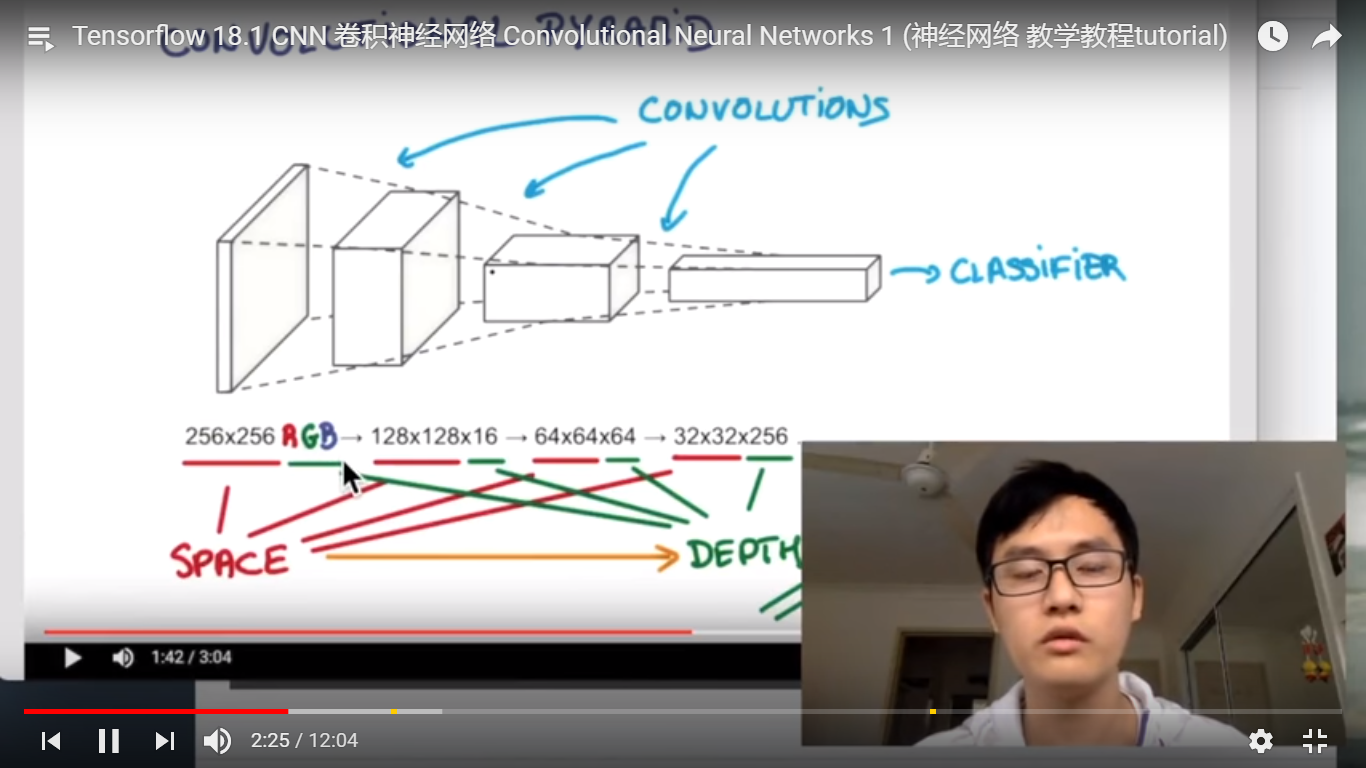
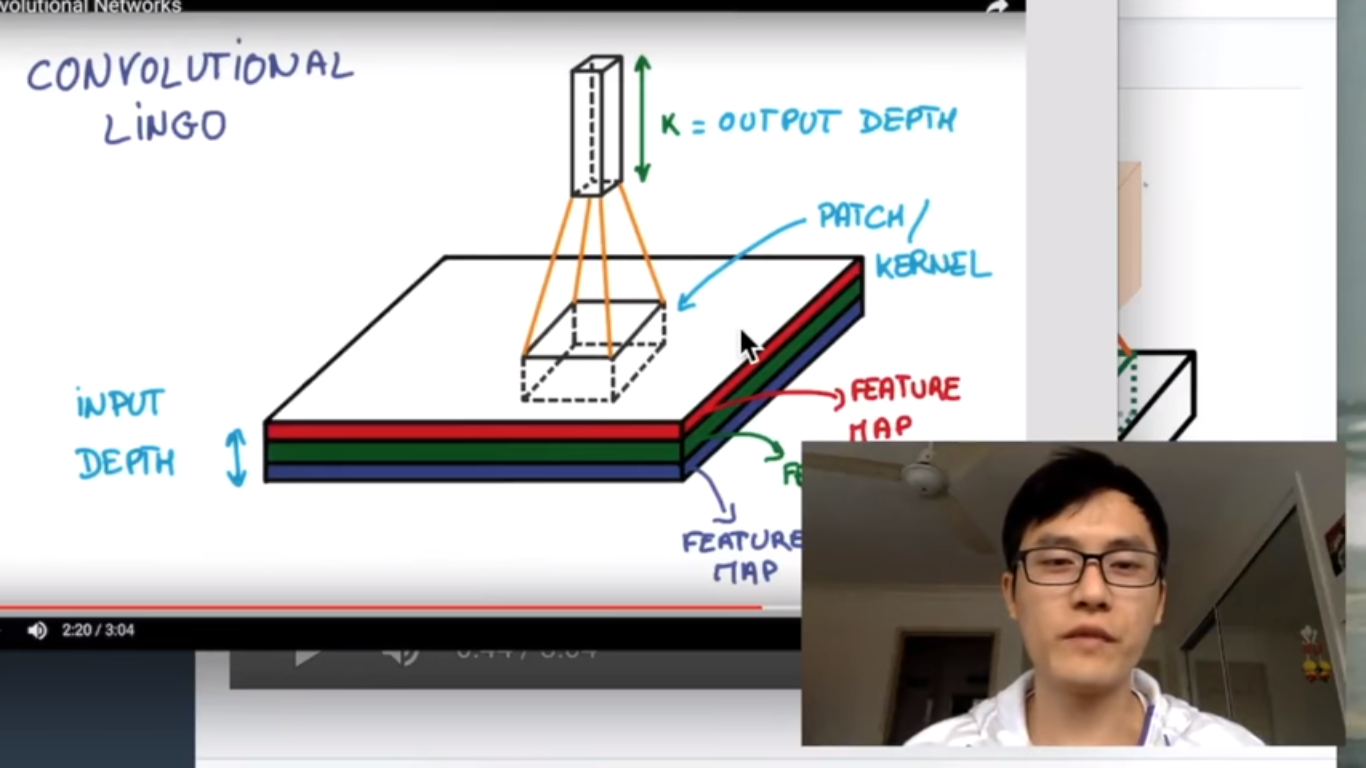
CNN学习

