也是人为设定的，一般随着卷积操作的进行越来越大。图中filter的尺寸为 3×3×3，filter的深度为 2。

卷积操作中，一个 3×3×3 的子节点矩阵和一个 3×3×3 的 filter 对应元素相乘，得到的是一个 3×3×3 的矩阵，此时将该矩阵所有元素求和，得到一个 1×1×1 的矩阵，将其再加上 filter 的 bias，经过激活函数得到最后的结果，将最后的结果填入到对应的输出矩阵中。

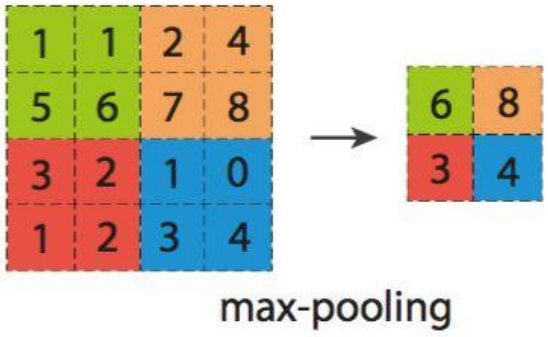
池化层

池化层可以非常有效地缩小矩阵的尺寸（主要减少矩阵的长和宽，一般不会去减少矩阵深度），从而减少最后全连接层中的参数。使用池化层既可以加快计算速度也有防止过拟合问题的作用。

与卷积层类似，池化层的前向传播过程也是通过一个类似 filter 的结构完成的。不过池化层 filter 中的计算不是节点的加权和，而是采用更加简单的最大值或者平均值运算。使用最大值操作的池化层被称为最大池化层（max pooling），这是使用最多的池化层结构。使用平均值操作的池化层被称为平均池化层（average pooling）。

与卷积层的 filter 类似，池化层的 filter 也需要人工设定 filter 的尺寸、是否使用全 0 填充 以及 filter 移动的步长等设置，而且这些设置的意义也是一样的。

卷积层和池化层中 filter 的移动方式是相似的，唯一的区别在于卷积层使用的 filter 是横跨整个深度的，而池化层使用的 filter 只影响一个深度上的节点。所以池化层的过滤器除了在长和宽两个维度移动之外，它还需要在深度这个维度移动。也就是说，在进行 max 或者 average 操作时，只会在同一个矩阵深度上进行，而不会跨矩阵深度进行。



图中，池化层 filter 的尺寸为 2×2，即 F=2F=2，padding 大小 P=0P=0，filter 移动的步长 S=2S=2。 池化层一般不改变矩阵的深度，只改变矩阵的长和宽。池化层没有 trainable 参数，只有一些需要人工设定的超参数。

