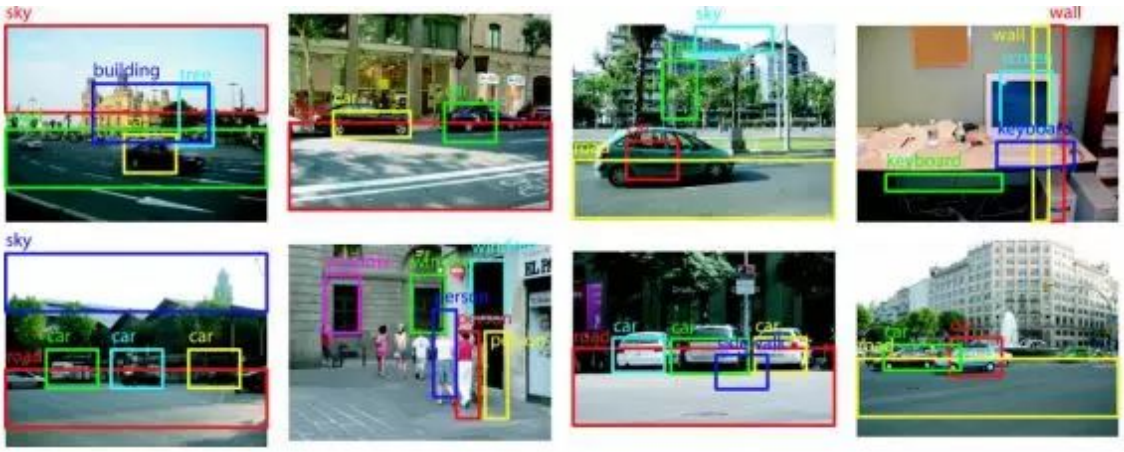
卷积神经网络实战（附数据集&学习资料）

深度学习是目前最热门的人工智能话题之一。它是部分基于生物学解释的算法合集，在计算机视觉、自然语言处理、语音识别等多个领域均展现了令人惊叹的成果。

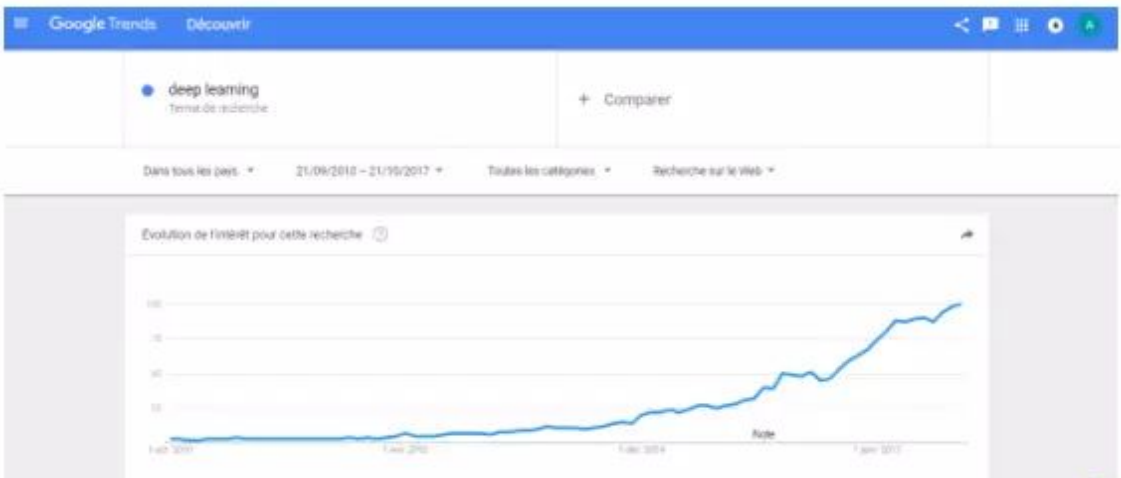
在过去的五年中，深度学习已经拓展到很多工业领域。

最近很多科技性突破都得归功于深度学习，例如特斯拉无人驾驶汽车、Facebook照片标记系统、Siri和Cortana等虚拟助手、聊天机器人和物体识别照相机。在**语言理解**和**图像分析**的认知领域，深度学习已经达到人类水平。

下面一个例子很好的说明了深度学习算法能做到什么：自动识别和标记场景里的不同物体。



深度学习都已经成了媒体上老生常谈的话题了。



我会在本文中跳过那些主流媒体的炒作内容并向你展现深度学习的实际应用案例。

我将告诉你如何搭建一个在图片识别分类上达到90%准确度的深度神经网络。在深度网络尤其是卷积神经网络（CNN）出现之前，这个看似简单的问题已经困扰计算机科学家很多年了。

本文分为四个部分：

1.呈现数据集和应用案例，解释图片分类的复杂性

2.详细说明卷积神经网络。分析其内在机制，并阐述其在图片分类方面相对普通神经网络的优越性。

3.基于AWS拥有强大GPU的EC2实例，搭建深度学习专门环境

4.训练两个深度学习模型：一个是从零开始在端对端管道中使用Keras和Tensorflow，另一个是使用大型数据集上的预训练网络。

这些部分互相独立。如果你对理论不感兴趣，可以跳过第一部分和第二部分。

深度学习是个很有挑战性的话题。作为一名机器学习实践者，我花了很长时间去了解这个学科。我会在本文结尾分享以前看过的学习资料，这样你们也可以自学并开始你们的深度学习之旅。

本文是一次整合我在神经网络领域所有知识的尝试。阅读时若发现不当之处，请及时指出。若有不明白之处和新想法，欢迎和我讨论。

本文的代码和训练模型都在我的Github账户里（https://github.com/ahmedbesbes/Understanding-deep-Convolutional-Neural-Networks-with-a-practical-use-case-in-Tensorflow-and-Keras），欢迎大家来查看和修改。

那就开始吧。

1.一个有趣的例子：如何区分猫和狗?

有很多图片集专门用来基准测试深度学习模型。本文选择的图片集是“Kaggle猫狗大战”（https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition）。通过名字你也能猜到，这是一个猫狗标记图片集。

像其他的Kaggle比赛一样，我们会有两个文件夹：

- **训练数据文件夹：**包含了25，000张猫和狗的图片。每张图片都在其文件名中有标记。我们会用它来训练和验证我们的模型。
- **测试数据文件夹：**包含了12，500张图片，根据数值ID来命名。对于数据集中的每一张照片，都要预测该图片中动物是狗的概率（1表示是狗，0表示是猫）。实际操作中，它被用来在Kaggle的排行榜上给模型打分。

你可以看到，我们有很多图片，分辨率、拍摄角度和焦距各不相同。所有猫狗的形态、位置和颜色互相迥异，或坐着，或站着，或开心，或难过，或睡觉，或吠叫。

特征可以是无穷无尽的。但人类可以不费力气的在一堆不同的照片里识别场景里的宠物。然而这对于机器来说并不简单。自动分类需要知道如何强有力地描述出猫和狗各自的特征。需要知道能够描述每只动物的本质特征。

深度神经网络在图片分类上非常有效是因为它在分类任务中多层提取类别特征的能力，并能抵抗失真和简单几何变形。

那深度神经网络是如何做到的?

2.全连接网络vs卷积神经网络

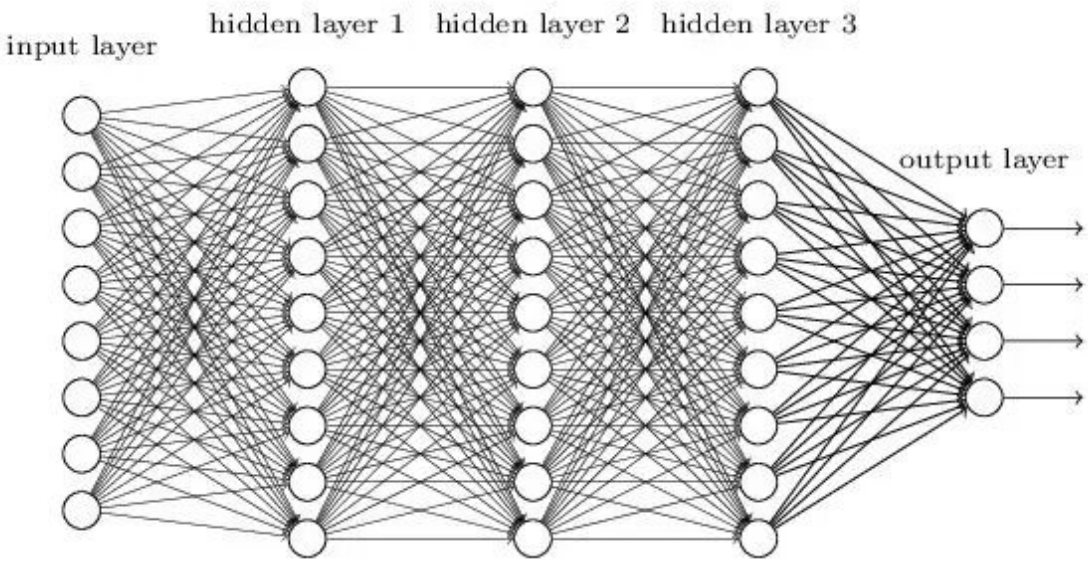
很多人一开始会选择全连接网络来解决图片分类问题。然而逐渐地他们发现这种网络并不是完成任务的最好方法。

