

# 本科生毕业设计（论文）参考文献译文本

译文出处：Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]. Advances in neural information processing systems. 2012, 25(2): 1097-1102.

院 系 软件学院

专业班级 软件工程1205

姓 名 陈吕劼

学 号 U201217478

指导教师 管乐

2016年1月

**译文要求**

1. 译文内容须与课题（或专业内容）联系，并需在封面注明详细出处。
2. 出处格式为

图书：作者.书名.版本（第×版）.译者.出版地：出版者，出版年.起页～止页  
期刊：作者.文章名称.期刊名称，年号，卷号（期号）：起页～止页

1. 译文不少于5000汉字（或2万印刷符）。
2. 翻译内容用五号宋体字编辑，采用A4号纸双面打印，封面与封底采用浅蓝色封面纸（卡纸）打印。要求内容明确，语句通顺。
3. 译文及其相应参考文献一起装订，顺序依次为封面、译文、文献。
4. 翻译应在第七学期完成。

**译文评阅**

|  |
| --- |
| **导师评语**  应根据学校“译文要求”，对学生译文翻译的准确性、翻译数量以及译文的文字表述情况等做具体的评价后，再评分。 |
| 评分：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_（百分制） 指导教师（签名）：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  2016年1月22日 |

**基于深层卷积神经网络的ImageNet图像分类**

**摘要**

我们使用了大型深层卷积神经网络将2010年ImageNet LSVRC比赛中的120万张高分辨率图像分类成1000个不同的类别。在测试数据集上，我们分别在首个匹配和前五匹配准则下达到了37.5%和17.0%的错误率，而这个结果相比于以往最好的结果有了相当大的提升。我们的神经网络包含有6000万个参数和650000个神经单元，由5个卷积层和3个全连接层以及一个1000维的softmax分类层的组成，部分卷积层后还跟有最大池化层。为了加快学习速度，我们使用了非饱和神经元和利用GPU实现的非常快速的卷积操作。为减少全连接层中的过拟合问题，我们采用了最新开发被证明非常有效的名为“dropout”的正则化方法。我们也在2012年ILSVRC比赛中使用了这个模型的变种并获得优胜，相比于第二名26.2%的错误率，我们前五匹配准则下的错误率为15.3%。

1. **简介**

当前物体识别的实现都充分利用了机器学习方法，为了提升它们的性能，我们可以收集大量的数据，选择更强大的学习模型以及使用更好的技术来预防过拟合。直到最近，被标记的图像数据集依旧较小，大约成千上万张图片左右(例如., NORB [16], Caltech-101/256 [8, 9], and CIFAR-10/100 [12])。在这样大小的数据集下，简单识别任务可以很好得被处理，尤其是在通过了类别不变转换的增强后，比如，当前对于MNIST数字识别任务最好处理的错误率（小于0.3%）已经达到人类的水平[4]。但是，在现实情况中的物体则有相当大的变化，所以学习并识别它们需要更庞大的训练集。而且实际上，数据集太小的缺点已经被广泛认可(例如,Pintoetal.[21])，但是直到最近收集数以百万计被标记的图像才成为可能。最新更大型的数据集包括LabelMe[23]，拥有成千上万的完全划分图像，以及拥有超过22000个种类1500万张高像素图像的ImageNet[6]。

为了在上百万张图片中学习上千种类别的物体，我们需要一个拥有强大学习能力的模型。然而，物体识别任务的巨大复杂性使得即使在像ImageNet这样庞大的数据集下问题都难以解决，所以我们的模型需要一些先进的知识来弥补我们所缺失的数据。卷积神经网络构成了这样一类模型[16, 11, 13, 18, 15, 22, 26]，它的能力随网络的深度和广度变化而改变。同时，它能对图像的性质做出强大和大致正确的假设。因此，相比较于同样规模的标准前馈神经网络，卷积神经网络拥有更少的来连接和参数，所以更易训练，同时它理论上最佳性能也只有略微变差。

尽管卷积神经网络有着吸引人的特质以及相对高效的网络结构，但是它应用在大型高分辨率图像上依旧十分耗费资源。幸运的是，如今的GPU已经配备了对二维卷积的高度优化实现以致足以支持大型卷积神经网络的训练，而且例如ImageNet的现代数据集也包含足够多的带标签的图像来避免训练模型中的严重过拟合。