1.压小一点，增高一点

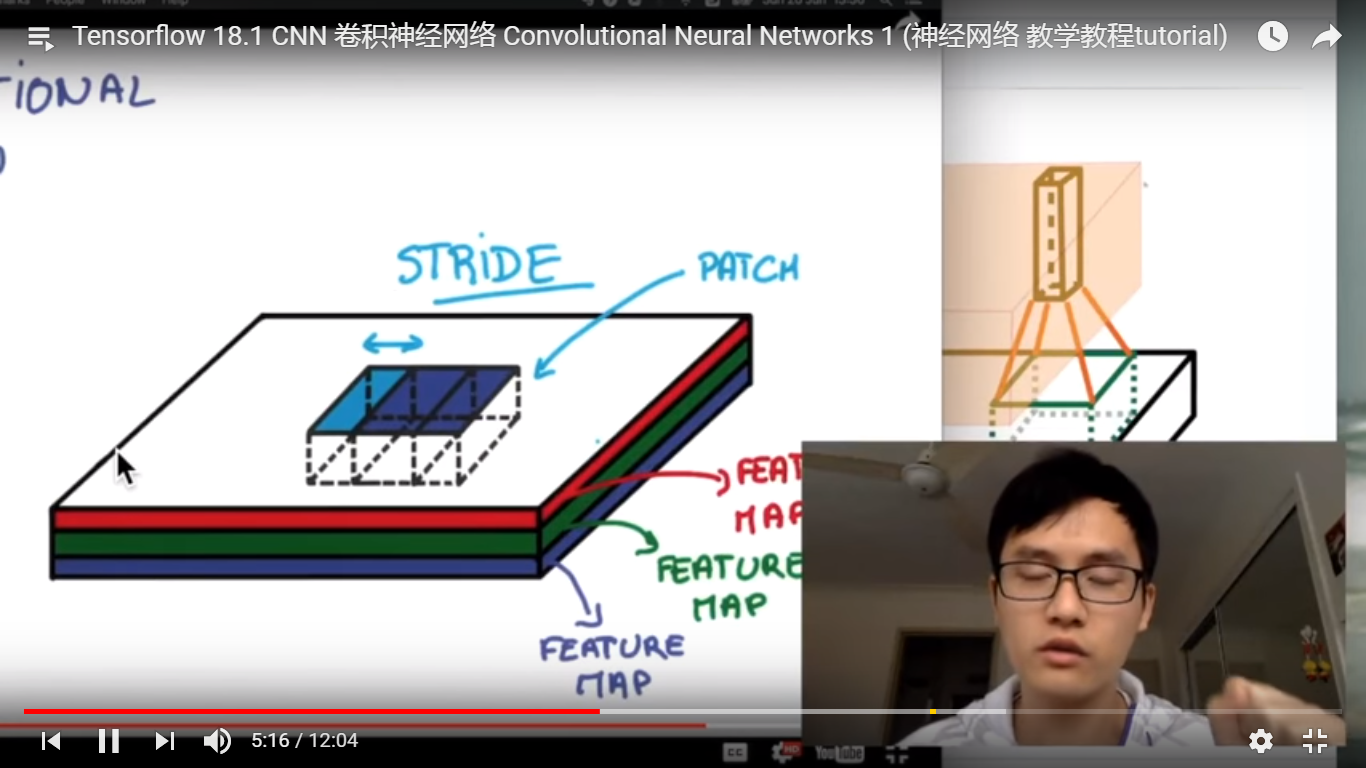


Classifier ---[0 0 0 0 1 0 0 0 0]

