實作論文名稱和簡介

Very Deep Convolutional Networks for Large-Scale Image Recognition（用於大規模圖像識別的深卷積網路）

為了達到更好的準確性，前人的改進：

使用了更小的感受窗口尺寸和更小的第一卷積層步長。（Zeiler & Fergus,2013;Sermanet等,2014）

整個圖像和多個尺度上對網路進行密集地訓練和測試。（Sermanet等，2014；Howard，2014）

論文的改進方案：

解決了另一個重要方面——深度。修正了架構的其他參數，並通過添加更多的卷積層來穩定地增加網路的深度，在所有層中使用非常小的（3×3）卷積濾波器。

要解決問題

1.研究了卷積網路深度在大規模的圖像識別環境下對準確性的影響。

2.使用非常小的（3×3）卷積濾波器架構對網路深度的增加進行了全面評估，表明通過將深度推到16-19加權層可以實現對現有技術配置的顯著改進。

3.對於其他數據集泛化的很好，在其他數據集上取得了最好的結果。

問題和困難

訓練數據尺寸不同，如何訓練？

首先將全連接層轉換到卷積層，第一個全連接層轉換到一個7x7的卷積層，後面兩個轉換到1x1的卷積層，這不僅僅讓全連接層應用到整個未裁剪的原始圖像上，而且能得到一個類別的得分圖，它的通道數等於類別數，還有一個取決於圖片尺寸的可變空間解析度。

LeNet的原理相反？

即使用大的卷積來獲得一張圖像中相似的特徵。和AlexNet的9或11篩檢程式不同，VGG的篩檢程式很小，離LeNet竭力所要避免的1的卷積異常接近--至少在該網路的第一層是這樣。但是VGG巨大的進展是通過依次採用多個3的卷積，能夠模仿出更大的感受野的效果，例如5或7.這些思想也被用在了最近的更多的網路架構上。如Inception與ResNet。

實做論文名稱和簡介（2）

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

結構分析

卷積層，池化層&全連通層是vgg的結構。

--->卷積層濾波器啟動函數為線性整流函數（relu）濾波器的尺寸為 3*3 ，步長為1，padding為1，圖像通過此層尺寸不會發生變化。vgg包括5個卷積段，平均每一段內有2或者3個卷積層，每段尾部會連接一個最大池化層。

--->池化層窗口大小 2*2 ，步長為2。圖像通過此層長寬都變成原來的一半。

輸入為(N,224,224,3)的圖像

通過池化層1--->(N,112,112,64)，通過池化層2--->(N,56,56,128)，通過池化層3--->(N,28,28,256)，通過池化層4--->(N,14,14,512)，通過池化層5--->(N,7,7,512)。然後鋪平為(N,25088)的全連層輸入，經過三層全連通層後輸出(N,1000)的邏輯。

論文的描述

我們的ConvNet的輸入是固定大小的224×224 RGB圖像。我們唯一的預處理是從每個像素中減去在訓練集上計算的RGB均值。圖像通過一堆卷積層，我們使用感受野很小的濾波器：3×3（這是捕獲左/右，上/下，中心概念的最小尺寸）。在其中一種配置中，我們還使用了1×1卷積濾波器，可以看作輸入通道的線性變換（後面是非線性）。卷積步長固定為1個像素；卷積層輸入的空間填充要滿足卷積之後保留空間解析度，即3×3卷積層的填充為1個像素。空間池化由五個最大池化層進行，這些層在一些卷積層之後（不是所有的卷積層之後都是最大池化）。在2×2像素窗口上進行最大池化，步長為2。

一堆卷積層（在不同架構中具有不同深度）之後是三個全連接（FC）層：前兩個每個都有4096個通道，第三個執行1000維ILSVRC分類，因此包含1000個通道（一個通道對應一個類別）。最後一層是soft-max層。所有網路中全連接層的配置是相同的。