本文具体的贡献如下：我们利用2010和2012年ILSVRC比赛中的ImageNet子训练集，训练了一个至今为止最大的卷积神经网络，并且获得了到目前为止在数据集上发表过的最好的成绩。我们写了一个对于二维卷积和其他所有训练卷积神经网络操作的高度优化的GPU实现，而这些都是开源可用的。在第3节，我们会详细介绍网络包含的一系列新鲜且特别的提高性能和减少训练时间的特性。即使拥有120万张图片的训练集，网络规模带来的过拟合依然成为一个重要的问题，所以在第四节，我们会介绍一些有效的技术来预防过拟合。我们最终的网络包含5个卷积层和3个全连接层，而且它的深度似乎非常重要。我们发现无论去除任何一个卷积层（每一层都包含不超过1%的模型参数），都会导致性能较差。

最后，网络的规模主要受限于目前GPU的可用内存和我们愿意容忍的训练时间量。在使用两个GTX 580 3GB GPU的情况下，我们的网络训练了五到六天的时间。我们所有的实验表明更快的GPU和更大的数据集就能简单得使我们的结果变好。

1. **数据集**

ImageNet是一个拥有大约22000个种类超过1500万标记好高质量图片的数据集。这些图片从网络上收集并通过亚马逊的Mechanical Turk众包工具被人标记。从2010年开始，作为帕斯卡可视对象挑战的一部分，名为ImageNet大型视觉识别挑战（ILSVRC）的竞赛每年举办一次。这项比赛使用ImageNet的一个子数据集，其中包含1000个种类各大约1000张图片总共大约120万张训练图片和50000张验证图像以及150000张测试图片。

2010年的ILSVRC比赛是唯一一届测试数据集标记可用的，所以我们在此基础上做了大多数的实验。由于我们同样参加了2012年的ILSVRC比赛，我们在第6节也呈现了这一年数据集的结果，而这一年的测试数据的标记是未知的。在ImageNet上，习惯呈现两个错误率：首次匹配错误率和前五匹配错误率。前五匹配错误率是指正确的标记不在模型给出的前五种可能的标记之中的概率。

虽然ImageNet的图像分辨率很高，但我们的系统要求一个固定的输入维数。因此，我们对图像进行采样得到固定大小256 x 256的图像。对于给定的矩形图像，我们首先将较短的边缩小到256像素长，然后裁剪正中间的256 x 256大小的一块作为结果。除了减去每个像素在训练过程中的平均活度，我们并未使用任何其他方法来预处理图像，所以都是直接使用每个像素的RGB值来训练网络。

1. **体系结构**

图二显示了我们网络的结构。它包含八个学习层——五个卷积层和三个全连接层。接下来，我将介绍一些关于网络结构的新颖或特别的特性。3.1到3.4节将根据重要性从高到低罗列这些特性。

3.1 ReLU非线性

根据输入x，建立一个神经元的激活函数f的一般方式为或者。考虑到使用梯度下降方法的训练时间，这些饱和非线性方法远慢于非饱和非线性方法。根据Nair和Hinton的研究成果[20]，我们侵向于使用整合线性单元（ReLU）非线性激活函数的神经单元。在训练深层卷积神经网络时使用ReLU要比使用tanh快上好几倍。从图1中显示了在CIFAR-10数据集上使用一个四层的卷积神经网络训练时达到25%错误率时的周期次数。这幅图显示出我们如果使用传统的饱和神经元模型是无法训练如此庞大的神经网络的。

