MATLAB을 활용한 딥러닝

실전 예제

MathWorks®

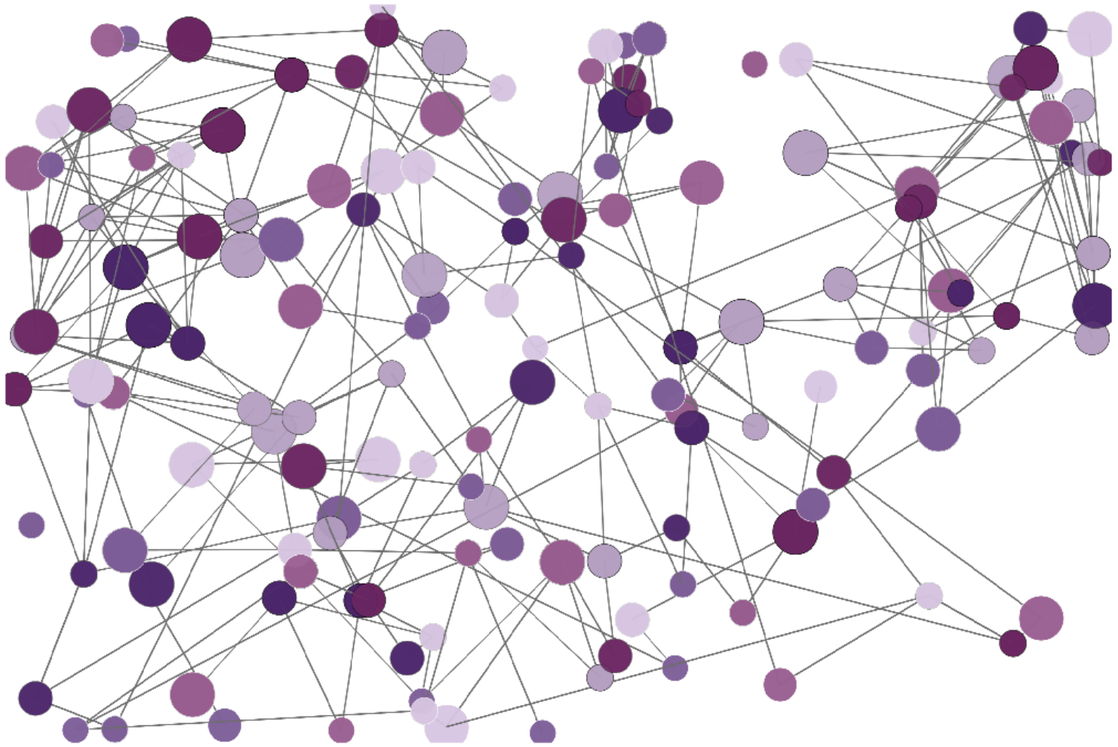
소개

이 ebook은 "딥러닝이란 무엇인가?"라는 질문에 답을 준 *MATLAB을 활용한 딥러닝* ebook의 후속편입니다. 본 ebook에서는 MATLAB을 활용한 딥러닝이 실제로 어떻게 이루어지는지 살펴보는 실전편입니다. 딥러닝 네트워크의 학습 과정을 세 가지로 나누어 살펴보겠습니다.

- 백지 상태의 네트워크를 학습시킵니다.
- 전이 학습을 이용하여 기존의 네트워크를 학습시킵니다.
- 기존의 네트워크를 학습시켜서 시멘틱 분할을 수행하도록 합니다.

여기에서는 주로 이미지 분류 예제를 다루지만, 딥러닝은 다른 응용 분야에도 널리 활용하고 있습니다. 본 ebook의 두 번째 파트에서는 이미지에 활용한 많은 딥러닝 기법을 신호 데이터에도 활용하는 두 가지 예제를 살펴보겠습니다.

모든 예제와 코드를 *(다운로드)*할 수 있습니다.

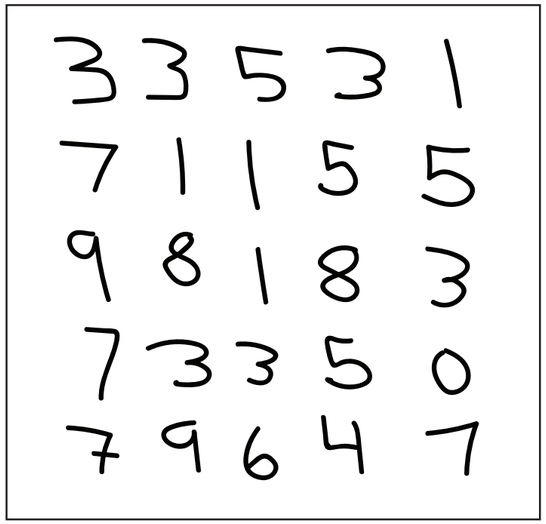


기본 내용 살펴보기

- *딥러닝이란?* 3:33
- *딥러닝과 머신 러닝 비교* 3:48

실전 예제 #1: 백지 상태부터 모델 학습시키기

이 예제에서는 *CNN(컨벌루션 뉴럴 네트워크)*을 이용하여 손글씨 숫자를 식별해 봅니다. 손글씨로 쓴 0부터 9까지의 숫자 이미지 60,000개가 수록된 *MNIST 데이터셋*을 활용합니다. MNIST 데이터셋에서 손글씨 숫자 25개를 임의로 추출했습니다.

