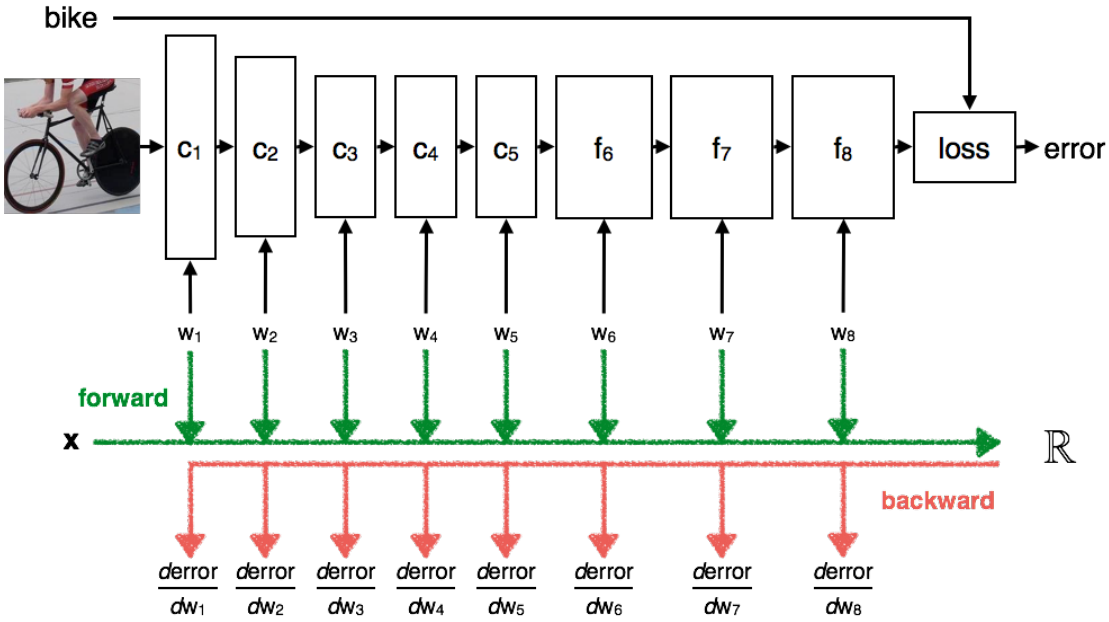
VGG 卷积神经网络实战

VGG Convolutional Neural Networks Practical

By Andrea Vedaldi and Andrew Zisserman

迷若烟雨 译



卷积神经网络是解决各类计算机视觉问题的一类有效方法。特别的，由几个处理层组成的深度卷积神经网络，每个包含非线性操作，它们共同的学习，以端到端的方式，来解决特定的任务。这些方法现在成为语音和纹理数据主流的方法。

这个实践教程解释了卷积神经网络的基本内容。第一部分介绍了卷积神经网络的组件，例如 ReLU 单元和线性滤波器。第二部分关注于学习两个基本的 CNNs。第一个是简单的捕捉图像结构的非线性滤波器，而第二个是识别手写字符的网络（使用了很多不同的字体）。这些例子解释了随机梯度下降、使用动量项、定义目标函数、成批处理数据的构建以及数据增强。最后一个部分展示了强大的 CNN 模型怎么可以离线下载及在应用中直接使用，跳过敖贵的训练过程。

开始

阅读并理解 requirements and installation instructions。本实验的下载链接是：

- Code and data: practical-cnn-2015a.tar.gz
- Code only: practical-cnn-2015a-code-only.tar.gz
- Data only: practical-cnn-2015a-data-only.tar.gz
- Git repository (for lab setters and developers)

在安装完成后，在 matlab 的编辑器中打开并编辑脚本 exercise1.m。这个脚本包含了建议的代码并且描述了这个实验所有步骤。你还可以剪切和黏贴这些代码到 matlab 的窗口执行，并且随着课程的深入需要修改这些代码。其他的文件 exercise2.m, exercise3.m, and exercise4.m 分别是给第二、三和四部分的。

第一部分：CNN 构建组件

1.1 卷积

一个前向神经网络可以被认为是几个函数组成：

$$f(\mathbf{x}) = f_L(\cdots f_2(f_1(\mathbf{x};\mathbf{w}_1);\mathbf{w}_2)\cdots),\mathbf{w}_L).$$

