PaddlePaddle与PyTorch的转换

1 常用的包

torch.nn paddle.nn 包括了神经网络相关的大部分函数

nn.Module nn.Layer 搭建网络时集成的父类，包含了初始化等基本功能

torch.optim paddle.optimizer 训练优化器

torchvision.transforms paddle.vision.transforms 数据预处理、图片处理

torchvision.datasets paddle.vision.datasets 数据集的加载与处理

2 网络结构

nn.Conv2d nn.Conv2D 2维卷积层

nn.BatchNorm2d nn.BatchNorm2D Batch Normalization 归一化

nn.ReLU nn.ReLU ReLU激活函数

nn.MaxPool2d nn.MaxPool2D 二维最大池化层

nn.AdaptiveAvgPool2d nn.AdaptiveAvgPool2D 自适应二维平均池化（只用给定输出形状即可）

nn.Linear nn.Linear 全连接层

nn.Sequential nn.Sequential 顺序容器，用来添加layers

torch.flatten paddle.flatten 展平处理

torch.softmax paddle.softmax softmax层

3 数据加载与处理

transforms.Compose transforms.Compose 图片处理打包

transforms.RandomResizedCrop transforms.RandomResizedCrop 随机裁剪

transforms.RandomHorizontalFlip transforms.RandomHorizontalFlip 随机水平翻转

transforms.ToTensor transforms.ToTensor 转化为tensor格式

transforms.Normalize transforms.Normalize 数据标准化

datasets.ImageFolder datasets.DatasetFolder 指定数据集文件夹

torch.utils.data.DataLoader paddle.io.DataLoader 加载数据集

4 模型训练

(net).train (net).train 训练模式

(loss).backward (loss).backward 反向传递误差

optim.Adam optim.Adam Adam优化器，注意paddlepaddle中的参数分别为parameters和learning \_rate，与PyTorch中是不同的

(optimizer).no\_grad (optimizer).zero\_grad 梯度清零

torch.save paddle.jit.save 说实话，这两个还是有点区别的，使用请看官方文档

(net).eval (net).eval 预测模式

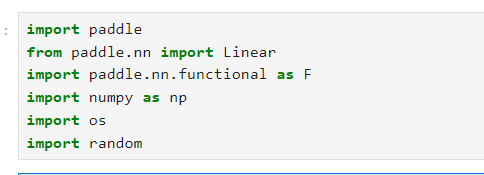
5 模型预测

torch.unsqueeze paddle.unsqueeze 增加数据维度

torch.no\_grad paddle.no\_grad 不计算梯度

6 其它

torch.device paddle.set\_device 指定设备



paddle：飞桨的主库，paddle 根目录下保留了常用API的别名，当前包括：paddle.tensor、paddle.framework、paddle.device目录下的所有API；

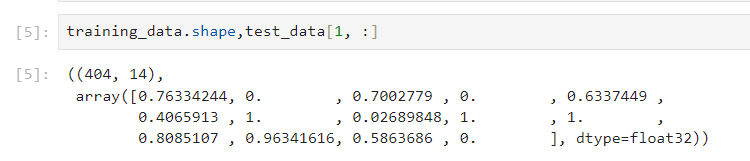
Linear：神经网络的全连接层函数，包含所有输入权重相加的基本神经元结构。在房价预测任务中，使用只有一层的神经网络（全连接层）实现线性回归模型。

paddle.nn：组网相关的API，包括 Linear、卷积 Conv2D、循环神经网络LSTM、损失函数CrossEntropyLoss、激活函数ReLU等；

paddle.nn.functional：与paddle.nn一样，包含组网相关的API，如：Linear、激活函数ReLU等，二者包含的同名模块功能相同，运行性能也基本一致。 差别在于paddle.nn目录下的模块均是类，每个类自带模块参数；paddle.nn.functional目录下的模块均是函数，需要手动传入函数计算所需要的参数。在实际使用时，卷积、全连接层等本身具有可学习的参数，建议使用paddle.nn；而激活函数、池化等操作没有可学习参数，可以考虑使用paddle.nn.functional。