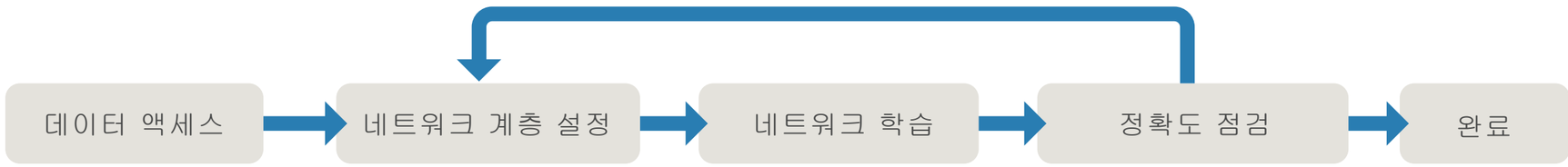


비교적 간단한 데이터셋을 이용하기 때문에, 메모리에 부담이 큰 처리 과정 없이도 딥러닝의 워크플로를 이루는 주요한 단계를 모두 살펴볼 수 있습니다. 여기에서 다루는 워크플로는 더 복잡한 딥러닝 문제나 더 큰 데이터셋에도 적용할 수 있습니다.

이제 막 딥러닝을 응용하기 시작했다면 이 데이터셋을 이용하여 값비싼 GPU를 구매하지 않고도 모델을 학습시킬 수 있다는 장점이 있습니다.

데이터셋이 비록 단순하지만, 정확한 딥러닝 모델과 학습 옵션을 이용한다면 99% 이상의 정확도를 달성할 수 있습니다. 그렇다면 이러한 모델은 어떻게 생성해야 할까요?

반복적인 과정을 거쳐야 할 것입니다. 앞에서 학습한 결과를 이용해서 학습 문제에 어떻게 접근할 것인가를 계획해야 합니다. 다음과 같은 단계를 거치게 됩니다.



1. 데이터에 액세스

우선 *MNIST*이미지를 MATLAB®에 다운로드합니다. 데이터셋이 다양한 파일 형식으로 저장됩니다. 이 데이터는 이진 파일로서 저장되고 MATLAB은 이 파일을 빠르게 사용하여 이미지 형상으로 변경합니다.

아래 코드로 원본 이진 파일을 읽어서 모든 예제 이미지로 이루어진 배열을 생성합니다.

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

% Reshape the data part into a 4D array
rawImgDataTrain = reshape(rawImgDataTrain, [numRows, numCols,...
    numImgs]);
imgDataTrain(:,:,1,ii) = uint8(rawImgDataTrain(:,:,ii));
```

명창에 **whos**를 입력하여 데이터의 크기와 클래스를 확인할 수 있습니다.