我们来分析下为何会这样。

2.1一个全连接（FC）神经网络

全连接神经网络是一种相邻层之间每个神经元都彼此连接的网络。它是标准的传统神经网络架构。若想了解神经网络背后更多的原理，请参考Andrej Karpathy在斯坦福大学非常棒的讲义（http://cs231n.github.io/neural-networks-1/；http://cs231n.github.io/neural-networks-2/）。

为了便于阐述，下面是一个三层隐藏层全连接神经网络。



使用全连接网络，图片会首先转成一维矢量，然后作为输入量输入网络。

例如，一张256x256的彩色照片，内容是一个(255, 255, 3)的物体，3是指颜色通道数目，转成一个256 x 256 x 3 = 196608的矢量。我们做的是把图片转成了长矢量，矢量的每个元素都是像素值。

使用全连接网络分析一系列256x256的彩色照片，我们可以得到：

- 一层尺寸为196608的输入层，其中每个神经元都编码了一个像素值
- 一层尺寸为2的输出层，每个神经元都显示着输出类别预估情况
- 隐藏层和其中的隐藏神经元

全连接网络是很好的分类工具。在监督算法领域，它可以学习复杂非线性图案，总结能力很强，当然前提是架构稳定，没有数据过度拟合。

对于处理图片，全连接网络就不是最合适的工具了。

主要有两点原因：

1. 我们来假设一个有1000个隐藏单元的隐藏层，考虑到输入层尺寸，1000是一个合理值。在这种情况下，连接输入层和第一隐藏层的参数数目达到196608 x 1000 = 196608000!不仅仅是数目太大的问题，由于神经网络一般需要不止一个隐藏层来确保稳固，所以网络不可能运作很好。说实在的，网络能够拥有这样的隐藏层和1000个隐藏单元，真的非常不错了。但是我们来计算下存储成本。一个参数是8字节的浮动值，196698000个参数就是1572864000个字节，大约**1,572 GB**。因此我们需要1,572 GB来存储仅第一隐藏层的参数。除非你电脑的RAM非常大，不然这个方案很明显不可拓展。

2. 使用全连接网络，我们会丢失图片本有的空间结构信息。事实上，把图片转成长矢量之后，每个像素值的处理方式都很相似。无法找到像素之间的空间关联。每个像素都扮演同样的角色。我们丢失了相近像素间的关联性和相似性，这是很严重的信息损失。这些信息我们都希望能够编码到模型中的。

为了克服这两个局限，很多工作都投入到创造既能拓展，又适合处理图片数据的复杂性的新型神经网络架构中。。

于是我们创造了了**卷积神经网络（CNN）**。

2.2卷积神经网络

卷积神经网络是种特殊的神经架构，专门用来处理图片数据。自1989年被LeCun et al引进后，卷积神经网络已经在手写数字分类和面部识别等方面表现优秀。在过去的几年中，好几篇论文都表明了它在更有挑战性的视觉分类中也有不俗表现。最有名的要属Krizhevsky在2012年ImageNet的分类测试中破纪录的表现，他的卷积神经网络模型AlexNet的错误率只有16.4%，而第二名的错误率是26.1%。

卷积神经网络不只是媒体炒作，多方面因素能解释为何这么多人会对它感兴趣。

1.开放的大型训练数据集，数以百万计的标签例子。最有名的数据库之一就是ImageNet。

2.强大的GPU性能，使得大型模型训练切实可行。

3.提升的模型规则化策略，如Dropout（https://www.youtube.com/watch?v=UcKPdAM8cnI）。

卷积神经网络在图片分类中作用非常强大，专门用于解决之前所说的全连接网络面临的两大局限。

卷积神经网络有自己的架构和属性。虽然和标准的全连接网络看起来不同，但是两者的机制是一样的。我们会谈到隐藏层、weight（权重）、biase（偏置）、隐藏层神经元、损失函数、反向传播算法和随机梯度下降。如果你对这些概念不是很懂，希望你可以看看Andrej Karpathy关于神经网络的讲解。

卷积神经网络由五个基本部分组成，理解他们会让你对整体机制有直观的理解。

1.输入层

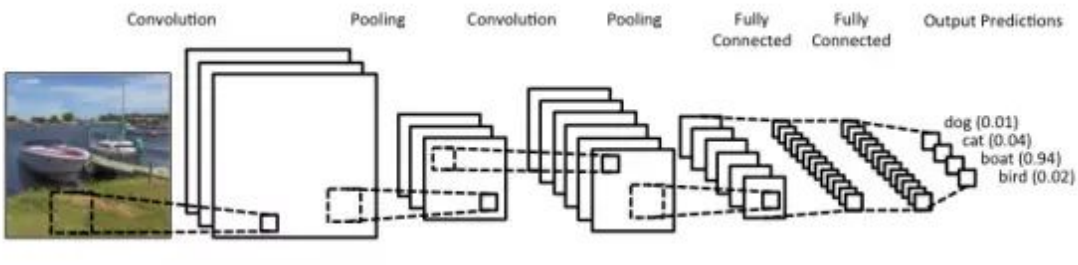
2.卷积层

3.激活函数层

4.池化层

5.全连接层

了解每个部分之前，先看看卷积神经网络的架构。



正如你所见，图片在网络中会经过多层处理，输出神经元包含每类的预期值。

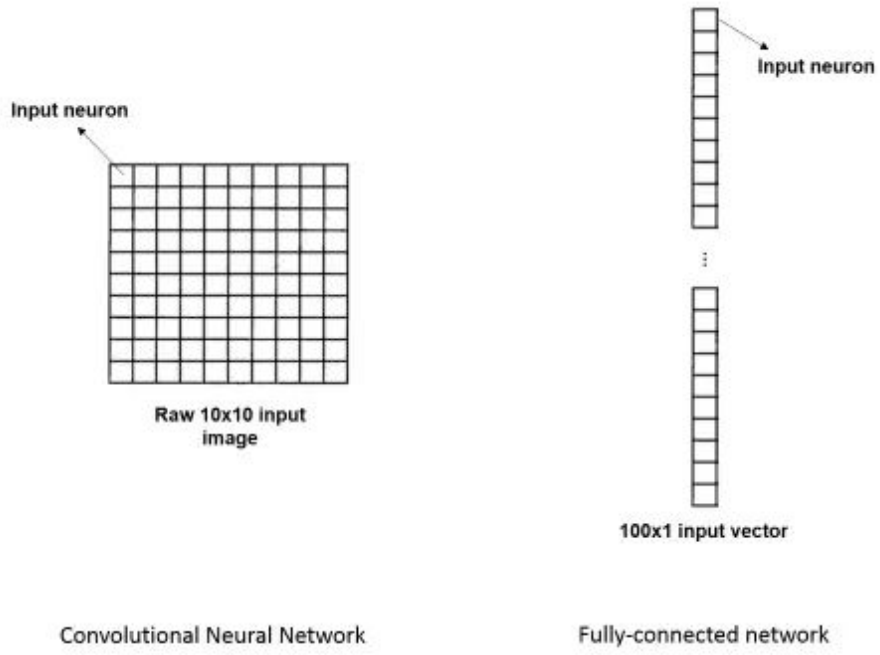
我们来详细介绍每一层的原理。

- **输入层**

在全连接网络中，输入量是被描述为一列神经元的向量。不管我们是否处理图片，我们都得调整数据来转换成这样。

而在卷积神经网络中，图片被看作许多小方格或者神经元，每个神经元代表一个像素值。卷积神经网络能基本上让图片保真，而不必去压缩成矢量。

下面的图表展示了差异：



- **卷积层**

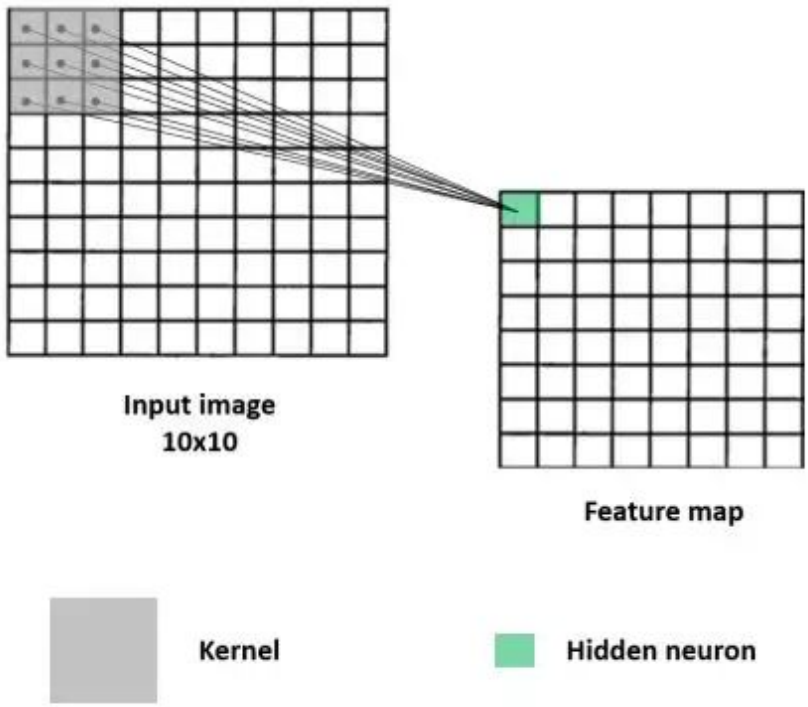
卷积层是卷积神经网络中最主要的部分。在解释它的功能前，我们得先理解卷积神经网络和全连接网络在连接方面的差异。这一点非常重要。我来阐述下：