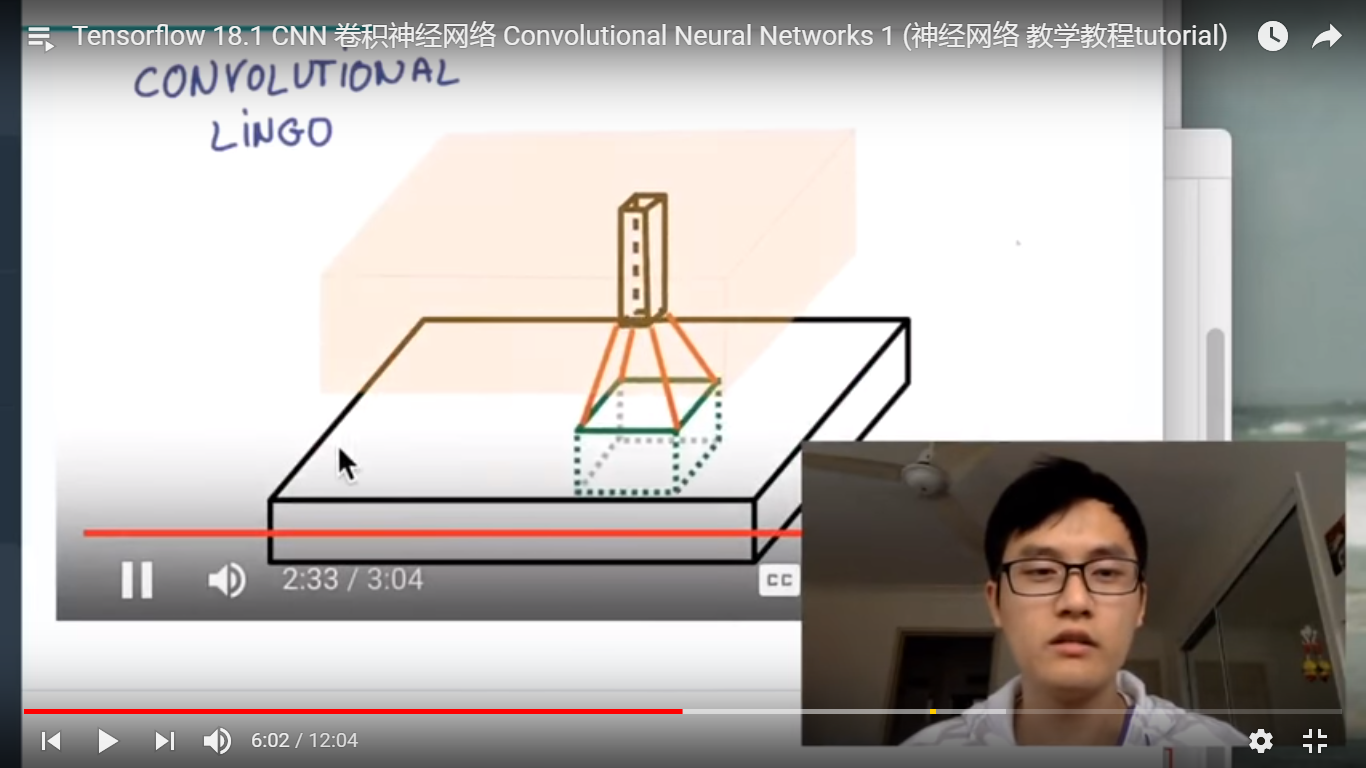


长度\*宽度\*厚度 ======厚度越来越大

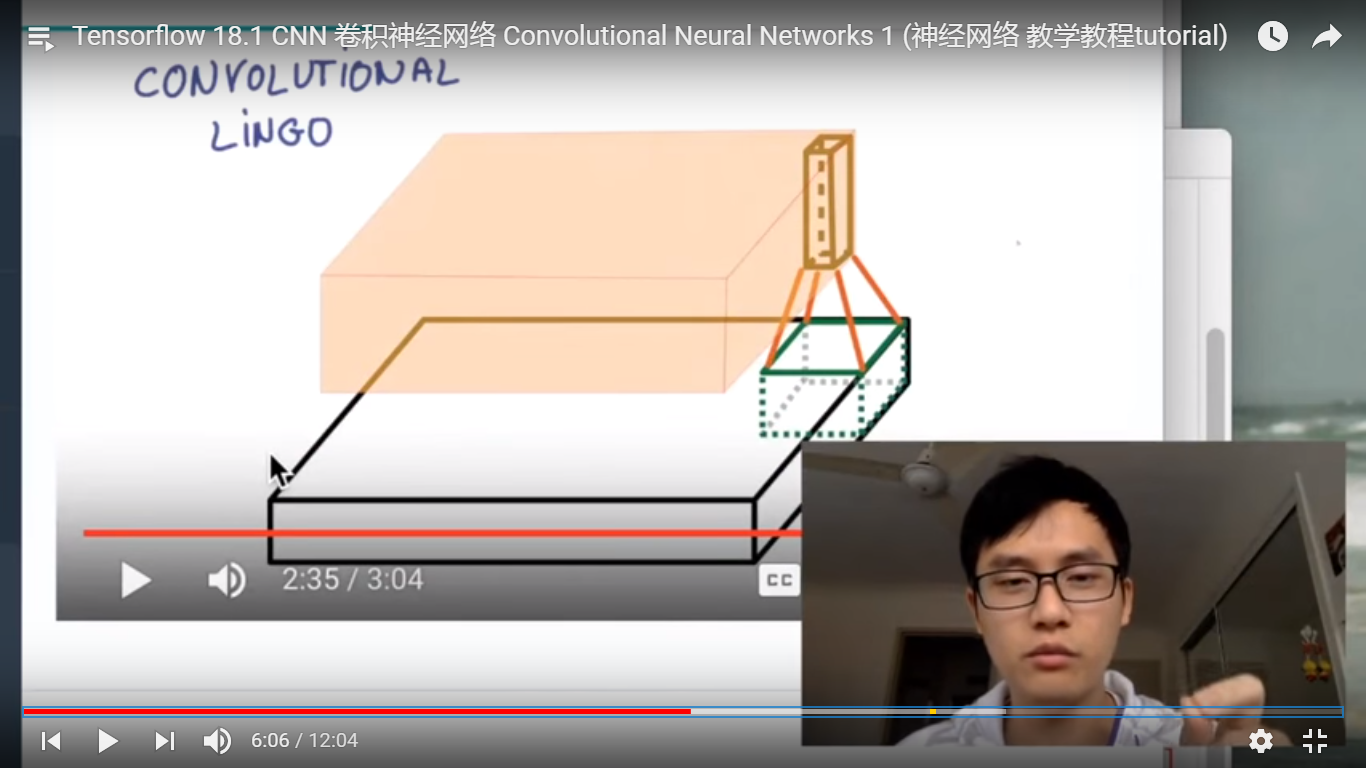
Patch：抽离一部分分析



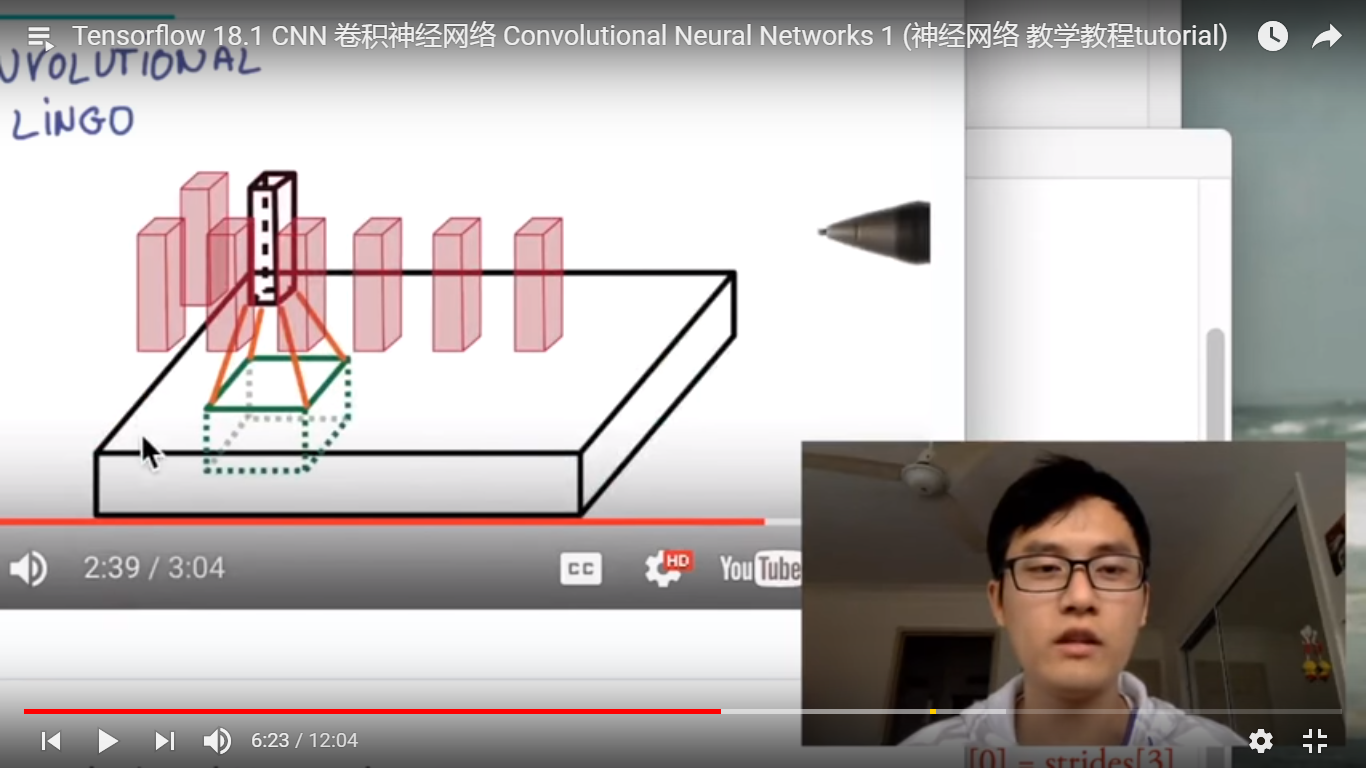
Stride 每隔一定的像素点去分析



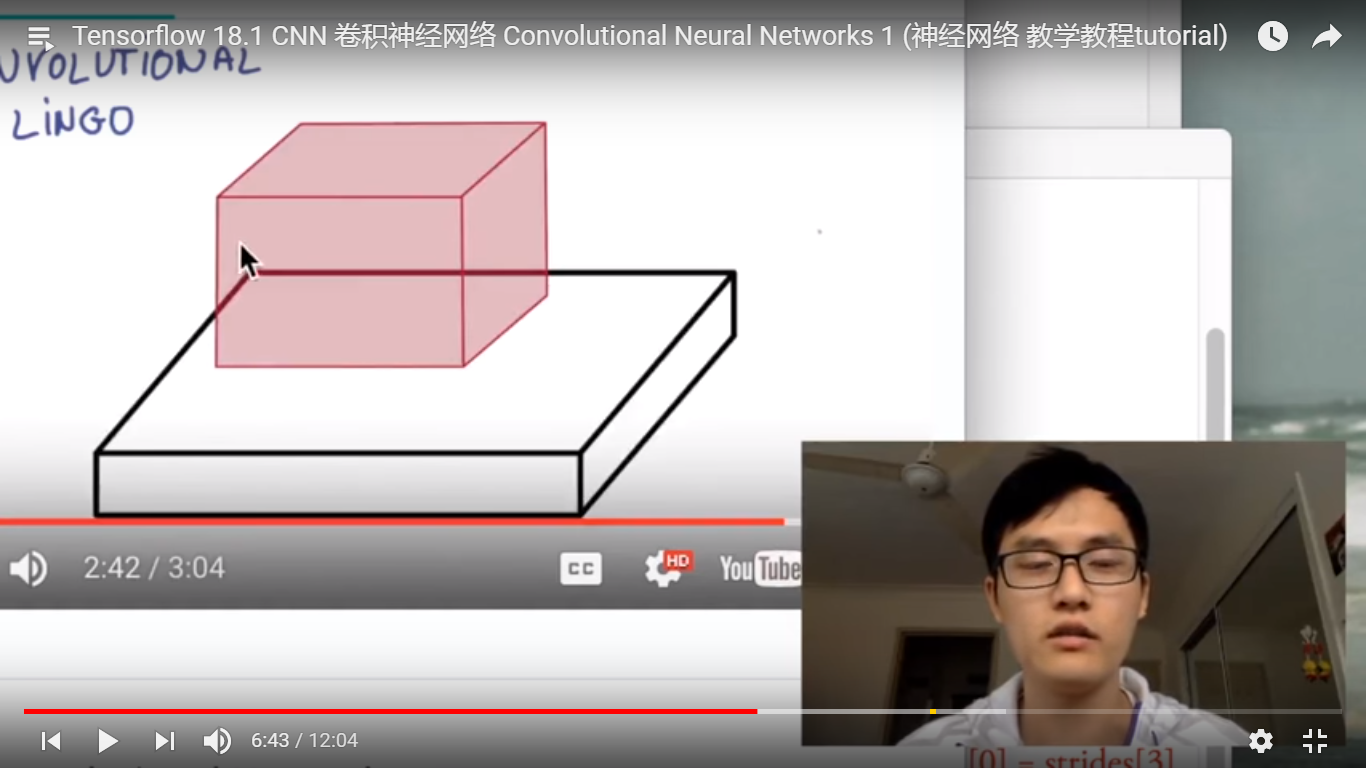