更多实验请见在线平台

本次实验作业：

基于TensorFlow搭建卷积神经网络进行MNIST手写体数字图像分类。

要求：能够正确训练，能够任意选择数据集内图像进行测试，输出类别。

实践五 TensorFlow 2 的高级 API—Keras

（一）实践目的

掌握 Keras 的两种常用序列模型构建方式、函数式 API、子类化 Keras 模型类。

（二）实践环境

计算机、希冀平台

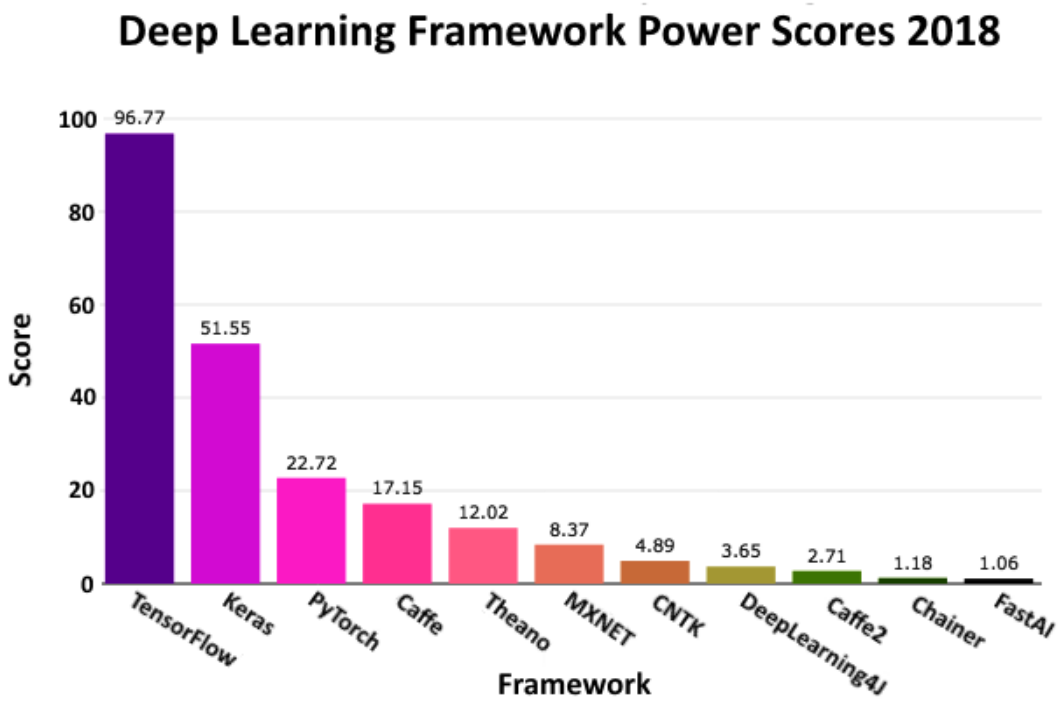
（三）实践学时

4 学时

（四）实践内容与步骤

1、Keras 的使用情况及优势

Keras 目前已被工业界和学术界广泛使用。下图为 Jeff Hale 制作的 2018 年深度学习框架的热度排名。



Keras 具有以下优势：

- 同时为新用户和专家提供简单且一致的 API；
- 对用户友好，具有简单且一致的界面，并针对常见用例进行了优化；
- 针对用户的错误提供了明确可行的反馈，并给出有效建议；
- 通过将可配置的构造块连接在一起即可构建 Keras 模型，几乎不受任何限制，具有模块化和易组合的特点；
- 可编写自定义构造模块，易于扩展；
- 无需单独导入，可通过 tensorflow.keras 直接获得。

2、Keras 特性

通过如下命令可查看 TensorFlow 附带的 Keras 版本。

```
import tensorflow as tf
print(tf.keras.__version__)
```

Keras 的其他特性：对多 GPU 数据并行化的内置支持；Keras 模型可以转换为 TensorFlow Estimator，并在谷歌云（Google Cloud）中的 GPU 集群上进行训练。Keras 作为独立开源项目进行维护，其参考实现可见 www.keras.io。

尽管 TensorFlow 在 tf.keras 模块中完全实现了 Keras，但该项目的维护与 TensorFlow 无关。默认情况下，该项目具有 TensorFlow 特定的附加功能，包括对动态图机制的支持等。

3、Keras 默认配置文件

Linux 系统的默认配置文件如下：

```
$ HOME/.keras/keras.jason
```

4、Keras 后端

Keras 是一个模型级的库，它可能有不同的张量操作引擎来处理各种低级操作，

例如卷积、张量乘积等。这些操作引擎被称为后端。更多有关 keras.backend 的方法参见 https://keras.io/backend。

使用 Keras 后端的规范方法如下：

```
from keras import backend as K
```

下面是通过 Keras 后端调用函数的函数签名：

```
K.constant(value, dtype = None, shape = None, name = None)
```

其中，value 是要赋给常数的值；dtype 是所创建的张量的类型；shape 是所创建的张量的形状；name 是一个可选名称。

大家可以尝试创建一个常量并将其进行输出显示。

5、Keras 模型

Keras 是一个基于神经网络模型概念的模型库，共包含两类模型：主要模型为序列模型（Sequence），该模型是层与层的线性堆叠；还有一个是使用 Keras 函数式 API 模型。

5.1 Keras 序列模型

构建 Keras 序列模型时，按照网络计算的相同顺序，向模型添加层。

模型建立完成后，对其进行编译；该操作可以优化要进行的计算，并未模型分配优化器和损失函数。

下一步，将模型与数据拟合。该过程通常被称为模型训练，所有的计算都将在此进行，数据可批量或一次性全部输入模型。

接下来，对模型进行评估，以获得准确率、损失和其他指标。

最后，利用训练后的模型对新数据进行预测。

创建一个序列模型有以下两种方法，下面来分别加以介绍。

5.1.1 构建序列模型方法一

直接在模型定义中进行层列表的传递，实例如下：

```
#导入数据
mnist = tf.keras.datasets.mnist
    (train_x,train_y), (test_x, test_y) = mnist.load_data()
batch_size = 32
epochs=10

#归一化数据
train_x, test_x = tf.cast(train_x/255.0, tf.float32), tf.cast(test_x/255.0, tf.float32)
train_y, test_y = tf.cast(train_y,tf.int64),tf.cast(test_y,tf.int64)

#模型构造
model1 = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512,activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10,activation=tf.nn.softmax)
])
```

模型定义如下：

Flatten 接受图像输入将其转化生成为一个 1D 向量；

Dense 是一个全连接层，表示在两个网络层中任意一对神经元都存在连接关系。上例中有 512 个神经元，其输入通过 ReLU（非线性）激活函数传递；

Dropout 会随机关闭上一层的一小部分神经元（本例随机概率设置为 0.2）。这样做是为了防止任何特定的神经元变得过于强大，而导致模型与数据过于拟合，从而影响模型对测试数据度量的准确率；