每个函数f_l把一个输入$\mathbf{x}_l$和参数向量$\mathbf{w}_l$结合，产生输出结果$\mathbf{x}_{l+1}$。虽然这些函数的类型和序列是手工选择的，参数$\mathbf{w} = (\mathbf{w}_1, \cdots, \mathbf{w}_L)$是从解决目标问题的数据学出来的，例如对图片或者语音分类。

在卷积神经网络中数据和函数有附加的结构特征。数据$\mathbf{x}_1, \cdots, \mathbf{x}_n$是图像、声音或者更一般的切片数据或者实数。特别的，由于这个教程剩下的部分关注于视觉应用，数据将会 2D 的数组。形式化描述为，每个$\mathbf{x}_i$将是$M \times N \times K$实数的数组，含有$M \times N$个像素以及 K 个通道。因此数组的前两维是数据后一维是通道.注意到网络的输入只有$\mathbf{x} = \mathbf{x}_1$时真实的数据，其他的都是中间的表达。

CNN 第二个属性是函数f_l有一个卷积的结构。这意味着作用到输入层$\mathbf{x}_l$的f_l操作是局部和平移不变的。卷积操作的例子把一系列滤波器作用于$\mathbf{x}_l$。

在这个部分，我们熟悉下一系列卷积和非线性操作算子。第一个是线性限制单元。我们通过聚焦我们的焦点在如下的单个函数：

$$f : \mathbb{R}^{M \times N \times K} \to \mathbb{R}^{M' \times N' \times K'}, \qquad \mathbf{x} \mapsto \mathbf{y}.$$

打开 example1.m 文件，选中下面的文本，并且在 matlab 中执行。

```
x = imread('peppers.png') ;

% Convert to single format
x = im2single(x) ;

% Visualize the input x
figure(1) ; clf ; imagesc(x) ;
```

这将会在图 1 中展示一幅图片：

使用 matlab 的 size 命令来获得数组 a 的大小。注意到数组 x 被转换成了单精度浮点数的格式。这是因为 matconvnet 使用的格式是单精度浮点的。

问题：

数组 x 的第三维是 3，为什么？

现在我们将创建 5*5*3 大小的滤波器

w = randn(5,5,3,10,'single') ;

这个滤波器也是单精度浮点的。注意到 w 有 4 个维度，10 个滤波器。注意到每个滤波器不是平面的，而是三个层的体卷积。下一步是把滤波器作用到图像上。这是通过 matconvvnet 里的 vl_nnconv 的函数完成的。

y = vl_nnconv(x, w, []) ;

提醒：可能你还没注意到 vl_nnconv 函数的第三个参数是空矩阵[]。它也可以被传入偏置向量来加入到滤波器的输出。

变量 y 包含了卷积的输出。注意到滤波器是三维的，直觉上它会在 K 个通道上操作。此外，有 K' 个这样的滤波器，生成了 K' 维的特征图，如下所示：

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

问题：

认真的学习这个表达式，并且回答下述问题：

给定输入特征图大小为 $M \times N \times K$，每个滤波器大小为 $M_f \times N_f \times K$，y 的维度是多少？

注意到 x 是由$i+i'$ and$j+j'$索引的，但是为什么在k andk'没有这个符号？

任务：检查 y 的大小和你计算的一样。

我们现在可以可视化卷积的输出 y。为了做到这一点，使用函数 vl_imarraysc 来显示 y 中每个通道的特征：

```
figure(2) ; clf ; vl_imarraysc(y) ; colormap gray ;
```

问题：

看下获得通道的特性。大多数都会在输入图像 x 的边缘有强烈的响应。回忆下我们是通过高斯分布来随机的获得参数的。你能解释这种现象吗？

至此滤波器保留的输入特征图的分辨率。然而，有时候对输出降采样很有用。这可以通过使用 vl_nnconv 里的 stride 操作。

```
y_ds = vl_nnconv(x, w, [], 'stride', 16) ;
figure(3) ; clf ; vl_imarraysc(y_ds) ; colormap gray ;
```

正如你在上一个问题中所注意到的那样，把滤波器作用的特征图会降低图像的大小。如果这你能理解的话，我们可以使用 pad 来在边界处补充 0.

