MATLAB深度学习实用示例

MathWorks®

简介

本电子书是在 *MATLAB 深度学习简介*电子书的基础上撰写，前者解答了“什么是深度学习?”这一问题，本书将向您展示其实现方式。我们将展示使用三种方式用于训练深度学习网络：

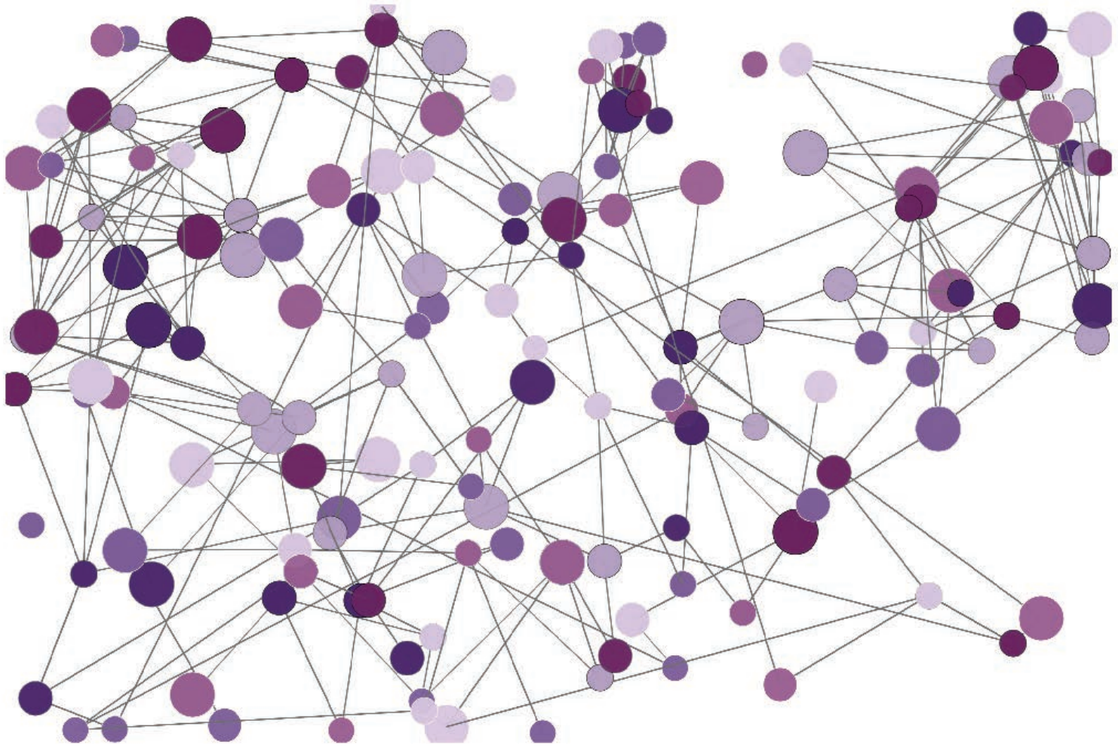
- 从零开始训练神经网络
- 使用迁移学习训练现有网络
- 训练现有网络以执行语义分割

这些示例大多是关于图像分类的应用。但是，深度学习也日益受到其他应用领域的青睐。在电子书的第二部分，我们将通过另外两个示例来展示用于图像处理的深度学习技巧也可以同时应用于信号数据。

所有示例和代码均可 *下载*。

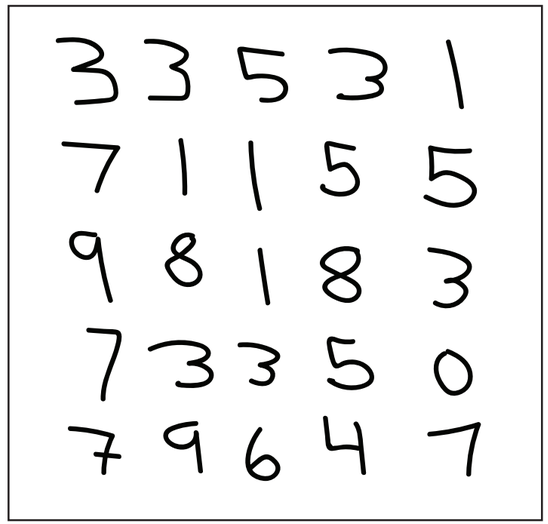
复习基础知识

- *什么是深度学习?* 3:33
- *深度学习 VS 机器学习* 3:48



实用示例 1：从头开始训练模型

在本例中，我们需要一个训练*卷积神经网络* (CNN) 来识别手写数字。我们将使用 *MNIST 数据集*中的数据。该数据集包含 60,000 个 0–9 的手写数字图像。下面是 MNIST 数据集中的 25 个手写数字随机样本：



使用简单的数据集，即可涵盖深度学习工作流程的所有关键步骤，无需再人工介入处理各种难题，例如处理能力或因太大而无法存入内存的数据集。此处介绍的工作流程可应用于更复杂的深度学习问题和更大的数据集。

如果您刚刚开始运用深度学习，那么使用此数据集的另一项优势便是不必购买昂贵的 GPU 即可训练该数据集。

尽管数据集很简单，但只要采用正确的深度学习模型和训练方案，便可以实现超过 99% 的准确率。那么，我们应如何创建模型，才能达到这样的准确率呢？

这将是一个迭代的过程，我们要利用以前的训练结果找出解决训练问题的方法。步骤如下所述：

1.访问数据

首先，将 *MNIST* 图像集下载到 MATLAB®。数据集以很多不同的文件类型来存储。此数据存储为二进制文件，方便 MATLAB 迅速使用和重构为图像。

以下代码行将读取二进制原文件并创建包含所有训练图像的阵列：

```
rawImgDataTrain = uint8 (fread(fid, numImg * numRows * numCols,...
    'uint8'));

% 将数据部分重构为 4D 阵列
rawImgDataTrain = reshape(rawImgDataTrain, [numRows, numCols,...
    numImgs]);
imgDataTrain(:,:,1,ii) = uint8(rawImgDataTrain(:,:,ii));
```

我们可以通过在命令窗口中输入 **whos**，检查数据的大小和类。

```
>> whos imgDataTrain

Name            Size                      Bytes      Class

imgDataTrain    28x28x1x60000             47040000    uint8
```

MNIST 图像很小，只有 28 x 28 像素，总共有 60,000 个训练图像。

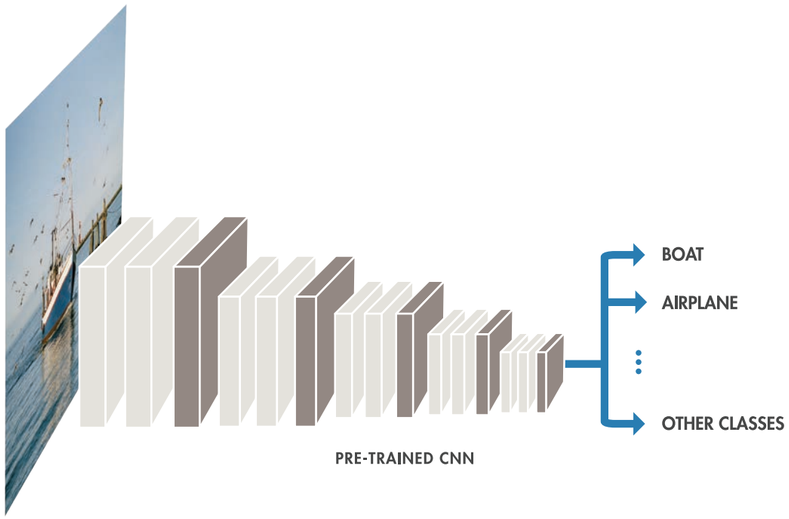
下一项任务是图像标注，但因为 MNIST 图像自带标签，所以我们可以跳过这一冗长的步骤，转而迅速构建我们的神经网络。