最后的 Dense 层包含一个 softmax 激活函数，该函数为每一个可能的输出（10

个单元）分配概率。

接下来对模型进行编译、训练和评估：

```
#优化器 optimizer 可以通过调整，模型权重来减少损失
optimiser = tf.keras.optimizers.Adam()
model1.compile (optimizer= optimiser, loss='sparse_categorical_crossentropy', metrics = ['accuracy'])

#用 fit()方法进行训练
model1.fit(train_x, train_y, batch_size=batch_size, epochs=epochs)

#通过 evaluate()方法评估训练后的模型准确性
model1.evaluate(test_x, test_y)
```

5.1.2 构建序列模型方法二

相同的网络结构，将层列表传递给序列模型构造函数的另一种方法是使用 add 方法，示例如下：

```
model2 = tf.keras.models.Sequential();
model2.add(tf.keras.layers.Flatten())
model2.add(tf.keras.layers.Dense(512, activation='relu'))
model2.add(tf.keras.layers.Dropout(0.2))
model2.add(tf.keras.layers.Dense(10,activation=tf.nn.softmax))
```

相应的优化拟合和测试实例如下：

```
optimiser = tf.keras.optimizers.Adam()
model2.compile (optimizer= optimiser, loss='sparse_categorical_crossentropy', metrics = ['accuracy'])
```

```
model2.fit(train_x, train_y, batch_size = batch_size, epochs=epochs)
model2.evaluate(test_x, test_y)
```

5.2 Keras 函数式 API

对于前文提到的仅对线性层进行简单堆叠的序列模型，函数式 API 允许用户构造比起复杂得多的结构，同时支持构造更高级的模型，包括多输入多输出模型、具有共享层的模型以及具有残差连接的模型。下例为使用了函数 API 的简短示例：

```
inputs = tf.keras.Input(shape=(28,28))
x = tf.keras.layers.Flatten()(inputs)
x = tf.keras.layers.Dense(512, activation='relu',name='d1')(x)
x = tf.keras.layers.Dropout(0.2)(x)
predictions = tf.keras.layers.Dense(10,activation=tf.nn.softmax, name='d2')(x)
model3 = tf.keras.Model(inputs=inputs, outputs=predictions)

#利用 summary 输出网络构造结果
model3.summary()
```

请注意观察这段代码是如何生成与 model 1 和 model 2 相同的体系结构的。

5.3 子类化 Keras 模型类

Keras 模型类可以按照下面的代码进行子类化：

```
#使用构造函数(.__init__())分别对神经层进行声明和命名
class MyModel(tf.keras.Model):
    def __init__(self, num_classes=10):
        super(MyModel, self).__init__()

        inputs = tf.keras.Input(shape=(28,28))
```

```
            self.x0 = tf.keras.layers.Flatten()
            self.x1 = tf.keras.layers.Dense(512, activation='relu',name='d1')
            self.x2 = tf.keras.layers.Dropout(0.2)
            self.predictions = tf.keras.layers.Dense(10,activation=tf.nn.softmax, na
me='d2')
  #在 call()方法中以函数形式将神经层连接在一起
        def call(self, inputs):
            x = self.x0(inputs)
            x = self.x1(x)
            x = self.x2(x)
            return self.predictions(x)
  #调用模型
  model4 = MyModel()
  optimiser = tf.keras.optimizers.Adam()
  model4.compile (optimizer= optimiser, loss='sparse_categorical_crossentropy', m
etrics = ['accuracy'])
  model4.fit(train_x, train_y, batch_size=32, epochs=epochs)

    model4.evaluate(test_x, test_y)
```

5.4 保存和加载 Keras 模型

保存模型的结构、权重、训练状态（损失、优化器）和优化器的状态，以便用户可以从中断的位置继续训练模型:

```
model.save('./model_name.h5')
```

加载已保存的模型：

```
from tensorflow.keras.models import load_model
new_model = load_model('./model_name.h5')
```

也可以仅保存模型权重及加载：

```
model.save_weights('./model_name.h5')
model.load_weights('./model_name.h5')
```

更多实验请见在线平台。

实践六 基于 TensorFlow 的无监督学习

（一）实践目的

掌握自动编码器的基本原理，部署简单的自动编码器并进行图像去噪。

（二）实践环境

计算机、希冀平台

（三）实践学时

4 学时

（四）实践内容与步骤

1、自动编码器

无监督学习的目的是发现无标签数据间的模式或关系，这与监督学习不同，监督学习同时提供了数据特征及标签，并希望依此预测出未见过的新特征的标签。而无监督学习的核心是挖掘数据见的潜在结构或规律。