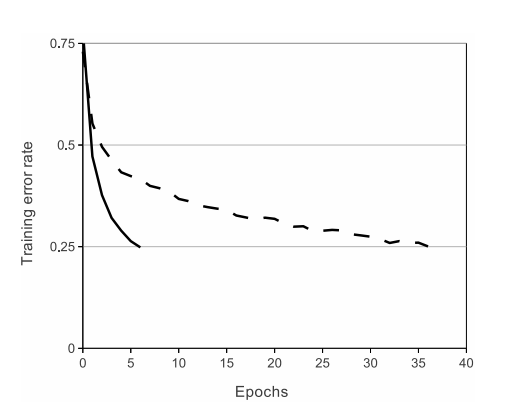


图1: 在一个四层卷积层的神经网络上对CIFAR-10使用ReLUs（实线）达到25%训练错误率是在同等网络上使用tanh神经元（虚线）速度的6倍。每一个神经网络的学习速率都是相互独立且尽可能得快并且没有使用任何的正规化方法。虽然对于不同的神经网络这种效果的大小各异，但是使用ReLU方法训练都要比同等环境下的饱和神经元方法快上几倍。

我们并不是最先想到要替换卷积神经网络中传统神经元的人。例如，Jarrett等人[11]宣称非线性方法在Caltech-101数据集上配合他们对应的正则化方法以及局部平均池化方法效果很好。然而，在这个数据上最主要的考虑是要避免过拟合，所以他们所努力探索的和我们之前提到使用ReLU来提升适应数据集的能力有所不同。更快的学习速度对于在大型数据集上使用大型模型训练的性能有很影响。

3.2 多GPU训练

一个GTX 580 GPU只拥有3GB的内存，这会限制可训练的模型的最大规模。事实证明120万张图片是足以训练神经网络的，但是这对于单一GPU来说规模太大，因此我们将网络拆分到两个GPU上。如今的GPU因为可以相互直接读取和写入对方的内存而不经过主机的机器内存，所以非常适合于跨GPU的并行运算。并行策略是将内核（或神经元）平分到每个GPU上，还有一个小策略是GPU仅在某些特定层通信。这意味着，例如第三层内核接受所有来自第二层的内核为输入，然而第四层内核只接受在同一个GPU上的第三层的内核为输入。选择连接模式是一个交叉验证的问题，但是这个使得我们可以精准地调整所有的通信直到所有结果的满足要求。

最终的结构在某种程度上和Ciresan等人[5]设计的“柱状”CNN很类似，除了我们每一栏都不是独立的（见图2）。使用这种策略，相比较于使用一个GPU在每一层仅使用一半的内核来训练，首次匹配和前五匹配的错误率分别降低了1.7%和1.2%。而两个GPU训练的时间只比单GPU略少一点。

3.3 局部响应归一化

ReLU拥有令人满意的特性使用我们不需要对输入进行归一化来防止饱和。只要至少有一些训练用例对ReLU产生了正输出，神经元就会得到学习。然而，我们依旧发现接下来的这种局部归一化策略能够促进统一化。用来表示在(x,y)位置的神经元应用了内核i和ReLU非线性变换后的活性，而响应归一化的活性可以由下面的表达式得到



其中在同一个位置使用n个“调整”内核计算总和，N是一层中所有内核的总数。内核映射的顺序肯定是任意的而且事先决定的。这种响应规范化实习了一种横向抑制，这种灵感来源于真正的神经元，创造出利用不同内核计算神经元输出的巨大活动竞争。常量，，和是根据验证集决定的超参数，我们设置，，，。在特定层上，我们应用了ReLU非线性变换后还应用了归一化（见3.5节）。

这个策略与Jarrett等人[11]的局部对比度归一化策略有某些相似之处，但是我们的策略更可被称为“亮度归一化”，因为我们没有去掉平均活度。响应归一化分别将首次匹配和前五匹配的错误率降低了1.4%和1.2%。我们同样在CIFAR-10数据集上验证了这种策略的有效性，一个四层的CNN在不适用归一化下的测试错误率为13%而使用了归一化后为11%。

3.5 整体架构

现在我们已经可以介绍我们CNN的整体架构了。正如图2所示，网络包含八个拥有权值的层，前五层为卷积层，剩下三层为全连接层。最后一个全连接层的输出是使用了1000维softmax来产生1000类标记的分布。我们的网络最大化多元逻辑回归目标，这等同于最大限度地提高训练用例的预测分布中正确标记的平均对数概率值。