任务：确保你自己看到之前代码的输出和现在的输出有不同的边界。你能解释为什么吗？

为了固定已经学到的东西，我们现在手工设计一个滤波器：

```
w = [0    1 0 ;
     1 -4 1 ;
     0    1 0 ] ;
w = single(repmat(w, [1, 1, 3])) ;
y_lap = vl_nnconv(x, w, []) ;
figure(5) ; clf ; colormap gray ;
subplot(1,2,1) ; imagesc(y_lap) ; title('filter output') ;
subplot(1,2,2) ; imagesc(-abs(y_lap)) ; title('- abs(filter output)')
```

问题：

我们实现了什么样的滤波器？

这个滤波器是怎样处理 RGB 颜色通道的？

图像什么样的结构被检测到了？

1.2 非线性门

正如我们在引言中讲的那样，CNNs 通过几个不同的函数获得的。除了上述所讲的线性滤波器外，还有很多的非线性滤波器。

问题:CNN 中一些函数必须是非线性的，为什么？线性的滤波器组合仍然是线性。

最简单的非线性是通过在非线性门单元后加一个线性滤波器，在特征图的每个组件上作用。最简单的函数是非线性限制单元（ReLU）：

$$y_{ijk} = \max\{0, x_{ijk}\}.$$

这个函数是通过 vl_relu 实现的，让我们试一下：

任务：

运行上面的代码，并且理解 w 滤波器到底做了什么？

解释结果 z。

1.3 池化

在 C NN 中有很多其他重要的操作，其中一个就是池化。池化算子子在每个特征通道上独立的操作，通过合适的算子结合几个相近的特征。公共的选择包括最大池化、求和池化。例如，最大池化定义为：

$$y_{ijk} = \max\{y_{i'j'k} : i \le i' < i + p, j \le j' < j + p\}$$

最大池化是通过 vl_nnpool 函数实现的。试下下面的：

```
y = vl_nnpool(x, 15) ;
figure(7) ; clf ; imagesc(y) ;
```

问题：

观察下结果图像，你能解释吗？

vl_nnpool 函数和 vl_nnconv 函数一样支持 padding。然而，对于最大池化来说，pad 是$-\infty$ instead of 0，为什么？

1.4 归一化

另一个构件事通道级别的归一化。这个算子在输入特征每个通道进行归一化。归一化算子形式如下所示：

$$y_{ijk'} = \frac{x_{ijk}}{\left(\kappa + \alpha \sum_{k \in G(k')} x_{ijk}^2\right)^\beta}$$

$$G(k) = \left[k - \lfloor \frac{\rho}{2} \rfloor, k + \lceil \frac{\rho}{2} \rceil\right] \cap \{1, 2, \cdots, K\}$$

其中 ρ 是 个相邻输入特征通道的组。

任务：理解算子做了什么。为了实现 L2 的归一化你需要怎样设置参数。

现在，让我们试一下：

```
rho = 5 ;
kappa = 0 ;
alpha = 1 ;
beta = 0.5 ;
y_nrm = vl_nnnormalize(x, [rho kappa alpha beta]) ;
figure(8) ; clf ; imagesc(y_nrm) ;
```

任务：

解释结果

计算 L2 归一化时，你注意到了什么？

解释和参数相关的选择。

第 2 部分反向传播和导数

CNN 的参数$\mathbf{w} = (\mathbf{w}_1, \cdots \mathbf{w}_L)$应当以 $\mathbf{z} = f(\mathbf{x}; \mathbf{w})$达到目标值来学习到。在有些情况下，目标是建模数据的分布，这导致生成式目标函数。然而，在这里，我们使用 f 作为一个回归器并且通过最小化判别目标函数得到。在简单的情况下，我们被给予：

一系列期望的输入-输出关系$(\mathbf{x}_1, \mathbf{z}_1), \cdots, (\mathbf{x}_n, \mathbf{z}_n)$ ，其中，$\mathbf{x}_i$ 是输入数据，$\mathbf{z}_i$ 是对应的输出数据。

一个损失函数，给出了预测为$\hat{\mathbf{z}}$ 的惩罚。

我们通过平均这些样本的经验损失定义误差函数：

$$L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{z}_i, f(\mathbf{x}_i; \mathbf{w}))$$

注意到 f 和 l 可以被认为是由多余一个层组成（称为损失层）。因此，稍有一点概念上的混淆，在剩下的部分我们把损失包含在 f 中，并且不再显式说明。

实践中最常用的最小化 L 的算法是梯度下降。想法很简单：计算目标函数的梯度，以及当前的解，然后通过他们的差值来在梯度下降最快的方向更新：

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \frac{\partial f}{\partial \mathbf{w}} (\mathbf{w}^t)$$