前文提到说全连接网络是真的“全部连接了”。意思是指每层隐藏层的每个神经元都和相邻隐藏层的所有神经元相连接。当一个多维数据点从一个隐藏层流向另一个隐藏层，隐藏层的每个神经元激活状态都是由上一层所有神经元的权重所决定的。

然而卷积神经网络情况大不相同。**它不是全连接的。**

也就是说，隐藏层的每个神经元并不是和上一个隐藏层所有的神经元都相连，而只是和上一个隐藏层某一小片相连。

下面是一个例子：



在该图中，第一层隐藏层的第一个神经元和输入层3x3像素区相连接。输入层到隐藏层的这种映射叫做**特征映射（Feature map）**。这个隐藏层神经元只依赖于3x3这一小片区域，也最终会通过学习来捕获这片区域的特征。

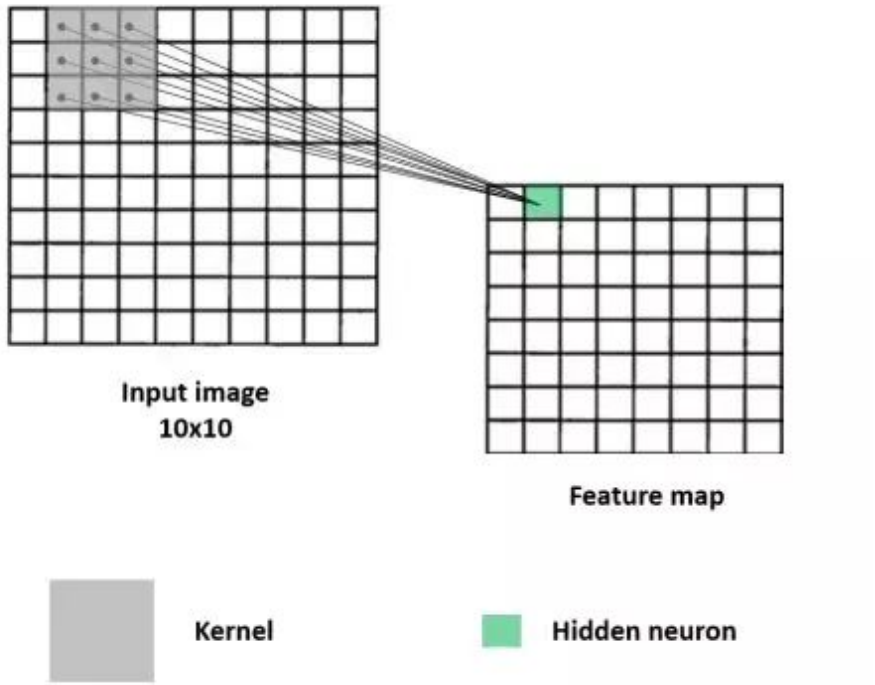
第一个隐藏层的神经元的数值代表了什么呢？灰色区域对应一个权重矩阵，称作**卷积核（kernal）**，图片中相同尺寸相同区域称作**局部感知域(receptive field)**，两者间卷积的结果就是第一个隐藏层神经元的数值了。

背后的操作非常简单，就是两个矩阵之间的元素相乘，再相加形成一个输出值。上述例子中，我们将9个相乘值相加就得到了第一个隐藏元的数值了。

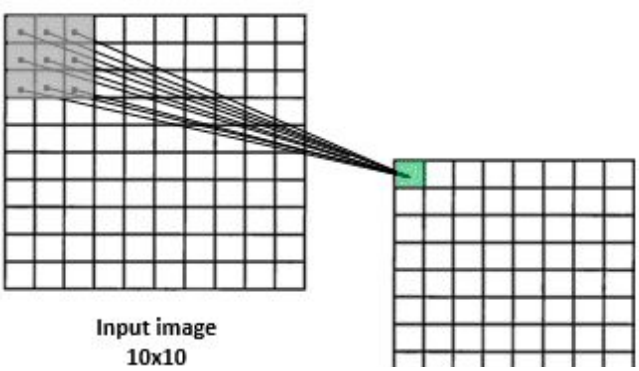
该神经元从局部感知域学习到视觉图形。你可以把它的值看作是代表图片中某个特征属性的强度。

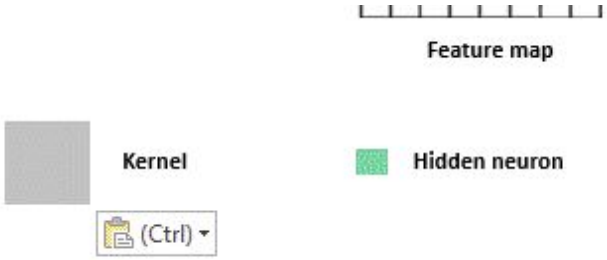
那么其他隐藏层的神经元又是怎么计算的呢？

为了计算第二隐藏层的神经元，卷积核会在输入图片上从坐到右移动一个单元（**stride=1**），用同样的过滤器来做卷积。看下图：



现在，我们可以想象卷积核移遍整个图片，每一步都做卷积，并将输出值存入特征映射。





这就是卷积层的作用：给定一个过滤器，卷积层会扫描输入层来阐述一个特征映射。

但是卷积操作到底代表着什么？如何理解最后形成的特征映射？

我一开始说卷积层能捕捉到图片里的视觉图形。那么我现在来证明下。

我会从数据集中下载一张猫的图片，然后变换卷积核来做多次卷积，并将结果视觉化。

In [1]:
```
%matplotlib inline
from scipy.signal import convolve2d
import numpy as np
import cv2
from matplotlib import pyplot as plt

image = cv2.imread('./data/train/cats/cat.46.jpg')
# converting the image to grayscale
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

我会定义一个函数，将卷积核作为输入量，对图片进行卷积，然后绘制原始图片和卷积后图片。

卷积核是一个小型矩阵（上文中的灰色方块）

In [2]:
```
def show_differences(kernel):
    convolved = convolve2d(image, kernel)
    fig = plt.figure(figsize=(15, 15))
    plt.subplot(121)
    plt.title('Original image')
    plt.axis('off')
    plt.imshow(image, cmap='gray')

    plt.subplot(122)
    plt.title('Convolved image')
    plt.axis('off')
    plt.imshow(convolved, cmap='gray')
    return convolved
```

我们先试试下面的过滤器：

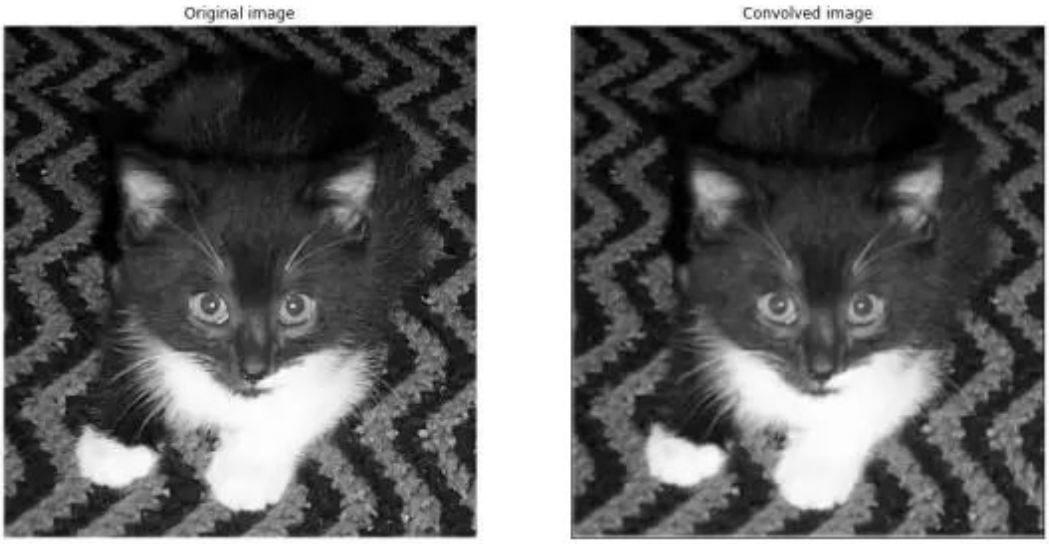
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

这叫做方框滤波（box blur）。 当该过滤器应用到输入图片中某个像素值上，它会截取该像素和其周围8个像素，并计算它们的平均像素值。数学上，这只是一个简单的平均算法。而视觉上，它能弱化图片转化的差异。

方框滤波被广泛应用于**噪音去除**。

我们来把方框滤波应用到一张猫的图片上，看看效果如何。