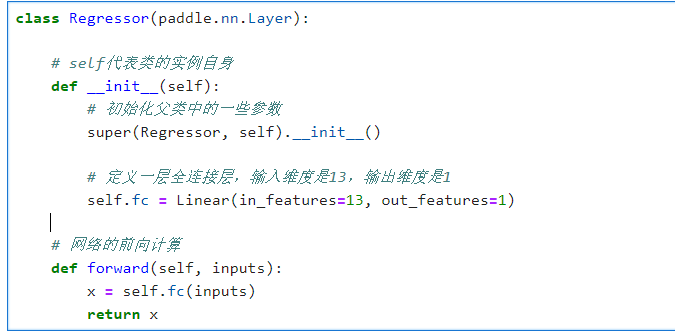
2.1 数据处理

数据处理的代码不依赖框架实现，与使用Python构建房价预测任务的代码相同



2.2 模型设计

模型定义的实质是定义线性回归的网络结构，飞桨建议通过创建Python类的方式完成模型网络的定义，该类需要继承paddle.nn.Layer父类，并且在类中定义init函数和forward函数。forward函数是框架指定实现前向计算逻辑的函数，程序在调用模型实例时会自动执行，forward函数中使用的网络层需要在init函数中声明。

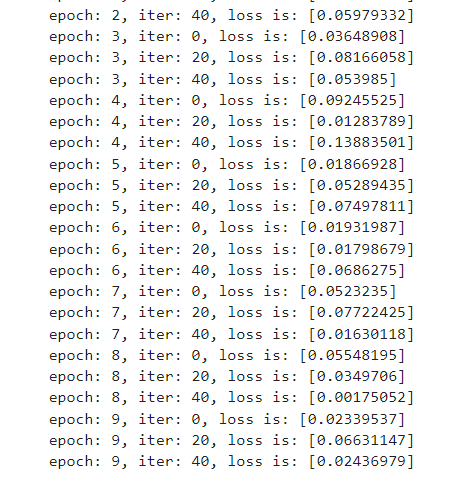


2.3 训练配置

声明定义好的回归模型实例为Regressor，并将模型的状态设置为train；

使用load\_data函数加载训练数据和测试数据；

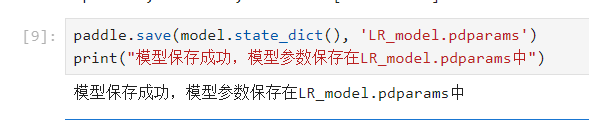
设置优化算法和学习率，优化算法采用随机梯度下降SGD，学习率设置为0.01。



2.5 保存并测试模型

2.5.1 保存模型

使用paddle.save API将模型当前的参数数据 model.state\_dict() 保存到文件中，用于模型预测或校验的程序调用。



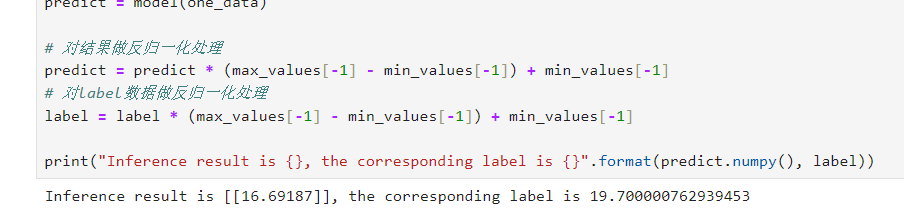
2.5.2 测试模型

下面选择一条数据样本，测试下模型的预测效果。测试过程和在应用场景中使用模型的过程一致，主要可分成如下三个步骤：

配置模型预测的机器资源。本案例默认使用本机，因此无需写代码指定。

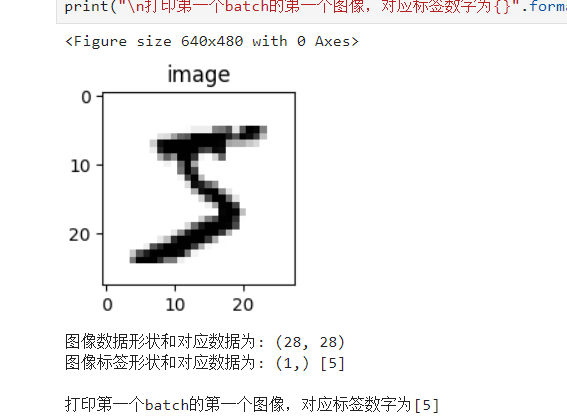
将训练好的模型参数加载到模型实例中。由两个语句完成，第一句是从文件中读取模型参数；第二句是将参数内容加载到模型。加载完毕后，需要将模型的状态调整为eval()（校验）。上文中提到，训练状态的模型需要同时支持前向计算和反向传导梯度，模型的实现较为臃肿，而校验和预测状态的模型只需要支持前向计算，模型的实现更加简单，性能更好。