无监督学习还可以用于数据压缩。得益于无监督学习，该数据的模式占用更少内存，且不会损害到数据的结构和完整性。比如自动编码器可用于数据压缩和图像去噪。

自动编码是一种使用人工神经网络(ANN)实现的数据压缩和解压缩算法。它是一种无监督形式的学习算法,只需给它提供未标记的数据即可。该算法的工作方式为强制输人通过瓶颈(即宽度小于原始输入的一层或多层神经网络)来生成输人的压缩版本。为了重构输入(即解压缩),要将过程逆转,使用反向传播在中间层创建输入的表征,然后将这种表征重构为输出。

自动编码是有损的压缩算法。与无损压缩算法不同,自动编码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。

自动编码与数据相关，这意味着自动编码器只能压缩与训练数据类似的数据。例如，使用汽车图片训练的自动编码器，在街道标志图片上就会表现很差，因为它学习的是汽车独有的特征。

2、部署一个自动编码器

下面结合一个例子来进行阐述：

```
#导入数据
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras import regularizers
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

2.1 数据预处理

包含归一化等数据预处理操作：

```
#加载数据
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

print(x_train.shape)
print(x_test.shape)

#为了将图像传入一个一维的全连接层，将图像展平
x_train = x_train.reshape(( x_train.shape[0], np.prod(x_train.shape[1:])))
x_test = x_test.reshape((x_test.shape[0], np.prod(x_test.shape[1:])))

print(x_train.shape)
print(x_test.shape)
```

```
#指定所需的尺寸
image_dim = 784
encoding_dim = 32 #压缩系数 = 784/32 = 24.5

#创建单层编码器和自动编码器模型
input_image = Input(shape=(image_dim, ))
encoded_image = Dense(encoding_dim, activation='relu', activity_regularizer=
regularizers.l1(10e-5))(input_image)
encoder = Model(input_image, encoded_image)
decoded_image = Dense(image_dim, activation='sigmoid')(encoded_image)
autoencoder = Model(input_image, decoded_image)

#创建解码器模型
encoded_input = Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer(encoded_input))

#编译自动编码器
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

#定义两个检查点。一个用于每轮训练后保存模型。如果 save_best_only =
True，则根据监测值（验证损失），保留最新的最佳模型。当监测值（验证损失）
的变化小于最小增量 min_data 时，第二个检查点停止训练。
checkpointer1 = ModelCheckpoint(filepath= 'model.weights.best.hdf5' , verbo
se =2, save_best_only = True)
checkpointer2 = EarlyStopping(monitor='val_loss',
                              min_delta=0.0005,
                              patience=2,
                              verbose=2, mode='auto')
```

2.2 训练

运行训练使用.fit 方法：

```
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256, callbacks=[checkpointer1], verbose=2,
                shuffle=True,
                validation_data=(x_test, x_test))
```

接下来对数据进行压缩和解压缩：

```
encoded_images = encoder.predict(x_test)
decoded_images = decoder.predict(encoded_images)
```

2.3 结果显示

```
number_of_items = 12 #要显示的图像数目
plt.figure(figsize=(20, 4))
for i in range(number_of_items):
    # display items before compression
    graph = plt.subplot(2, number_of_items, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    graph.get_xaxis().set_visible(False)
    graph.get_yaxis().set_visible(False)