2.创建和配置网络层

我们将构建一个卷积神经网络，这是最常见的深度学习网络。

关于卷积神经网络（CNN）

CNN 将一张图前向传输通过网络，然后输出一个最终的分类结果。网络可以有几十层或几百层构成，每一层都学习检测不同的特征。滤波器会以不同分辨率应用到各个训练图像，且每个卷积图像的输出会用作下一层的输入。滤波器最初可以是非常简单的特征，例如亮度和边缘，然后增加复杂度，随着层数的增加，直至可以唯一地确定目标特征。



了解更多

*卷积神经网络是什么？*4:44

常用网络层

卷积层将输入图像放进一组卷积滤波器，每个滤波器激活图像中的某些特征。

ReLU 层通过将负值映射到零和保持正数值，实现更快、更高效的训练。

池化层通过执行非线性下采样，减少网络需要学习的参数个数，从而简化输出。

全连接层将网络 2D 空间特征“扁平化”为 1D 矢量，为分类目的而表示图像级特征。

Softmax 层为数据集中的每个类别提供概率。

从零开始构建网络时，宜采用常用层简单组合的形式，降低复杂度将使调试更方便，但我们可能需要添加一些层来达到所需要的准确率。

```
layers = [  imageInputLayer([28 28 1])
            convolution2dLayer(5,20)
            reluLayer
            maxPooling2dLayer(2, 'Stride', 2)
            fullyConnectedLayer(10)
            softmaxLayer
            classificationLayer()    ]
```

3.训练网络

训练前，我们需要选择训练方案。可以使用的方案有很多。

下表显示了最常用的方案。

训练方案	定义	提示
训练进度图	该图显示了每批 mini-batch 的损失和准确率函数。它包括可随时暂停网络训练的停止按钮。	('Plots','training-progress') 绘制网络训练的进度。
最大训练代数（Epoch）	一次迭代是指训练算法完全通过整个训练集。	('MaxEpoch',20) 指定的训练代数越多，网络训练持续时间越长，但准确率也会随着每次迭代而提高。
最小批大小 (Mini-batch size)	最小批是同时在 GPU 上处理的训练数据集的子集。	('MiniBatchSize',64) 最小批越大，训练速度越快，但最大大小将由 GPU 内存决定。如果在训练时出现内存错误，请减小最小批大小。
学习速率	这是控制训练速度的主要参数。	学习速率越低，训练结果准确率越高，但网络训练时间会更长。

我们指定两个方案：进度图和最小批大小。

```
miniBatchSize = 8192;
options = trainingOptions( 'sgdm',...
    'MiniBatchSize', miniBatchSize,...
    'Plots', 'training-progress');
net = trainNetwork(imgDataTrain, labelsTrain, layers, options);
```

然后，我们会运行网络并监控其进度。

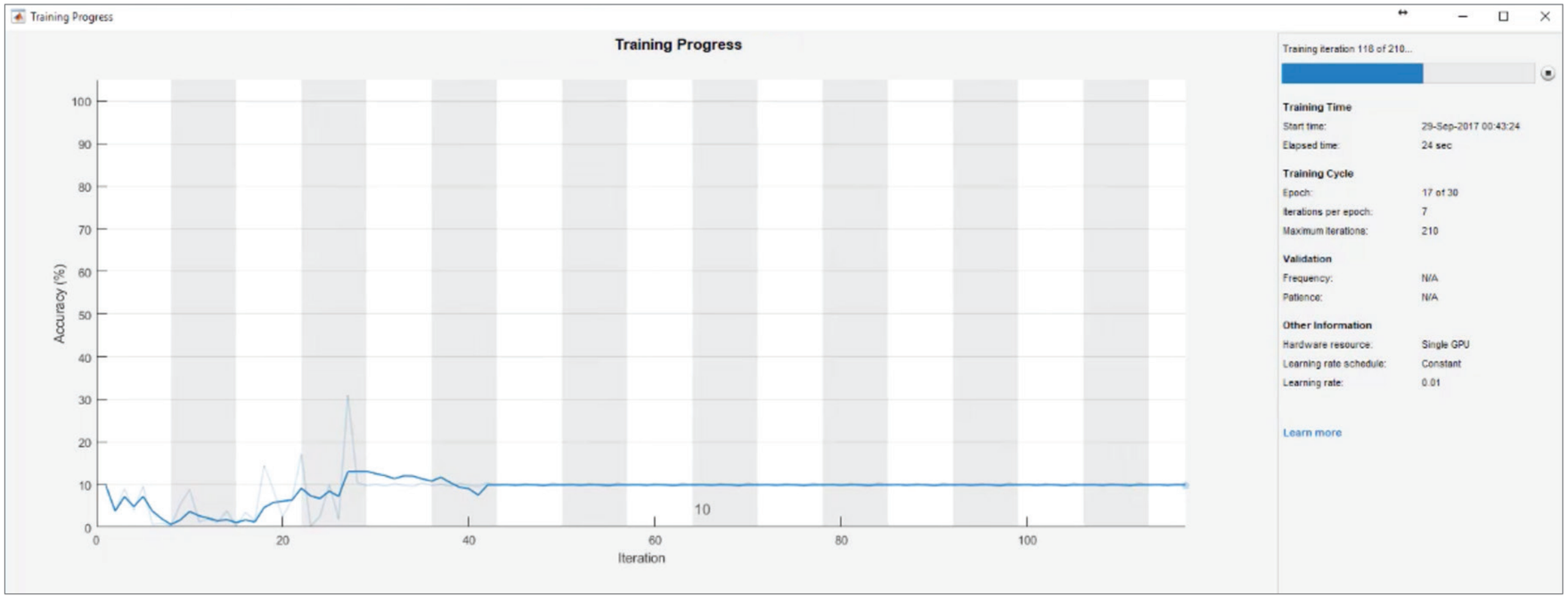
技巧

大数据集会降低处理速度。但是，深度学习网络可以利用 GPU 的大规模并行架构。确切的加速效果取决于硬件、数据集大小和网络配置等因素，但您会发现训练时间从几小时缩短到了几分钟。

在 MATLAB 的训练选项 (training option)中，您可以快速更改用于训练网络的硬件资源。如果未指定此 option，则训练将默认为单个 GPU（如果可用）。

4.检查网络准确率

我们的目标是随时间一步步提高模型的准确率。在网络训练过程中，会出现进度图。



我们的模型似乎在第 28 次迭代后停止改进，随后掉落至约 10% 的准确率。在从头开始训练网络时，这是常见现象。这意味着网络无法在一个解决方案上收敛。准确率已经达到平台期，再也不能改进。此时，已没有继续的必要，可停止训练，尝试其他方法。