所有隱藏層都配備了修正（ReLU）非線性。我們注意到，我們的網路（除了一個）都不包含局部回應規範化（LRN）：將在第4節看到，這種規範化並不能提高在ILSVRC數據集上的性能，但增加了記憶體消耗和計算時間。

局部回應歸一化（Local Response Normalization）

$$b_{x,y}^{i} = a_{x,y}^{i} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^{j})^{2} \right)^{\beta}$$

局部回應歸一化原理是仿造生物學上活躍的神經元對相鄰神經元的抑制現象（側抑制）

這個公式中的a表示卷積層（包括卷積操作和池化操作）後的輸出結果，這個輸出結果的結構是一個四維數組[batch,height,width,channel]，這裏可以簡單解釋一下，batch就是批次數(每一批為一張圖片)，height就是圖片高度，width就是圖片寬度，channel就是通道數可以理解成一批圖片中的某一個圖片經過卷積操作後輸出的神經元個數(或是理解成處理後的圖片深度)。ai(x,y)表示在這個輸出結構中的一個位置[a,b,c,d]，可以理解成在某一張圖中的某一個通道下的某個高度和某個寬度位置的點，即第a張圖的第d個通道下的高度為b寬度為c的點。論文公式中的N表示通道數(channel)。a,n/2,k,α,β分別表示函數中的input,depth_radius,bias,alpha,beta，其中n/2,k,α,β都是自定義的，特別注意一下∑疊加的方向是沿著通道方向的，即每個點值的平方和是沿著a中的第3維channel方向的，也就是一個點同方向的前面n/2個通道（最小為第0個通道）和後n/2個通道（最大為第d-1個通道）的點的平方和(共n+1個點)。而函數的英文注解中也說明了把input當成是d個3維的矩陣，說白了就是把input的通道數當作3維矩陣的個數，疊加的方向也是在通道方向。（參考資料：https://blog.csdn.net/sinat_21585785/article/details/75087768）

測試分析

我們輸入的圖片的尺寸可以與訓練圖片的尺寸相同，也可以不同。尺寸不同的圖片經過插值變換直接送入模型進行訓練。

先讓全連接層覆蓋到原始圖像上，文章中訓練輸入圖像尺寸均為224x224，這樣網路的參數就已經固定了。圖像尺寸只會與pool層相關，經過5層pool層後，最後輸出為7x7x512的尺寸。

由於全連通層參數已經固定了。我們把全連通層繼續看作卷積層，第一個全連通層的參數 W=(7*7*512,4096) ,將其轉換一下，變成 W'=(7,7,512,4096) 的濾波器，得到一個一共4096個通道的特徵映射，如果測試的圖像大小剛好為 224*224 ，則此時輸出一個 (N,1,1,4096) 的特徵映射。

本文與其它論文的討論

我們的ConvNet配置與ILSVRC-2012（Krizhevsky等，2012）和ILSVRC-2013比賽（Zeiler＆Fergus，2013；Sermanet等，2014）表現最佳的參賽提交中使用的ConvNet配置有很大不同。不是在第一卷積層中使用相對較大的感受野（例如，在（Krizhevsky等人，2012）中的11×11，步長為4，或在（Zeiler＆Fergus，2013；Sermanet等，2014）中的7×7，步長為2），我們在整個網路使用非常小的3×3感受野，與輸入的每個像素（步長為1）進行卷積。很容易看到兩個3×3卷積層堆疊（沒有空間池化）有5×5的有效感受野；三個這樣的層具有7×7的有效感受野。

Ciresan等人（2011）以前使用小尺寸的卷積濾波器，但是他們的網路深度遠遠低於我們的網路，他們並沒有在大規模的ILSVRC數據集上進行評估。Goodfellow等人（2014）在街道號識別任務中採用深層ConvNets（11個權重層），顯示出增加的深度導致了更好的性能。GooLeNet（Szegedy等，2014），ILSVRC-2014分類任務的表現最好的專案，是獨立於我們工作之外的開發的，但是類似的是它是基於非常深的ConvNets（22個權重層）和小卷積濾波器（除了3×3，它們也使用了1×1和5×5卷積）。然而，它們的網路拓撲結構比我們的更複雜，並且在第一層中特徵圖的空間解析度被更積極地減少，以減少計算量。

