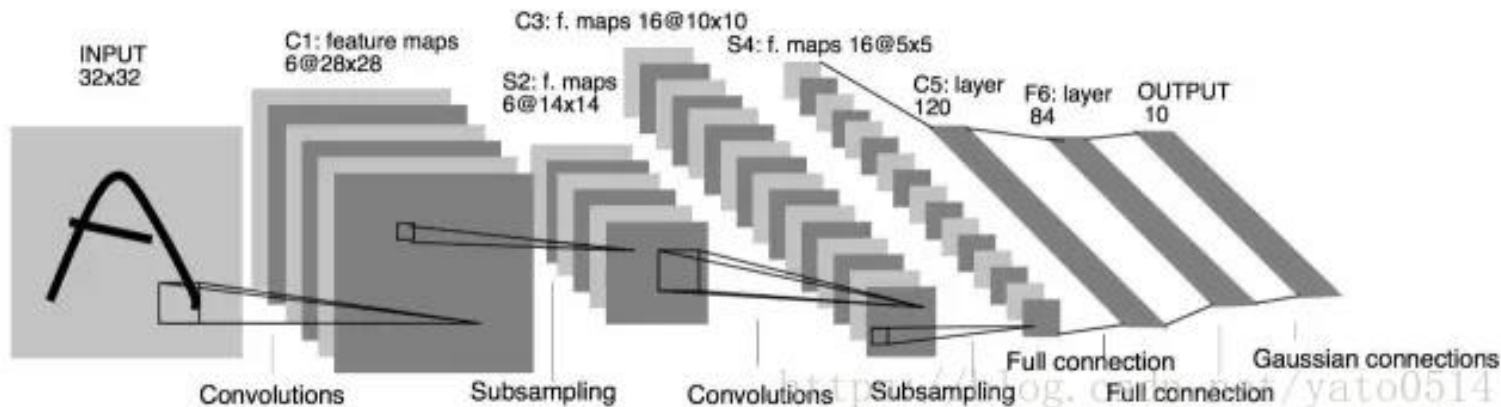


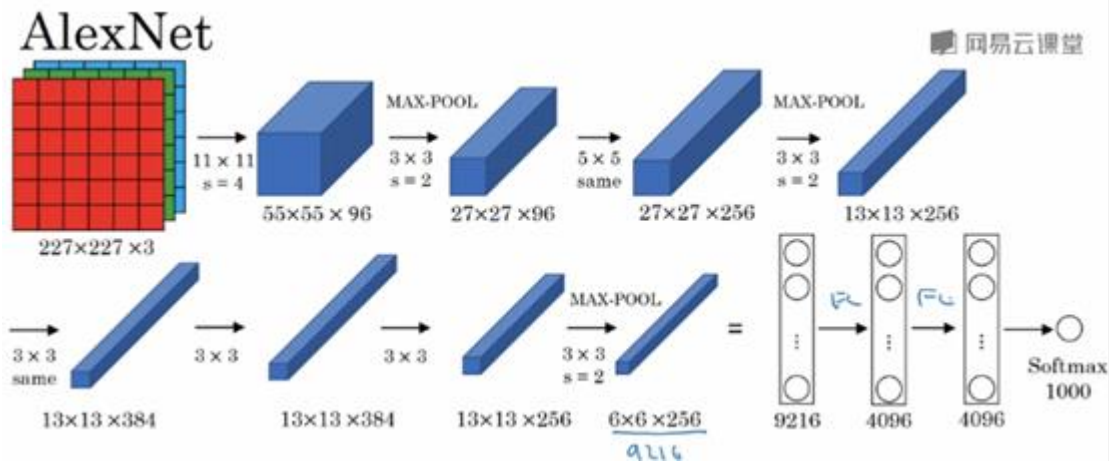
LeNet5



Lenet5由两个卷积层，两个池化层，以及两个全连接层组成。卷积都是**5*5**的**filter**，步长为**1**。池化都是max-pooling Lenet5特征能够总结为如下几点：

- (1) 卷积神经网络使用三个层作为一个系列：卷积、池化、非线性
- (2) 使用卷积提取空间特征
- (3) 使用映射到空间均值下采样
- (4) 双曲线（**tanh**）或s型（**sigmoid**）形式的非线性
- (5) 多层神经网络（**MLP**）作为最后的分类器
- (6) 层与层之间的稀疏链接矩阵避免大的计算成本