通过单击屏幕右上角的停止按钮，可停止训练，返回网络的当前状态。一旦执行停止，便需要从头开始重新启动训练，而无法从训练停止点恢复训练。

调整网络准确率的方法很多。例如，我们可以：

- 增加训练图像的数量
- 提高训练图像的质量
- 改变训练选项 (training option)
- 改变网络配置（例如，通过添加、删除或重组层）

我们将尝试改变训练选项 (training option) 和网络配置。

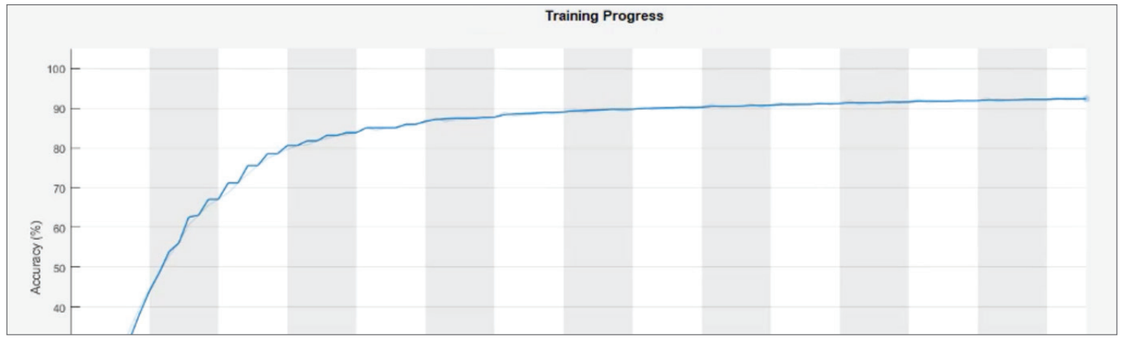
更改训练方案

首先，我们将调整学习速率。我们要设置的初始学习速率比 0.01 这个默认速率低得多。

```
'InitialLearnRate', 0.0001
```

尽管只更改了这一个参数，效果却立竿见影 — 几乎 90% 的准确率！

对于某些应用场合而言，这样的结果是令人满意的，但您可能会记得我们的目标原本是 99%。



进阶技巧

您可以使用贝叶斯优化来识别训练参数的最优值。贝叶斯优化将多次运行网络（并且可以并行处理此过程）。

更改网络配置

准确率从 90% 提高到 99% 需要加强网络深度和多轮试错。我们添加更多层，包括批归一化层，将帮助加快网络收敛的速度。

```
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

现在，网络“更深”了。此时，我们将更换网络，但训练选项 (training option) 保持不变。

完成网络训练后，我们要在 10,000 个图像上进行测试。

```
predLabelsTest = net.classify(imgDataTest);
accuracy = sum(predLabelsTest == labelsTest) / numel(labelsTest)
```

```
accuracy = 0.9880
```

此网络达到最高准确率 — 约 99%。现在，我们可以用它识别在线图像甚至实时视频流中的手写字母。

了解更多

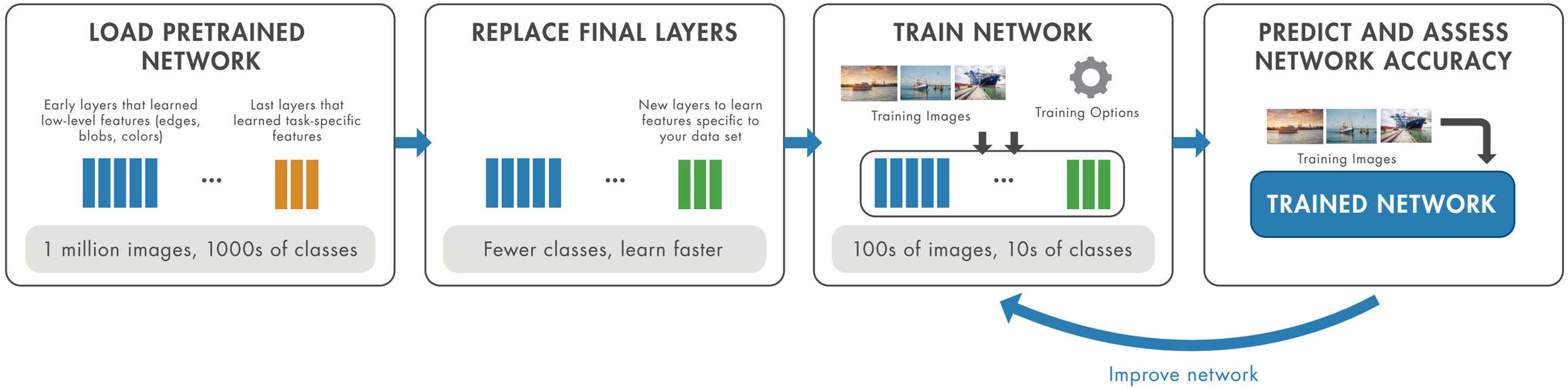
使用 MATLAB 从头开始训练神经网络 5:13
用 11 行 MATLAB 代码实现深度学习 2:38

实用示例 2:迁移学习

在本例中,我们将修改预先训练好的网络并使用迁移学习进行训练以执行新识别任务。微调预先训练好的网络比构造和训练新网络更快、更容易:您可以使用更少数量的训练图像迅速迁移学习到新任务。迁移学习的优势是预先训练好的网络因经过大量图像训练已经学习了一系列丰富的特征。

我们将使用 GoogLeNet。该网络完成了 1000 类对象(包括自行车、汽车和狗)的训练。我们需要重新训练此网络以识别 5 类食物。下面是训练步骤:

1. 导入预先训练好的网络。
2. 配置最后三层以执行新识别任务。
3. 利用新数据训练网络。
4. 检验结果。



1.导入预先训练好的网络

我们可以通过一行代码导入 GoogLeNet：

```
% 加载预先训练好的网络
net = googlenet;
```

预先训练好的网络已完成多数繁重的网络设置（选择和组织层结构）。这意味着我们无需执行任何重新配置操作即可利用网络最初受训练类别的图像检验网络：

```
%% 利用一个图像进行检验
img = imread('peppers.png');
imgLabel = net.classify(imresize(img, [224 224]));
```

技巧

使用以下这行代码查看 GoogLeNet 受训练所用的全部 1000 个类别：

```
class_names = net.Layers(end).ClassNames;
```

迁移学习技巧

- 使用准确率高的网络。如果网络的原始识别任务准确率只有 50%，在执行新识别任务时便不可能准确。
- 如果新识别类别与原始类别具有类似特征，模型准确率可能更高。例如，针对狗进行过训练的网络在学习其他动物时可能相对快一些。

2.配置网络以执行新任务

要训练 GoogLeNet 对新图像分类，我们只需重新配置网络的最后三层。这些层包含组合网络提取到类概率和标签的特征所需的信息。GoogLeNet 有 144 个层。此处显示的是网络的最后 5 个层。

```
>>net.Layers(end-4:end)
```

```
140    'pool5-7x7_s1'          平均池化
141    'pool5-drop_7x7_s1'     丢弃
142    'loss3-classifier'      全连接
143    'prob'                  Softmax
144    'output'                分类输出
```

我们将重置第 143 和 144 层，即 softmax 层和分类输出层。这些层负责将正确的类别分配给输入图像。我们需要这些层对应新类别，而不是原始网络学习过的类别。我们将最终的全连接层设置为与新数据集中的类数量相同的大小 — 本例为 5 个类。

技巧

为了使在新增层的学习速度比在原始层中学习的速度快，请提高全连接层的学习速率。

```
lgraph = removeLayers(lgraph, {'loss3-classifier', 'prob', ...
    'output'});
numClasses = numel(unique(categories(trainDS.Labels)));
newLayers = [
    fullyConnectedLayer(numClasses, 'Name','fc',...
    'WeightLearnRateFactor',20,'BiasLearnRateFactor',20)
    softmaxLayer('Name','softmax')
    classificationLayer('Name','classoutput')];
lgraph = addLayers(lgraph,newLayers);
```

3.利用新数据训练网络

从零开始训练网络时，为提高网络的准确率，我们要调整某些训练选项 (training option)（在本例中是批大小、学习速率和验证数据）。

```
opts = trainingOptions('sgdm','InitialLearnRate',0.001,...
    'ValidationData',valDS,...
    'Plots','training-progress',...
    'MiniBatchSize',64,...
    'ValidationPatience',3);