将待预测的样本特征输入到模型中，打印输出的预测结果。



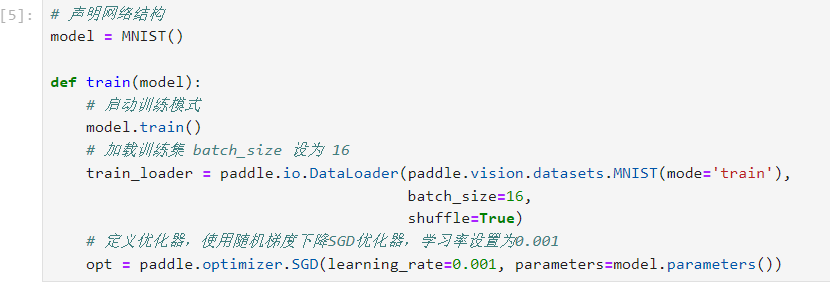
通过极简方案构建手写数字识别模型

通过代码读取任意一个数据内容，观察打印结果。



下面以类的方式组建手写数字识别的网络

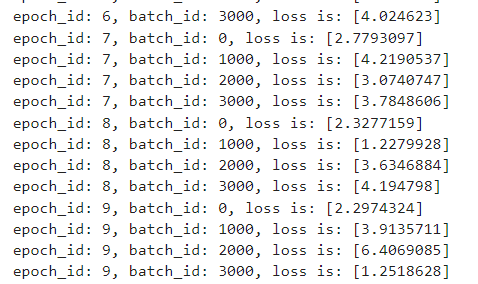
训练配置需要先生成模型实例（设为“训练”状态），再设置优化算法和学习率（使用随机梯度下降SGD，学习率设置为0.001）



训练过程采用二层循环嵌套方式，训练完成后需要保存模型参数，以便后续使用。

内层循环：负责整个数据集的一次遍历，遍历数据集采用分批次（batch）方式。

外层循环：定义遍历数据集的次数，本次训练中外层循环10次，通过参数EPOCH\_NUM设置



模型测试的主要目的是验证训练好的模型是否能正确识别出数字，包括如下四步：

声明实例

加载模型：加载训练过程中保存的模型参数，

灌入数据：将测试样本传入模型，模型的状态设置为校验状态（eval），显式告诉框架我们接下来只会使用前向计算的流程，不会计算梯度和梯度反向传播。

获取预测结果，取整后作为预测标签输出。

在工业实践中，我们面临的任务和数据环境千差万别，通常需要自己编写适合当前任务的数据处理程序，一般涉及如下五个环节：

读入数据

划分数据集

生成批次数据

训练样本集乱序

校验数据有效性

data包含三个元素的列表：train\_set、val\_set、 test\_set，包括50 000条训练样本、10 000条验证样本、10 000条测试样本。每个样本包含手写数字图片和对应的标签。

train\_set（训练集）：用于确定模型参数。

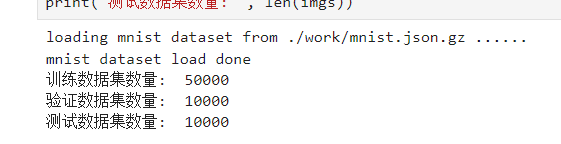
val\_set（验证集）：用于调节模型超参数（如多个网络结构、正则化权重的最优选择）。