如何跨度：从左到右，从前到后，每隔像素点跨度



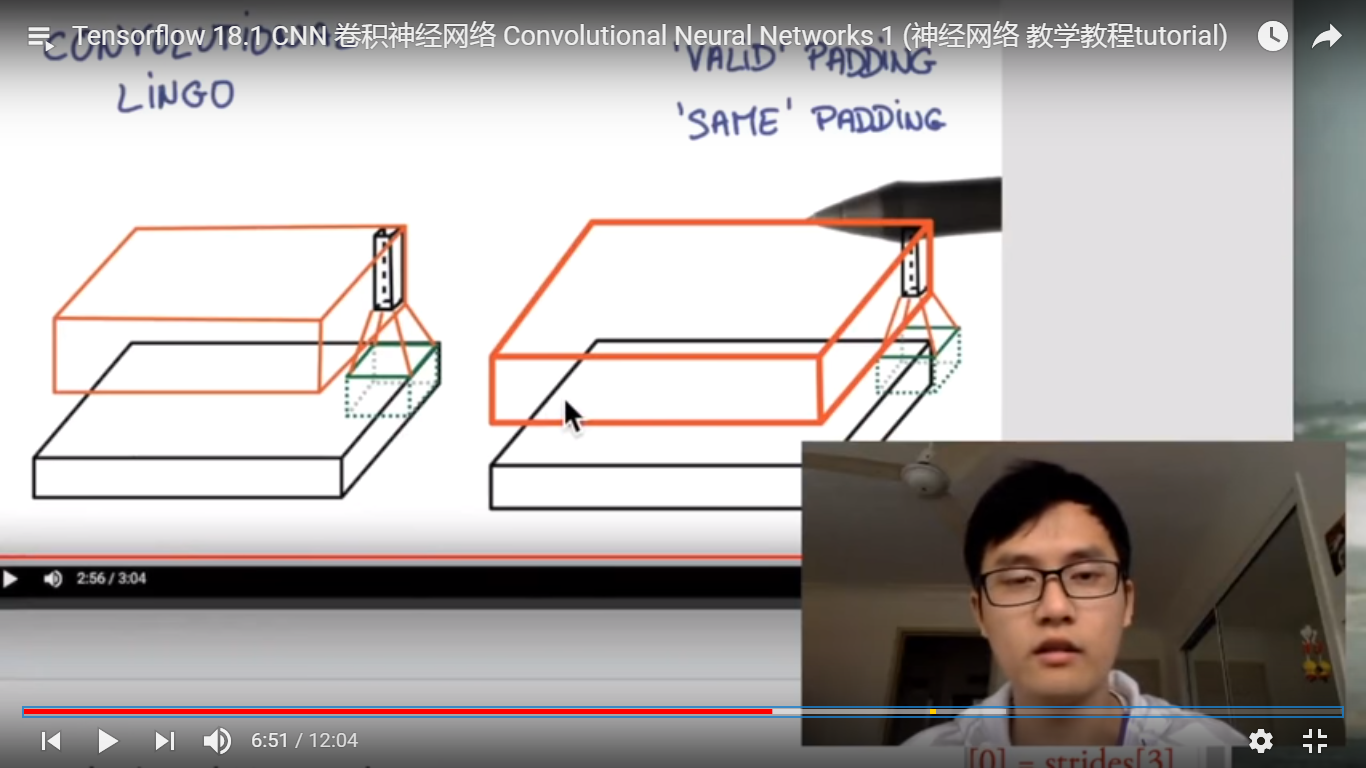
一次跨两步，长和宽进行压缩，但是高增加了：



压缩完合并，成为更小的一块立方体，这包含了图片中的所有信息



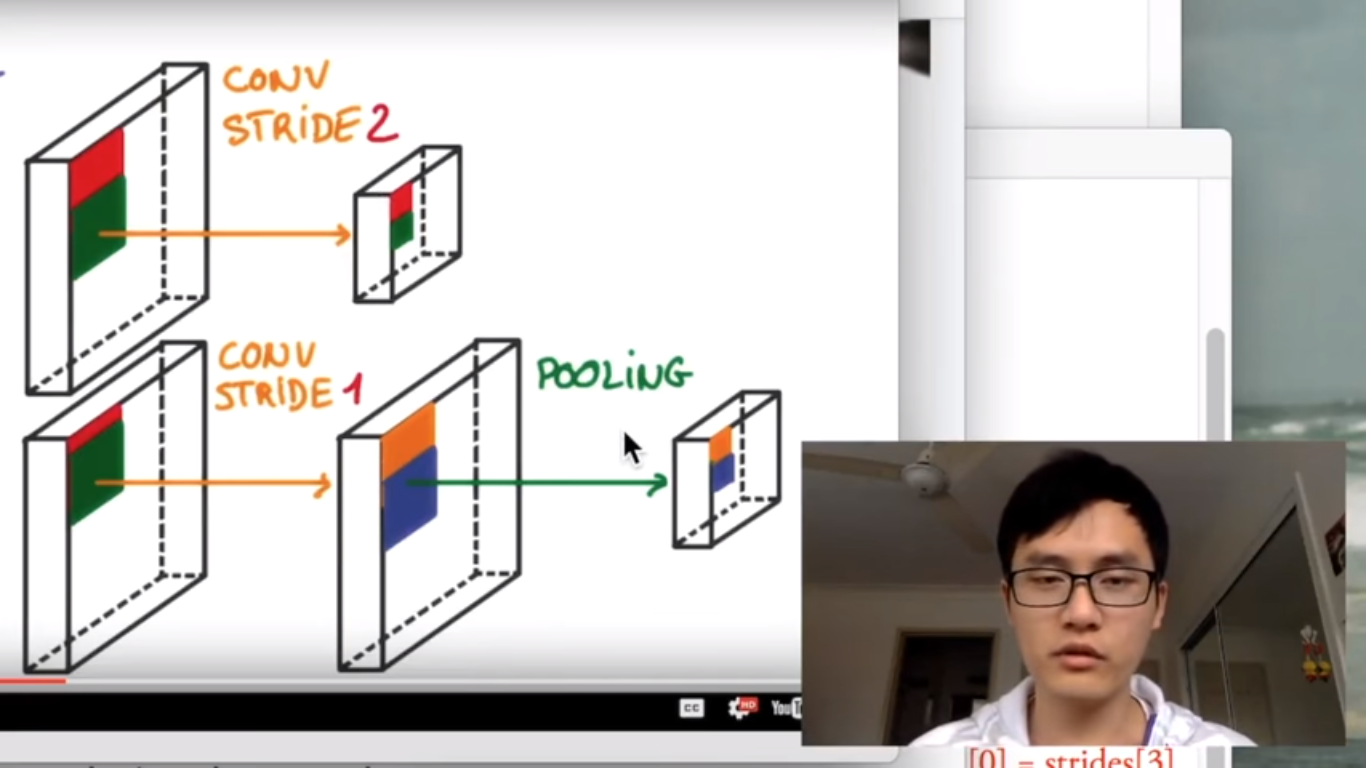
抽离信息的方式：padding



Valid padding：抽出来的这一层可能比之前的尺寸要小一点

Same padding：抽离出来的这一层和我图像是一样的长和宽