% 使用优化的超参数集进行训练
tic
disp('训练开始前，初始化可能需要一分钟时间')
net = trainNetwork(trainDS, layers_train, opts);
toc
```

此模型的训练时间可能会因所使用的硬件和其他因素而显著不同。一个 Tesla P100 GPU 训练此模型可能需要大约 20 分钟。

技巧

使用 `tic` 和 `toc` 可快速查看开展训练所需的时长。`tic` 可启动秒表计时器以测量性能。`toc` 可停止计时器并读取命令窗口中显示的用时。

技巧

如果收到 GPU 内存不足错误，请减小 `'MiniBatchSize'` 值。

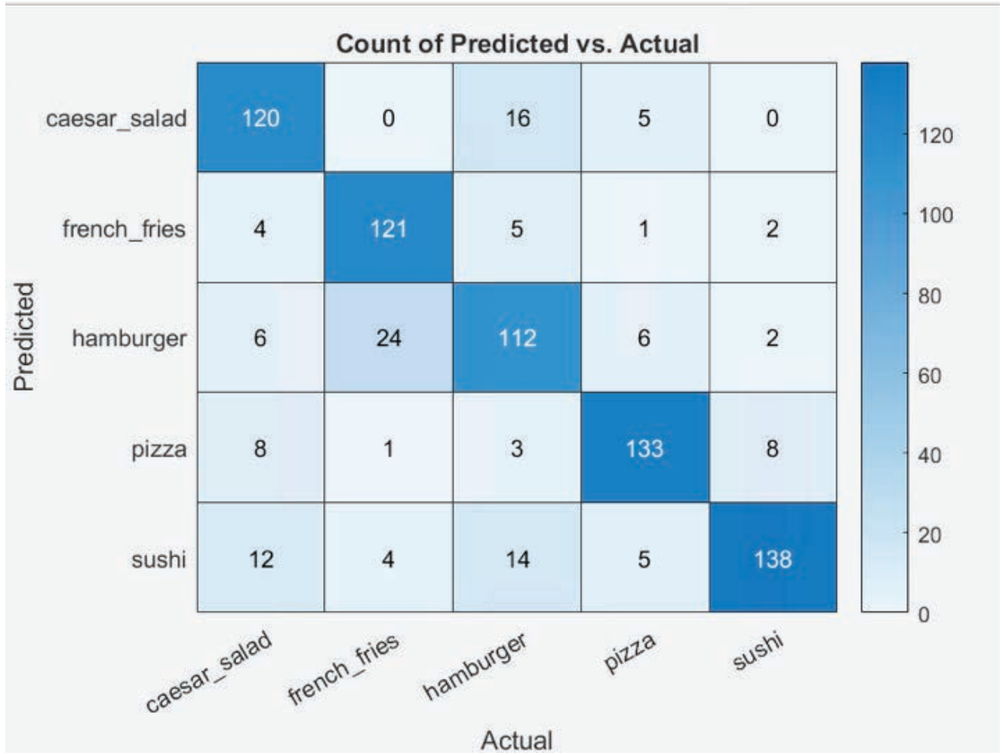
4.评估网络

网络经过训练后，即可查看其处理新数据的表现。

```
% 对测试数据集的所有图像分类
[labels,err_test] = classify(net, testDS);

accuracy_default = sum(labels == testDS.Labels)/numel(labels);
disp(['测试准确率为 ' num2str(accuracy_default)])
```

混淆矩阵显示了网络对于每一类别 150 个图像的预测。如果对角线上的所有值都是 150，则表示每个测试图像均已正确分类。显然，我们的网络不属于这种情况。通过对角线外的值可以判断哪一个类别分类不当。这可以帮助我们对相应数据展开调查。



模型训练后的最终准确率为 83%。尽管这个准确率对做示范来说够用了，但在现实应用中是不可接受的。为了提高现实应用的模型准确率，我们要继续迭代，重新研究训练方案、检查数据以及重新配置网络。

最后，我们要直观验证网络处理新图像的性能。

```
[label,conf] = classify(net,im);
% 对随机图像分类
imshow(im_display);
title(sprintf('%s %.2f, actual %s', ...
     char(label),max(conf),char(actualLabel))
```

即使最终选择从零开始创建自己的网络，迁移学习仍不失为了解深度学习的绝佳起点：您可以利用领域专家们开发的网络，更换几个层，然后开始训练。既然模型已经从原始训练数据集学习了许多特征，训练用时和训练图像必然比从零开始开发模型的需求要低。

了解更多

预先训练好的卷积神经网络
使用 GoogLeNet 的迁移学习
用 10 行 MATLAB 代码完成迁移学习 4:00
在 MATLAB 中利用神经网络完成迁移学习 4:06

实用示例 3：语义分割

语义分割是深度学习方面较新的进步之一，对图像的特征提供了精细的、像素级理解。传统的 CNN 会将图像的特征分类，而语义分割则将每个像素与特定类别（例如花、道路、人或汽车）关联。结果类似于下面这种情况：

注意，利用语义分割，可以很好地定义形态不规则的对象，例如道路。

语义分割可以作为对象检测的一种有用替代方法，因为它允许感兴趣对象跨越图像中的多个区域。这种技术可以清楚地检测到形态不规则的对象，相比之下，对象检测要求对象必须位于有边界的方框内。

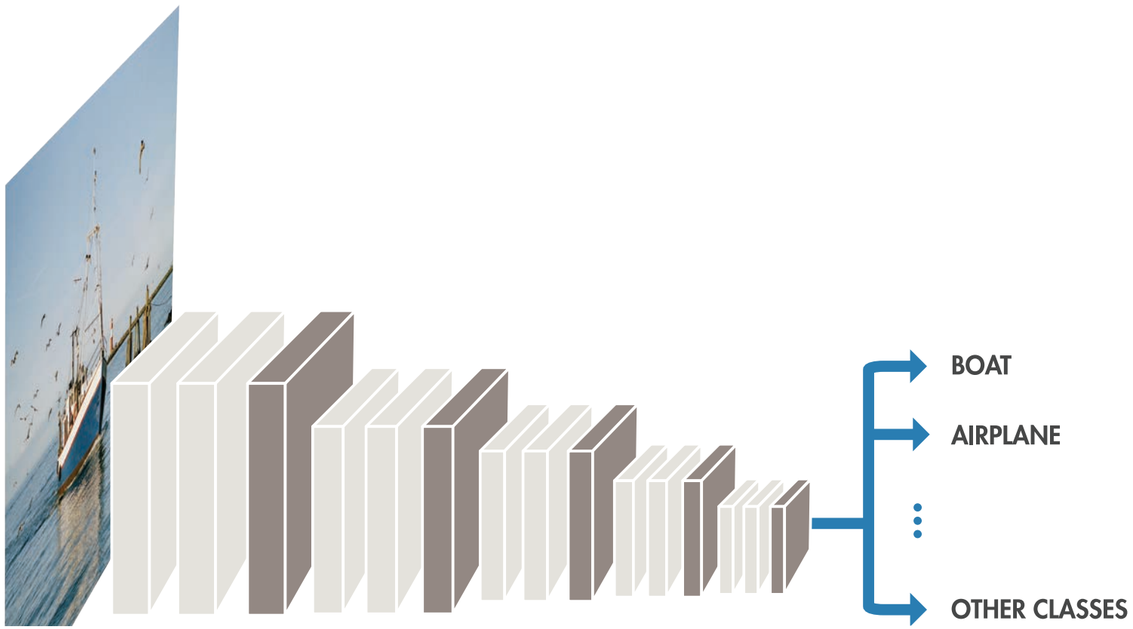
常用语义分割应用

- 自动驾驶：通过区分道路与障碍物，比如行人、人行道、路灯和其他汽车，让汽车识别可行驶的路径
- 工业检测：用于检测材料中的缺陷，如晶圆检验
- 卫星影像：用于识别高山、河流、沙漠和其他地形
- 医学成像：用于分析和检测细胞中的癌变

在我们开始分析示例之前，先快速看一下语义分割网络的架构。

语义分割网络架构

正如我们在示例 1 和 2 中所见，传统的 CNN 采集一个图像，通过层层网络前向传输，然后输出最终分类。



语义分割网络在这个过程中构建一个向上采样网络，该网络的架构与逆向 CNN 相似。

此新层系列对预先训练好的网络结果进行*向上采样*，然后回馈到图像。结果图像的每一个像素均分配一个分类标签。

1.导入预先训练好的网络

在本例中，我们需要构建一个能够用于检测清晰道路空间、行车道和人行道的自动驾驶系统。

步骤如下所述：

1. 导入预先训练好的网络。
2. 载入数据集。
3. 设置网络。
4. 训练网络。
5. 评估网络的准确率。

我们可以从头开始训练网络，但在本例中，我们要使用一个预先训练好的网络。正如我们在此前的示例中所见，我们可以利用最初受训练类别的图像来检验预先训练好的网络，而不必重新配置。

我们预先训练好的网络是 VGG-16。VGG-16 用于 *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC)，通过了超过 100 万个图像的训练，可以将图像分为 1000 个对象类别。

导入 VGG-16 只需一行 MATLAB 代码：