第二、四、五层卷积层的内核仅仅和处在同一GPU上的前一层的内核相关联（见图2），第三层卷积层的内核则和第二层的所有内核相关联。响应归一化层分别跟在第一和第二层卷积层后。在3.4节介绍的最大池化层跟在两个响应归一化层和第五层卷积层后。ReLU非线性方法被应用到每一个卷积层和全连接层的输出中。

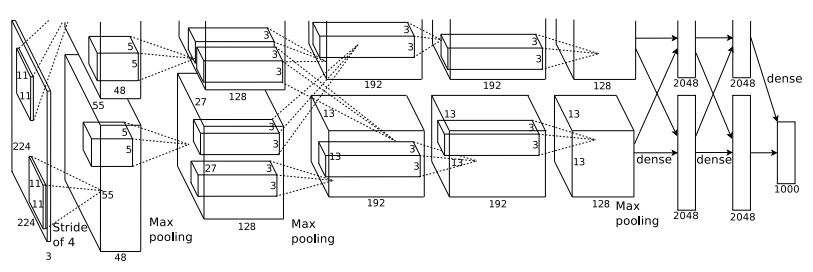


图2 CNN的架构图解，详细展示了两个GPU的责任划分。一个GPU负责处理网络的上半部另一个处理网络的下半部，两个GPU仅在特定层有交流。网络的输入时150528维的，而且剩下每层的神经元数量分别是253440-186624-64896-64896-43264-4096-4096-1000。

第一个卷积层以4个像素为步进（这是一个内核映射相邻神经元的感受野中心的距离）使用96个11 x 11 x 3的内核来过滤224 x 224 x 3的输入图像。第二层卷积层使用第一层的输出（响应归一化和池化后）作为输入，并使用256个5 x 5 x 48大小的内核进行过滤。第三、四、五层卷积层相互连接，没有任何干预的池化和归一化层。第三层卷积层有384个3 x 3 x 256的内核和第二层的输出（归一化和池化后）连接。第四层卷积层有384个3 x 3 x 192大小的内核，第五层有256个3 x 3 x 192大小的内核，全连接层每层有4096个神经元。

1. **降低过拟合**

我们的神经网络有6000万个参数，即使1000个种类的ILSVRC使得每个训练用例都施加10位从图像到标签的映射约束，这也难以摆脱如此多参数下可观的过拟合问题。下面，我们介绍两种主要的方法来对抗过拟合。

4.1 数据增强

最简单且常用减少图像数据过拟合的方法就是人工使用保留标签变换的方法增大数据集的规模（例如，[25，4，5]）。我们采用了两种不同形式的数据加强方法，两种方法都只需要对原始图像做简单的运算即可得到转换后的图像，所以这些转换后的图像并不要存储在硬盘中。在实现过程中，我们使用Python代码在CPU中转换图像，同时GPU会对前一批图像进行训练。所以这种图像增强的策略实际上是计算自由的。

第一种数据增强由生成图像转换和水平反射组成。我们通过从256 x 256图像上随机提取224 x 224大小的区块（以及他们的水平反射），并通过这些区块训练我们的网络。这种方法将我们训练集扩大了2048倍，虽然我们得到的训练实例是高度互相依赖的。没有这种策略的话，我们的网络将遭受大量的过拟合，这将迫使我们使用更小规模的网络。在测试阶段，我们网络通过提取的5个224 x 224大小的区块（四个角上的区块和正中间的区块）以及它们的水平反射（因此总共十个区块）来进行预测，而且通过网络的softmax层来均衡10个区块的预测结果。

第二种数据增强通过改变RGB通道的强度来训练图像。具体来说，我们对ImageNet训练集中每个RGB像素值使用PCA操作。对于每个训练图像，我们将添加已发现的主成分的倍数，以及大小成比例的相应特征值和从均值为0标准差为0.1的高斯分布中选择的一个随机变量的乘积。因此对于每一个RGB图像像素，我们如下增强：



其中和分别是RGB像素值对应的3 x 3协方差矩阵的第i个特征向量和特征值，是前面提到过的随机变量。每一个对于一个特定的图像上的所有像素点都只被选取一次直到这个图像被用于再次训练而重绘。这种策略近似捕捉自然图像的一个重要特性，即物体的特性是不变的，只是在亮度和颜色的照明有变化，它将首次匹配的错误率降低了超过1%。