- ALEXNET
- AlexNet中提出让步长比池化核的尺寸小
- 五个卷积层与三个全连接层
- 2-GPU并行，1，2，5跟随max-pooling层，两个全连接上使用dropout
- 前面卷积核大后面卷积核小
- LeNet的一种更深更宽的版本
- 最后使用softmax函数输出识别的结果
- AlexNet包含大约6000万个参数。
- AlexNet使用了ReLU激活函数；
- AlexNet也使用了LRN层（Local Response Normalization，局部响应归一化层），但是由于LRN可能作用并不大，应用的比较少。



```
Extracting MNIST_data\train-images-idx3-ubyte.gz
Extracting MNIST_data\train-labels-idx1-ubyte.gz
Extracting MNIST_data\t10k-images-idx3-ubyte.gz
Extracting MNIST_data\t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From D:\anaconda\envs\tensorflow\lib\site-packages\tensorflow\python\util\tf_should_use.py:193: initialize_all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Use `tf.global_variables_initializer` instead.
Iter 1280, Minibatch Loss= 114642.609375, Training Accuracy= 0.10938
Iter 2560, Minibatch Loss= 59842.164062, Training Accuracy= 0.25000
Iter 3840, Minibatch Loss= 74583.578125, Training Accuracy= 0.34375
Iter 5120, Minibatch Loss= 39104.605469, Training Accuracy= 0.48438
Iter 6400, Minibatch Loss= 44752.148438, Training Accuracy= 0.45312
Iter 7680, Minibatch Loss= 32665.726562, Training Accuracy= 0.64062
Iter 8960, Minibatch Loss= 33024.949219, Training Accuracy= 0.60938
Iter 10240, Minibatch Loss= 33857.218750, Training Accuracy= 0.67188
Iter 11520, Minibatch Loss= 26790.498047, Training Accuracy= 0.62500
Iter 12800, Minibatch Loss= 22727.763672, Training Accuracy= 0.67188
Iter 14080, Minibatch Loss= 26776.789062, Training Accuracy= 0.65625
Iter 15360, Minibatch Loss= 27913.441406, Training Accuracy= 0.64062
Iter 16640, Minibatch Loss= 28523.515625, Training Accuracy= 0.56250
Iter 17920, Minibatch Loss= 8054.070312, Training Accuracy= 0.78125
Iter 19200, Minibatch Loss= 22678.341797, Training Accuracy= 0.65625
Iter 20480, Minibatch Loss= 22683.085938, Training Accuracy= 0.73438
Iter 21760, Minibatch Loss= 17268.744141, Training Accuracy= 0.68750
Iter 23040, Minibatch Loss= 15282.295898, Training Accuracy= 0.68750
Iter 24320, Minibatch Loss= 9494.058594, Training Accuracy= 0.78125
Iter 25600, Minibatch Loss= 21166.041016, Training Accuracy= 0.65625
Iter 26880, Minibatch Loss= 11474.026367, Training Accuracy= 0.73438
Iter 28160, Minibatch Loss= 20742.148438, Training Accuracy= 0.68750
Iter 29440, Minibatch Loss= 13449.431641, Training Accuracy= 0.73438
```

```
-----  
Iter 179200, Minibatch Loss= 681.391968, Training Accuracy= 0.96875  
Iter 180480, Minibatch Loss= 2278.456299, Training Accuracy= 0.90625  
Iter 181760, Minibatch Loss= 1724.046631, Training Accuracy= 0.93750  
Iter 183040, Minibatch Loss= 621.620300, Training Accuracy= 0.95312  
Iter 184320, Minibatch Loss= 3401.261719, Training Accuracy= 0.87500  
Iter 185600, Minibatch Loss= 1380.598389, Training Accuracy= 0.89062  
Iter 186880, Minibatch Loss= 1924.352539, Training Accuracy= 0.93750  
Iter 188160, Minibatch Loss= 2756.644287, Training Accuracy= 0.85938  
Iter 189440, Minibatch Loss= 99.180420, Training Accuracy= 0.98438  
Iter 190720, Minibatch Loss= 1260.076782, Training Accuracy= 0.92188  
Iter 192000, Minibatch Loss= 1144.575195, Training Accuracy= 0.89062  
Iter 193280, Minibatch Loss= 918.109497, Training Accuracy= 0.90625
```