    # display items after decompression
    graph = plt.subplot(2, number_of_items, i + 1 + number_of_items)
    plt.imshow(decoded_images[i].reshape(28, 28))
    plt.gray()
    graph.get_xaxis().set_visible(False)
    graph.get_yaxis().set_visible(False)
plt.show()
```

图 6.1 压缩前的图像

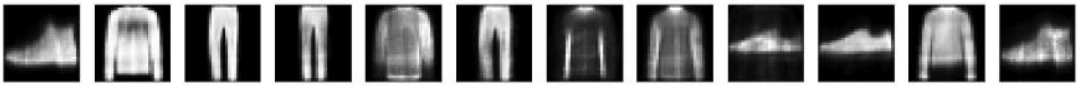


图 6.2 解压缩后的图像

显而易见，压缩/解压缩是有损的。

本次实验作业：

基于 TensorFlow 搭建自动编码器对 fashion_minist 或 minist 数据集进行图像去噪，展示去噪前后的图像以及损失变化曲线。

要求：首先在图像中加入人工噪声，然后使用自动编码器将其进行消除。

**提示：

- 可采用由 np.random.normal 产生的高斯数组来对原图引入噪声；
- 注意采用.fit 方法时选择什么作为输入和标签，才能实现去噪的效果。

实践七 基于 TensorFlow 的迁移学习

（一）实践目的

掌握迁移学习的基本原理，使用预训练的网络进行迁移再训练。

（二）实践环境

计算机、希冀平台

（三）实践学时

4 学时

（四）实践内容与步骤

所谓迁移学习，就是将一个问题上训练好的模型通过简单的调整使其适用于一个新的问题。本次实验将通过介绍如何利用 ImageNet 数据集上的训练好的 Inception-v3 模型来解决一个新的图像分类问题。根据论文 DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition 中的结论，可以保留训练好的 Inception-v3 模型中所有卷积层的参数，只是替换最后一层全连接层。在最后这一层全连接层之前的网络层称之为瓶颈层（bottleneck）.

将新的图像通过训练好的卷积神经网络直到瓶颈层的过程可以看成是对图像进行特征提取的过程。在训练好的 Inception-v3 模型中，因为瓶颈层的输出在通过一个单层的全连接层神经网络可以很好地区分 1000 种类别的图像，所以有理由认为瓶颈层输出的节点向量可以被作为任何图像的一个更加精简且表达能力更强的特征向量。于是，在新数据集上，可以直接利用这个训练好的神经网络对图像进行特征提取，然后再将提取得到的特征向量作为输入作为一个新的单层全连接神经网络处理新的分类问题。

一般来说，在数据量足够的情况下，迁移学习的效果不如完全重新训练。但是迁移学习所需要的训练时间和训练样本数要远远小于训练完整的模型。

下面结合一个具体例子来介绍如何通过 TensorFlow 实现迁移学习。

```
import  glob
```

```
import os.path
import random
import numpy as np
import tensorflow as tf
from tensorflow.python.platform import gfile

#模型和样本路径设置
BOTTLENECK_TENSOR_SIZE = 2048 #瓶颈层的节点个数设置
BOTTLENECK_TENSOR_NAME = 'pool_3/_reshape:0'
JPEG_DATA_TENSOR_NAME = 'DecodeJpeg/contents:0'



MODEL_DIR = '../../datasets/inception_dec_2015'
MODEL_FILE= 'tensorflow_inception_graph.pb'

CACHE_DIR = '../../datasets/bottleneck'
INPUT_DATA = '../../datasets/flower_photos'

VALIDATION_PERCENTAGE = 10
TEST_PERCENTAGE = 10

#神经网络参数的设置
LEARNING_RATE = 0.01
STEPS = 4000
BATCH = 100

#此处省略数据集构造过程，仅对主函数进行分析



def main():
```

```
image_lists = create_image_lists(TEST_PERCENTAGE, VALIDATION_PER
    CENTAGE)
n_classes = len(image_lists.keys())

# 读取已经训练好的 Inception-v3 模型。
with gfile.FastGFile(os.path.join(MODEL_DIR, MODEL_FILE), 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
bottleneck_tensor, jpeg_data_tensor = tf.import_graph_def( graph_def, return
    _elements = [BOTTLENECK_TENSOR_NAME, JPEG_DATA_TENSO
    R_NAME])

# 定义新的神经网络输入
bottleneck_input = tf.placeholder(tf.float32, [None, BOTTLENECK_TENSO
    R_SIZE], name='BottleneckInputPlaceholder')
ground_truth_input = tf.placeholder(tf.float32, [None, n_classes], name='Gro
    undTruthInput')

# 定义一层全连接层
with tf.name_scope('final_training_ops'):
    weights = tf.Variable(tf.truncated_normal([BOTTLENECK_TENSOR_SI
        ZE, n_classes], stddev=0.001))
    biases = tf.Variable(tf.zeros([n_classes]))
    logits = tf.matmul(bottleneck_input, weights) + biases
    final_tensor = tf.nn.softmax(logits)

# 定义交叉熵损失函数。
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=
    ground_truth_input)
cross_entropy_mean = tf.reduce_mean(cross_entropy)
```

```
train_step = tf.train.GradientDescentOptimizer(LEARNING_RATE).minimize
    (cross_entropy_mean)

# 计算正确率。
with tf.name_scope('evaluation'):
    correct_prediction = tf.equal(tf.argmax(final_tensor, 1), tf.argmax(ground
        _truth_input, 1))
    evaluation_step = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
    # 训练过程。
    for i in range(STEPS):
        train_bottlenecks, train_ground_truth = get_random_cached_bottlene
            cks(sess, n_classes, image_lists, BATCH, 'training', jpeg_data_
            tensor, bottleneck_tensor)
        sess.run(train_step, feed_dict={bottleneck_input: train_bottlenecks,
            ground_truth_input: train_ground_truth})

        if i % 100 == 0 or i + 1 == STEPS:
            validation_bottlenecks, validation_ground_truth = get_random_
                cached_bottlenecks(sess, n_classes, image_lists, BATCH, '
                validation', jpeg_data_tensor, bottleneck_tensor)
            validation_accuracy = sess.run(evaluation_step, feed_dict={
                bottleneck_input: validation_bottlenecks, ground_truth_inp
                ut: validation_ground_truth})
            print('Step %d: Validation accuracy on random sampled %d
            examples = %.1f%%' %
                (i, BATCH, validation_accuracy * 100))
```