```
% 下载和安装用于 VGG-16 网络支持包的 Neural Network Toolbox 模型。
vgg16;
```

2.载入数据集

我们使用 *CamVid 数据集*。该数据集包括了行驶过程中采集的街景图像，为 32 个语义类（包括汽车、行人和道路）提供像素级标签。

数据集中的每个图像都有一副彩图，图像中的每个像素都有一个标签图像。

将大量图像放入内存很麻烦。数据存储是导入、访问和管理大数据文件的便捷方式。任何数据存储（图像、像素甚至电子表格）均可作为存储库使用，只要所有存储的数据具有相同的结构和格式。在我们的语义分割示例中，我们创建了两个数据存储对象：

- **`ImageDatastore`**，用于管理图像文件，其中每个图像都可以放入内存中，但整个集合则不可以。
- **`pixelLabelDatastore`**，用于放入包含像素标签的图像目录。

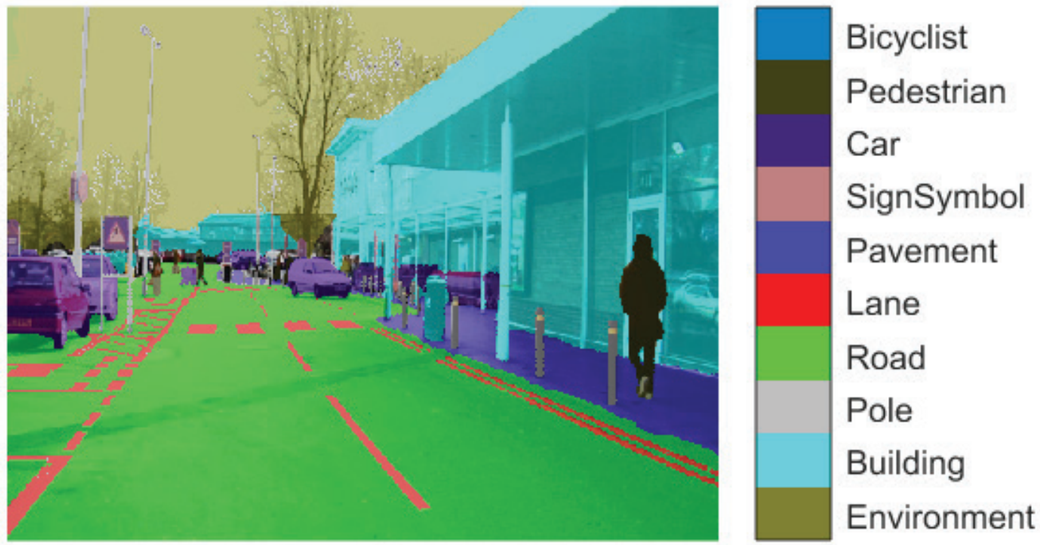
技巧

如果无法找到与要识别的类别对应的、预先标注的数据集，则需要自行创建一个数据集。使用 Image Labeler App，可轻松完成这一耗时的任务：您只需选择一组像素，该 App 便会自动使用一种颜色和对应的类别予以标注。

将图像数据和像素标签数据导入 MATLAB 后，我们会采集一个样本图像，然后查看原始图像与像素标签的合成图。

原始图像

合成图像



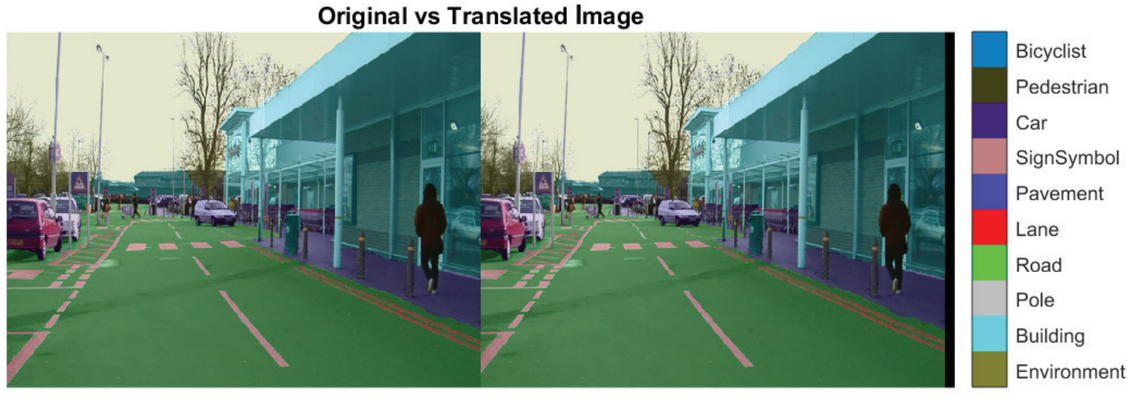

```
% 将分割结果叠加在原始图像上。
B = labeloverlay(I,C,'ColorMap',cmap);
```

数据增强是提高训练模型准确率的有用技术。在数据增强中，通过添加改变版本的原始图像可增加训练图像中的变体数量。最常见的数据增强类型是图像转换：旋转、平移和缩放。

在本例中，我们使用了随机平移。

```
augmenter = imageDataAugmenter('RandXTranslation',...
    [-10 10],'RandYTranslation',[-10 10]);
```

下例中，通过将原始图像向左移动 10 个像素创建了一个新图像。



虽然这种平移很微小，但它可以迫使神经网络去学习和理解很可能发生在现实生活中的微小变化来增加深度学习网络的鲁棒性。

3.设置分割网络

回忆一下，语义分割网络由一个图像分类网络和创建最终像素分类的向上采样部分组成。

我们可以使用 MATLAB **segnetLayers()** 函数自动创建网络的向上采样部分。

这会形成一个有向无环图 (DAG) 网络。

```
% segnetLayers 返回 SegNet 网络层：lgraph，已利用预先训练好的模型的层和权重进行预先初始化。
lgraph = segnetLayers(imageSize,numClasses,'vgg16');
```

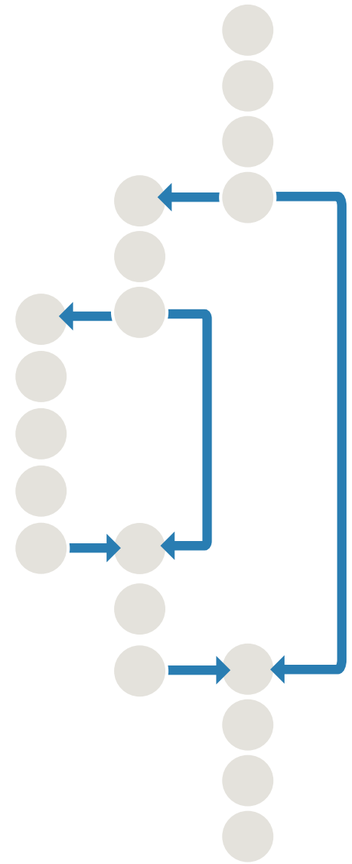
与串联网络不同，DAG 网络可以有多层输入或输出。DAG 允许在各层之间建立更复杂的连接，因而可以在不同的分类任务上形成更高的准确率。

注意此结构中的分支：一个输入节点可以转到多个输出。

通过调用此行代码，可对任意 DAG 网络结构实现可视化：