```
In [3]: kernel = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])/9
        output = show_differences(kernel)
```



如果你仔细查看卷积后的图片，会发现它更加平滑，上面的白色像素点更少（噪音）。

现在来试试更进一步的方框滤波。

$$\frac{1}{64}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

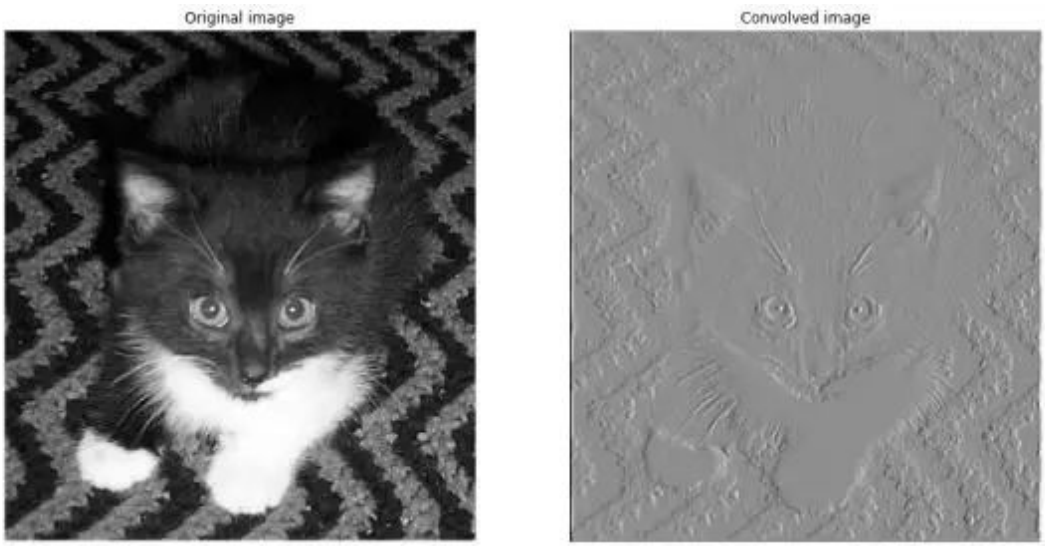
```
In [4]: kernel = np.ones((8,8), np.float32)/64
        dx = show_differences(kernel)
```

某些过滤器专门用来捕获图片的一些细节，像边缘。

下面这个例子在计算图片A中竖直方向变化的近似值。

$$G_x = A * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

In [5]:
```
kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
dx = show_differences(kernel)
```

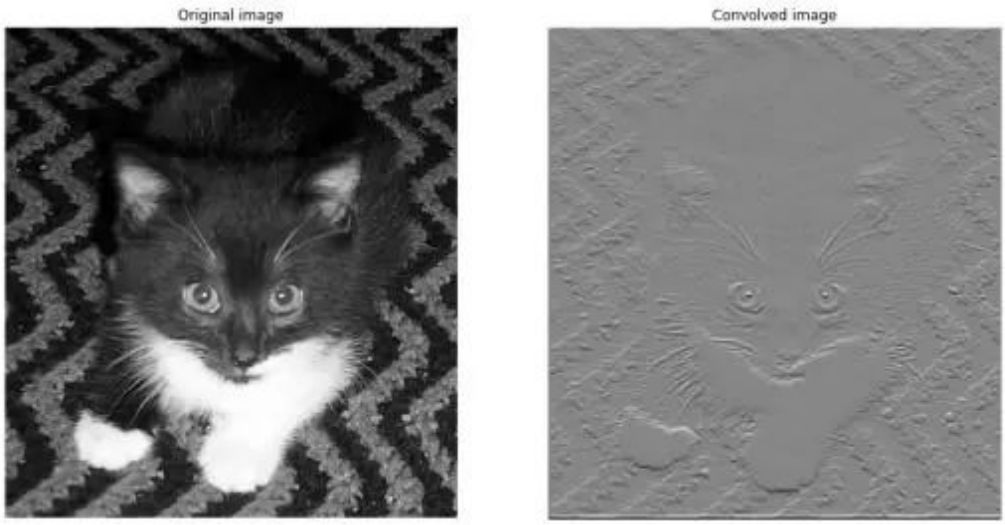


白色部分更好的回应了过滤器，显示了竖直边缘的存在。请仔细看猫左耳的边缘是怎么被捕获的。

很酷吧！下面的例子是同样的操作，只是方向换成了水平方向。

$$G_y = A * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

```
In [6]: kernel = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)
        dy = show_differences(kernel)
```



请仔细看猫胡须信息是怎么被捕获的。

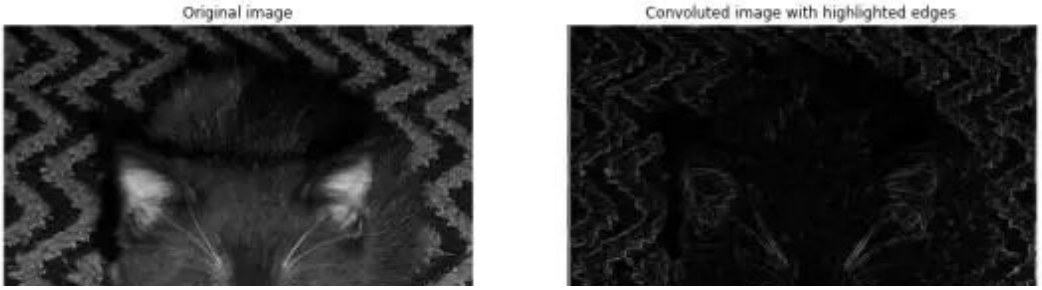
前面这两种过滤器是梯度算符。某种程度上，它们能够展示某个方向上图片的内在结构。

然而，当G_x和G_y用如下公式合并后：

$$G = \sqrt{G_x^2 + G_y^2}$$

会有更好的边缘捕获效果。

In [7]:
```
mag = np.hypot(dx, dy)  # magnitude
mag *= 255.0 / np.max(mag)  # normalize (Q&D)

fig = plt.figure(figsize=(15, 15))
plt.subplot(121)
plt.title('Original image')
plt.axis('off')
plt.imshow(image, cmap='gray')