4.2 舍弃

结合许多不同模型的预测是一个用来降低测试错误率非常成功的方法[1，3]，但是这显然对于一个已经训练很多天了的大型神经网络来说耗费太大。然而，有一种非常高效的模型组合在训练过程中仅仅只有一两个影响的因素。最新推出名为“Dropout”[10]的技术，让每个神经元都有0.5的概率使结果被设置为零。被这种方式“舍弃”的神经元不会对下一层有影响也不会参与到反向传播过程。所以每次一个输入被确定，神经网络都使用一个不同的架构，但是所有这些加收共享权值。这项技术降低了神经元之间互相适应的复杂度，因为一个神经元不需要依赖其他任何特定的神经元。因此，它被迫学习更多健壮的特性，有利于和很多不同的随机的神经元子集相连。在测试时，我们将所有神经元的输出都乘以0.5，这是对指数级dropout网络产生的预测分布的几何平均的一个合理估计。

我们对前两个全连接层使用了dropout，见图2。缺少了dropout，我们的网络将具有大量的过拟合。Dropout大约将使收敛的迭代次数翻倍。

**参考文献原文**

**ImageNet Classification with Deep Convolutional** **Neural Networks**

**Abstract**

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

**1 Introduction**

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labeled images were relatively small — on the order of tens of thousands of images (e.g., NORB [16], Caltech-101/256 [8, 9], and CIFAR-10/100 [12]). Simple recognition tasks can be solved quite well with datasets of this size, especially if they are augmented with label-preserving transformations. For example, the current best error rate on the MNIST digit-recognition task (<0.3%) approaches human performance [4]. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets. And indeed, the shortcomings of small image datasets have been widely recognized (e.g., Pinto et al. [21]), but it has only recently become possible to collect labeled datasets with millions of images. The new larger datasets include LabelMe [23], which consists of hundreds of thousands of fully-segmented images, and ImageNet [6], which consists of over 15 million labeled high-resolution images in over 22,000 categories.

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don’t have. Convolutional neural networks (CNNs) constitute one such class of models [16, 11, 13, 18, 15, 22, 26]. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Despite the attractive qualities of CNNs, and despite the relative efficiency of their local architecture, they have still been prohibitively expensive to apply in large scale to high-resolution images. Luckily, current GPUs, paired with a highly-optimized implementation of 2D convolution, are powerful enough to facilitate the training of interestingly-large CNNs, and recent datasets such as ImageNet contain enough labeled examples to train such models without severe overfitting.

The specific contributions of this paper are as follows: we trained one of the largest convolutional neural networks to date on the subsets of ImageNet used in the ILSVRC-2010 and ILSVRC-2012 competitions [2] and achieved by far the best results ever reported on these datasets. We wrote a highly-optimized GPU implementation of 2D convolution and all the other operations inherent in training convolutional neural networks, which we make available publicly1. Our network contains a number of new and unusual features which improve its performance and reduce its training time, which are detailed in Section 3. The size of our network made overfitting a significant problem, even with 1.2 million labeled training examples, so we used several effective techniques for preventing overfitting, which are described in Section 4. Our final network contains five convolutional and three fully-connected layers, and this depth seems to be important: we found that removing any convolutional layer (each of which contains no more than 1% of the model’s parameters) resulted in inferior performance.

In the end, the network’s size is limited mainly by the amount of memory available on current GPUs and by the amount of training time that we are willing to tolerate. Our network takes between five and six days to train on two GTX 580 3GB GPUs. All of our experiments suggest that our results can be improved simply by waiting for faster GPUs and bigger datasets to become available.

**2 The Dataset**

ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories. The images were collected from the web and labeled by human labelers using Amazon’s Mechanical Turk crowd-sourcing tool. Starting in 2010, as part of the Pascal Visual Object Challenge, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held. ILSVRC uses a subset of ImageNet with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images.

ILSVRC-2010 is the only version of ILSVRC for which the test set labels are available, so this is the version on which we performed most of our experiments. Since we also entered our model in the ILSVRC-2012 competition, in Section 6 we report our results on this version of the dataset as well, for which test set labels are unavailable. On ImageNet, it is customary to report two error rates: top-1 and top-5, where the top-5 error rate is the fraction of test images for which the correct label is not among the five labels considered most probable by the model.