运行时间长，迭代过程中准确率有波动。随着网络深度的加深，训练错误会先减少，然后增多

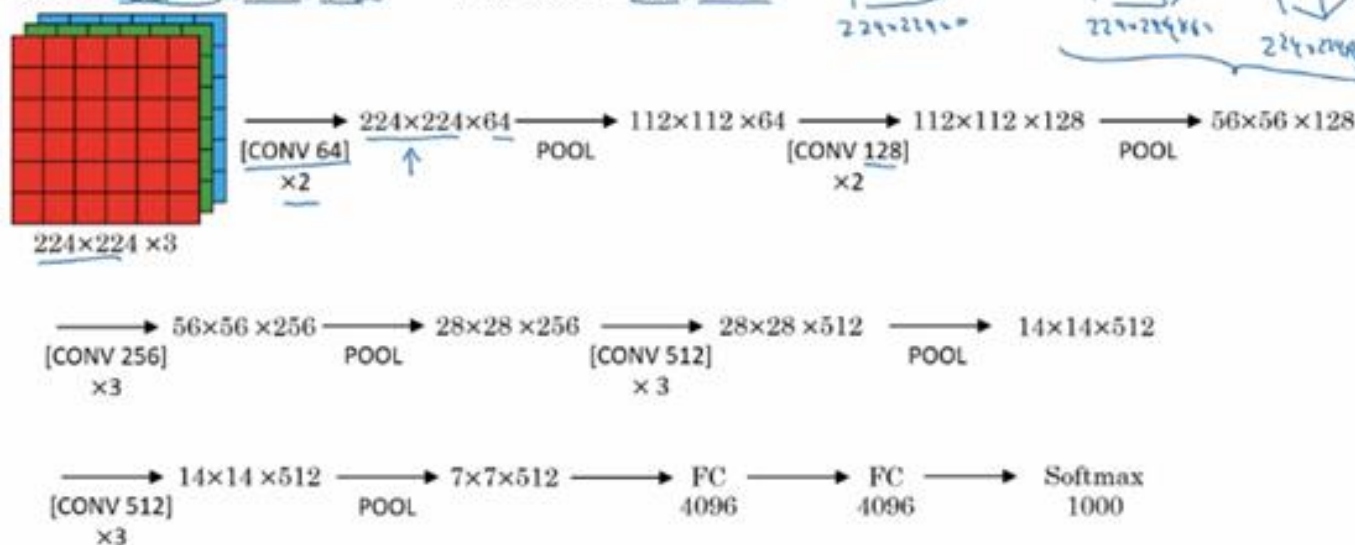
- **VGG-16**

- VGG16网络包含了16个卷积层和全连接层。
- VGG网络的一大优点是：简化了神经网络结构。
- VGG网络使用的统一的卷积核大小：3x3，stride=1，padding=same，统一的Max-Pool：2x2，stride=2。
- VGG16是一个很大的网络，总共包含1.38亿个参数。因此其主要缺点就是需要训练的特征数量非常巨大。
- 另外也有VGG19网络，由于VGG16表现几乎和VGG19不分高下，所以很多人还是会使用VGG16。

VGG - 16

CONV = 3x3 filter, s = 1, same

MAX-POOL = 2x2, s = 2



```
In [1]: import tensorflow as tf
import os
import pickle
import numpy as np

# CIFAR_DIR = "../../../cifar-10-batches-py"
CIFAR_DIR = "F:\\神经网络\\cifar-10-batches-py"
print(os.listdir(CIFAR_DIR))

['batches.meta', 'data_batch_1', 'data_batch_2', 'data_batch_3', 'data_batch_4', 'data_batch_5', 'readme.html', 'test_batch']
```

```
train_filenames = [os.path.join(CIFAR_DIR, 'data_batch_%d' % i) for i in range(1, 6)]
test_filenames = [os.path.join(CIFAR_DIR, 'test_batch')]

train_data = CifarData(train_filenames, True)
test_data = CifarData(test_filenames, False)

(50000, 3072)
(50000,)
(10000, 3072)
(10000,)
```

```
[3]: print(train_data)

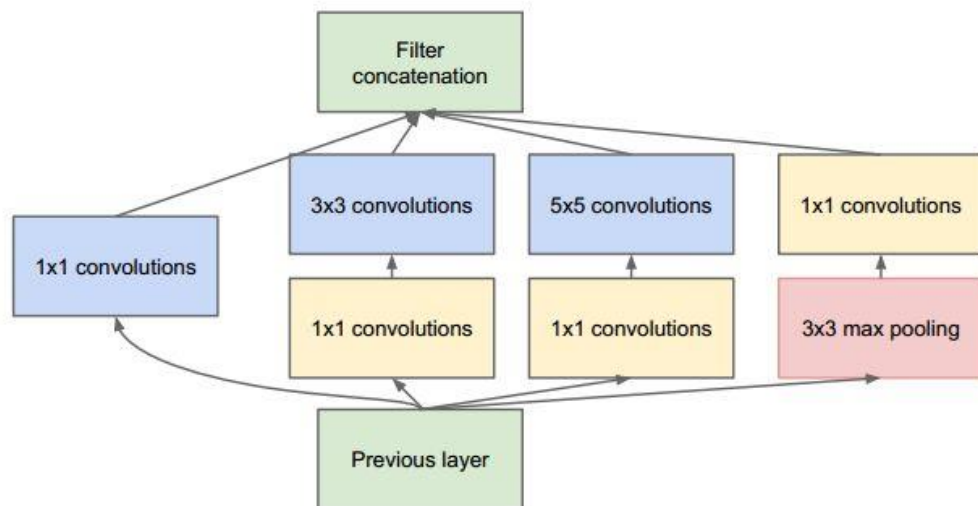
<__main__.CifarData object at 0x000001B7111E07B8>
```

```
[Train] Step: 8700, loss: 1.36943, acc: 0.55000
[Train] Step: 8800, loss: 0.51276, acc: 0.80000
[Train] Step: 8900, loss: 0.82282, acc: 0.70000
[Train] Step: 9000, loss: 1.17819, acc: 0.40000
(10000, 3072)
(10000,)
[Test ] Step: 9000, acc: 0.71850
[Train] Step: 9100, loss: 1.23025, acc: 0.55000
[Train] Step: 9200, loss: 1.23958, acc: 0.70000
[Train] Step: 9300, loss: 0.88120, acc: 0.70000
[Train] Step: 9400, loss: 0.58785, acc: 0.90000
[Train] Step: 9500, loss: 0.62823, acc: 0.80000
[Train] Step: 9600, loss: 0.31032, acc: 0.95000
[Train] Step: 9700, loss: 0.84536, acc: 0.65000
[Train] Step: 9800, loss: 0.71056, acc: 0.75000
[Train] Step: 9900, loss: 0.40845, acc: 0.90000
[Train] Step: 10000, loss: 0.83654, acc: 0.55000
(10000, 3072)
(10000,)
[Test ] Step: 10000, acc: 0.72900
```