plt.subplot(122)
plt.title('Convoluted image with highlighted edges')
plt.axis('off')
plt.imshow(mag, cmap='gray')
```



这叫做索贝尔算子，是两种简单卷积的非线性组合。之后我们会看到卷积层能够非线性整合多个特征映射，从而实现这样的边缘检测。

还有很多先进的过滤器，更多信息请看维基百科上的说明（https://en.wikipedia.org/wiki/Kernel_(image_processing)）。

我们现在已经理解了卷积层在卷积神经网络中的作用了，就是产生一个卷积后的值来对应输入量中的视觉元素。输出值可能尺寸变小，因此你可以把它当作是关于某个特征输入量的缩小版。

卷积核决定了卷积是在寻找哪种特征。它起到特征检测器的功能。我们可以想到很多过滤器来检测边缘、半圆和角落等。

卷积层不止一个过滤器？

在经典的卷积神经网络架构中，每一层卷积层一般不止一个过滤器。有时，是10个，16个或32个，有时，甚至更多。这种情况下，我们在每一卷积层做的卷积次数等同于过滤器数目。思路就是制造不同的特征映射，每个特征映射定位图片中某个特定特征。我们有越多的过滤器，我们就能提取越多的本质细节。

记住，我们现在提取的这些简单特征之后会在网络中合并起来来检测更复杂的图案。

那我们该如何选择过滤器权重了？

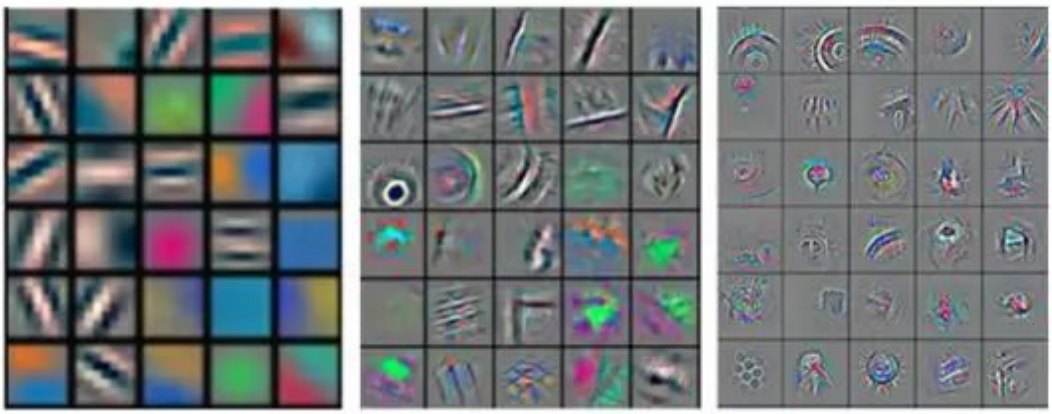
我们并不基于已有数据集主流知识来计算过滤器权重。

实际上，当训练卷积神经网络时，我们并不会人工设置过滤器权重。这些值都是网络**自动习得**的。你还记得在经典全连接网络中权重是如何通过反向传播算法习得的么？是的，卷积神经网络也是同样的原理。

不是我们来给每层设置大型权重矩阵，而是卷积神经网络习得过滤器权重。换言之，当我们从随机值来调整权重来降低分类错误时，网络会找出正确的过滤器，适用于寻找我们感兴趣的物体的特征。这是个影响力巨大的思路，颠覆了整个过程。

我发觉卷积神经网络既富有感染力，又很神秘。下面是首次卷积神经网络习得的三个特征映射，即有名的AlexNet。

注意从简单边缘提取特征形成更奇怪的形状有多复杂。



权重共享？

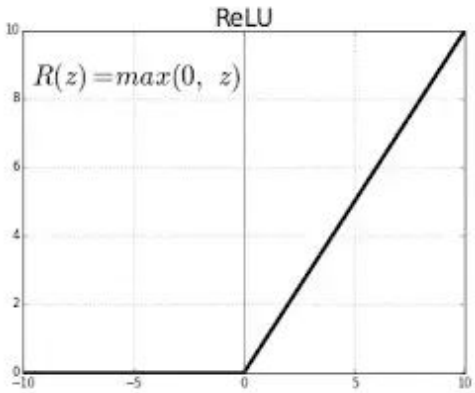
特征映射是仅由一个过滤器来产生的。所有隐藏层神经元共享同样的权重，因为是同一个过滤器在生成所有的神经元数值。这就是权重共享。这种属性大量减少习得参数数目，从而提升卷积神经网络训练速度。

除了为提升训练速度，共享权重概念的提出也是出于一个事实，即同一个视觉图案会在图片的不同区域多次出现，因此在整个图片中用同一个过滤器来检测就很合理了。

- **激活函数层**

一旦特征映射从卷积层提取后，下一步就是把它们移动到激活函数层。激活函数层一般是和卷积层绑定在一起，共同运作。

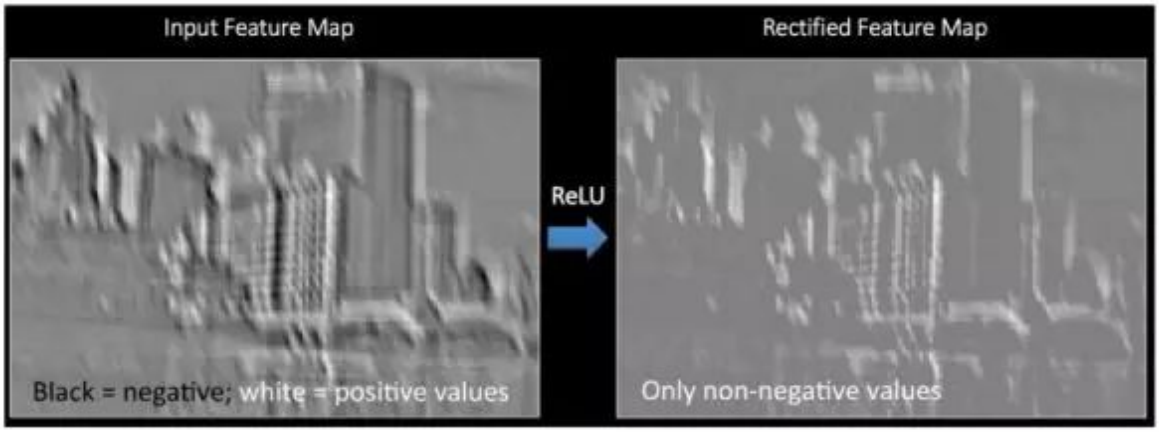
激活函数层会对特征映射使用激活函数，使得所有的负值都是0.该操作的输出被称作修订特征映射。



激活函数有两个主要优点：

- 它们给网络引入了非线性。实际上，目前提到的运算都是线性的，像卷积、矩阵相乘和求和。如果我们没有非线性，那么最终我们会得到一个线性模型，不能完成分类任务。
- 它们通过防止出现梯度消失来提升训练进程。

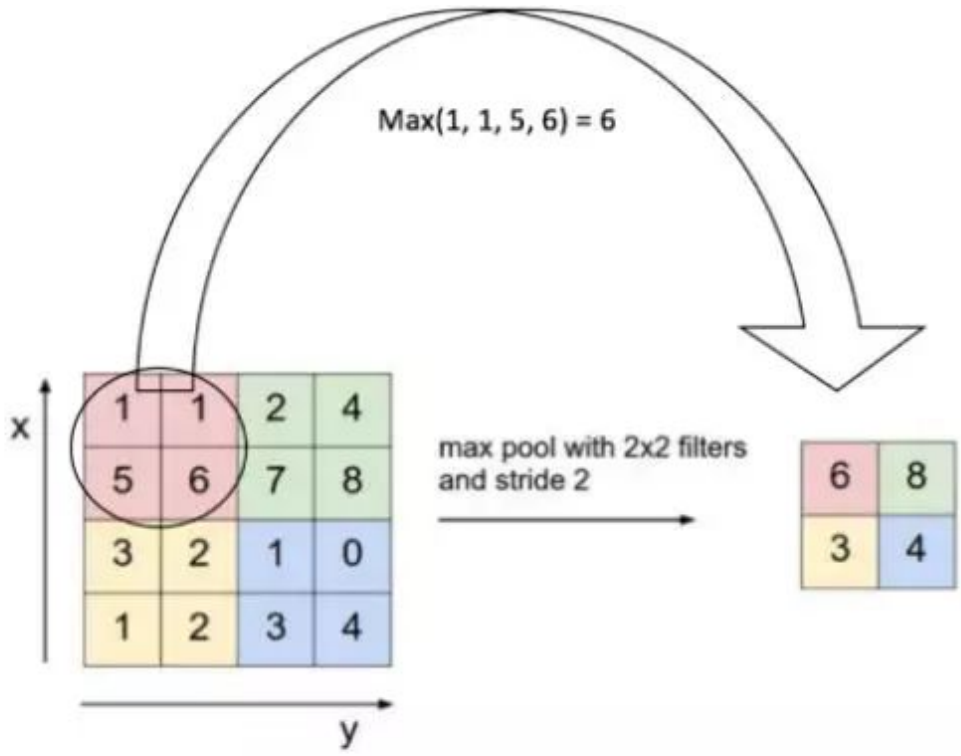
下面是函数激活层如果影响图片的视觉化展示。



- **池化层**

修订特征映射现在来到池化层了。池化是一个下采样操作，能让特征映射降维。

最普通的池化操作是Max-pooling。它选取一个小窗口，一般是2x2 ，以stride（步幅）为2在修订特征映射上滑动，每一步都取最大值。



例如一个10x10的修订特征映射会转变成一个5x5的输出。

Max-pooling有很多优点：

- 减小了特征映射的尺寸，减少了训练参数数目，从而控制过度拟合。
- 获取最重要特征来压缩特征映射。
- 对于输入图片的变形、失真和平移具有网络不变性，输入小的失真不会改变池化的输出，因为我们是取最大值。

Max-pooling还不是特别直观。根据我的理解，我总结如下：

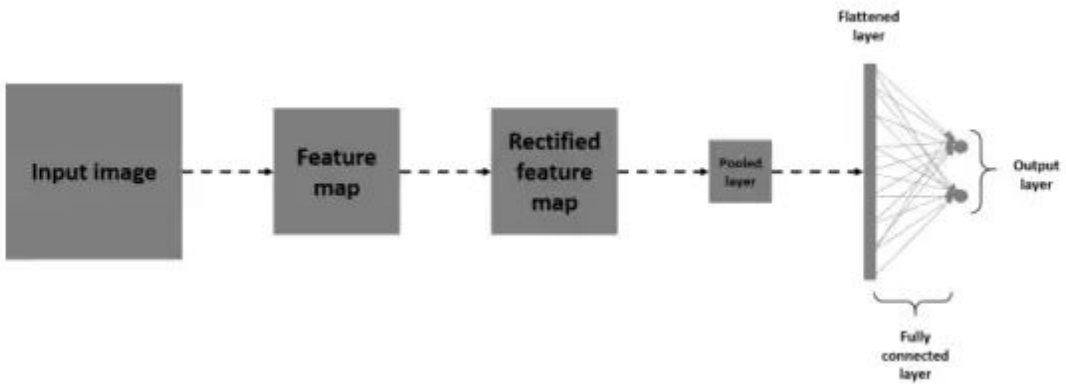
如果检测到一个特征，例如边缘，以上面例子中2x2红色方块为例，我们并不在乎具体是哪些像素使得特征显示出来。相反，我们只提取最大值，并假设它代表了视觉特征。这个方法看起来很大胆，因为很多空间信息都被舍弃了。但事实上，它相当有效，实践操作中表现不错。当然，你不用在意这个4x4图片的例子。因为max-pooling在被应用到相对高分辨率的图片上时，主要空间信息仍然被保留了，只有不重要的细节被舍弃了。这也是为什么max-pooling能够防止过度拟合。它使得网络聚焦于图片最相关的信息上面。

- **全连接层**

卷积神经网络也有一层全连接层，就是我们在经典全连接网络中看到的那种。它一般在网络的末尾，最后一层池化层已经扁平化为一个矢量，和输出层全连接。而输出层是预测矢量，尺寸和类别数目相同。

如果你对整个框架还有印象，即输入->卷积->函数激活->Max pooling，那么最后给输出添加一个全连接层。你会得到一个完整的小型卷积神经网络，把基本部分都整合在一起。

大概是这样的：



注意，这里扁平化层就是之前的池化层矢量化的结果。

我们为何需要全连接层？

全连接层起着**分类**的作用，而前面那些层都是在**提取特征**。

全连接层收到卷积、修订和池化的压缩产物后，整合它们来实施分类。

除为了分类，添加一层全连接层也是一种学习这些特征非线性组合的方法。从卷积层和池化层来的大部分特征可能已经能够完成分类，但是整合这些特征后效果会更好。

把全连接层看作是添加到网络的一种附加提取。

总结

我们来总结下目前提到的层。

层

功能

输入层

输入图片，保存空间结构信息

卷积层

从输入提取特征映射，对应一个特定图案

函数激活层

将像素负值设置为0，给网络引入非线性

Max-pooling层

下采样修订特征映射，降维和保留重要特征，防止过度拟合。

全连接层

学习特征非线性组合，实施分类

在一个经典全连接架构中，每种类型不会只有一种。实际上，如果你考虑有两个连续的卷积-池化层的网络，就会知道第二个卷积是在获得图片的压缩版，而图片包含了某一个特征的情况。因此，你可以把第二层卷积层当作原始输入图片的抽象压缩版，依然保留着很多空间结构信息。

我们一般会设置2到3个全连接层，这样在实施分类前就可以学习非线性特征组合。

关于这点可以引用Andrej Karpathy的话：

> 最普通的卷积神经网络架构形式是包含好几层卷积-函数激活层的，池化层紧随其后，这种结构不断重复直到图片已经完全融合成小尺寸。在某个阶段，转化到全连接层也是很正常的事情。最后一

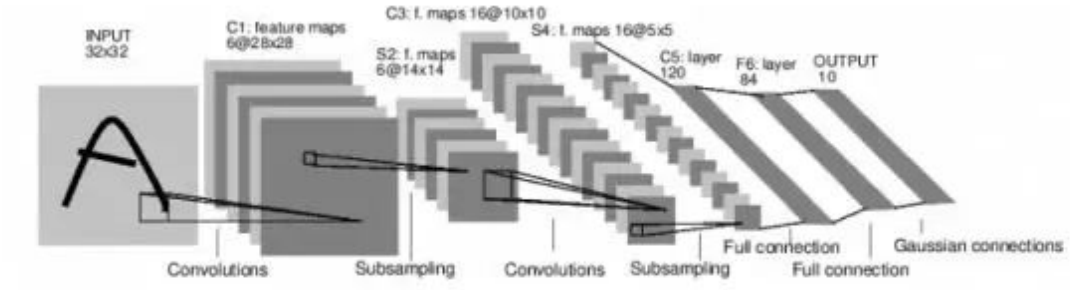
> 层全连接层持有输出，如类别值。换言之，最普通的卷积神经网络架构允许这样的形式：

INPUT -> [[CONV -> RELU]*N* -> *POOL?*]M -> [FC -> RELU]*K -> FC

现在我们已经知晓了卷积神经网络的基本部分，就来看看一个经典网络吧。

- **LeNet5**

下面是一个非常有名的卷积神经网络，叫**LeNet5**，由LeCun et al.在1998年设计。



我来解释下这个网络架构。当然我希望你能先自己试着分析下。

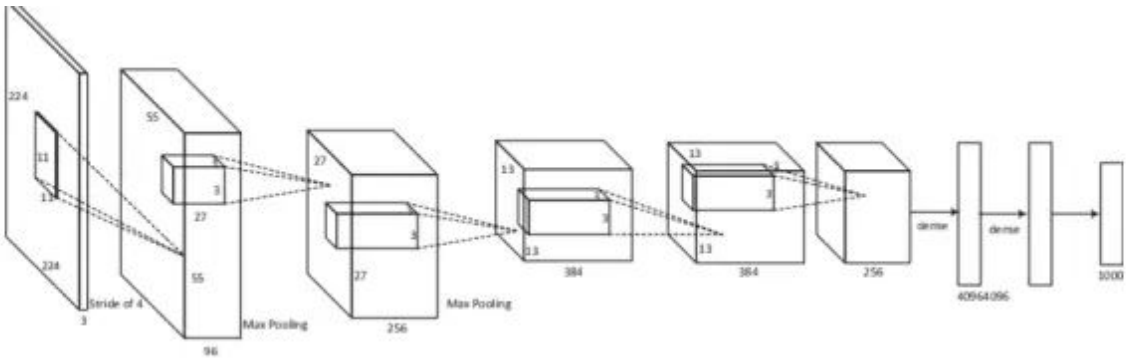
- **输入层：**一个32x32的灰度图，一个颜色通道
- **卷积层n°1：**对图片采用6个5x5过滤器，形成6个28x28的特征映射，该层的激活函数不是ReLU。
- **池化层：**池化这6个28x28的特征映射，形成6个14x14池化的特征映射，尺寸变为1/4。
- **卷积层n°2：**对这6个14x14特征映射采用16个不同的5x5过滤器，形成16个尺寸为 10x10的特征映射。每个过滤器和6个输入卷积，形成6个矩阵，并求和形成一个特征映射，总共16个特征映射。

- **池化层：**池化这16个 10x10特征映射为16个5x5特征映射。
- **第一层全连接层：**包含120个神经元，每个神经元都连接16个5x5特征映射的所有像素。这一层有16x5x5x120=48000个学习权重。
- **第二层全连接层：**包含84个神经元，这一层和上一层全连接，有120x84=10080个学习权重。
- **全连接层：**链接输出层，有84x10=840个学习权重。

更先进的卷积神经网络架构

你如果对更复杂更详细的卷积神经网络架构感兴趣，你可以看看这个博客（https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html）。

下面是赢得2012 ImageNet比赛的卷积神经网络——AlexNet。



3.搭建深度学习专门环境

深度学习的计算量很庞大。当你准备在你的笔记本电脑上搭建模型时就会发现这点。

然而，使用GPU就可以大幅度加快训练。因为GPU在如矩阵乘积之类的并行任务中效率非常高。由于神经网络都是关于矩阵乘积，因此使用GPU会大大提升性能。

我的笔记本电脑上没有强大的GPU，因此我选择了Amazon Web Services (AWS)上的虚拟机**p2.xlarge**。p2.xlarge是Amazon EC2（亚马逊弹性计算云）实例的一部分，有Nvidia GPU，12GB的视频存储，61GB的RAM，4vCPU和2,496 CUDA核。性能非常强大，每小时花费0.9美元。

当然还有更强大的实例，但是对于目前要处理的任务，一个p2.xlarge绝对够了。

Instance Name	GPU Count	vCPU Count	Memory	Parallel Processing Cores	GPU Memory	Network Performance
p2.xlarge	1	4	61 GiB	2,496	12 GiB	High
p2.8xlarge	8	32	488 GiB	19,968	96 GiB	10 Gigabit
p2.16xlarge	16	64	732 GiB	39,936	192 GiB	20 Gigabit

我在Deep Learning AMI CUDA 8 Ubuntu Version上开始实例，更多信息可以参考这个（https://aws.amazon.com/marketplace/pp/B06VSPXKDX）。基本上安装一个Ubuntu 16.04服务器，就包含了所有需要的深度学习框架，像Tensorflow, Theano, Caffe, Keras，也包含了听说安装异常难的GPU驱动。

AWS给你提供强大的实例和随时可用的深度学习专门环境，这样你就能快速开始你的项目了。这太棒了。

如果你对AWS不是很熟悉，可以看看下面两个帖子。

- https://blog.keras.io/running-jupyter-notebooks-on-gpu-on-aws-a-starter-guide.html
- https://hackernoon.com/keras-with-gpu-on-amazon-ec2-a-step-by-step-instruction-4f90364e49ac

它们能让你：

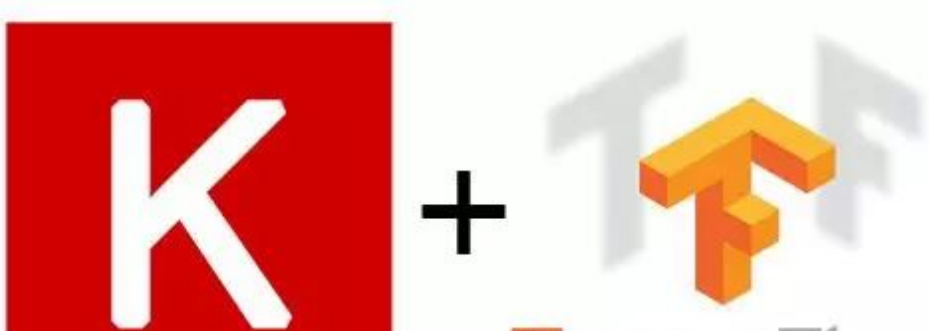
- 搭建EC2虚拟机，并连接上
- 设置网络安全属性来远程使用jupyter notebook

4.使用Tensorfow和Keras来建造猫狗分类器

环境已经搭建好了。我们可以把目前学到的知识运用到实战中了，那就是建造一个卷积神经网络来分类猫狗图片。

我们会使用Tensorflow深度学习框架和Keras

Keras是高等级的神经网络应用程序编程接口（KPI），用Python编写，能够在TensorFlow、CNTK和Theano上运行。它的设计初衷是让快速实验成为可能。能够尽量不拖延把想法变成结果的执行力是做好研究的秘诀。



4.1从零开始建造卷积神经网络

第一部分，我们会搭建端对端管道来训练卷积神经网络。我们会涉及到数据准备、数据集扩增、架构设计、训练和验证。我们会在训练集和验证集上实验测试损失和准确度矩阵。这可以让我们通过训练估测模型改善情况。

准备数据

开始建造卷积神经网络前第一件事就是从Kaggle下载和解压训练集。

In [8]:
```
%matplotlib inline
from matplotlib import pyplot as plt
from PIL import Image
import numpy as np
import os
import cv2
from tqdm import tqdm_notebook
from random import shuffle
import shutil
import pandas as pd
```

我们得组织下这些数据，从而keras能够容易的处理它们。

我们会建一个data文件夹，里面包含两个子文件夹：

- train
- validation

这两个子文件夹又各自包含两个文件夹。

- cats
- dogs

最终我们有如下的文件夹结构：