```
plot(lgraph);
```

lgraph 是描述 DAG 网络架构的层次图，包括所有层及其互连。



4.训练网络

像其他示例一样，我们有很多训练选项 (training option)。我们指定的选项包括：

- 优化算法。我们使用动量随机梯度下降 (SGDM)。这是一种常用的 CNN 训练方法。
- 批大小。我们使用的最小批大小为 4，可以在训练过程减少内存使用。批大小可根据 GPU 内存可用量提高或降低。
- 处理器。此网络利用 NVIDIA™ Tesla K40c 进行训练。通过指定更多高级硬件（例如，可以使用多 GPU 集群，而不是只有一个 GPU 的台式机），可大幅缩短训练时间。

技巧

有多款 GPU 设备可帮助加快训练速度。如何找到合适的 GPU 取决于各种因素，包括速度要求和价格。在 MATLAB 中使用 GPU 的最低要求是支持 3.0 及以上计算能力的 NVIDIA GPU。

```
options = trainingOptions('sgdm', ...
    'Momentum', 0.9, ...
    'InitialLearnRate', 1e-2, ...
    'L2Regularization', 0.0005, ...
    'MaxEpochs', 120,...
    'MiniBatchSize', 4, ...
    'Shuffle', 'every-epoch', ...
    'Verbose', false,...
    'Plots','training-progress');
```

使用此类方案训练网络大约会耗用 19 小时。为缩短训练时间，我们可以调整某些参数。例如：

- 迭代次数。如果迭代次数减少 20 次，训练约耗用 10.5 小时。
- 最小批大小。增加最小批大小将缩短训练时间，因为 GPU（或 CPU）可同时处理更多数据。增加 1 便可将训练时间从 19 小时缩短至 12 小时。
- 学习速率。每一数量级的学习速率降低 (0.1 → 0.01) 都会使总训练时间增加约半小时。

5.评估网络

就网络准确率而言，我们需要通过在测试数据上运行以及编译指标进行定量评估，同时需要通过测试数据结果可视化来进行定性评估。

我们将使用在训练之前留出的测试数据来计算全局准确率：正确分类的像素与总像素的比率，无论是何种类。

```
metrics.DataSetMetrics
```

```
ans =  1×5 table
```

	GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
1	0.9220	0.8976	0.6709	0.8511	0.7833

全局准确率指标显示 92% 的像素将会予以正确标注，但单独的图像类又会怎样呢?如果网站能正确识别每一个路标，但不能识别行人，这个结果是否可以接受呢?

网络准确率测量

- MeanAccuracy：每个类中正确分类的像素数与总像素数的比率，取所有类的平均值。该值等于 **`ClassMetrics.Accuracy`** 的平均值。
- MeanIoU：所有类的平均交并比 (IoU)。该值等于 **`ClassMetrics.IoU`** 的平均值。
- WeightedIoU：所有类的平均交并比 IoU，取类中像素数的加权值。
- MeanBFScore：所有图像的平均边界 F1 (BF) 得分。该值等于 **`ImageMetrics.BFScore`** 的平均值。

了解更多

语义分割指标

要查看网络识别单独图像类的准确率，我们可以查看类指标。

```
metrics.ClassMetrics
```

```
ans =  10×3 table
```

	Accuracy	IoU	MeanBFScore
1 Environment	0.9485	0.9059	0.7871
2 Building	0.9055	0.8456	0.7828
3 Pole	0.8156	0.3104	0.7683
4 Road	0.9165	0.8912	0.8825
5 Lane	0.9310	0.4550	0.8242
6 Pavement	0.9288	0.8187	0.8673
7 SignSymbol	0.8274	0.4727	0.6441
8 Car	0.9403	0.8283	0.8212
9 Pedestrian	0.8895	0.4860	0.6717
10 Bicyclist	0.8729	0.6952	0.7875

图表显示了汽车、环境和道路分类的精确率达到 90% 以上。路灯、路标和骑车人的分类准确率低于 90%。

视具体应用而定，这可能是可以接受的结果，或者说，网络可能需要在再次训练时提高对分类不当类别的重视程度。

故障成本决定了所需的准确率水平。例如，小型机器人可视系统偶尔对人类误分类或可接受，但自动驾驶汽车对行人误分类则肯定是不能接受的。

最后，我们把经过训练的网络输出图像结果与原始、手工标注的图像，并排显示。