GoogLeNet/Inception

由于卷积层的通道数过大，VGG并不高效。

GoogLeNet设计了一种称为inception的模块，这个模块使用密集结构来近似一个稀疏的CNN。



GoogLeNet使用了不同大小的卷积核来抓取不同大小的感受野。

Inception模块的另外一个特点是使用了一中瓶颈层（实际上就是**1x1**卷积）来降低计算量。

GoogLeNet的另外一个特殊设计是最后的卷积层后使用全局均值池化层替换了全连接层，所谓全局池化就是在整个**2D**特征图上取均值。这大大减少了模型的总参数量。要知道在AlexNet中，全连接层参数占整个网络总参数的**90%**。使用一个更深更大的网络使得GoogLeNet移除全连接层之后还不影响准确度。


```
train_data = CifarData(train_data,
test_data = CifarData(test_data,
```

```
(50000, 3072)
(50000,)
(10000, 3072)
(10000,)
```

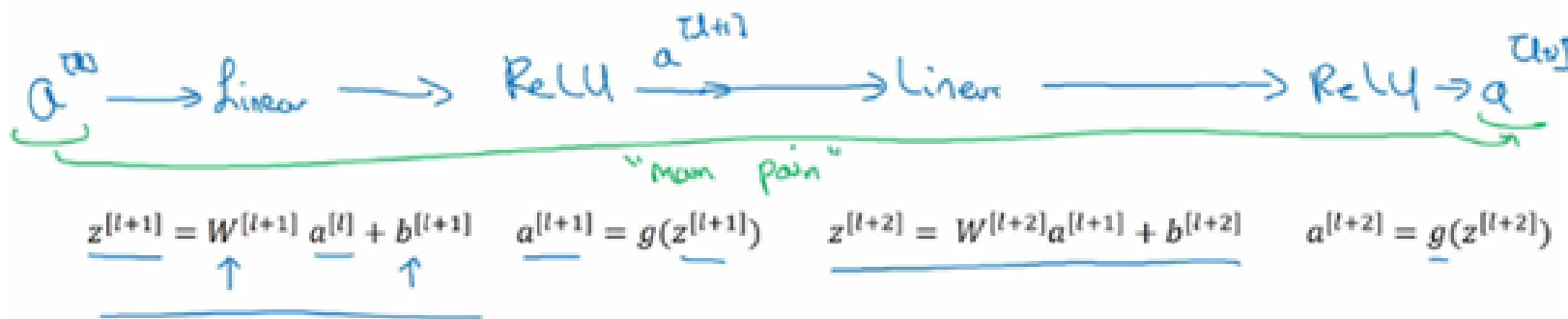
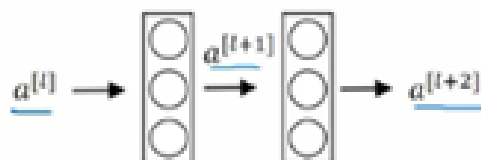
```
def inception_block(x,
```

```
[Train] Step: 8700, loss: 0.56888, acc: 0.75000
[Train] Step: 8800, loss: 0.65166, acc: 0.85000
[Train] Step: 8900, loss: 0.61417, acc: 0.80000
[Train] Step: 9000, loss: 0.41577, acc: 0.85000
(10000, 3072)
(10000,)
[Test ] Step: 9000, acc: 0.73700
[Train] Step: 9100, loss: 0.25869, acc: 0.95000
[Train] Step: 9200, loss: 1.00814, acc: 0.65000
[Train] Step: 9300, loss: 0.59531, acc: 0.75000
[Train] Step: 9400, loss: 0.42882, acc: 0.90000
[Train] Step: 9500, loss: 0.99593, acc: 0.55000
[Train] Step: 9600, loss: 0.70374, acc: 0.70000
[Train] Step: 9700, loss: 0.71996, acc: 0.70000
[Train] Step: 9800, loss: 0.75936, acc: 0.70000
[Train] Step: 9900, loss: 0.63364, acc: 0.85000
[Train] Step: 10000, loss: 0.57122, acc: 0.80000
(10000, 3072)
(10000,)
[Test ] Step: 10000, acc: 0.74600
```