```
data/
    train/
        dogs/
            dog001.jpg
            dog002.jpg
            ...
        cats/
            cat001.jpg
            cat002.jpg
            ...
    validation/
        dogs/
            dog001.jpg
            dog002.jpg
            ...
        cats/
            cat001.jpg
            cat002.jpg
```

这样的文件夹结构可以让我们的模型在不管是训练还是验证的时候都可以知道从哪个文件夹里提取图片和标签。

这有一个函数可以让你来建文件树。它有两个参数：图片总数n和验证集比例r。

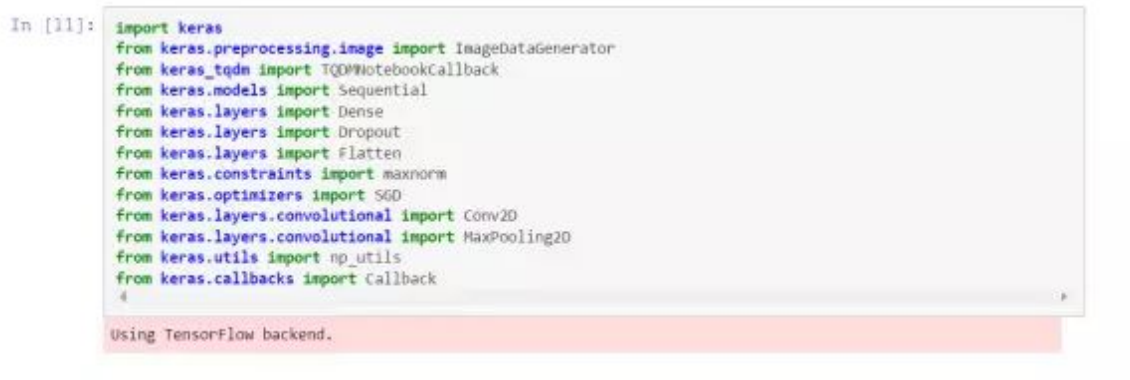
In [9]:
```
def organize_datasets(path_to_data, n=4000, ratio=0.2):
    files = os.listdir(path_to_data)
    files = [os.path.join(path_to_data, f) for f in files]
    shuffle(files)
    files = files[:n]