```
pic_num = 200;
I = readimage(imds, pic_num);
Ib = readimage(pxds, pic_num);
IB = labeloverlay(I, Ib, 'Colormap', cmap, 'Transparency',0.8);
figure
% 显示语义分割的结果
C = semanticseg(I, net);
CB = labeloverlay(I, C, 'Colormap', cmap, 'Transparency',0.8);
figure
imshowpair(IB,CB,'montage')
HelperFunctions.pixelLabelColorbar(cmap, classes);
title('真实路况与预测路况对比')
```

我们看到存在一些差异，例如右图中的路灯误分类为人行道。

视最终应用而定，此网络可能足够准确，或者我们需要返工，针对有意提高检测准确率的差异部分，展开更多图像训练。

了解更多

揭开深度学习的神秘面纱：语义分割和部署 47:10
分析训练数据用于语义分割

不仅仅用于图像

截至目前，我们分析的三个示例着重于图像识别。但深度学习正日益应用于其他应用领域，例如语音识别和文本分析便使用信号数据，而不是图像数据。在接下来的几节中，我们将简要回顾一下用于信号数据分类的两个常用技巧：

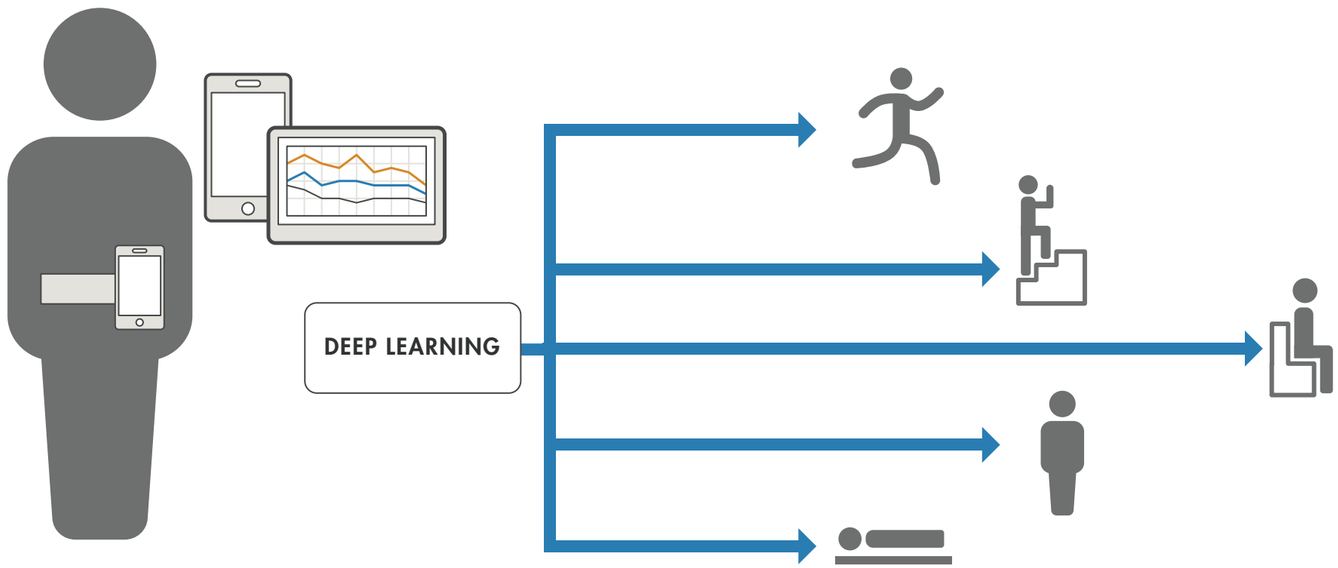
- 使用长短期记忆 (LSTM) 对在智能手机上捕获的信号数据进行分类
- 使用频谱图对音频文件数据进行分类

使用 LSTM 网络对人类活动进行分类

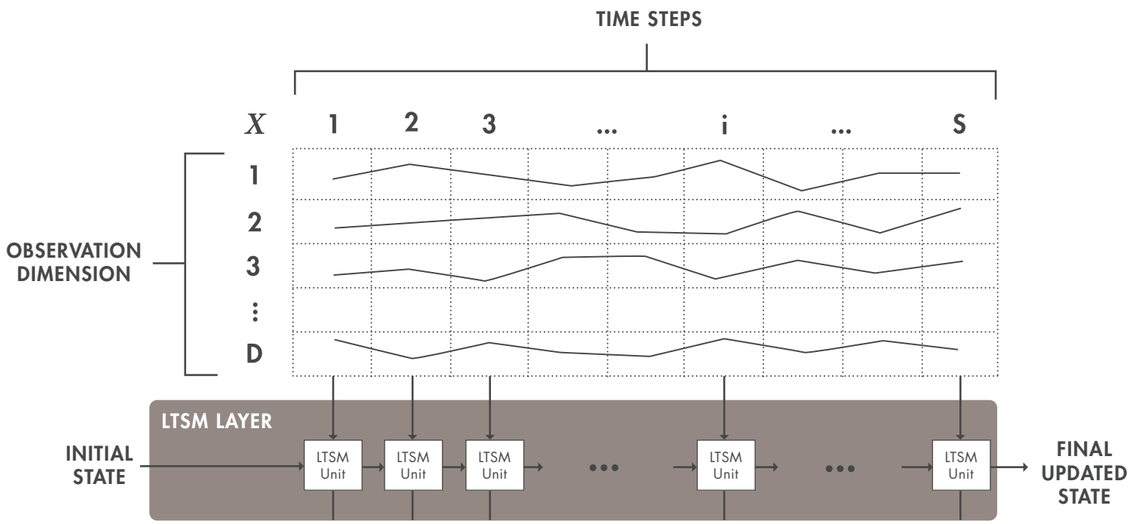
在本例中，我们要使用从智能手机上捕获的信号数据来对六项活动进行分类：在平地上行走、上楼梯、下楼梯、坐姿、站立和躺卧。

LSTM 网络十分适合此类分类任务，因为任务涉及序列数据：使用 LSTM，可基于单独的序列数据时间步长进行预测。

> LSTM 网络是一种循环神经网络 (RNN)，可学习时间步长序列数据之间的长期依赖关系。与传统的 CNN 不同，LSTM 可以记住预测之间的网络状态。



LSTM 架构



LSTM 网络由一系列输入层定义，每个输入层均用于一个收集数据的通道。第一个 LSTM 单元采用初始网络状态和序列的第一个时间步长进行预测，并将更新后的网络状态发送给下一个 LSTM 单元。

LSTM 网络的核心组件是序列输入层和 LSTM 层。序列输入层将序列或时序数据输入网络。LSTM 层学习序列数据时间步长之间的长期依赖关系。

下面以图示的方式介绍简单 LSTM 网络架构如何进行分类。

网络的第一层是序列输入层，然后是 LSTM 层。其余层则与以前示例中创建的图像分类模型相同。(要预测类标签，网络必须以全连接层、softmax 层和分类输出层结束。)

在整合了两个新层 (序列层和 LSTM 层) 之后，我们的信号数据便可用于训练可对新活动信号进行分类的模型。

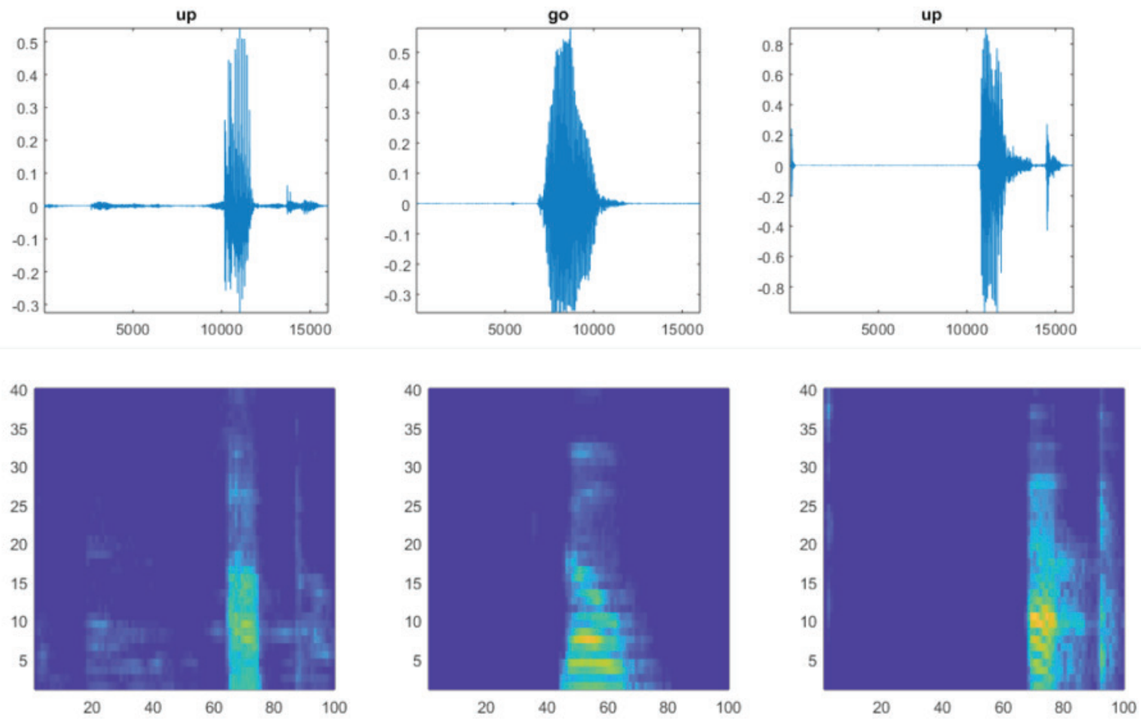
当训练好的网络在新数据上运行时，便能达到 95% 的准确率。对我们的活动跟踪应用来说，这个结果是令人满意的。

了解更多

长短期记忆网络
使用 LSTM 网络对序列数据进行分类
使用 LSTM 网络对文本数据进行分类

使用频谱图进行语音识别

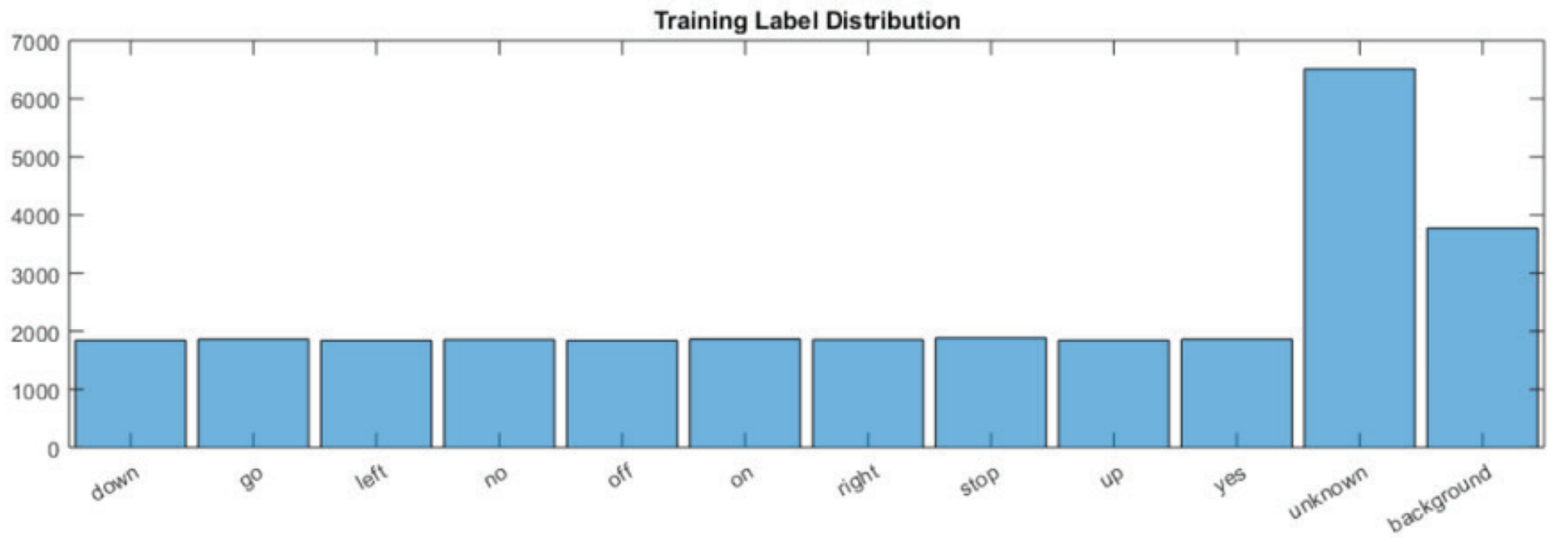
在本例中，我们要根据语音文件对应的文字类对这些文件进行分类。我们将使用频谱图将 1D 音频文件转换为可用作传统 CNN 输入的 2D 图像。



上：原始语音信号。下：相应频谱图。