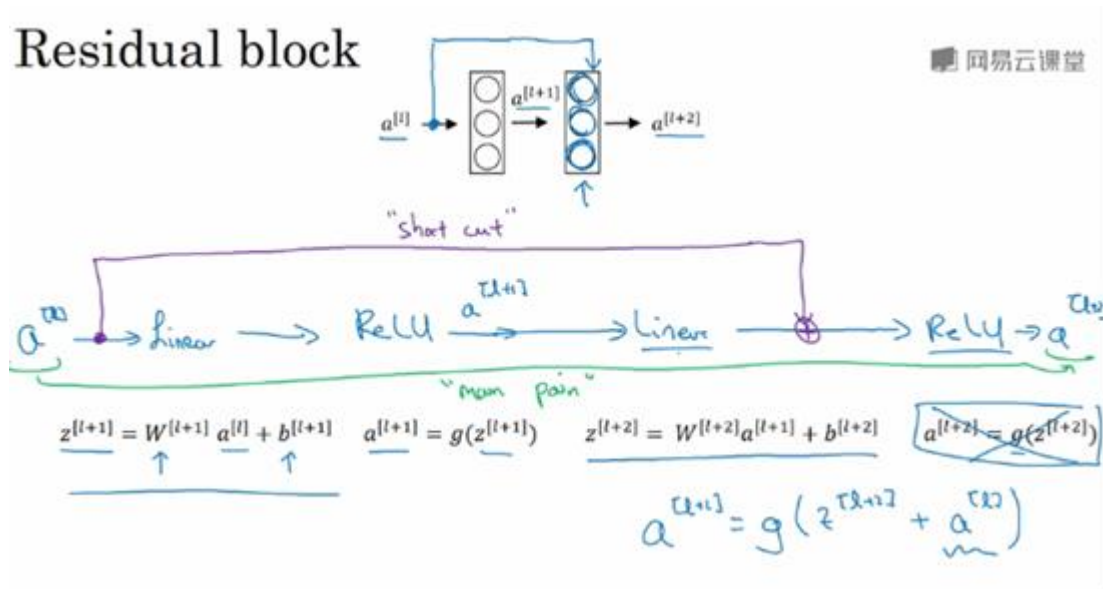
- 残差网络（Residual Networks, ResNet）
- 因为存在梯度消失和梯度爆炸（vanishing and exploding gradients），非常深的网络是很难训练的。
- ResNet由残差块（Residual block）组成，
- 信息流从 $a[l]$ 到 $a[l+2]$ ，普通的网络需要经过以下几个步骤，称为主路径。

Residual block

网易云课堂



- Shortcut/skip connection指 $a[l]$ 跳过一层或者好几层，从而将信息传递到神经网络的更深层。
- Shortcut/skip connection在进行ReLU非线性激活之前加上，如下图所示：



ResNet即使网络再深，训练的表现却不错

Shortcut使得我们很容易得出 $a[l+2]=a[l]$ ，这意味着即使给神经网络增加了这两层，它的效率也并不逊色于更简单的神经网络。因为只要使得新添加的两层的权重和偏置为0，那么新网络就跟原始网络效果是一样的。但是如果新添加的这些隐层单元学到一些有用信息，那么它可能比学习恒等函数表现更好。

```
test_filenames = [os.path.join(CIFAR10_PATH, f) for f in test_filenames]

train_data = CifarData(train_filenames, 50000)
test_data = CifarData(test_filenames, 10000)

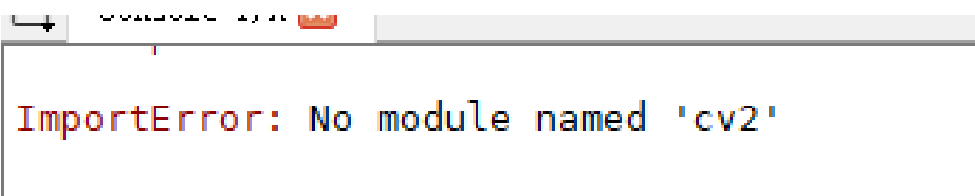
(50000, 3072)
(50000,)
(10000, 3072)
(10000,)
```

```
In [21]: def residual_block(x, output_channel):
```

```
[Train] Step: 8700, loss: 0.69903, acc: 0.75000
[Train] Step: 8800, loss: 0.57168, acc: 0.90000
[Train] Step: 8900, loss: 1.03144, acc: 0.65000
[Train] Step: 9000, loss: 1.16891, acc: 0.65000
(10000, 3072)
(10000,)
[Test ] Step: 9000, acc: 0.73650
[Train] Step: 9100, loss: 0.38002, acc: 0.85000
[Train] Step: 9200, loss: 0.56189, acc: 0.75000
[Train] Step: 9300, loss: 0.71686, acc: 0.65000
[Train] Step: 9400, loss: 0.76256, acc: 0.75000
[Train] Step: 9500, loss: 0.70114, acc: 0.80000
[Train] Step: 9600, loss: 0.67687, acc: 0.80000
[Train] Step: 9700, loss: 0.68563, acc: 0.70000
[Train] Step: 9800, loss: 1.21756, acc: 0.50000
[Train] Step: 9900, loss: 0.69737, acc: 0.85000
[Train] Step: 10000, loss: 0.44321, acc: 0.80000
(10000, 3072)
(10000,)
[Test ] Step: 10000, acc: 0.76750
```