```
>> whos imgDataTrain

Name            Size                     Bytes      Class

imgDataTrain    28x28x1x60000            47040000   uint8
```

MNIST 이미지는 크기가 28 x 28 픽셀로 상당히 작고, 학습용 이미지는 총 60,000개입니다.

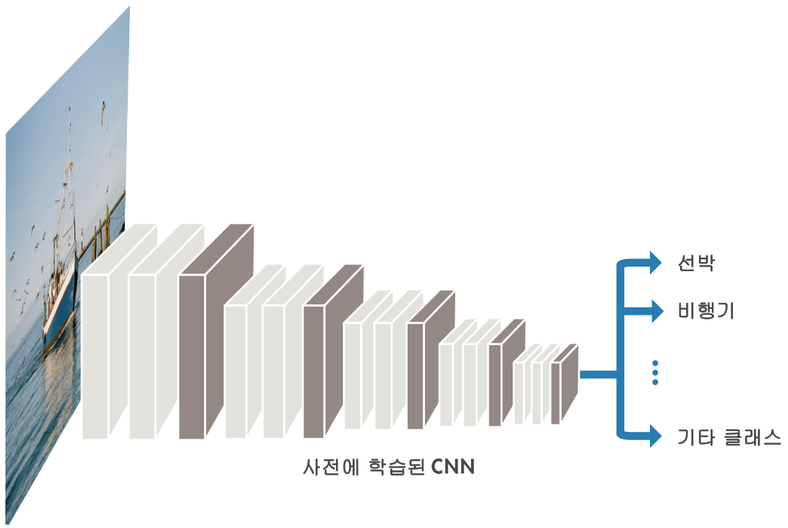
다음 작업은 이미지 레이블링이지만 MNIST 이미지에는 레이블이 달려 있기 때문에 이 과정을 건너 뛰고 신경망 구축 과정으로 바로 넘어갑니다.

2. 네트워크 계층의 생성과 설정

다음으로는 가장 널리 사용되는 딥러닝 네트워크인 CNN을 구축합니다.

CNN 소개

CNN은 네트워크 계층에 이미지를 통과시켜 최종적인 클래스를 산출합니다. 네트워크에는 수십, 수백 개의 계층이 있고 각 계층이 여러 가지 특징을 검출하게 됩니다. 필터는 해상도가 서로 다른 각 학습 이미지에 적용되고, 각 나선 이미지의 출력은 다음 계층의 입력으로 활용됩니다. 필터는 밝기 및 가장자리 등과 같이 매우 단순한 특징에서 시작하여, 계층이 진행되면서 객체만의 고유한 특징으로 더 복잡하게 발전할 수 있습니다.



사전에 학습된 CNN

추가 정보

컨벌루션 뉴럴 네트워크란? 4:44

널리 사용하는 네트워크 계층

컨벌루션은 입력 이미지를 컨벌루션 필터들에 넣고, 각각의 필터는 이미지의 특정한 특징을 활성화합니다.

ReLU(Rectified Linear Unit)는 음수 값을 0에 매핑하고 양수 값을 유지하여 더 빠르고 효과적인 학습을 가능하게 합니다.

풀링(Pooling)은 비선형 다운샘플링을 수행하고 네트워크에서 학습해야 하는 파라미터 개수를 줄여서 출력을 간소화합니다.

Flatten **완전 연결** 계층은 2차원적 공간 특징이 1차원 벡터로 단순화됩니다. 이러한 1차원 벡터는 이미지 수준의 특징을 나타내어 분류할 때 사용됩니다.

소프트맥스(Softmax)는 데이터셋에 있는 각 범주의 확률을 제공합니다.

처음부터 네트워크를 구축할 때에는 자주 사용하는 계층을 단순히 조합하는 것이 좋습니다. 복잡하지 않기 때문에 디버깅이 훨씬 쉬워지기 때문입니다. 그렇지만 원하는 정확도를 달성하려면 계층 몇 개를 추가해야 할 것입니다.

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

3. 네트워크 학습시키기

학습을 시작하기 전에 학습 옵션을 선택합니다. 다양한 옵션을 이용할 수 있습니다.

가장 흔히 사용하는 옵션이 아래 표에 정리되어 있습니다.

학습 옵션	정의	힌트
학습 과정을 그래프로 작성(Plot of training progress)	그래프는 미니배치 손실(minibatch loss)과 정확도를 나타냅니다. 언제라도 네트워크 학습을 중단시킬 수 있는 정지 버튼이 있습니다.	('Plots','training-progress') 네트워크가 학습해가는 진전상황을 그래프로 그립니다.