2.1 反向传播的理论

基本的问题是计算参数 w 的梯度函数。由于 f 是有几个函数组成的，关键的一点在于链式法则：

$$\begin{aligned} \frac{\partial f}{\partial \mathbf{w}_l} &= \frac{\partial}{\partial \mathbf{w}_l} f_L(\cdots f_2(f_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2) \cdots), \mathbf{w}_L) \\ &= \frac{\partial \operatorname{vec} f_L}{\partial \operatorname{vec} \mathbf{x}_L^\top} \frac{\partial \operatorname{vec} f_{L-1}}{\partial \operatorname{vec} \mathbf{x}_{L-1}^\top} \cdots \frac{\partial \operatorname{vec} f_{l+1}}{\partial \operatorname{vec} \mathbf{x}_{l+1}^\top} \frac{\partial \operatorname{vec} f_l}{\partial \mathbf{w}_l^\top} \end{aligned}$$

表达式需要一些解释。回忆每个函数f_l 是$M \times N \times K$ 的数组到$M' \times N' \times K'$ 数组的映射。符号$\partial \operatorname{vec} f_l / \partial \operatorname{vec} \mathbf{x}_l^\top$ 代表输出向量对输入向量的导数。注意到$\mathbf{w}_l$ 已经是一个列向量，因此它不需要显示的向量化。

问题：

确保你已经理解这个形式的结构回答下列问题：

$\partial \operatorname{vec} f_l / \partial \operatorname{vec} \mathbf{x}_l^\top$ 是一个矩阵，它的维度是多少？

这个形式可以稍微改成别的形式通过吧 f_l 换为 $\mathbf{x}_{l+1}$，如果你这样做的话，你发现什么消减项了没？

形式只包含导数符号。然而，这些导数必须在一个已经定义好的点计算，这个点是哪儿？

为了应用链式法则，我们必须能够对每个 f_l 计算导数。虽然可以暴力的计算，但是由于高达 $M'N'K' \times MNK$ 复杂度。我们将会引入一个技巧，它可以降为 MNK，并且导出反向传播的算法。

关键的一点在于 $\partial f / \partial \mathbf{w}_l^\top$ 而不是 $f_l / \partial \mathbf{w}_l^\top$

$$\frac{\partial f}{\partial \mathbf{w}_l^\top} = \frac{\partial g_{l+1}}{\partial \operatorname{vec} \mathbf{x}_{l+1}^\top} \frac{\partial \operatorname{vec} f_l}{\partial \mathbf{w}_l^\top}$$

其中 $g_{l+1} = f_L \circ \cdots \circ f_{l+1}$ 是 CNN 的尾端。

问题：解释上式为什么成立？特别的，$\partial g_{l+1} / \partial \mathbf{x}_{l+1}$ 和 $\mathbf{x}_{l+1}$ 大小相同。

命中：注意到最后一层就是损失。

因此，算法可以聚焦于计算 g_l 的导数而不是 f_l，这就低了很多维。为了展示怎么迭代的进行，把 g_l 分解为：

$$\mathbf{x}_l \longrightarrow \boxed{f_l} \longrightarrow \mathbf{x}_{l+1} \longrightarrow \boxed{g_{l+1}} \longrightarrow \mathbf{x}_L$$
$$\uparrow$$
$$\mathbf{w}_l$$

随后迭代的关键在于获得 $l+1$ 层相对于 l 层的导数。

输入：

$\partial g_{l+1} / \partial \mathbf{x}_{l+1}$

输出：

$\partial g_l / \partial \mathbf{x}_l$

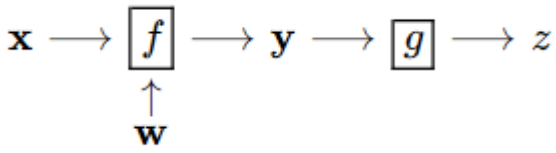
$\partial g_l / \partial \mathbf{w}_l$

问题：

Suppose that f_l is the function $\mathbf{x}_{l+1} = A\mathbf{x}_l$ where $\mathbf{x}_l$ and $\mathbf{x}_{l+1}$ are column vectors. Suppose that $B = \partial g_{l+1} / \partial \mathbf{x}_{l+1}$ is given. Derive an expression for $C = \partial g_l / \partial \mathbf{x}_l$ and an expression for $D = \partial g_l / \partial \mathbf{w}_l$.