即使网络再深，训练的表现却不错，准确率不会降。

- Yolo v1

A terminal window with a title bar containing a close button, a maximize button, and a red warning icon. The terminal displays the text "ImportError: No module named 'cv2'" in a monospaced font. A vertical cursor line is positioned at the end of the text.

```
ImportError: No module named 'cv2'
```

```
self._send_output(message_body)
File "/usr/lib/python2.7/httplib.py", line 897, in _send_output
    self.send(msg)
File "/usr/lib/python2.7/httplib.py", line 859, in send
    self.connect()
File "/usr/lib/python2.7/httplib.py", line 1278, in connect
    server_hostname=server_hostname)
File "/usr/lib/python2.7/ssl.py", line 353, in wrap_socket
    _context=self)
File "/usr/lib/python2.7/ssl.py", line 601, in __init__
    self.do_handshake()
File "/usr/lib/python2.7/ssl.py", line 830, in do_handshake
    self._sslobj.do_handshake()
IOError: [Errno socket error] [Errno 104] Connection reset by peer
(tensorflow) (cfolder) prj04@quo-PowerEdge-T630:~$
```

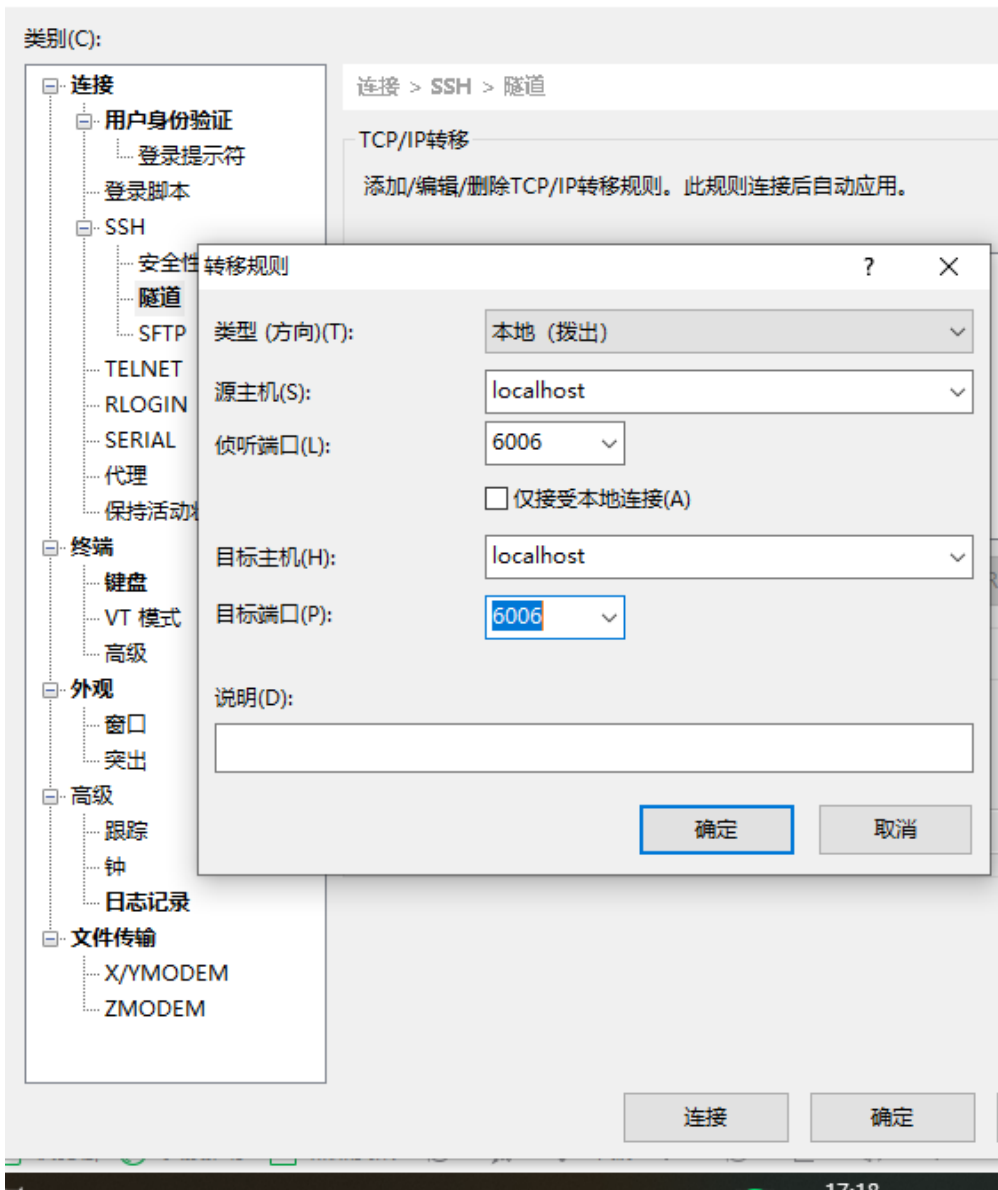
发现训练集import不进去

```
wget: 未找到命令
(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~$
(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~$ 没关系 http_proxy解救你，来这么一句：
没关系：未找到命令
(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~$
(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~$ wget https://s3.amazonaws.com/lasagne/recipes/datasets/mnist/
--2019-05-07 15:11:52-- https://s3.amazonaws.com/lasagne/recipes/datasets/mnist/
正在解析主机 s3.amazonaws.com (s3.amazonaws.com)... 52.216.232.21
正在连接 s3.amazonaws.com (s3.amazonaws.com)|52.216.232.21|:443... ^C
(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~$ wget https://s3.amazonaws.com/lasagne/recipes/datasets/mnist/
--2019-05-07 15:13:56-- https://s3.amazonaws.com/lasagne/recipes/datasets/mnist/
正在解析主机 s3.amazonaws.com (s3.amazonaws.com)... 52.216.165.221
正在连接 s3.amazonaws.com (s3.amazonaws.com)|52.216.165.221|:443... █
```