```
        # 在最后的测试数据上测试正确率。
        test_bottlenecks, test_ground_truth = get_test_bottlenecks(
            sess, image_lists, n_classes, jpeg_data_tensor, bottleneck_tensor)
        test_accuracy = sess.run(evaluation_step, feed_dict = {bottleneck_input:
                test_bottlenecks, ground_truth_input: test_ground_truth})
        print('Final test accuracy = %.1f%%' % (test_accuracy * 100))

if __name__ == '__main__':
    main()
```

本次实验作业：

基于 TensorFlow 利用预训练好的网络模型在新的数据集上完成迁移学习。

要求：能够正确训练，并对有无采用迁移学习训练的结果进行对比分析。

实践八 基于 TensorFlow 的图像风格迁移

（一）实践目的

掌握图像风格迁移的基本原理，采用预训练模型实现风格迁移。

（二）实践环境

计算机、希冀平台

（三）实践学时

4 学时

（四）实践内容与步骤

神经网络风格迁移时一种利用神经网络，将一副图像的艺术风格施加给另一幅图像内容的技术，最终结果是两幅图像的混合。继承艺术风格的图像称为内容图像，施加艺术风格的图像称为风格图像，生成的图像称为混合图像。

图像风格迁移通过定义两个损失函数来实现：一个用来描述两幅图像内容之间的差异；另一个用来描述两幅图像风格之间的差异。

首先，用内容图像初始化混合图像。接着，采用反向传播方法，最小化混合图像和内容图像的内容差异（也称为损失或距离）及与风格图像的风格差异。最终，生成同时具有“风格图像”的风格和“内容图像”的内容的新图像（即混合图像）。

进行图像风格迁移时，推荐使用 VGG19 架构（VGG19 架构已在 ImageNet 数据集上完成与训练，该数据集包含超过 1400 万张图像和 1000 个类别）的特征层。

下面结合实例来介绍如何通过 TensorFlow 进行图像风格迁移。

1、导入配置

为了实现图像风格迁移功能，需要先导入所需模块

```
import numpy as np
from PIL import Image
```

```
import time
import functools

import matplotlib.pyplot as plt
import matplotlib as mpl
# 设置图像显示
mpl.rcParams['figure.figsize'] = (10,10)
mpl.rcParams['axes.grid'] = False
```

以 pip 方式安装 pillow 模块，并导入 TensorFlow 模块

```
import tensorflow as tf
from tensorflow.keras.preprocessing import image as kp_image
from tensorflow.keras import models
from tensorflow.keras import losses
from tensorflow.keras import layers
from tensorflow.keras import backend as K
from tensorflow.keras import optimizers
```

2、图像预处理

将图像进行预处理，统一调整图像大小并将调用 img_to_array()将 PIL 图像转换为 NumPy 数组。

为了兼容后续操作，图像需要沿 axis=0 扩展一个维度。可通过调用 np.expand_dims()实现该过程：

```
image = np.expand_dims(image, axis=0)
```

3、使用 VGG19 架构

为了更好地理解接下来的代码片段，可以查阅下面这个网址学习 VGG19 网络架构：https://github.com/fchollet/deep-learning-models/blob/master/vgg19.py

VGG19 的结构比较简单，由多个卷积层 block 组成，每个 block 后面接一个

最大池化层。

内容层用 block5（第 5 个卷积层 block）的第 2 个卷积层（block_conv2）表示。同时，使用每个 block 的第 1 个卷积层来表示风格层。、

```
# 内容层
content_layers = ['block5_conv2']
# 风格层
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1'
]
number_of_content_layers = len(content_layers)
number_of_style_layers = len(style_layers)
```

4、创建模型

首先加载 vgg_model 模型，无须包含该模型的分类层（include_top = False）。接着，将其冻结（vgg_model.trainable = False）。

用列表推导式过去风格层和内容曾的输出，并用输出值和 VGG 的输入共同创建一个可访问 VGG 层的新模型，该模型输出与训练模型 VGG19 的风格中间层及内容中间层。

之后，创建一个输出图像，使得输出风格和输入风格在相应的特征层上的和距离最小。

```
def get_model():
    # 加载预训练好的 VGG 模型
    vgg_model = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
    vgg_model.trainable = False
```