최대 에포크 (Max epochs)	에포크(epoch)란 학습 알고리즘이 전체 학습 세트를 완전히 통과하는 것을 말합니다.	('MaxEpoch',20) 에포크를 많이 지정하면 네트워크의 학습 시간이 길어지지만 각 에포크의 정확도가 개선될 것입니다.
미니배치 사이즈 (Minibatch size)	미니배치(minibatch)란 GPU에서 동시에 처리되는 학습 데이터셋의 부분 집합입니다.	('MiniBatchSize',64) 미니배치가 커질수록 학습이 더 빨라지지만 최대 크기는 GPU 메모리에 따라 결정될 것입니다. 학습 중에 메모리 오류가 발생하면 미니배치의 크기를 줄입니다.
학습 속도 (Learning rate)	학습 속도를 조절하는 주요한 매개 변수입니다.	학습 속도를 낮추면 결과가 더 정확해지지만 네트워크의 학습 시간이 길어질 것입니다.

옵션으로 진전상황 그래프와 미니배치 사이즈를 규정합니다.

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

이어서 네트워크를 실행하고 진전상황을 모니터링합니다.

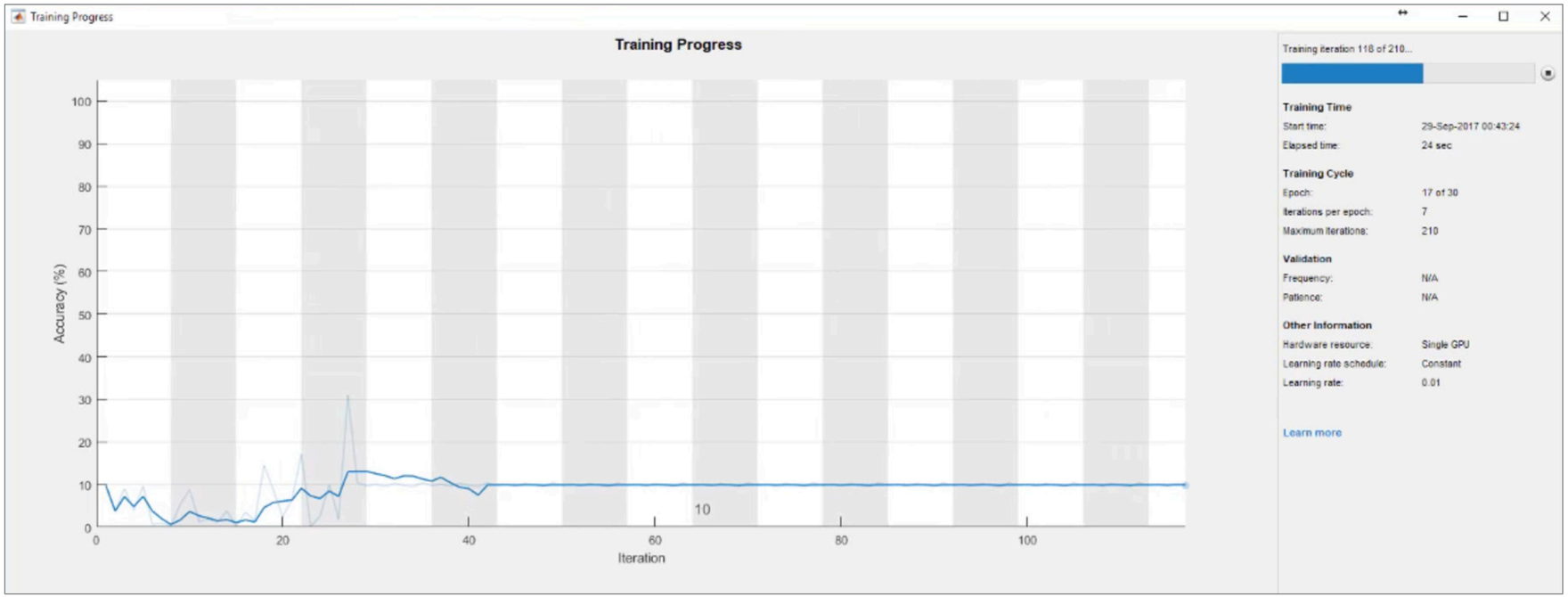
팁

데이터셋이 크면 처리시간이 느려집니다. 그렇지만 딥러닝 네트워크는 병렬 구조로 되어 있는 GPU를 활용할 수 있습니다. 정확한 속도향상 정도는 하드웨어, 데이터셋의 크기, 네트워크 설정 등의 인자에 따라 결정되지만, 학습 시간이 수 시간에서 단 몇 분으로 줄어드는 것을 알 수 있습니다.

MATLAB의 학습 옵션에서는 하드웨어 리소스를 네트워크 학습용으로 빠르게 변경할 수 있습니다. 이 옵션을 지정하지 않으면 기본적으로 사용할 수 있는 GPU 하나로 학습이 이루어집니다.

4. 네트워크 정확도 점검

시간이 지남에 따라 모델의 정확도가 높아지도록 하는 것이 목표입니다. 네트워크의 학습이 진행되면서 진전상황 그래프가 나타납니다.



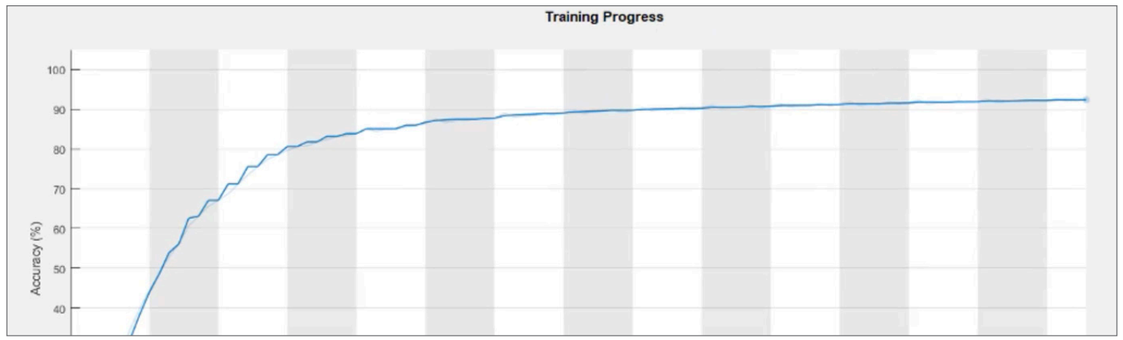
모델이 28번 반복한 뒤로는 개선을 멈추고, 이어서 정확도가 약 10% 감소하는 것으로 보입니다. 백지 상태부터 네트워크 학습을 시작하면 이런 모습이 흔히 나타납니다. 즉 네트워크가 어떤 솔루션으로 수렴하지 못한다는 것을 의미합니다. 정확도가 안정기에 도달해서 더 이상 개선되지 않습니다. 이 때에는 계속할 필요가 없습니다. 학습을 마치고 다른 방법을 시도해봅니다.

화면 우측 상단에 있는 정지 버튼을 클릭해서 학습을 중단하고 네트워크의 현재 상태로 복귀합니다. 실행이 중단되면 처음부터 다시 학습을 시작해야 합니다. 중단한 시점부터 다시 시작할 수는 없습니다.

네트워크의 정확도를 조정할 수 있는 다양한 방법이 있습니다.
예를 들면

- 학습 이미지의 개수를 늘립니다.
- 학습 이미지의 품질을 높입니다.
- 학습 옵션을 변경합니다.
- 네트워크 설정을 변경합니다(예를 들면 계층을 추가, 삭제 또는 재구성).

학습 옵션과 네트워크 설정을 변경해 봅시다.



학습 옵션의 변경

우선 학습 속도를 조정합니다. 초기 학습 속도를 기본 속도인 0.01보다 훨씬 낮게 설정합니다.