找到网址发现wget不成功

一种是网管登录

一种是防火墙外网



尝试在本地通过配置xshell的转移规则在本地打开tensorboard
从而更加形象化的查看训练情况，
例如分类训练率与识别准确率曲线等

后面通过一直开着对应服务，即可在本机端看见最近运行的模型的有关数据；代码也需要加；


```
100 weight: [0.29980946] bias: [0.10009873]
(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~$ tensorboard --logdir logs
TensorBoard 1.13.1 at http://guo-PowerEdge-T630:6006 (Press CTRL+C to quit)
^C(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~$ tensorboard --logdir=='model_dir logs
> '
TensorBoard 1.13.1 at http://guo-PowerEdge-T630:6006 (Press CTRL+C to quit)
I0507 17:22:23.870249 140216495912704 _internal.py:122] ::ffff:127.0.0.1 - - [07/May/2019 17:22:23] "GET / HTTP/1.1" 200 -
I0507 17:22:25.811170 140216504305408 _internal.py:122] ::ffff:127.0.0.1 - - [07/May/2019 17:22:25] "GET / HTTP/1.1" 200 -
```

远程转移规则成功，200即OK

```
1.11 min/avg/max/mdev = 2.595/2.719/2.902/0.171 ms
(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~$ git clone https://github.com/hizhangp/yolo_tensorflow.git
正克隆到 'yolo_tensorflow'...
remote: Enumerating objects: 117, done.
remote: Total 117 (delta 0), reused 0 (delta 0), pack-reused 117
接收对象中: 100% (117/117), 199.65 KiB | 191.00 KiB/s, 完成.
处理 delta 中: 100% (60/60), 完成.
检查连接... 完成.
(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~$
```