```
    # 获取风格层和内容层
    style_outputs = [vgg_model.get_layer(name).output for name in style_laye
rs]
    content_outputs = [vgg_model.get_layer(name).output for name in content
_layers]
    model_outputs = style_outputs + content_outputs
    # 创建模型
    return models.Model(vgg_model.input, model_outputs)
```

5、计算损失

使用均方误差分别计算两个图像间的内容损失和风格损失。

```
def rms_loss(image1,image2):
    loss = tf.reduce_mean(input_tensor=tf.square(image1 - image2))
    return loss
```

定义内容损失：

```
def content_loss(content, target):
  return rms_loss(content, target)
```

风格损失用 Gram 矩阵（格拉姆矩阵）定义。Gram 矩阵作为一个度量标准，时风格图像矩阵与其自身转置的点积。执行点积运算时，图像矩阵的每一行与每一列相乘，是的该矩阵原始表示中包含的空间信息被分发。最终的结果是关于图像的非局部信息，即使图像中所有像素的位置被打乱，仍然能够保留的图片信息，如纹理、形状和权重，即其风格。

生成 Gram 矩阵的代码如下：

```
def gram_matrix(input_tensor):
  channels = int(input_tensor.shape[-1])
  tensor = tf.reshape(input_tensor, [-1, channels])
  number_of_channels = tf.shape(input=tensor)[0]
```

```
    gram = tf.matmul(tensor, tensor, transpose_a=True)
    return gram / tf.cast(number_of_channels, tf.float32)
```

风格损失代码如下：

```
def style_loss(style, gram_target):
    gram_style = gram_matrix(style)
    return rms_loss(gram_style, gram_target)
```

除此之外，还会添加返回总损失来减少混合图像局部的边缘效应。

6、执行风格迁移

首先，采用预训练模型来提取内容特征和风格特征：

```
def get_feature_representations(model, content_path, style_path):
    content_image = load_and_process_image(content_path)
    content_outputs = model(content_image)
   content_features = [content_layer[0] for content_layer in content_outputs[n
        umber_of_style_layers:]]

    style_image = load_and_process_image(style_path)
    style_outputs = model(style_image)
    style_features = [style_layer[0] for style_layer in style_outputs[:number_o
f_style_layers]]

    return style_features, content_features
```

对风格特征循环遍历，针对每一个风格特征使用 gram 矩阵操作，从而得到 gram_style_feature。

然后，加载内容图像并将其转化为张量，得到初始化图像。该图像将会融合风格图像作为算法的输出，即混合图像。

```
initial_image = load_and_process_image(content_path)
initial_image = tf.Variable(initial_image, dtype=tf.float32)
```

定义优化器：

```
optimiser = tf.compat.v1.train.AdamOptimizer(learning_rate=5, beta1=0.99,
    epsilon=1e-1)
```

接着，初始化混合图像和最佳损失，用于保存最佳图像和最有损失：

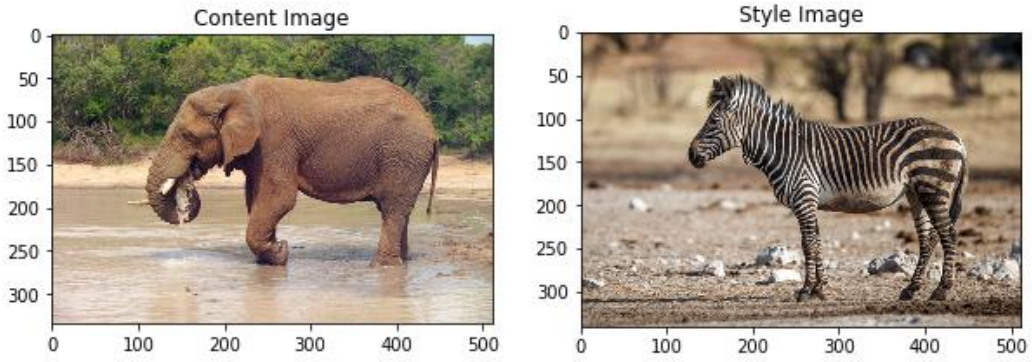
```
best_loss, best_image = float('inf'), None
# 设置配置字典
loss_weights = (style_weight, content_weight)
config = {
    'model': model,
    'loss_weights': loss_weights,
    'init_image': initial_image,
    'gram_style_features': gram_style_features,
    'content_features': content_features
}
number_rows = 2
number_cols = 5
display_interval = number_of_iterations/(number_rows*number_cols)

norm_means = np.array(channel_means)
minimum_vals = -norm_means
maximum_vals = 255 - norm_means
```