test\_set（测试集）：用于估计应用效果（没有在模型中应用过的数据，更贴近模型在真实场景应用的效果）。

train\_set包含两个元素的列表：train\_images、train\_labels。

train\_images：[50 000, 784]的二维列表，包含50 000张图片。每张图片用一个长度为784的向量表示，内容是28\*28尺寸的像素灰度值（黑白图片）。

train\_labels：[50 000, ]的列表，表示这些图片对应的分类标签，即0~9之间的一个数字。



训练样本乱序、生成批次数据

训练样本乱序： 先将样本按顺序进行编号，建立ID集合index\_list。然后将index\_list乱序，最后按乱序后的顺序读取数据。

说明：

通过大量实验发现，模型对最后出现的数据印象更加深刻。训练数据导入后，越接近模型训练结束，最后几个批次数据对模型参数的影响越大。为了避免模型记忆影响训练效果，需要进行样本乱序操作。

生成批次数据： 先设置合理的batch\_size，再将数据转变成符合模型输入要求的np.array格式返回。同时，在返回数据时将Python生成器设置为yield模式，以减少内存占用。

在执行如上两个操作之前，需要先将数据处理代码封装成load\_data函数，方便后续调用。load\_data有三种模型：train、valid、eval，分为对应返回的数据是训练集、验证集、测试集。

人工校验

人工校验是指打印数据输出结果，观察是否是预期的格式。实现数据处理和加载函数后，我们可以调用它读取一次数据，观察数据的shape和类型是否与函数中设置的一致。

封装数据读取与处理函数

上文，我们从读取数据、划分数据集、到打乱训练数据、构建数据读取器以及数据数据校验，完成了一整套一般性的数据处理流程，下面将这些步骤放在一个函数中实现，方便在神经网络训练时直接调用。

下面定义一层神经网络，利用定义好的数据处理函数，完成神经网络的训练。

数据增强/增广

数据增强是一种挖掘数据集潜力的方法，可以让数据集蕴含更多让模型有效学习的信息。这些方法是领域和任务特定的，通过分析任务场景的复杂性和当前数据集的短板，对现有数据做有针对性的修改，以提供更加多样性的、匹配任务场景复杂性的新数据。下面以计算机视觉相关问题的数据增强为例，给读者一些具体增强方法和背后的思考。