ImageNet consists of variable-resolution images, while our system requires a constant input dimensionality. Therefore, we down-sampled the images to a fixed resolution of 256 × 256. Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image. We did not pre-process the images in any other way, except for subtracting the mean activity over the training set from each pixel. So we trained our network on the (centered) raw RGB values of the pixels.

**3 The Architecture**

The architecture of our network is summarized in Figure 2. It contains eight learned layers — five convolutional and three fully-connected. Below, we describe some of the novel or unusual features of our network’s architecture. Sections 3.1-3.4 are sorted according to our estimation of their importance, with the most important first.

**3.1 ReLU Nonlinearity**

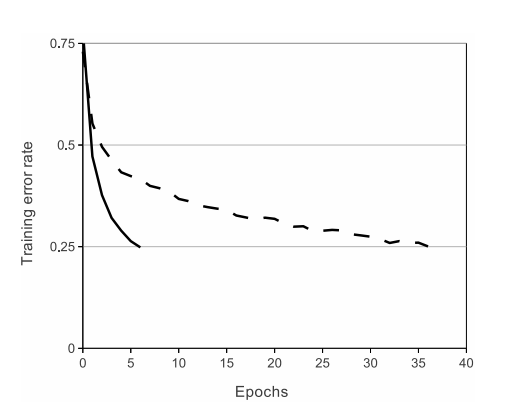


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

The standard way to model a neuron’s output f as a function of its input x is with f(x) = tanh(x) or f(x) = (1 + e−x)−1. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity f(x) = max(0, x). Following Nair and Hinton [20], we refer to neurons with this nonlinearity as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units. This is demonstrated in Figure 1, which shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network. This plot shows that we would not have been able to experiment with such large neural networks for this work if we had used traditional saturating neuron models.

We are not the first to consider alternatives to traditional neuron models in CNNs. For example, Jarrett et al. [11] claim that the nonlinearity f(x) = |tanh(x)| works particularly well with their type of contrast normalization followed by local average pooling on the Caltech-101 dataset. However, on this dataset the primary concern is preventing overfitting, so the effect they are observing is different from the accelerated ability to fit the training set which we report when using ReLUs. Faster learning has a great influence on the performance of large models trained on large datasets.

**3.2 Training on Multiple GPUs**

A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it. It turns out that 1.2 million training examples are enough to train networks which are too big to fit on one GPU. Therefore, we spread the net across two GPUs. Current GPUs are particularly well-suited to cross-GPU parallelization, as they are able to read from and write to one another’s memory directly, without going through host machine memory. The parallelization scheme that we employ essentially puts half of the kernels (or neurons) on each GPU, with one additional trick: the GPUs communicate only in certain layers. This means that, for example, the kernels of layer 3 take input from all kernel maps in layer 2. However, kernels in layer 4 take input only from those kernel maps in layer 3 which reside on the same GPU. Choosing the pattern of connectivity is a problem for cross-validation, but this allows us to precisely tune the amount of communication until it is an acceptable fraction of the amount of computation.

The resultant architecture is somewhat similar to that of the “columnar” CNN employed by Ciresan et al. [5], except that our columns are not independent (see Figure 2). This scheme reduces our top-1 and top-5 error rates by 1.7% and 1.2%, respectively, as compared with a net with half as many kernels in each convolutional layer trained on one GPU. The two-GPU net takes slightly less time to train than the one-GPU net2.

**3.3 Local Response Normalization**

ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, we still find that the following local normalization scheme aids generalization. Denoting by the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity  is given by the expression



where the sum runs over n “adjacent” kernel maps at the same spatial position, and N is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants k, n, α, and β are hyper-parameters whose values are determined using a validation set; we used k = 2, n = 5, α = 10−4, and β = 0.75. We applied this normalization after applying the ReLU nonlinearity in certain layers (see Section 3.5).