```
'InitialLearnRate', 0.0001
```

매개 변수 하나를 변경해서 훨씬 나은 결과를 얻게 됩니다. 정확도가 거의 90%에 도달합니다.

일부 응용 분야에서는 이 정도로 충분할 수 있지만 저희 목표는 99% 였습니다.

고급 팁

학습 매개 변수의 최적 값을 알아내기 위해 베이지안 최적화를 이용할 수 있습니다. 베이지안 최적화(Bayesian Optimization)를 하면 네트워크를 여러 차례 실행합니다(프로세스를 병렬로 진행할 수도 있습니다).

네트워크 설정 변경

정확도를 90%에서 99%로 올리려면 네트워크의 심도를 높이고 시행착오 횟수를 늘려야 합니다. 배치 정규화 계층을 비롯한 계층을 추가하면 네트워크 수렴(새로운 입력에 정확히 반응하는 시점) 속도를 높일 수 있습니다.

```
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];
```

이제 네트워크의 심도가 높아졌습니다. 이제 네트워크를 변경하고 학습 옵션은 이전과 같은 상태로 둡니다.

네트워크가 학습을 마친 뒤에 이미지 10,000개로 네트워크를 시험합니다.

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

```
accuracy = 0.9880
```

이 네트워크는 약 99%의 최고 정확도를 달성합니다. 이제 이 네트워크로 온라인 이미지나 라이브 비디오 스트림에 있는 손글씨를 식별해 봅니다.

추가 정보

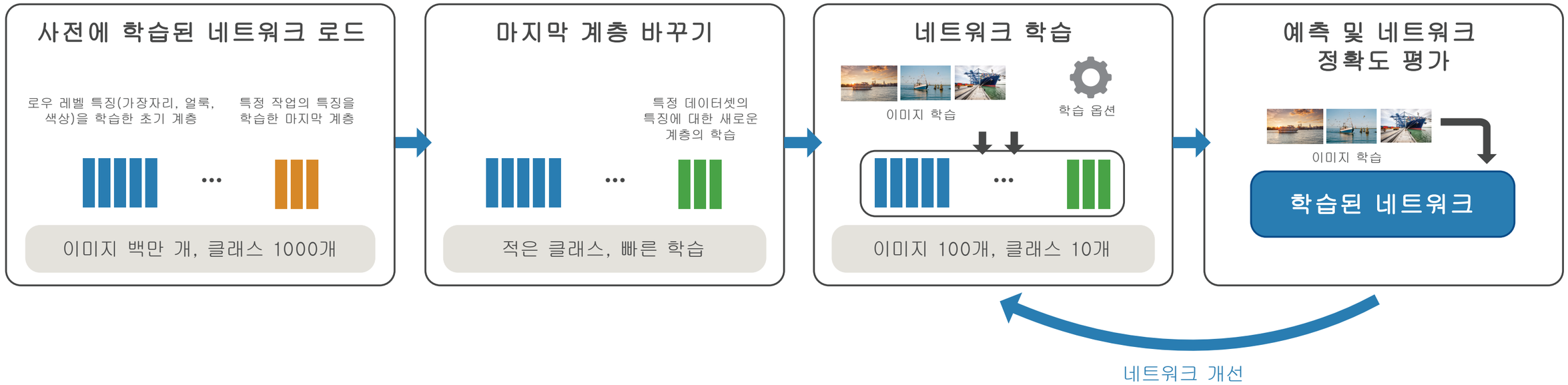
MATLAB으로 백지 상태의 네트워크 학습시키기 5:13
단 11줄의 MATLAB 코드로 딥러닝하기 2:38

실전 예제 #2: 전이 학습

이 예제에서는 미리 학습한 네트워크를 수정하고 전이 학습을 이용하여 이 네트워크가 새로운 인지 작업을 수행하도록 합니다. 사전 학습된 네트워크를 미세하게 조정하면 새로운 네트워크를 구축하고 학습시키는 것보다 훨씬 빠르고 쉽습니다. 훨씬 적은 학습 이미지를 이용하여 새로운 작업으로 빠르게 학습을 전이할 수 있습니다. 전이 학습의 장점은 사전 학습된 네트워크는 이미 많은 이미지로부터 특징을 잘 추출하도록 학습되었다는 것입니다.

자전거, 자동차, 개 등의 1000가지 객체를 이미 학습한 GoogLeNet을 사용해 봅니다. 이 네트워크를 다시 학습시켜서 다섯 가지 음식을 식별해 보도록 합니다. 다음과 같은 단계를 거칩니다.

1. 사전 학습된 네트워크를 가져옵니다.
2. 마지막 세 계층을 설정하여 새로운 인식 작업을 수행하도록 합니다.
3. 네트워크가 새로운 데이터를 학습하도록 합니다.
4. 결과를 점검합니다.



1. 사전 학습된 네트워크 가져오기

코드 한 줄로 GoogLeNet을 가져올 수 있습니다.

```
% Load a pretrained network
net = googlenet;
```

사전 학습된 네트워크는 네트워크를 설정하는 막대한 작업(계층 선정과 조직화)이 이미 이루어져 있습니다. 그렇기 때문에 다시 설정할 필요 없이, 네트워크가 원래 학습한 범주에 있는 이미지로 네트워크를 점검할 수 있습니다.