Pooling：

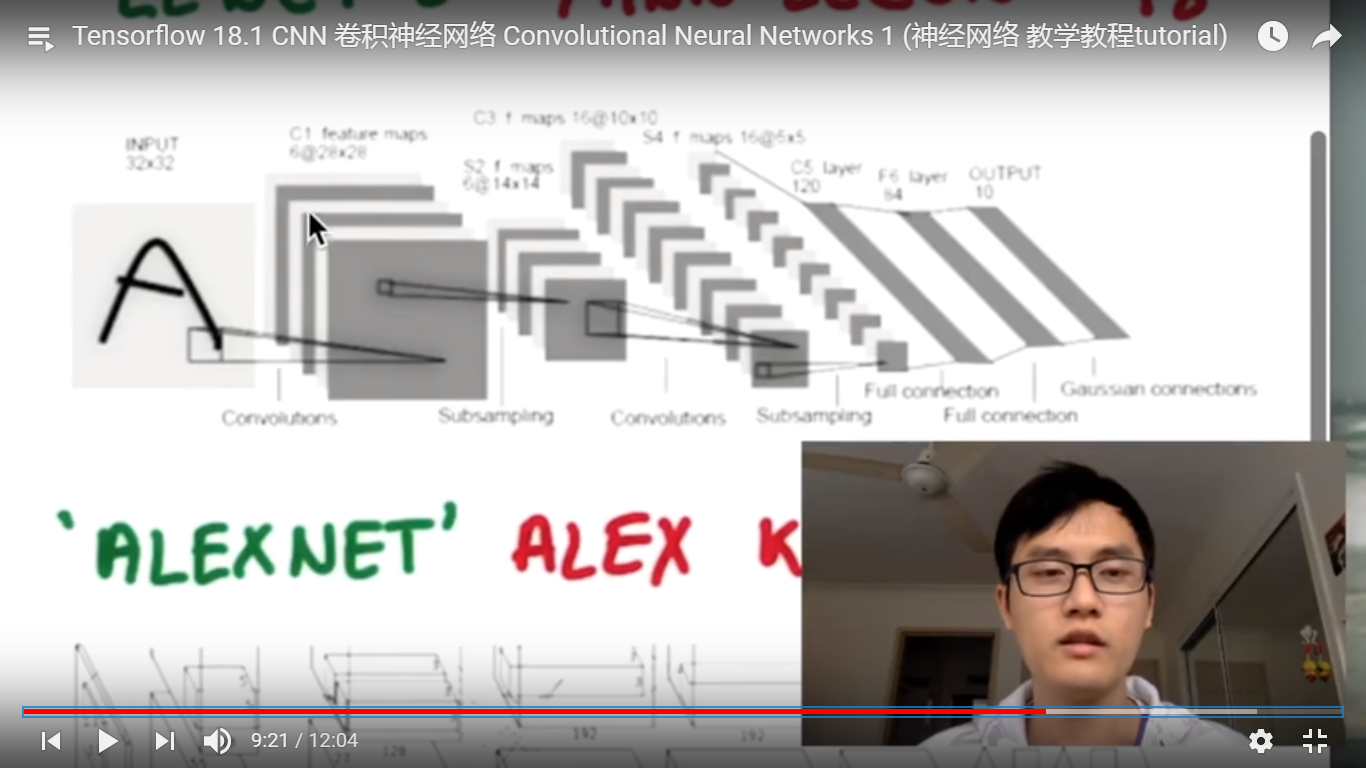


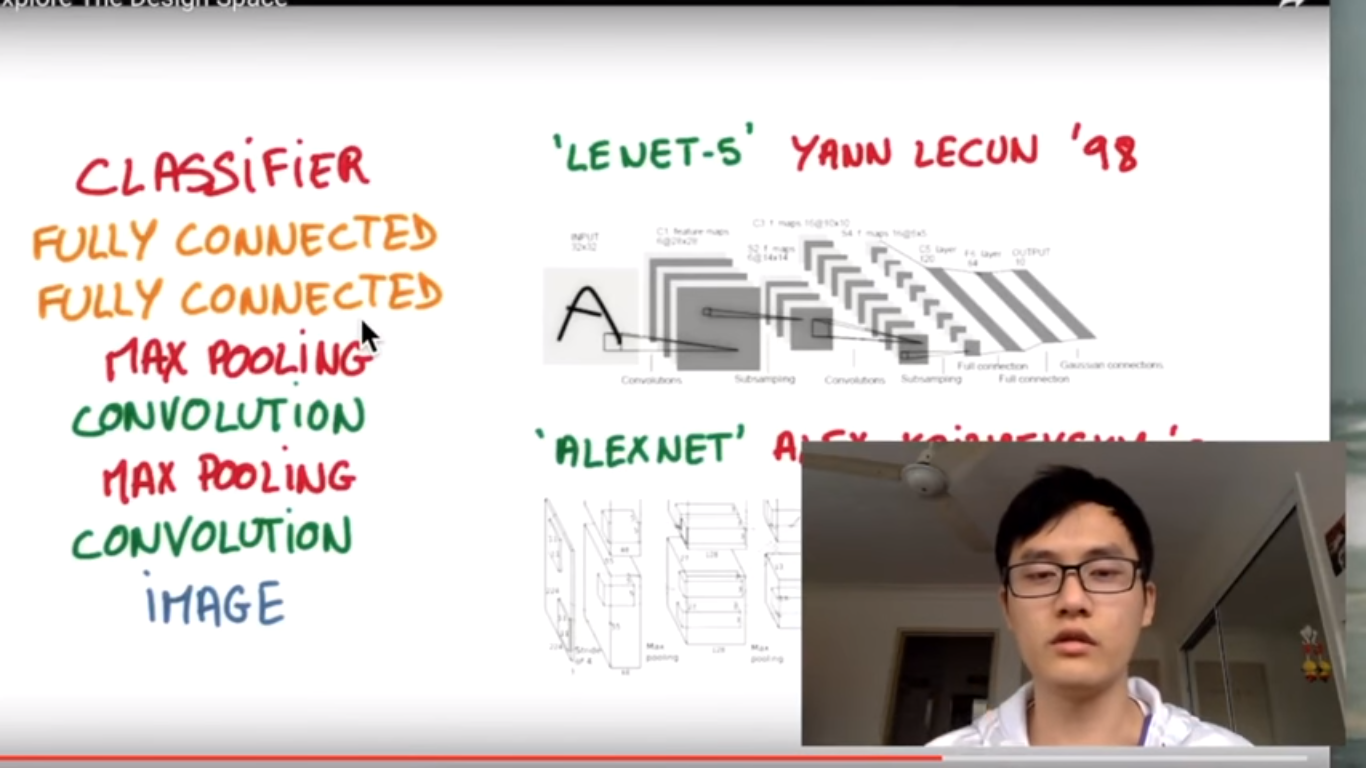
Pooling分两种：max pooling 和average pooling

用pooling处理跨度大的问题：

跨度2 == 跨度1 + pooling 2\*2 处理结果形状一样，但是可以保留更多的图片信息

CNN 工作方式：