数据处理

在介绍网络结构前，需要先进行数据处理针对手写数字识别的任务，网络层的设计如下：

输入层的尺度为28×28，但批次计算的时候会统一加1个维度（大小为batch size）。

中间的两个隐含层为10×10的结构，激活函数使用常见的Sigmoid函数。

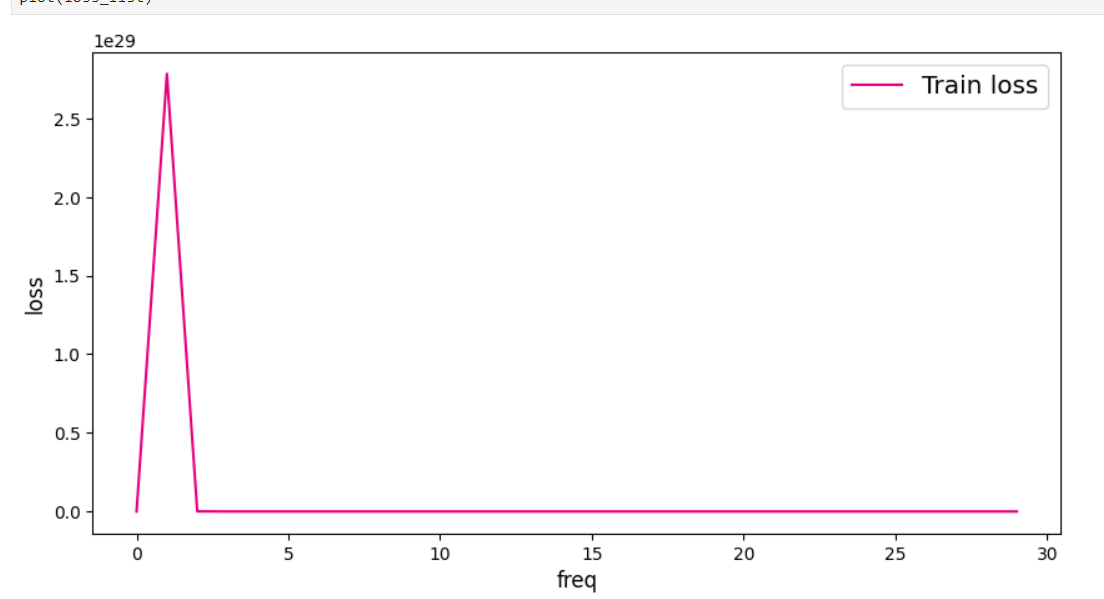
与房价预测模型一样，模型的输出是回归一个数字，输出层的尺寸设置成1。

卷积神经网络

虽然使用经典的全连接神经网络可以提升一定的准确率，但其输入数据的形式导致丢失了图像像素间的空间信息，这影响了网络对图像内容的理解。对于计算机视觉问题，效果最好的模型仍然是卷积神经网络。卷积神经网络针对视觉问题的特点进行了网络结构优化，可以直接处理原始形式的图像数据，保留像素间的空间信息，因此更适合处理视觉问题。

卷积神经网络由多个卷积层和池化层组成，如 图4 所示。卷积层负责对输入进行扫描以生成更抽象的特征表示，池化层对这些特征表示进行过滤，保留最关键的特征信息。

可视化损失变化：



损失函数是模型优化的目标，用于在众多的参数取值中，识别最理想的取值。损失函数的计算在训练过程的代码中，每一轮模型训练的过程都相同，分如下三步：

先根据输入数据正向计算预测输出。

再根据预测值和真实值计算损失。

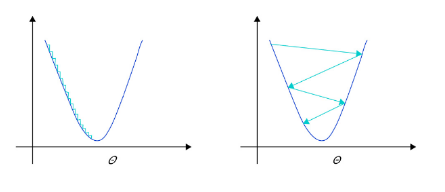
最后根据损失反向传播梯度并更新参数。

设置学习率

在深度学习神经网络模型中，通常使用标准的随机梯度下降算法更新参数，学习率代表参数更新幅度的大小，即步长。当学习率最优时，模型的有效容量最大，最终能达到的效果最好。学习率和深度学习任务类型有关，合适的学习率往往需要大量的实验和调参经验。探索学习率最优值时需要注意如下两点：

学习率不是越小越好。学习率越小，损失函数的变化速度越慢，意味着我们需要花费更长的时间进行收敛，如左图所示。

学习率不是越大越好。只根据总样本集中的一个批次计算梯度，抽样误差会导致计算出的梯度不是全局最优的方向，且存在波动。在接近最优解时，过大的学习率会导致参数在最优解附近震荡，损失难以收敛，如右图所示。



单GPU训练

通过paddle.device.set\_device API，设置在GPU上训练还是CPU上训练。

paddle.device.set\_device (device)

参数 device (str)：此参数确定特定的运行设备，可以是cpu、 gpu:x或者是xpu:x。其中，x是GPU或XPU的编号。当device是cpu时， 程序在CPU上运行；当device是gpu:x时，程序在GPU上运行。

模型并行

模型并行是将一个网络模型拆分为多份，拆分后的模型分到多个设备上（GPU）训练，每个设备的训练数据是相同的。模型并行的实现模式可以节省内存，但是应用较为受限。

模型并行的方式一般适用于如下两个场景：

模型架构过大： 完整的模型无法放入单个GPU。如2012年ImageNet大赛的冠军模型AlexNet是模型并行的典型案例，由于当时GPU内存较小，单个GPU不足以承担AlexNet，因此研究者将AlexNet拆分为两部分放到两个GPU上并行训练。

网络模型的结构设计相对独立： 当网络模型的设计结构可以并行化时，采用模型并行的方式。如在计算机视觉目标检测任务中，一些模型（如YOLO9000）的边界框回归和类别预测是独立的，可以将独立的部分放到不同的设备节点上完成分布式训练。