2.2 实践中使用的反向传播算法

MatConvNet 类似的关键特征是支持反向传播。为了展示如何使用，让我们关注于一个简单的计算块



其中 z 假设为一个标量。然后每个计算块可以计算在给定输入 $\mathbf{x}$ and $\partial z / \partial \mathbf{y}$ 的情况下计算 $\partial z / \partial \mathbf{x}$ and $\partial z / \partial \mathbf{w}$ 。让我们练习下：

```
x = im2single(imread('peppers.png')) ;

% Create a bank of linear filters and apply them to the image
w = randn(5,5,3,10,'single') ;
y = vl_nnconv(x, w, []) ;

% Create the derivative dz/dy
dzdy = randn(size(y), 'single') ;

% Back-propagation
[dzdx, dzdw] = vl_nnconv(x, w, [], dzdy) ;
```

任务：运行上面的代码，检查 dzdx 和 dzdy 的维度，看看他们是否和你计算的一样。

这种方式的一个好处就是新的块可以很容易的键入。然而，很容易导致导数计算上的错误。因此，数值上检验非常重要。考虑下述的代码：

```
ex = randn(size(x), 'single') ;
eta = 0.0001 ;
xp = x + eta * ex    ;
yp = vl_nnconv(xp, w, []) ;

dzdx_empirical = sum(dzdy(:) .* (yp(:) - y(:)) / eta) ;
dzdx_computed = sum(dzdx(:) .* ex(:)) ;

fprintf(...
   'der: empirical: %f, computed: %f, error: %.2f %%\n', ...
   dzdx_empirical, dzdx_computed, ...
```

```
  abs(1 - dzdx_empirical/dzdx_computed)*100) ;
```

问题:

上述代码中 ex 代表啥意思。

`dzdx_empirical and dzdx_computed` 的导数是啥？

任务：

运行上述代码，并且确保 vl_nnconv 是正确的。

创建这个代码的新版本，并且分别测试上面的导数。

我们现在可以构建第一个 CNN 了，只是由两个层组成，并且计算它们的导数：

问题：注意到最后一个 CNN 是 `dzdx3` 。在这里，为了简单起见，这个导数是随机初始化的。在实际的应用中，这个导数代表了什么？

我们现在可以使用相同的技术来检查反向传播的计算是否正确：

```
ew1 = randn(size(w1), 'single') ;
eta = 0.0001 ;
w1p = w1 + eta * ew1    ;

x1p = x1 ;
x2p = vl_nnconv(x1p, w1p, []) ;
x3p = vl_nnpool(x2p, rho2) ;

dzdw1_empirical = sum(dzdx3(:) .* (x3p(:) - x3(:)) / eta) ;
dzdw1_computed = sum(dzdw1(:) .* ew1(:)) ;

fprintf(...
  'der: empirical: %f, computed: %f, error: %.2f %%\n', ...
  dzdw1_empirical, dzdw1_computed, ...
  abs(1 - dzdw1_empirical/dzdw1_computed)*100) ;
```

第 3 部分学习一个简单的 CNN

在这部分里，我们将学习一个非常简单的 CNN。这个 CNN 是由两层组成的：一个卷积层和一个最大池化层：

$$\mathbf{x}_2 = W * \mathbf{x}_1 + b, \qquad \mathbf{x}_3 = \operatorname{maxpool}_\rho \mathbf{x}_2 .$$

我们包含了 3*3 的方形滤波器，因此 b 是标量。输入图像 x=x1 是单通道图像。

任务：

打开 tinycnn.m 并执行。确保代码计算了 CNN 的节结果。

查看使用的 padding。如果 x1 维度为 M*N，输出特征图 x3 的维度是多少？

3.1 训练数据和标注

第一步是加载图像。使用 extractBlackBlobs 函数来提取图像出的圆点。

Pos 和 neg 数组里面包含了 CNN 监督学习所需的像素数据和标注。这些标注可以通过下列代码可视化：

```
figure(1) ; clf ;
subplot(1,3,1) ; imagesc(im) ; axis equal ; title('image') ;
subplot(1,3,2) ; imagesc(pos) ; axis equal ; title('positive points (blob centres)') ;
subplot(1,3,3) ; imagesc(neg) ; axis equal ; title('negative points (not a blob)') ;
colormap gray ;
```

任务：观察 pos 和 neg，并且确保：

Pos 包含了每个位置中心的单个 true 值。

Neg 对每个远离圆点的像素都为 true。

由没有同时 pos 和 neg 为假的数据？没有。

3.2 图像处理

在我们训练 CNN 之前，图像需要减去均值的预处理。它还会通过 3*3 的高斯滤波进行处理：

```
im = vl_imsmooth(im,3) ;

% Subtract median value
im = im - median(im(:)) ;
```