```
%% Test it on an image
img = imread('peppers.png');
imgLabel = net.classify(imresize(img, [224 224]));
```

팁

이 코드를 이용하여 GoogLeNet이 이미 학습한 1000가지 범주를 모두 살펴봅니다.

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

전이 학습 팁

- 매우 정확한 네트워크로 시작합니다. 원래 인식한 작업에 대한 정확도가 50%에 불과하다면 그 네트워크는 새로운 인식 작업에도 정확성을 나타낼 가능성이 낮습니다.
- 새로운 인식 범주가 원래의 범주와 비슷한 특징을 갖고 있다면 모델이 정확할 가능성이 높습니다. 예를 들면 개에 대해 학습한 네트워크는 다른 동물들도 비교적 빠르게 학습할 것입니다.

2. 네트워크가 새로운 작업을 수행하도록 설정하기

GoogLeNet이 새로운 이미지를 분류하도록 학습시키기 위해서 네트워크의 마지막 세 계층만 다시 설정합니다. 이 계층들에는 네트워크가 추출한 특징들을 클래스 확률 및 레이블에 조합하는데 필요한 정보가 담겨 있습니다. GoogLeNet에는 144개 계층이 있습니다. 여기에 마지막 5개 계층을 표시합니다.

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

```
140        'pool5-7x7_s1'                  Average Pooling
141        'pool5-drop_7x7_s1'             Dropout
142        'loss3-classifier'              Fully Connected
143        'prob'                          softmax
144        'output'                        Classification Output
```

계층 143 소프트맥스 계층과 계층 144 분류 출력 계층을 재설정합니다. 이 계층들은 정확한 범주를 입력 이미지에 배정하는 역할을 합니다. 이 계층들이 원래 학습한 범주가 아닌, 새로운 범주에 상응하도록 하려 합니다. 완전히 연결된 최종 계층을 새로운 데이터셋에 있는 클래스의 개수와 같은 크기로 설정합니다. 이 예에서는 5개입니다.

팁

새 계층에서의 학습 속도가 원래의 계층보다 빠르게 하려면 완전히 연결된 계층의 학습 속도를 높입니다.

```
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. 네트워크에 새로운 데이터 학습시키기

백지 상태의 네트워크와 마찬가지로, 네트워크의 정확도를 높이기 위해 학습 옵션 몇 개를 조정합니다(이 예에서는 배치 크기, 학습 속도, 유효성 검사 데이터).

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

% Training with the optimized set of hyperparameters
tic
disp('Initialization may take up to a minute before training
begins')
net = trainNetwork(trainDS, layers_train, opts);
toc
```

이 모델의 학습 시간은 특히 하드웨어에 따라 크게 달라질 수 있습니다. Tesla P100 GPU 한 대를 사용하면 이 모델이 학습하는 데 대략 20분이 걸립니다.

팁

`tic`과 `toc`을 이용하여 학습 실행에 얼마나 걸리는지 빠르게 알아봅니다. `tic`은 성능 측정을 위해 스톱워치 타이머를 시작하고`toc`은 타이머를 중지하고 명령 창에 표시된 경과 시간을 읽습니다.

팁

GPU에 메모리 부족(out-of-memory) 오류가 발생하면 `'MiniBatchSize'` 값을 낮춥니다.

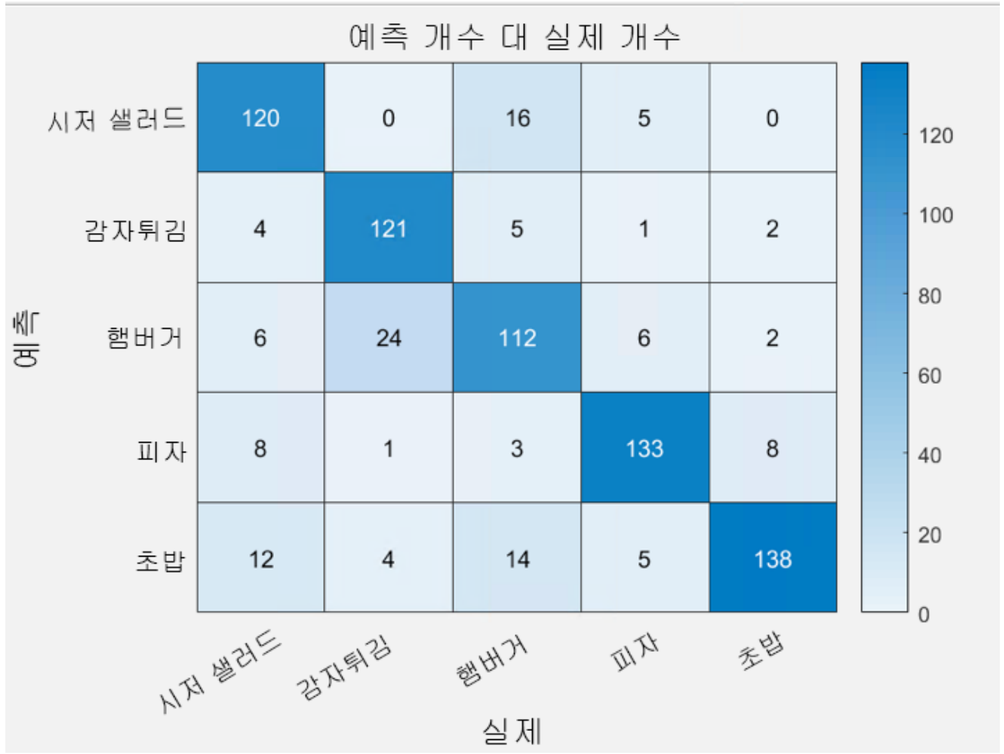
4. 네트워크 평가하기

네트워크가 학습을 마쳤다면 이제 새로운 데이터에 대해 어떤 성능을 발휘하는지 살펴봅니다.