数据并行

数据并行与模型并行不同，数据并行每次读取多份数据，读取到的数据输入给多个设备（GPU）上的模型，每个设备上的模型是完全相同的，飞桨采用的就是这种方式。

训练过程优化思路主要有如下五个关键环节：

1. 计算分类准确率，观测模型训练效果。

交叉熵损失函数只能作为优化目标，无法直接准确衡量模型的训练效果。准确率可以直接衡量训练效果，但由于其离散性质，不适合做为损失函数优化神经网络。

2. 检查模型训练过程，识别潜在问题。

如果模型的损失或者评估指标表现异常，通常需要打印模型每一层的输入和输出来定位问题，分析每一层的内容来获取错误的原因。

3. 加入校验或测试，更好评价模型效果。

理想的模型训练结果是在训练集和验证集上均有较高的准确率，如果训练集的准确率低于验证集，说明网络训练程度不够；如果训练集的准确率高于验证集，可能是发生了过拟合现象。通过在优化目标中加入正则化项的办法，解决过拟合的问题。

4. 加入正则化项，避免模型过拟合。

飞桨框架支持为整体参数加入正则化项，这是通常的做法。此外，飞桨框架也支持为某一层或某一部分的网络单独加入正则化项，以达到精细调整参数训练的效果。

5. 可视化分析。

用户不仅可以通过打印或使用matplotlib库作图，飞桨还提供了更专业的可视化分析工具VisualDL，提供便捷的可视化分析方法。

在训练过程中，我们会发现模型在训练样本集上的损失在不断减小。但这是否代表模型在未来的应用场景上依然有效？为了验证模型的有效性，通常将样本集合分成三份，训练集、校验集和测试集。

训练集 ：用于训练模型的参数，即训练过程中主要完成的工作。

校验集 ：用于对模型超参数的选择，比如网络结构的调整、正则化项权重的选择等。

测试集 ：用于模拟模型在应用后的真实效果。因为测试集没有参与任何模型优化或参数训练的工作，所以它对模型来说是完全未知的样本。在不以校验数据优化网络结构或模型超参数时，校验数据和测试数据的效果是类似的，均更真实的反映模型效果。

训练模型时，经常需要观察模型的评价指标，分析模型的优化过程，以确保训练是有效的。可选用这两种工具：Matplotlib库和VisualDL。

Matplotlib库：Matplotlib库是Python中使用的最多的2D图形绘图库，它有一套完全仿照MATLAB的函数形式的绘图接口，使用轻量级的PLT库（Matplotlib）作图是非常简单的。

VisualDL：如果期望使用更加专业的作图工具，可以尝试VisualDL，飞桨可视化分析工具。VisualDL能够有效地展示飞桨在运行过程中的计算图、各种指标变化趋势和数据信息。

卷积（Convolution）

这一小节将为读者介绍卷积算法的原理和实现方案，并通过具体的案例展示如何使用卷积对图片进行操作，主要涵盖如下内容：

卷积计算

填充（padding）

步幅（stride）

感受野（Receptive Field）

多输入通道、多输出通道和批量操作

飞桨卷积API介绍

卷积算子应用举例

卷积计算

卷积是数学分析中的一种积分变换的方法，在图像处理中采用的是卷积的离散形式。这里需要说明的是，在卷积神经网络中，卷积层的实现方式实际上是数学中定义的互相关 （cross-correlation）运算，与数学分析中的卷积定义有所不同，这里跟其他框架和卷积神经网络的教程保持一致，都使用互相关运算作为卷积的定义

填充（padding）

在上面的例子中，输入图片尺寸为3×3，输出图片尺寸为2×2，经过一次卷积之后，图片尺寸变小

池化（Pooling）

池化是使用某一位置的相邻输出的总体统计特征代替网络在该位置的输出，其好处是当输入数据做出少量平移时，经过池化函数后的大多数输出还能保持不变。比如：当识别一张图像是否是人脸时，我们需要知道人脸左边有一只眼睛，右边也有一只眼睛，而不需要知道眼睛的精确位置，这时候通过池化某一片区域的像素点来得到总体统计特征会显得很有用。由于池化之后特征图会变得更小，如果后面连接的是全连接层，能有效的减小神经元的个数，节省存储空间并提高计算效率。 如 图15 所示，将一个2×22\times 22×2的区域池化成一个像素点。通常有两种方法，平均池化和最大池化。