我们待会还会回来预处理这个步骤。

3.3 使用梯度下降学习

我们现在将设置一个简单的学习问题 W and b 来学习图像中的块。回想 CNN 计算 $\text{pixel}(u,v)$ a score $f(\mathbf{x};\mathbf{w},b)_{(u,v)}$，我们希望这个分数：

至少在块的中心很接近 1

在远离块的中心接近 0

我们通过优化下列目标函数得到：

$$E(\mathbf{w},b) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{|\mathcal{P}|}\sum_{(u,v)\in\mathcal{P}} \max\{0, 1 - f(\mathbf{x};\mathbf{w},b)_{(u,v)}\} + \frac{1}{|\mathcal{N}|}\sum_{(u,v)\in\mathcal{N}} \max\{0, f(\mathbf{x};\mathbf{w},b)_{(u,v)}\}.$$

问题：

如果 $\lambda = 0$ and $E(\mathbf{w},b) = 0$ 意味着什么？

注意到目标函数强制了正例和反例之间的 margin，它是多大？

我们现在通过分别最小化 w 和 b 来最小化目标函数。我们可以通过带有动量项的梯度下降来做到。给定当前的解并且更新它，这是通过寻找下降最快的方向得到的。然而，梯度更新是要考虑动量项 $(\bar{\mathbf{w}}_t, \bar{\mu}_t)$ 的，产生了更新的等式：

$$\bar{\mathbf{w}}_{t+1} \leftarrow \mu\bar{\mathbf{w}}_t + \eta\frac{\partial E}{\partial \mathbf{w}_t}, \qquad \mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \bar{\mathbf{w}}_t.$$

b 也是如此。这里 μ 是动量率二 η 是学习率。

问题：解释问什么动量率必须小于 1.它接近 1 是的作用是什么？

学习率决定了算法最小化目标函数时的速度。使用大的学习率会有什么问题？

这个算法的参数如下所示：

任务：

理解 exercise3.m 文件。确保代码实现了上面的算法。特别注意前向和反向传播是怎样计算的。

运行上述的算法，获得结果。回答下述问题：

学习到的滤波器应该和熟知的差分算子很像，哪个？

和绝对值的平均相比滤波器的输出平均是多少？

重新运行算法， 回答下面的问题：

目标函数时单调递减的吗？

随着直方图的演化，你能发现优化中的两个阶段吗？

一旦收敛，结果是你预期的那样吗？

命中：plotPeriod 可以换位 plot 来诊断高低频率的图。这可能会大大影响算法的速度。

3.4 使用 tiny CNN 实验

在这个部分我们将使用学习到的网络进行几个实验。首先，我们学习图片平滑的影响：

任务：

不使用平滑再来训练一次。回答下面的问题：

学习到的滤波器和原来的是否大不相同？

如果是的，你能找出哪儿错了吗？

认真的查看第一层的输出，并且使用放大境工具。最大的响应是否出现在块的中心？

命中：

拉普拉斯高斯算子只有在块的大小和滤波器大小相同的时候才响应最大。把这个事实带入到预平滑的图片，应用 3*3 的滤波器。

现在重新使用平滑，但是减去输入图片进行中值滤波。

任务：

不使用中值滤波重新训练 CNN。回答下面的问题：

算法收敛了吗？

减少学习率，增大动量项，能不能得到更好的结果？

解释下为什么增加常数能大幅提高优化的性能。

刚才所示的就是通用的准则：中心化数据使得学习问题更易求解。

现在我们将发掘算法中的几个参数：

任务：重新运行 experiment4.m，试下下面的做法：

试试下面的做法：

试着增加学习率。你能在 500 代时获得好的结果吗？

通过 momentum = 0 禁能动量项。尽最大努力选择不同 eta 来获得好的结果。你成功了吗？

最后，考虑正则化的影响：

任务：

在 experiment4.m 重新设置学习率和动量项。然后增加缩减率和手工的折数。

对收敛速度有何影响？

对最后的目标函数和平均损失有何影响？

第 4 部分：学习一个字符 CNN

在本部分我们将学习一个用来识别字符的 CNN。

4.1 准备数据

打开 exercise4.m 并且执行 4.1 部分。代码应该加载 imdb，它包含了 931 种的字符。查看它的结构：

```
imdb = load('data/charsdb.mat') ;
ans =
          id: [1x24206 double]
        data: [32x32x24206 single]
       label: [1x24206 double]
         set: [1x24206 double]
```

imdb.images.id 存储的是一个 24206 维的字符，每个字符包含 32*32 大小的图片，形成了 32x32x24206 大小的数组。imdb.images.label 存储了字符的标注，指示它属于哪一个集合。imdb.images.set 代表了它用来进行训练，2 代表用来进行验证。