    n = int(len(files) * ratio)
    val, train = files[:n], files[n:]


    shutil.rmtree('./data/')
    print('/data/ removed')

    for c in ['dogs', 'cats']:
        os.makedirs('./data/train/{0}/'.format(c))
        os.makedirs('./data/validation/{0}/'.format(c))

    print('folders created !')

    for t in tqdm_notebook(train):
        if 'cat' in t:
            shutil.copy2(t, os.path.join('.', 'data', 'train', 'cats'))
        else:
            shutil.copy2(t, os.path.join('.', 'data', 'train', 'dogs'))

    for v in tqdm_notebook(val):
        if 'cat' in v:
            shutil.copy2(v, os.path.join('.', 'data', 'validation', 'cats'))
        else:
            shutil.copy2(v, os.path.join('.', 'data', 'validation', 'dogs'))

    print('Data copied!')
```

我的参数是：

- n : 25000 (整个数据集)
- r : 0.2

In [10]:
```
ratio = 0.2
n = 25000
organize_datasets(path_to_data='./train/', n=n, ratio=ratio)
```

我们来下载Keras和它的dependencies。



图片生成器和数据扩增

当我们训练一个模型的时候，不会下载整个数据集放在存储里。这样效率非常低，特别是使用你自己的电脑来操作。

我们会使用ImageDataGenerator函数。它能让你从训练集和验证集中无限批量的得到图片。每一批图片流过网络，做正向传播和反向传播，然后参数随着验证数据的测试而更新。接着下一批图片会有同样的操作。

在ImageDataGenerator object（目标）中我们会给每批图片做随机修改。这个过程叫做数据扩增（**data augmentation**）。它能生成更多的数据，从而我们的模型绝不会看到两张一模一样的照片。这能防止过度拟合，并让模型总结能力更强。

我们会建两个ImageDataGenerator object。

train_datagen是训练集，**val_datagen**是验证集。这两个集会对图片做变形调整。而train_datagen会做更多的调整。

```
In [12]: batch_size = 32
```

```
In [13]: train_datagen = ImageDataGenerator(rescale=1/255.,
                                            shear_range=0.2,
                                            zoom_range=0.2,
                                            horizontal_flip=True
                                            )
         val_datagen = ImageDataGenerator(rescale=1/255.)
```

从前面两个object中，我们会创建两个文件生成器：

- train_generator
- validation_generator

每个生成器都会从目录生成批量张量图片数据，实时数据扩增。并且数据会无限批量循环。

```
In [14]: train_generator = train_datagen.flow_from_directory(
                './data/train/',
                target_size=(150, 150),
                batch_size=batch_size,
                class_mode='categorical')

         validation_generator = val_datagen.flow_from_directory(
                './data/validation/',
                target_size=(150, 150),
                batch_size=batch_size,
                class_mode='categorical')
```

```
Found 20000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
```

模型架构

我会使用拥有三层卷积/池化层和两层全连接层的卷积神经网络。三层卷积层分别使用32、32和64个3x3过滤器。

我对两层全连接层使用dropout来防止过度拟合。

In [15]:
```
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
```

我使用随机梯度下降优化器，学习速率为0.01，momentum系数为0.9。

因为我们是在做二进制分类，所以我使用了二进制交叉熵代价函数。

In [16]:
```
epochs = 50
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

Keras提供了很方便的方法来展示模型总结。对于每一层，它都会展现输出形状和训练参数数目。

这是拟合模型前的可用性测试：

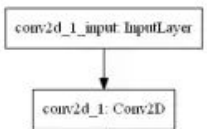
In [17]:

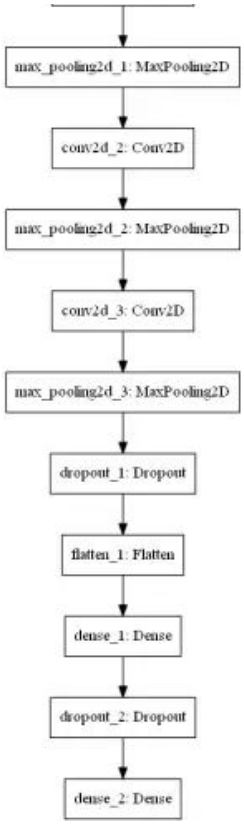
```
In [47]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 150, 150, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 75, 75, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 37, 37, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 18, 18, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 18, 18, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 20736)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                1327168
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 130
=================================================================
Total params: 1,355,938
Trainable params: 1,355,938
Non-trainable params: 0
```

我们来看看网络架构：

- **视觉化呈现架构：**





训练模型

在训练模型之前，我定义了两个回调函数，训练的时候会被回调。

- 一个回调函数用于在损失函数不能改善验证数据的时候及时停止训练
- 另一个回调函数用于储存每个epoch（使用训练集中的全部样本训练一次）验证损失和准确度。这可以来实验测试训练误差。

In [18]:

```
## Callback for loss logging per epoch
class LossHistory(Callback):
    def on_train_begin(self, logs={}):
        self.losses = []
        self.val_losses = []

    def on_epoch_end(self, batch, logs={}):
        self.losses.append(logs.get('loss'))
        self.val_losses.append(logs.get('val_loss'))

history = LossHistory()
```

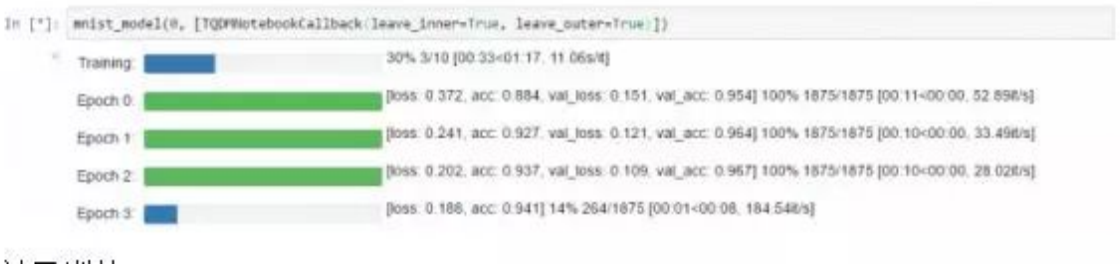
```
## Callback for early stopping the training
early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss',
                               min_delta=0,
                               patience=2,
                               verbose=0, mode='auto')
```

我还使用了keras-tqdm（https://github.com/bstriner/keras-tqdm），它是和Keras完美整合的超棒进度条。

它可以让你很轻松的监督你的模型训练。

你需要做的事情很简单，就是在keras_tqdm上下载TQDMNotebookCallback，然后把它当作第三个回调函数。

下面是简单例子来呈现keras-tqdm的样子。



关于训练：

- 我们会使用fit_generator方法。它是标准拟合方法的一种变化形式，把生成器作为输入。
- 我们会训练模型50个epoch。每个epoch中，会有20000张独特扩增照片以每批32张的方式流入网络，做正向传播和反向传播，用SGD来调整权重。使用多个epoch也是为了防止过度拟合。

计算量非常庞大：

- 如果你是使用你自己的笔记本电脑，那么每个epoch会花费15分钟。
- 如果你是像我一样在使用p2.xlarge EC2实例，那么每个epoch会花费大概2分钟。

tqdm能够让你监督每个epoch验证损失和准确度。这对检查模型质量大有帮助。

分类结果

在验证集上我们的准确率达到89.4%。训练/验证误差和准确率显示在下文。

考虑到我并没有在设计网络架构上投入太多时间，所以这个结果已经非常棒了。

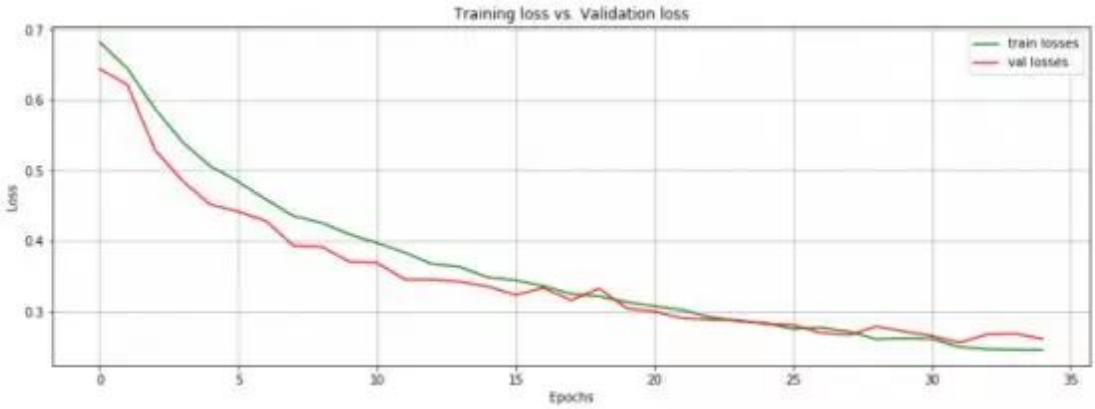
现在我们来保存模型以作后用。

```
In [20]: model.save('./models/model4.h5')
```