```
% Classify all images from test dataset
[labels,err_test] = classify(net, testDS);

accuracy_default = sum(labels == testDS.Labels)/numel(labels);
disp(['Test accuracy is ' num2str(accuracy_default)])
```

정오분류표에는 각 범주에 있는 150개 이미지에 대한 네트워크의 예측결과가 나타나 있습니다. 대각선에 있는 모든 값이 150이었다면 테스트 이미지를 정확하게 분류한 것입니다. 아직은 정확하지 않네요. 그럼 어떤 범주가 잘못 분류되었는지 대각선 이외의 값들을 이용하여 알 수 있습니다. 이 값들이 어떤 데이터를 자세히 살펴보아야 할지 알려 줍니다.



모델을 학습시킨 뒤의 최종 정확도는 83%입니다. 예제에서는 충분한 수치지만 실제 세계에 적용하기에는 부족할 것입니다. 실제 세계에 적용하기 위해 모델의 정확도를 높이려면 계속 반복 학습을 시키고 학습 옵션을 다시 살펴보며 데이터를 검사해 보고, 네트워크를 다시 설정해야 할 것입니다.

마지막으로는 새로운 이미지에 대한 네트워크의 성능을 육안으로 확인해 봅니다.

```
[label,conf] = classify(net,im);
% classify a random image
imshow(im_display);
title(sprintf('%s %.2f, actual %s', ...
     char(label),max(conf),char(actualLabel))
```

백지 상태의 네트워크를 구축하는 경우에도, 전이 학습은 딥러닝을 배우기 위한 좋은 시작점이 될 수 있습니다. 이 분야의 전문가들이 개발한 네트워크를 활용하여 몇 가지 계층을 수정하고 학습을 시작하는 경우, 해당 모델은 원래의 데이터셋에서 많은 특징을 학습했기 때문에 백지 상태에서 개발한 모델보다는 적은 학습 이미지로 더 빠르게 학습할 수 있습니다.

추가 정보

사전 학습된 컨벌루션 뉴럴 네트워크
GoogLeNet을 이용한 전이 학습
단 10줄의 MATLAB 코드를 사용한 전이 학습 4:00
MATLAB의 신경망을 사용한 전이 학습 4:06

실전 예제 #3: 시맨틱 분할

딥러닝 분야에 새롭게 등장한 시맨틱 분할을 이용하면 이미지의 특징을 입자와 픽셀 수준에서 이해할 수 있습니다. 기존의 CNN이 이미지의 특징을 분류한다면 시맨틱 분할은 특정한 범주(예: 꽃, 도로, 사람, 자동차)와 각각의 픽셀을 연계시킵니다. 아래와 같은 결과가 산출됩니다.

시맨틱 분할을 하면 도로와 같은 불규칙한 형상의 객체가 잘 규정되는 것을 알 수 있습니다.

시맨틱 분할에서는 원하는 객체가 이미지의 여러 영역에 걸쳐 있어도 되기 때문에 객체 감지의 유용한 대안이 될 수 있습니다. 객체가 경계 상자 안에 있어야 하는 객체 감지와 달리, 시맨틱 분할에서는 형상이 불규칙한 객체도 깔끔하게 감지됩니다.

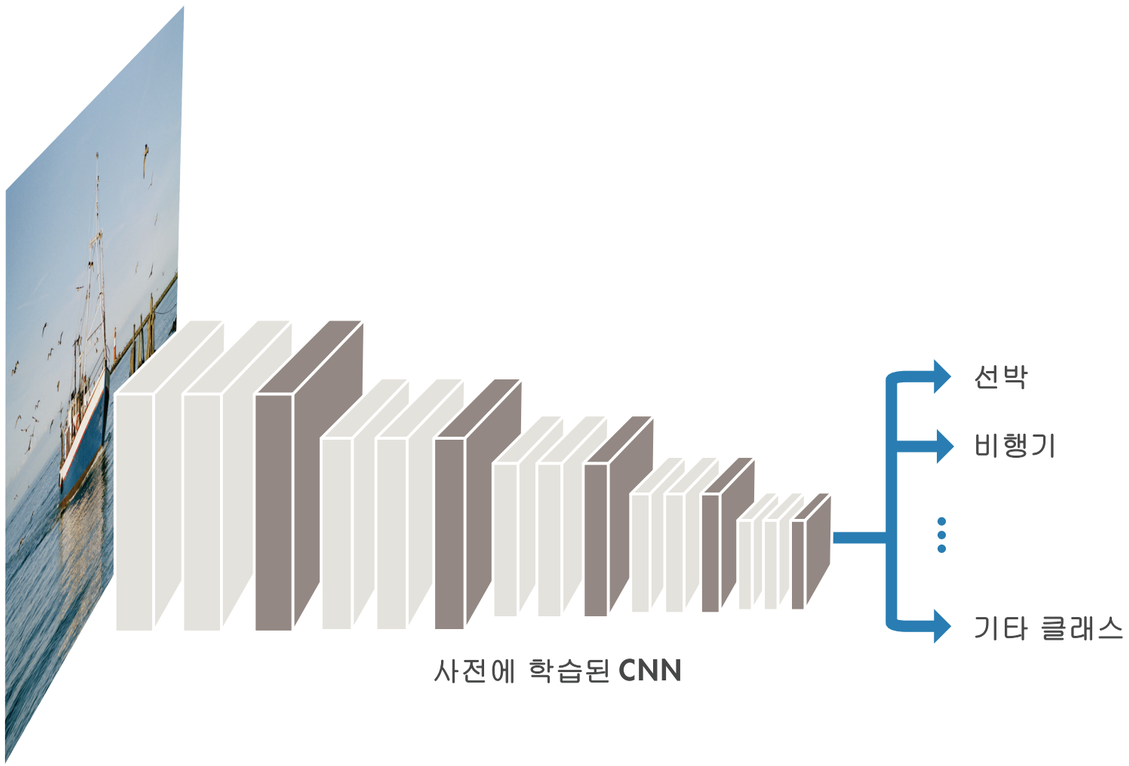
시맨틱 분할의 적용 사례

- 자율주행: 보행자, 인도, 기둥, 다른 자동차와 같은 장애물과 도로를 분리하여 차량이 주행 가능한 경로를 식별