訓練尺度

我們考慮兩種方法來設置訓練尺度S。第一種是修正對應單尺度訓練的S（注意，採樣裁剪圖像中的圖像內容仍然可以表示多尺度圖像統計）。在我們的實驗中，我們評估了以兩個固定尺度訓練的模型：$S = 256$（已經在現有技術中廣泛使用（Krizhevsky等人，2012；Zeiler＆Fergus，2013；Sermanet等，2014））和$S = 384$。給定ConvNet配置，我們首先使用$S=256$來訓練網路。為了加速$S = 384$網路的訓練，用$S = 256$預訓練的權重來進行初始化，我們使用較小的初始學習率$10^{−3}$。

設置S的第二種方法是多尺度訓練，其中每個訓練圖像通過從一定範圍$[S_{min}，S_{max}]$（我們使用$S_{min} = 256$和$S_{max} = 512$）隨機採樣S來單獨進行歸一化。由於圖像中的目標可能具有不同的大小，因此在訓練期間考慮到這一點是有益的。這也可以看作是通過尺度抖動進行訓練集增強，其中單個模型被訓練在一定尺度範圍內識別對象。為了速度的原因，我們通過對具有相同配置的單尺度模型的所有層進行微調，訓練了多尺度模型，並用固定的$S = 384$進行預訓練。

實做過程（1）

圖像預處理1：用caffe訓練的模型使用BGR，所以需要把圖像的RGB順序顛倒。

```
red, green, blue = tf.split(axis=3, num_or_size_splits=3, value=rgb_scaled)
    assert red.get_shape().as_list()[1:] == [224, 224, 1]
    assert green.get_shape().as_list()[1:] == [224, 224, 1]
    assert blue.get_shape().as_list()[1:] == [224, 224, 1]
```

圖像預處理2：減去的均值是數據集所有圖片的RGB三個通道的均值構成的向量[Rmean, Gmean, Bmean]，每個通道各一個均值。然後所有圖像都減去此向量。

```
VGG_MEAN = [103.939, 116.779, 123.68]
...
bgr = tf.concat(axis=3, values=[ blue - VGG_MEAN[0],green - VGG_MEAN[1],red - VGG_MEAN[2], ])
```

實做過程（2）

conv1: 224x224x64

```
self.conv1_1 = self.conv_layer(bgr, 3, 64, "conv1_1")
self.conv1_2 = self.conv_layer(self.conv1_1, 64, 64, "conv1_2")
self.pool1 = self.max_pool(self.conv1_2, 'pool1')
```

conv2: 112x112x128

```
self.conv2_1 = self.conv_layer(self.pool1, 64, 128, "conv2_1")
self.conv2_2 = self.conv_layer(self.conv2_1, 128, 128, "conv2_2")
self.pool2 = self.max_pool(self.conv2_2, 'pool2')
```

conv3: 56x56x256

```
self.conv3_1 = self.conv_layer(self.pool2, 128, 256, "conv3_1")
self.conv3_2 = self.conv_layer(self.conv3_1, 256, 256, "conv3_2")
self.conv3_3 = self.conv_layer(self.conv3_2, 256, 256, "conv3_3")
self.conv3_4 = self.conv_layer(self.conv3_3, 256, 256, "conv3_4")
self.pool3 = self.max_pool(self.conv3_4, 'pool3')
```