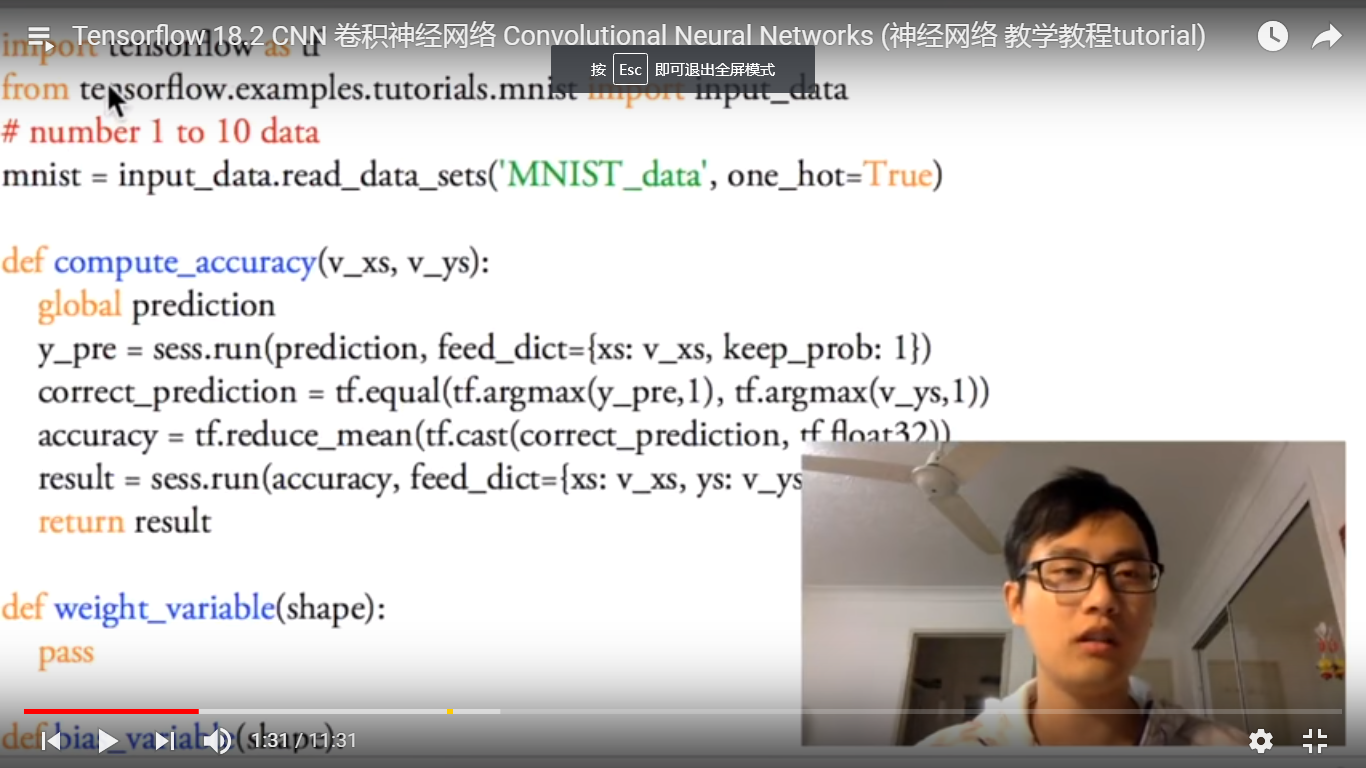




原来厚度=3为RGB，然后长和宽不断减小，厚度不断增加

然后进行fully connection 然后进行分类的处理

Code：



*#使用CNN训练MNIST数据集  
#2018.03.06 21:14  
#https://www.youtube.com/watch?v=JCBe\_yjDmY8&index=27&list=PLXO45tsB95cKI5AIlf5TxxFPzb-0zeVZ8***import** tensorflow **as** tf  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** tensorflow.examples.tutorials.mnist **import** input\_data  
*#载入数据集*mnist = input\_data.read\_data\_sets(**'MNIST\_data'**, one\_hot=**True**)  
  