`spectrogram()` 命令是将音频文件转换为相应时间定位频率的简单方式。然而，语音是音频处理的一种特殊形式，具有以特定频率定位的重要特征。因为我们需要 CNN 关注这些位置，所以我们要使用 Mel 频率倒谱系数。这些倒谱系数专门针对语音最相关的频率区域。

我们将训练数据均匀分布在要分类的文字类之间。



为减少误报，我们将包括可能与预期类别混淆的文字类别。例如，如果目标文字是“上”，那么与“上”发音类似或容易与“上”混淆的文字，比如“忙”、“当”和“让”便放置在“未知”类别中。

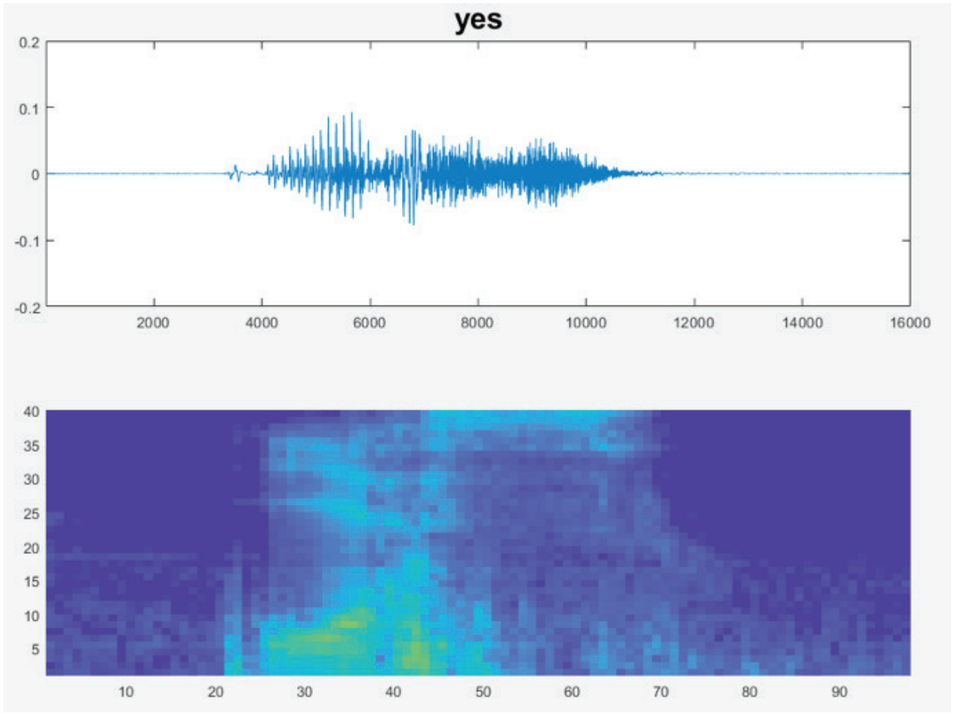
网络不需要识别这些文字，只要知道它们不是需要识别的目标文字即可。

然后，我们要定义 CNN。因为我们使用频谱图作为输入，实质上它是 1D 信号的 2D 表示，所以 CNN 的结构可以与我们用于图像处理的结构非常相似。

技巧

如果特征与原始训练集不同，迁移学习的效果便会打折扣。这意味着预先训练好的 AlexNet 或 GoogLeNet（通过图像进行训练）迁移到频谱图的效果不会太好。

模型完成训练后，将采集输入图像（频谱图），然后划分为适当的类别。验证集的准确率约为 96%。



在 Audio System Toolbox™ 上，可以使用 **`audioDeviceReader`** 针对麦克风发出的持续实时信号运行最终模型。

了解更多

训练一个深度学习语音识别模型
利用时序和序列数据进行深度学习的示例

部署深度学习网络

现在，您已训练好一个符合准确率目标的网络，接下来便可将其部署为一个应用程序。深度学习模型可部署到生产系统（现场或云端）、桌面上以及嵌入设备（例如 NVIDIA Tegra GPU 或 Intel® 或 ARM® 处理器）上。

当然，您选择的部署方案将取决于正在开发的应用程序种类。下面是工程师选择部署深度学习模型的四种最常用方式：

将模型部署为桌面应用程序

使用 MATLAB Compiler™ 将模型打包为最终用户可以在本地计算机上运行的独立应用程序。

部署到服务器或云端

使用 MATLAB Production Server™ 将模型部署为可以从 C、C++、Java®、.NET 或 Python® 调用的 API。

面向基于桌面的 GPU，提高性能速度

使用 GPU Coder™ 生成 CUDA 代码进行训练和预测。

部署到嵌入设备

使用 GPU Coder 生成可以在 MATLAB 之外运行的优化 CUDA 代码。

了解更多

共享和部署 MATLAB 应用程序 26:15

便利的深度学习工具

正如本电子书中的示例所示，使用 MATLAB 即可构建深度学习模型，而无需成为专家 — 而且，MATLAB 可以借助用于管理和标记数据、监控训练进度以及可视化结果的工具和函数，在深度学习中轻松完成更耗时或更令人厌烦的任务。下面是示例讲解过程中所用工具的快速指南。

工具或函数	描述
`ImageDataStore`	管理用于深度学习模型的大型训练和测试图像集。 创建一个自定义读取函数以自动执行对图像的预处理。
`ImageLabeler`	将感兴趣目标以方框标示出来。 也可在像素级快速标记图像，供语义分割。
`imageDataAugmenter`	通过自动平移、旋转和缩放现有图像的方式创建更多测试图像，从而扩展训练数据集。
`heatmap`	使用这个简单的混淆矩阵在训练好的模型中实现每个类别准确率的可视化。
`deepDreamImage` / `activations`	实现模型层可视化，以及实现通过层传输的图像输出可视化。
`spectrogram`	处理信号数据时，可以轻松将音频文件转换为相应时间定位频率。

了解更多

在 MATLAB 中进行深度学习的工具和资源指南