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

實做過程（3）

conv4: 28x28x512

```
self.conv4_1 = self.conv_layer(self.pool3, 256, 512, "conv4_1")
self.conv4_2 = self.conv_layer(self.conv4_1, 512, 512, "conv4_2")
self.conv4_3 = self.conv_layer(self.conv4_2, 512, 512, "conv4_3")
self.conv4_4 = self.conv_layer(self.conv4_3, 512, 512, "conv4_4")
self.pool4 = self.max_pool(self.conv4_4, 'pool4')
```

conv5: 14x14x512

```
self.conv5_1 = self.conv_layer(self.pool4, 512, 512, "conv5_1")
self.conv5_2 = self.conv_layer(self.conv5_1, 512, 512, "conv5_2")
self.conv5_3 = self.conv_layer(self.conv5_2, 512, 512, "conv5_3")
self.conv5_4 = self.conv_layer(self.conv5_3, 512, 512, "conv5_4")
self.pool5 = self.max_pool(self.conv5_4, 'pool5')
```

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

實做過程（4）

fc6: 25088(=7x7x512)x4096

```
self.fc6 = self.fc_layer(self.pool5, 25088, 4096, "fc6")  # 25088 = ((224 // (2 ** 5)) ** 2) * 512
        self.relu6 = tf.nn.relu(self.fc6)
        if train_mode is not None:
            self.relu6 = tf.cond(train_mode, lambda: tf.nn.dropout(self.relu6, self.dropout),
lambda: self.relu6)
        elif self.trainable:
            self.relu6 = tf.nn.dropout(self.relu6, self.dropout)
```

fc7: 4096x4096

```
        self.fc7 = self.fc_layer(self.relu6, 4096, 4096, "fc7")
        self.relu7 = tf.nn.relu(self.fc7)
        if train_mode is not None:
            self.relu7 = tf.cond(train_mode, lambda: tf.nn.dropout(self.relu7, self.dropout),
lambda: self.relu7)
        elif self.trainable:
            self.relu7 = tf.nn.dropout(self.relu7, self.dropout)
```

fc8: 4096x1000

```
        self.fc8 = self.fc_layer(self.relu7, 4096, 1000, "fc8")
        self.prob = tf.nn.softmax(self.fc8, name="prob")
```

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

實做過程（5）

一些基礎功能和工具

```
def avg_pool(self, bottom, name):
def max_pool(self, bottom, name):
def conv_layer(self, bottom, in_channels, out_channels, name):
def fc_layer(self, bottom, in_size, out_size, name):
def get_conv_var(self, filter_size, in_channels, out_channels, name):
def get_fc_var(self, in_size, out_size, name):
def get_var(self, initial_value, name, idx, var_name):
def save_npy(self, sess, npy_path="./vgg19-save.npy"):
def get_var_count(self):
```

論文的實驗結果（1）

單尺度評估

LRN並不能改善A網路性能

分類誤差隨著深度增加而降低

在訓練的時候採用圖像尺度抖動可以改善圖像分類效果

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

論文的實驗結果（2）

多尺度評估

相對於單一尺度評估，多尺度評估提高了分類精度

在訓練的時候採用圖像尺度抖動可以改善圖像分類效果

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

我的實驗結果（1）

下載ImageNet 1000種分類
以及排列
使用vgg網路識別右邊圖片，
識別結果中最大概率5分類
名稱和概率值：

n02231487 walking stick,
walkingstick, stick insect 0.131548
n01608432 kite 0.110931
n02236044 mantis, mantid 0.0947501
n01784675 centipede 0.0609114
n02226429 grasshopper, hopper 0.0406519

我的實驗結果（2）

本文介紹的方法略好於AlexNet，但訓練時間多一些

Type	train	test	Time	top-3 66%HIT
ConvNet F	10000	1000	1	82.33%
AlexNet	10000	1000	0.77	76.25%

實作總結

優點：簡潔易懂。從另一個角度提高了識別效率。

缺點：參數太多，計算耗費時間多。調試不太容易。

實做時，計算機的顯存只有2G，不夠放置所有的參數，所以沒有使用顯卡，而是轉爲使用CPU，同時，項目耗費了大量的時間。

演示結束，謝謝。