**def** computer\_accuracy(v\_xs,v\_ys):  
 **global** prediction  
 y\_pre = sess.run(prediction,feed\_dict={xs:v\_xs, keep\_prob :1})*#dropout =1* correct\_prediction = tf.equal(tf.arg\_max(y\_pre,1),tf.arg\_max(v\_ys,1))  
 accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,tf.float32))  
 result = sess.run(accuracy,feed\_dict= {xs:v\_xs,ys:v\_ys,keep\_prob:1})  
 **return** result  
  
**def** weight\_variable(shape):  
 initial = tf.truncated\_normal(shape,stddev=0.1)*#normal distrubition* **return** tf.Variable(initial)  
  
**def** bias\_variables(shape):  
 initial = tf.constant(0.1,shape=shape)*#bias usually>0 =0.1 is good* **return** tf.Variable(initial)  
  
**def** conv2d(x,W):  
 *#卷积神经网络层，x:图片，2d:二维图片  
 #stride[1,x\_movement,y\_movement,1]  
 #Must have strids[0] = strides[3] = 1* **return** tf.nn.conv2d(x, W, strides=[1,1,1,1], padding=**'SAME'**)  
  
  
  
  
  
**def** max\_pool\_2x2(x):  
 *#X:input：指卷积需要输入的参数，具有这样的shape[batch, in\_height, in\_width, in\_channels]，  
 # 分别是[batch张图片, 每张图片高度为in\_height, 每张图片宽度为in\_width, 图像通道为in\_channels]。  
 #stride[1,x\_movement,y\_movement,1]  
 #ksize =filter：指用来做卷积的滤波器，当然滤波器也需要有相应参数  
 #滤波器的shape为[filter\_height, filter\_width, in\_channels, out\_channels]，  
 # 分别对应[滤波器高度, 滤波器宽度, 接受图像的通道数, 卷积后通道数]，  
 # 其中第三个参数 in\_channels需要与input中的第四个参数 in\_channels一致，  
 # out\_channels第一看的话有些不好理解，如rgb输入三通道图，  
 # 我们的滤波器的out\_channels设为1的话，就是三通道对应值相加，最后输出一个卷积核。* **return** tf.nn.max\_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding=**'SAME'**)  
  
  
  
*#define placeholder for inputs to network*xs = tf.placeholder(tf.float32,[**None**,784])*#28\*28*ys = tf.placeholder(tf.float32,[**None**,10])*#0-9*keep\_prob = tf.placeholder(tf.float32)*#dropout*x\_image = tf.reshape(xs,[-1,28,28,1])*#[样本数-1为不管其为多少，28,28,1通道数，只有黑色为1，grb为3]*print(x\_image.shape) *#[n\_samples,28,28,1]  
  
  
##conv1 layer*W\_conv1 = weight\_variable([5,5,1,32])*#patch = 5\*5 , insize =1 image的厚度为1，outsize 高度 32*b\_conv1 = bias\_variables([32])  
*#CNN第一个卷积层*h\_conv1 = tf.nn.relu(conv2d(x\_image,W\_conv1)) + b\_conv1 *#output = 28\*28\*32  
#tf.nn.relu 非线性话处理,计算修正线性单元(非常常用)  
#ReLU（Rectified Linear unit）激活函数最近变成了神经网络中隐藏层的默认激活函数。  
#这个简单的函数包含了返回max(0,x)，所以对于负值，它会返回0，其它返回x。  
#pooling*h\_pool1 = max\_pool\_2x2(h\_conv1) *#output = 14\*14\*32  
  
##conv2 layer*W\_conv2 = weight\_variable([5,5,32,64]) *#patch = 5\*5,insize = 32,outsize =64*b\_conv2 = bias\_variables([64])  
h\_conv2 = tf.nn.relu(conv2d(h\_pool1,W\_conv2)) + b\_conv2 *#outsize = 14\*14\*64*h\_pool2 = max\_pool\_2x2(h\_conv2) *#outsize = 7\*7\*64  
  
##fun1 layer  
#神经网络层定义，接pooling2后面*W\_fc1 = weight\_variable([7\*7\*64,1024]) *#1024使其更高*b\_fc1 = bias\_variables([1024])  
*#[n\_samples,7,7,64] ->> [n\_samples,7\*7\*64]*h\_pool2\_flat = tf.reshape(h\_pool2,[-1,7\*7\*64])  
h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat,W\_fc1) + b\_fc1)  
*#有overfitting过拟合处理，加dropout处理*h\_fc1\_drop = tf.nn.dropout(h\_fc1,keep\_prob)  
  
*##fun2 layer*W\_fc2 = weight\_variable([1024,10])  
b\_fc2 = bias\_variables([10])  
prediction = tf.nn.softmax(tf.matmul(h\_fc1\_drop,W\_fc2) + b\_fc2)  
  
  
*#the erroe between prediction and real data  
  
#交叉信息熵损失*cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(ys \* tf.log(prediction),reduction\_indices=[1])) *#loss*train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)  
  
sess = tf.Session()  
  
*#important step  
#init = tf.initialize\_all\_variables()*init = tf.global\_variables\_initializer()  
sess.run(init)  
  
**for** i **in** range(1000):  
 batch\_xs,batch\_ys = mnist.train.next\_batch(100)  
 sess.run(train\_step, feed\_dict={xs:batch\_xs, ys:batch\_ys, keep\_prob: 0.5})  
 **if** i % 50 == 0:  
 print(computer\_accuracy(mnist.test.images,mnist.test.labels))