在同一个表格上我们来试试训练和验证损失：

```
In [21]: losses, val_losses = history.losses, history.val_losses
         fig = plt.figure(figsize=(15, 5))
         plt.plot(fitted_model.history['loss'], 'g', label="train losses")
         plt.plot(fitted_model.history['val_loss'], 'r', label="val losses")
         plt.grid(True)
         plt.title('Training loss vs. Validation loss')
         plt.xlabel('Epochs')
         plt.ylabel('Loss')
```

```
plt.legend()
```



连续两个epoch验证损失都没有提高，我们暂停了训练。

现在我们在训练集和验证集上测试下准确度。

两个矩阵持续增加直到达到一个水平状态，模型最终过度拟合了（从epoch34开始）。

4.2下载预训练模型

目前为止一切都挺好。我们设计了一个特定卷积神经网络，在验证数据上表现不错，准确率达89%。

然而有一个办法可以得到更好的分数。在一个包含猫狗照片的大型数据集上，下载预训练卷积神经网络的权重。这样的网络会已经学习了和我们分类相关的有关特征。

我会下载VGG16网络的权重。更准确的是，我会下载对应最后一层卷积层的网络权重。这个网络部分起着特征探测的作用，我们会为了分类任务而添加全连接层。

和LeNet5相比，VGG16是一个非常大的网络。它有16层，拥有可训练权重和将近1.4亿参数。想学习更多关于VGG16，请参考这个pdf（https://arxiv.org/pdf/1409.1556.pdf）。

我们先下载在ImageNet训练过的VGG16权重。这说明了我们对最后三层全连接层不是很感兴趣。

```
In [23]: from keras import applications
         # include_top: whether to include the 3 fully-connected layers at the top of the network.
         model = applications.VGG16(include_top=False, weights='imagenet')
         datagen = ImageDataGenerator(rescale=1. / 255)
```

我们把图片放入网络来获得特征表示，再把特征表示输入神经网络分类工具中。

我们做这个是为了训练集和验证集。

In [24]:
```
generator = datagen.flow_from_directory('./data/train/',
                                        target_size=(150, 150),
                                        batch_size=batch_size,
                                        class_mode=None,
                                        shuffle=False)

bottleneck_features_train = model.predict_generator(generator, int(n * (1 - ratio)) // batch_size)
np.save(open('./features/bottleneck_features_train.npy', 'wb'), bottleneck_features_train)
```

```
Found 20000 images belonging to 2 classes.
```

In [25]:
```
generator = datagen.flow_from_directory('./data/validation/',
                                        target_size=(150, 150),
                                        batch_size=batch_size,
                                        class_mode=None,
                                        shuffle=False)

bottleneck_features_validation = model.predict_generator(generator, int(n * ratio) // batch_size,)
np.save('./features/bottleneck_features_validation.npy', bottleneck_features_validation)
```

```
Found 5000 images belonging to 2 classes.
```

当图片经过网络的时候，是以正确的顺序显示的。因此，我们可以很容易的把标签和它们关联起来。

In [26]:
```
train_data = np.load('./features/bottleneck_features_train.npy')
train_labels = np.array([0] * (int((1-ratio) * n) // 2) + [1] * (int((1 - ratio) * n) // 2))

validation_data = np.load('./features/bottleneck_features_validation.npy')
validation_labels = np.array([0] * (int(ratio * n) // 2) + [1] * (int(ratio * n) // 2))
```

现在我们设计了一个小型全连接网络，把从VGG16提取出来的特征都插入进去，来作为卷积神经网络的分类部分。

In [27]:
```
model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy', metrics=['accuracy'])
```

In [28]:
```
fitted_model = model.fit(train_data, train_labels,
          epochs=15,
          batch_size=batch_size,
          validation_data=(validation_data, validation_labels[:validation_data.shape[0]]),
          verbose=0,
          callbacks=[TQDMNotebookCallback(leave_inner=True, leave_outer=False), history])
```

只15个epoch，准确率就达到了90.7%。效果很不错。而且在我的笔记本电脑上，每个epoch只需要大概1分钟。

In [29]:
```
fig = plt.figure(figsize=(15, 5))
plt.plot(fitted_model.history['loss'], 'g', label="train losses")
plt.plot(fitted_model.history['val_loss'], 'r', label="val losses")
plt.grid(True)
plt.title('Training loss vs. Validation loss - VGG16')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

In [30]:

```
fig = plt.figure(figsize=(15, 5))
plt.plot(fitted_model.history['acc'], 'g', label="accuracy on train set")
plt.plot(fitted_model.history['val_acc'], 'r', label="accuracy on validation sete")
plt.grid(True)
plt.title('Training Accuracy vs. Validation Accuracy - VGG16')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

很多深度学习开拓者们都提倡使用预训练网络来完成分类任务。实际上这是在充分利用大型数据集上的大型网络训练。

Keras能让你轻松下载预训练网络，像VGG16, GoogleNet和ResNet。更多信息，请看这个（https://keras.io/applications/）。

座右铭：**别逞能。别浪费时间去做无用功。使用预训练网络吧！**

我们还能做什么？

如果你对改善特定卷积神经网络很感兴趣：

- 给数据集引入更多的数据扩增
- 使用网络超参数：卷积层数目、过滤器数目、过滤器尺寸。用验证数据集来测试每种组合。
- 更换或改进优化器
- 试试不同的成本函数
- 使用更多的全连接层
- 引入更大胆的dropout

如果你对使用预训练网络来得到更好结果很感兴趣：

- 使用不同的网络架构
- 使用更多的全连接层和更多的隐藏元

如果你想挖掘卷积神经网络学到了什么：

- 视觉化特征映射。我还没有试过。不过有一篇很有趣的论文在讨论这方面（https://arxiv.org/pdf/1311.2901.pdf）。
- 如果你想使用训练模型：
- 推出产品，并使用到新的猫狗图片上。这个方法很好的测验了模型总结能力好不好。

总结

这篇文章概述了卷积神经网络背后的理论机制，详细解释了每个基本部分。

这也是一个上手指南，指导如何在AWS上搭建深度学习专门环境，如何从零开始建造端对端模型和基于预训练的增强模型。

使用python来做深度学习研究非常有趣。Keras让预处理数据和建层变得更加容易。记住一点，如果有天你想搭建特定神经网络的某些部分，就得换成别的框架。

我希望本文能够让你直观了解卷积神经网络机制，并激发你的兴趣来继续学习。

卷积神经网络真的非常强大。任何从事计算机的人士都坚信它的强大和高效。

它的应用领域不断在扩大。现在自然语言处理行业人士也在转向卷积神经网络。下面是他们的一些应用：

- 使用卷积神经网络实施文本分类：https://chara.cs.illinois.edu/sites/sp16-cs591txt/files/0226-presentation.pdf
- 自动图片描述（图片+文本）：https://cs.stanford.edu/people/karpathy/sfmltalk.pdf
- 字符层面的文本分类：https://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf

参考：

在之前学习神经网络和卷积神经网络的时候，我参考过下列链接：

http://1.neuralnetworksanddeeplearning.com

（http://neuralnetworksanddeeplearning.com/）：

包含目前为止关于神经网络和深度学习最好的笔记内容。我强烈推荐该网站给那些想学习神经网络的人。

2.CS231n Convolutional Neural Networks for Visual Recognition

（http://cs231n.github.io/convolutional-networks/）：

Andrej Karpathy在斯坦福大学的讲义，很注重数学。

3.A Beginner Guide to Understanding Neural Networks

（http://cs231n.github.io/convolutional-networks/）：

一篇包含三个部分用来解释卷积神经网络的文章，从基本的直观感受说到架构细节。读起来非常有趣，获益颇丰。

- 第一部分

（https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/）

- 第二部分

（https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/）

- 第三部分

（https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html）

4.Running jupyter notebooks on GPU on AWS

（https://blog.keras.io/running-jupyter-notebooks-on-gpu-on-aws-a-starter-guide.html）

5.Building powerful image classification models using very little data

（https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html）

6.CatdogNet - Keras Convnet Starter

（https://www.kaggle.com/jeffd23/catdognet-keras-convnet-starter）

7.A quick introduction to Neural Networks

(https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/)

8.An Intuitive Explanation of Convolutional Neural Networks

(https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/)

9. Visualizing parts of Convolutional Neural Networks using Keras and Cats

(https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59)

原文地址：

https://ahmedbesbes.com/understanding-deep-convolutional-neural-networks-with-a-practical-use-case-in-tensorflow-and-keras.html