This scheme bears some resemblance to the local contrast normalization scheme of Jarrett et al. [11], but ours would be more correctly termed “brightness normalization”, since we do not subtract the mean activity. Response normalization reduces our top-1 and top-5 error rates by 1.4% and 1.2%, respectively. We also verified the effectiveness of this scheme on the CIFAR-10 dataset: a four-layer CNN achieved a 13% test error rate without normalization and 11% with normalization3.

**3.4 Overlapping Pooling**

Pooling layers in CNNs summarize the outputs of neighboring groups of neurons in the same kernel map. Traditionally, the neighborhoods summarized by adjacent pooling units do not overlap (e.g., [17, 11, 4]). To be more precise, a pooling layer can be thought of as consisting of a grid of pooling units spaced s pixels apart, each summarizing a neighborhood of size z × z centered at the location of the pooling unit. If we set s = z, we obtain traditional local pooling as commonly employed in CNNs. If we set s < z, we obtain overlapping pooling. This is what we use throughout our network, with s = 2 and z = 3. This scheme reduces the top-1 and top-5 error rates by 0.4% and 0.3%, respectively, as compared with the non-overlapping scheme s = 2, z = 2, which produces output of equivalent dimensions. We generally observe during training that models with overlapping pooling find it slightly more difficult to overfit.

**3.5 Overall Architecture**

Now we are ready to describe the overall architecture of our CNN. As depicted in Figure 2, the net contains eight layers with weights; the first five are convolutional and the remaining three are fully connected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels. Our network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction distribution.

The kernels of the second, fourth, and fifth convolutional layers are connected only to those kernel maps in the previous layer which reside on the same GPU (see Figure 2). The kernels of the third convolutional layer are connected to all kernel maps in the second layer. The neurons in the fully connected layers are connected to all neurons in the previous layer. Response-normalization layers follow the first and second convolutional layers. Max-pooling layers, of the kind described in Section 3.4, follow both response-normalization layers as well as the fifth convolutional layer. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

The first convolutional layer filters the 224×224×3 input image with 96 kernels of size 11×11×3 with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size 5 × 5 × 48. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size 3 × 3 × 256 connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size 3 × 3 × 192, and the fifth convolutional layer has 256 kernels of size 3 × 3 × 192. The fully-connected layers have 4096 neurons each.

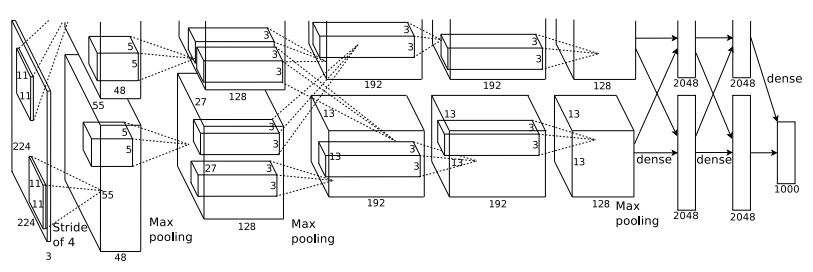


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264– 4096–4096–1000.

**4 Reducing Overfitting**

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

**4.1 Data Augmentation**

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random 224 × 224 patches (and their horizontal reflections) from the 256×256 images and training our network on these extracted patches4. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five 224 × 224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1. Therefore to each RGB image pixel  we add the following quantity:



where pi and λi are ith eigenvector and eigenvalue of the 3 × 3 covariance matrix of RGB pixel values, respectively, and αi is the aforementioned random variable. Each αi is drawn only once for all the pixels of a particular training image until that image is used for training again, at which point it is re-drawn. This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.

**4.2 Dropout**

Combining the predictions of many different models is a very successful way to reduce test errors [1, 3], but it appears to be too expensive for big neural networks that already take several days to train. There is, however, a very efficient version of model combination that only costs about a factor of two during training. The recently-introduced technique, called “dropout” [10], consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in backpropagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. At test time, we use all the neurons but multiply their outputs by 0.5, which is a reasonable approximation to taking the geometric mean of the predictive distributions produced by the exponentially-many dropout networks.

We use dropout in the first two fully-connected layers of Figure 2. Without dropout, our network exhibits substantial overfitting. Dropout roughly doubles the number of iterations required to converge.