http://www.robots.ox.ac.uk/~vgg/practicals/cnn/images/chars.png

任务：确保你理解图 10 的内容。

4.2 初始化一个 CNN 架构。

initializeCharacterCNN.m 使用随机权重初始化 CNN。

任务：

通过查看 initializeCharacterCNN.m 理解训练网络的结构。有多少层？滤波器是多少？

使用 vl_simplenn_display 来产生总结结构的表。

注意到倒数第二层有 26 个输出维度，每个代表一个字符。字符识别寻找最大的响应作为输出。

然而，最后一层是 vl_nnsoftmaxloss，反过来是由 vl_nnsoftmax 函数和对数损失 vl_nnlos 组成的。Softmax 算子如下：

$$y_{ijk'} = \frac{e^{x_{ijk'}}}{\sum_k e^{x_{ijk}}}$$

其中对数损失为：

$$y_{ij} = -\log x_{ijc_{ij}}$$

c_{ij} 是由空间位置给出的真值类别。

任务：

理解 softmax 算法做了什么。使用对数损失后数据一定在 0,1 之间。

理解最小化对数损失的作用。哪个神经元的响应应该变大。

你认为为什么 MatConvNet 提供一个第三方的函数把两个函数融合到一个函数里。

4.3 训练和评估 CNN

我们现在可以开始训练 CNN 了。在这里我们使用 MatConvNet 里提供的随机梯度下降。这个函数需要一些设置：

```
trainOpts.batchSize = 100 ;
trainOpts.numEpochs = 15 ;
trainOpts.continue = true ;
trainOpts.useGpu = false ;
trainOpts.learningRate = 0.001 ;
trainOpts.expDir = 'data/chars-experiment' ;
trainOpts = vl_argparse(trainOpts, varargin);
```

这意味着函数将会以 100 个数据为单位，运行 15 代。即使被中断也会继续训练。它不会使用 GPU，学习率为 0.0001.并在保存文件到 data/chars-experiment 文件夹下。

在开始训练之前，要减去图像的均值。

```
imdb = load('data/charsdb.mat') ;
imageMean = mean(imdb.images.data(:)) ;
imdb.images.data = imdb.images.data - imageMean ;
```

这和我们在第三部分做的一样。

训练通过调用

```
[net,info] = cnn_train(net, imdb, @getBatch, trainOpts) ;
```

这儿就是关键所在了。除了 trainOpts 结构，另外一个就是@getBatch 函数句柄了。这是 cnn_train 函数获取要操作数据的一份拷贝。看下这个函数：

```
function [im, labels] = getBatch(imdb, batch)
im = imdb.images.data(:,:,batch) ;
im = 256 * reshape(im, 32, 32, 1, []) ;
labels = imdb.images.label(1,batch) ;
```

这个函数把 m 个图片提取到一个向量中。它还把他们重新变形为 $32\times32\times1\times m$ 的数组，并且把他们的值乘了 256.最后他还返回每个图片的标注。

任务：

运行学习的代码，看下生成的图片。随着训练的结束，回答下面的问题：

在你处理的过程中，每秒能处理多少数据？

有两个曲线：能量和预测精度。你认为这有什么不同？什么是能量？

有些曲线标注为训练，有的是验证。他们是一样的吗？哪个应该更低一些。

top-1 and top-5 的预测精度都被画了出来。他们代表了什么？区别是什么？

一旦训练结束。模型被存回：

```
net.layers(end) = [] ;
net.imageMean = imageMean ;
save('data/chars-experiment/charscnn-jit.mat', '-struct', 'net') ;
```

注意到我们存下了 imageMean 以便将来使用。在保存之前 softmaxloss 层被移除。

4.4 可视化学习到的滤波器

下一步是看下学习到的滤波器。

```
figure(4) ; clf ;
decodeCharacters(net, imdb, im, vl_simplenn(net, im)) ;
```

任务：从滤波器你能看出什么？

4.5 应用模型

我们现在可以把这个模型应用到字符序列中。有一张 data/sentence-lato.png 图片。

```
net = load('data/chars-experiment/charscnn.mat') ;
im = im2single(imread('data/sentence-lato.png')) ;
im = 256 * (im - net.imageMean) ;
% Apply the CNN to the larger image
res = vl_simplenn(net, im) ;
```