```
images = [] #用于存储混合图像
```

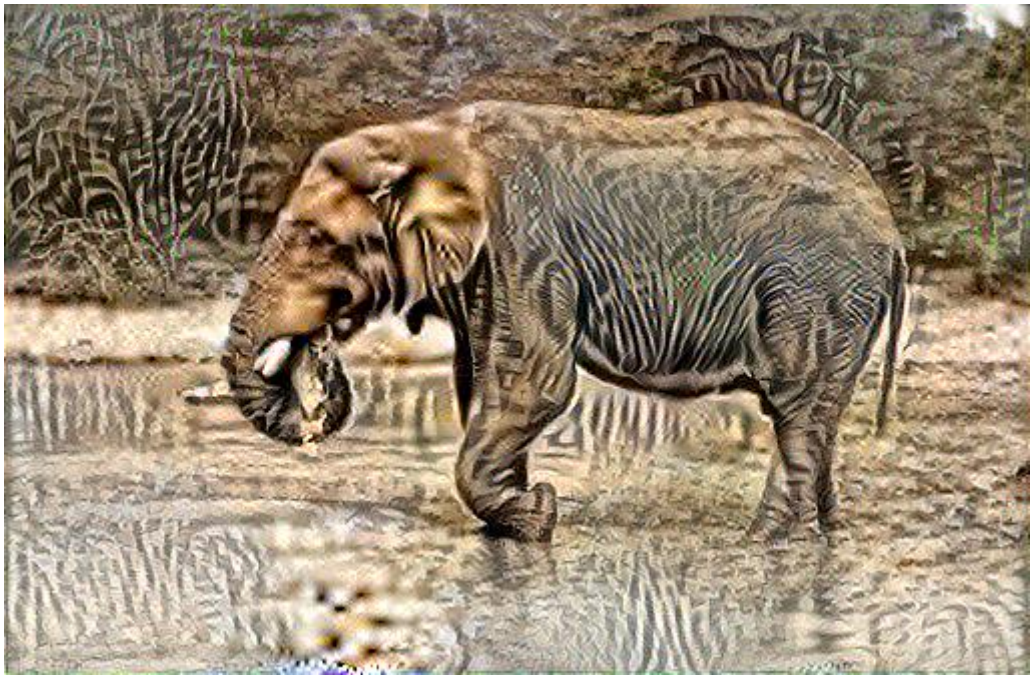
正式进入训练循环：

```
for i in range(number_of_iterations):
    grads, all_loss = compute_grads(config)
    loss, style_score, content_score = all_loss
    optimiser.apply_gradients([(grads, initial_image)]) #将计算出的梯度应用到变量上
    clipped_image = tf.clip_by_value(initial_image, minimum_vals,
        maximum_vals)
    initial_image.assign(clipped_image)

    if loss < best_loss:
      # 从损失中更新最优损失和最佳图像
      best_loss = loss
      best_image = deprocess_image(initial_image.numpy())
```

结果展示：



上图中，内容图像（左图）和风格图像（右图）进行风格迁移后得到的混合图像如下：



本次实验作业：

基于 TensorFlow 利用预训练的深度卷积网络（如 VGG19）完成图像风格迁移。

要求：能够正确训练，并展示内容图像、风格图像，以及训练中不同迭代次数以及最佳的混合图像。

实践九 基于 TensorFlow 的深度学习应用实践

（一） 实践目的

根据实验一到实验四的所学内容，以及学生自身所具有的深度学习与TensorFlow知识和编程技能，结合数学知识、算法知识、软件工程知识、自行选定某一特定应用实践，完成该应用的开发与实现。

（二） 实践环境

计算机、希冀平台

（三） 实践学时

4学时

（四） 实践内容与步骤

学生可以选做以下四个固定题目之一即可：

（1）题目一：

基于TensorFlow搭建卷积神经网络进行花卉图像分类；

（2）题目二：

基于TensorFlow搭建循环神经网络进行文本分类；

（3）题目三：

基于TensorFlow搭建U型网络实现医学图像分割算法；

（4）题目四：

此题目为开放命题，请结合自己的特长和能力自行选择题目，内容必须基于深度学习与TensorFlow，例如人脸识别、车辆识别、图像分类、生成对抗网络等。

实践报告要求说明

1. 实验报告模板参见文档“深度学习原理与TensorFlow实践-最终报告-模板”。

2. 每个人独立完成，不允许组成小组进行实践设计和报告撰写。

3. 所有实践内容写成一份报告，该报告分为三个部分：一、深度学习与实践基础实验，其中包含实践二和实践四中的实验作业；二、深度学习与实践进阶实验，其中包含实践六、实践七和实践八中的实验作业；三、深度学习与实践创新实验，其中包含实践九TensorFlow的深度学习应用实践。报告格式参见模板。

4. 报告正文篇幅不允许超过65页。其中 “深度学习与实践基础实验”不超过15页； “深度学习与实践进阶实验”不超过25页；“深度学习与实践进阶实验”不超过25页。

5. 报告最终版本只需提交电子版，只能是word格式，WPS和PDF等其他格式不予接受。