ReLU激活函数

前面介绍的网络结构中，普遍使用Sigmoid函数做激活函数。在神经网络发展的早期，Sigmoid函数用的比较多，而目前用的较多的激活函数是ReLU。这是因为Sigmoid函数在反向传播过程中，容易造成梯度的衰减。让我们仔细观察Sigmoid函数的形式，就能发现这一问题。

图像分类是根据图像的语义信息对不同类别图像进行区分，是计算机视觉的核心，是物体检测、图像分割、物体跟踪、行为分析、人脸识别等其他高层次视觉任务的基础。图像分类在许多领域都有着广泛的应用，如：安防领域的人脸识别和智能视频分析等，交通领域的交通场景识别，互联网领域基于内容的图像检索和相册自动归类，医学领域的图像识别等。

LeNet：Yan LeCun等人于1998年第一次将卷积神经网络应用到图像分类任务上[1]，在手写数字识别任务上取得了巨大成功。

AlexNet：Alex Krizhevsky等人在2012年提出了AlexNet[2], 并应用在大尺寸图片数据ImageNet上，获得了2012年ImageNet比赛冠军(ImageNet Large Scale Visual Recognition Challenge，ILSVRC）。

VGG：Simonyan和Zisserman于2014年提出了VGG网络结构[3]，是当前最流行的卷积神经网络之一，由于其结构简单、应用性极强而深受广大研究者欢迎。

GoogLeNet：Christian Szegedy等人在2014提出了GoogLeNet[4]，并取得了2014年ImageNet比赛冠军。

ResNet：Kaiming He等人在2015年提出了ResNet[5]，通过引入残差模块加深网络层数，在ImagNet数据集上的错误率降低到3.6%，超越了人眼识别水平。ResNet的设计思想深刻地影响了后来的深度神经网络的设计。

AlexNet

通过上面的实际训练可以看到，虽然LeNet在手写数字识别数据集上取得了很好的结果，但在更大的数据集上表现却并不好。自从1998年LeNet问世以来，接下来十几年的时间里，神经网络并没有在计算机视觉领域取得很好的结果，反而一度被其它算法所超越。原因主要有两方面，一是神经网络的计算比较复杂，对当时计算机的算力来说，训练神经网络是件非常耗时的事情；另一方面，当时还没有专门针对神经网络做算法和训练技巧的优化，神经网络的收敛是件非常困难的事情。

随着技术的进步和发展，计算机的算力越来越强大，尤其是在GPU并行计算能力的推动下，复杂神经网络的计算也变得更加容易实施。另一方面，互联网上涌现出越来越多的数据，极大的丰富了数据库。同时也有越来越多的研究人员开始专门针对神经网络做算法和模型的优化，Alex Krizhevsky等人提出的AlexNet以很大优势获得了2012年ImageNet比赛的冠军。这一成果极大的激发了产业界对神经网络的兴趣，开创了使用深度神经网络解决图像问题的途径，随后也在这一领域涌现出越来越多的优秀成果。

AlexNet与LeNet相比，具有更深的网络结构，包含5层卷积和3层全连接，同时使用了如下三种方法改进模型的训练过程：

数据增广：深度学习中常用的一种处理方式，通过对训练随机加一些变化，比如平移、缩放、裁剪、旋转、翻转或者增减亮度等，产生一系列跟原始图片相似但又不完全相同的样本，从而扩大训练数据集。通过这种方式，可以随机改变训练样本，避免模型过度依赖于某些属性，能从一定程度上抑制过拟合。

使用Dropout抑制过拟合。

使用ReLU激活函数减少梯度消失现象。

VGG是当前最流行的CNN模型之一，2014年由Simonyan和Zisserman提出，其命名来源于论文作者所在的实验室Visual Geometry Group。AlexNet模型通过构造多层网络，取得了较好的效果，但是并没有给出深度神经网络设计的方向。VGG通过使用一系列大小为3x3的小尺寸卷积核和池化层构造深度卷积神经网络，并取得了较好的效果。VGG模型因为结构简单、应用性极强而广受研究者欢迎，尤其是它的网络结构设计方法，为构建深度神经网络提供了方向。