问题：这个图片比 32 像素大多了。你能把它用在之前学习的 32*32 大小的块吗？

现在使用 decodeCharacters()函数来可视化结果：

```
figure(3) ; clf ;
decodeCharacters(net, imdb, im, res) ;
```

任务：看下 decodeCharacters()函数，并且回答下面的问题：

识别结果很好吗？

和你在识别率和验证率中的期望一样吗？

4.6 使用扰动训练

之前 CNN 的一个主要问题是没有识别其他字符的情况下训练的。此外，字符是完美的在图片的中央。完美可以放松这些假设通过使训练数据更“真实”些。在这个部分，完美将会使用扰动训练第二个网络：

随机的在字符左边或者右边添加一个字符。

水平和垂直方向随机的平移 5 或者 2 个像素。

这是通过 getBatchWithJitter()函数实现的。

任务：

加上扰动训练第二个模型。

看下训练误差和验证误差，他们的差距还和之前那样大吗？

使用新的模型来识别之前的图片，工作的好些了吗？

高级。还有什么方法能使识别的性能更好些？

4.7 使用 GPU 加速训练

如果你不想使用 GPU 硬件训练的话跳过这个部分。

深度学习一个关键的挑战在于大量的数据训练的大的模型。最好方法的模型，需要在 GPU 上训练数周，他们在 CPU 是不不可能完成的。因此，实践中学习使用这些硬件是很重要的。

在 MatConvNet 中几乎不费什么力气就可以添加 GPU 支持。你可以试下：

清除之前训练的模型和缓存。

确保 MatConvNet 有 GPU 支持。为了做到这点，使用

setup('useGpu', true) ;

重新使用 exercise4.m 训练，把 useGpu 设为 true。

任务：遵循上述步骤，现在你每秒能处理多少图片了？

对于小的图片来说，只能获得 2-倍的加速，但是对于大的模型，加速将会很大（数十倍）。

第 5 部分：使用预先训练好的模型

深度学习一个主要的特征是学习到了数据的表示。这些表达趋于有一致的值，至少能够用来处理模型训练的数组中。使用 1 块或者多块 GPU 或者上百块 CPU 训练模型，然后在很多应用中重用。不需要另外的工作，真是幸事。

在这部分我们将看到如何下载和运行图片分类中的高性能的模型。这些模型在包含 1000 类的 1.2M 的 ImageNet 图片上训练而来。

几个预先训练好的模型可以在 MatConvNet 网站上下载，包括几个使用其他模型诸如 caffe 等训练的模型。一个模型被包含在 data/imagenet-vgg-verydeep-16.mat，这是 ILSVCR2014 挑战赛中最好的模型之一。

5.1 加载预训练的模型

第一步是加载模型本身。这是使用 vl_simplenn 的 CNN 包装器完成的：

net = load('data/imagenet-vgg-verydeep-16.mat') ;

vl_simplenn_display(net) ;

任务：

观察 vl_simplenn_display 的输出，理解这个模型。你能理解为什么他被称为“很深的模型”了吗？

观察下 imagenet-vgg-verydeep-16.mat 的大小，这只是模型本身的大小，就占了这么多。

5.2 使用模型来分类图片

我们现在可以使用模型来对图片进行分类了。我们选择了 matlab 自带的 peppers.png。

```
im = imread('peppers.png') ;
im_ = single(im) ; % note: 255 range
im_ = imresize(im_, net.normalization.imageSize(1:2)) ;
im_ = im_ - net.normalization.averageImage ;
```

代码把图片变为 net 兼容的图片格式。这意味着：把图片转为单精度格式，缩放到固定的大小，减去均值。

现在总算可以调用 CNN 了：

```
res = vl_simplenn(net, im_) ;
```

一如之前所述，res 包含了计算结果，以及中间层的输出。最后一个可以用来分类：

```
scores = squeeze(gather(res(end).x)) ;
[bestScore, best] = max(scores) ;
figure(1) ; clf ; imagesc(im) ; axis image ;
title(sprintf('%s (%d), score %.3f',...
net.classes.description{best}, best, bestScore)) ;
```

这也意味着本实验的结束。

链接和将来的工作

- The code for this practical is written using the software package MatConvNet. This is a software library written in MATLAB, C++, and CUDA and is freely available as source code and binary.
- The ImageNet model is the *VGG very deep 16* of Karen Simonyan and Andrew Zisserman.

致谢

- Beta testing by: Karel Lenc and Carlos Arteta.
- Bugfixes/typos by: Sun Yushi.

历史

- Used in the Oxford AIMS CDT, 2014-15.