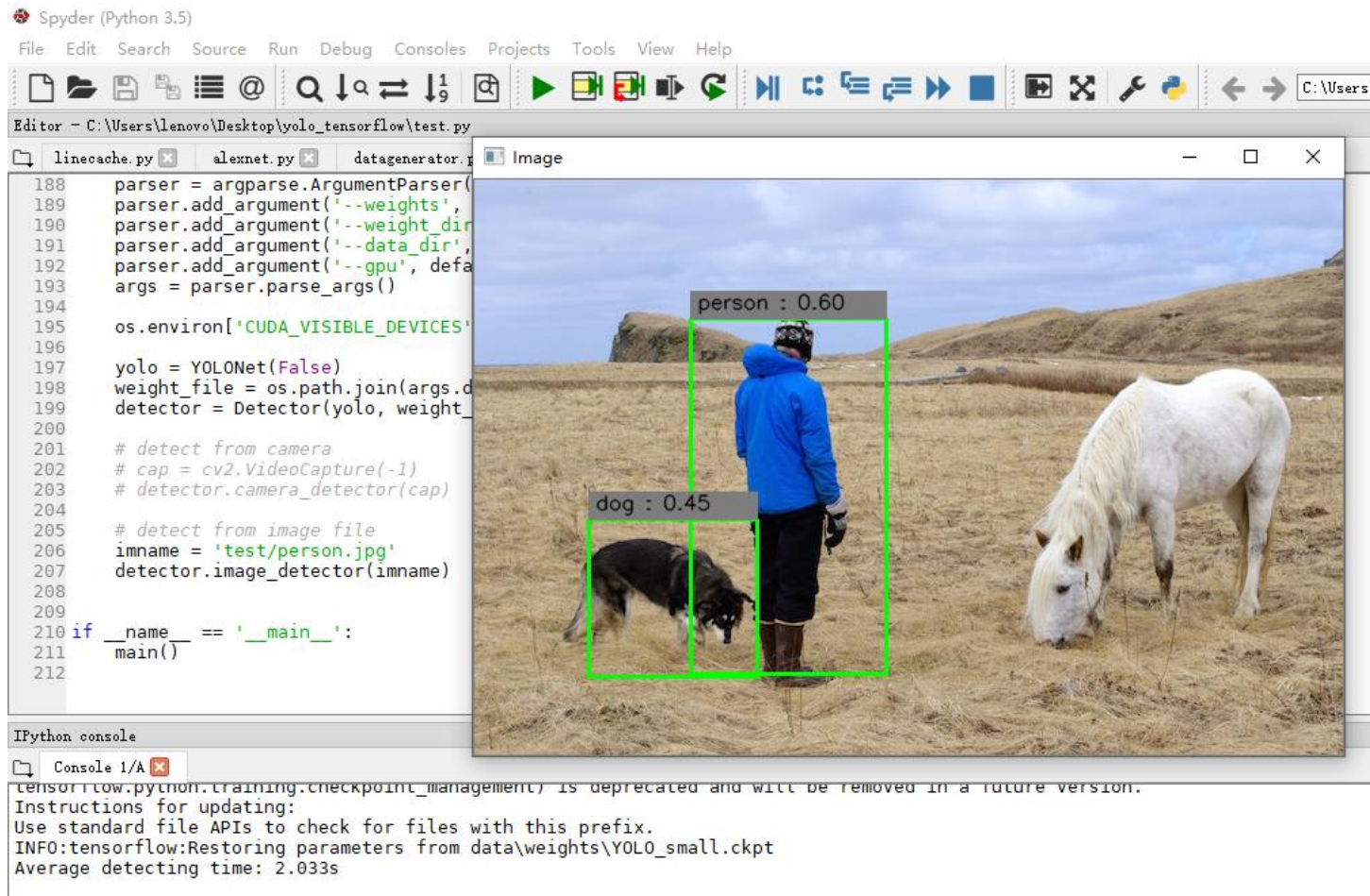
从github上下载代码，关于git安装与配置，之前实验报告已说明

```
(tensorflow) (cfolder) prj04@guo-PowerEdge-T630:~/yolo_tensorflow$ ./download_data.sh
Creating data directory...
Downloading Pascal VOC 2012 data...
URL transformed to HTTPS due to an HSTS policy
--2019-05-07 19:13:19-- https://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar
正在解析主机 pjreddie.com (pjreddie.com)... 128.208.4.108
正在连接 pjreddie.com (pjreddie.com)|128.208.4.108|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 460032000 (439M) [application/octet-stream]
正在保存至: "VOCtrainval_06-Nov-2007.tar"

VOCtrainval_06-Nov-2007.tar          7%[====>] 33.63M 3.34MB/s
```

用linux系统通过sh文件进行文件下载，其实打开sh文件以后，发现其中其实就是关于新建文件夹，用wget获取数据，以及解压tar文件（即训练集），此处是预训练模型，可以直接运行test.py文件；所以下一步可以去尝试下自己去使用前面的模型去分类；因为c10那个层次只有四层，识别10类，所以考虑大改下。

成功界面



- 还是有很多中途的问题（此处GPU配置问题就不多说了；ANA+TEN+CV2）
- 首先，在windows系统中直接下载训练好的模型面临着有关sh文件的调用问题，与LINUX直接输入不同，本机端需要运行git下的sh.exe，或者运行git bash（相当于window10中附属小型虚拟机），不过缺点是本身环境过小，连weget都要自己安。
- 还有一个方法就是用虚拟机下下来以后用winscp传到本机（比我之前用lrzsz方便）
- 在Gpu下载文件当中也出现了问题，首先访问cloud google网需要外网，而我没有权限关防火墙；于是我想到了用代理服务器，网上有每几分钟就能更新到的访问外网对应代理服务器及端口，但还是很慢，有的连着连着就丢包；解决方法转向百度云，但百度云一下载大文件会自动转到百度网盘下载，很难捕捉到可以直接用wget - -refere的网址；
- 有以下方法，理论上可直接用浏览器下载记录获取地址；用页面审查，输入代码改变下载方式，通过对NETWORK中报文302类型捕捉，找到对应类型的200的网址，即可用-o进行下载，不过百度云做的防护措施是所有云里最严的，很容易下不完整；于是我最终投降，转向本机上传。。。不过百度云好慢，后期考虑与虚拟机结合改变下载方式？