- 산업 검사: 재료에 포함된 결함 감지(웨이퍼 검사 등)
- 위성 영상: 산, 강, 사막과 같은 다양한 지형을 식별
- 의료 영상: 세포에서 암과 같은 이형을 분석 및 감지

예제로 진행하기 전에 시맨틱 분할 네트워크의 구조를 살펴봅시다.

시맨틱 분할 네트워크 구조

예제 1과 2에서 살펴본 것처럼 기존의 CNN은 이미지를 네트워크 계층에 통과시키고 최종 클래스로 산출합니다.



사전에 학습된 **CNN**

시맨틱 분할 네트워크는 CNN을 역전시킨 것과 비슷한 구조로 된 업샘플링 네트워크를 이용하여 이 과정을 활용합니다.

이러한 일련의 새로운 계층들은 미리 학습된 네트워크의 결과를 다시 이미지로 *업샘플링*합니다. 그렇게 하면 모든 픽셀에 분류 레이블이 할당된 이미지를 얻게 됩니다.

1. 사전 학습된 네트워크 가져오기

이 예제에서는 자율주행 시스템에서 주행 가능한 도로 공간, 차선, 인도를 감지할 수 있는 네트워크를 구축합니다.

다음과 같은 단계를 거치게 됩니다.

1. 사전 학습된 네트워크를 가져옵니다.
2. 데이터셋에 로딩합니다.
3. 네트워크를 설정합니다.
4. 네트워크에게 학습을 시킵니다.
5. 네트워크의 정확도를 평가합니다.

백지 상태의 네트워크를 학습시킬 수도 있지만, 이 예제에서는 사전에 학습된 네트워크를 활용합니다. 앞의 예제에서 살펴보았듯이, 네트워크를 재설정하지 않고도 원래 학습한 범주의 이미지를 미리 학습된 네트워크로 시험해 볼 수 있습니다.

여기에서 사전에 학습된 네트워크는 VGG-16입니다. VGG-16은 *ILSVRC(ImageNet Large-Scale Visual Recognition Challenge)*에 사용하는 네트워크로서 백만 개 이상의 이미지를 학습하였고, 이미지를 1000개의 객체 범주로 분류할 수 있습니다.

MATLAB 코드에 단 한 줄만 추가하면 VGG-16을 가져올 수 있습니다.

```
% Download and install Neural Network Toolbox Model for VGG-16
Network support package.
vgg16;
```

2. 데이터셋에 로딩하기

주행 중에 얻은 도로 수준의 장면으로 이루어진 이미지 컬렉션 *CamVid 데이터셋*을 이용합니다. 이 데이터셋은 자동차, 행인, 도로 등 32개 시맨틱 클래스에 대한 픽셀 수준의 레이블을 제공합니다.

데이터셋에 있는 각각의 이미지에는 컬러 이미지 및 이미지에 있는 각 픽셀에 대한 레이블 이미지가 있습니다.

많은 이미지를 메모리에 넣는 작업은 번거로울 수 있습니다. 큰 데이터 파일은 데이터 저장소를 이용하여 편리하게 가져오고 접근, 관리할 수 있습니다. 이미지, 픽셀, 스프레드시트 데이터 저장소는 저장된 데이터가 동일한 구조와 양식으로 되어 있으면 리포지토리 역할을 할 수 있습니다. 이 시맨틱 분할 예제에서는 두 가지 데이터 저장소 객체를 생성합니다.

- **`ImageDatastore`**: 전체 컬렉션은 아니어도 각각의 개별 이미지를 메모리에 넣어서 이미지 파일을 관리합니다.
- **`pixelLabelDatastore`**: 픽셀 레이블이 포함된 이미지 디렉터리를 가져옵니다.

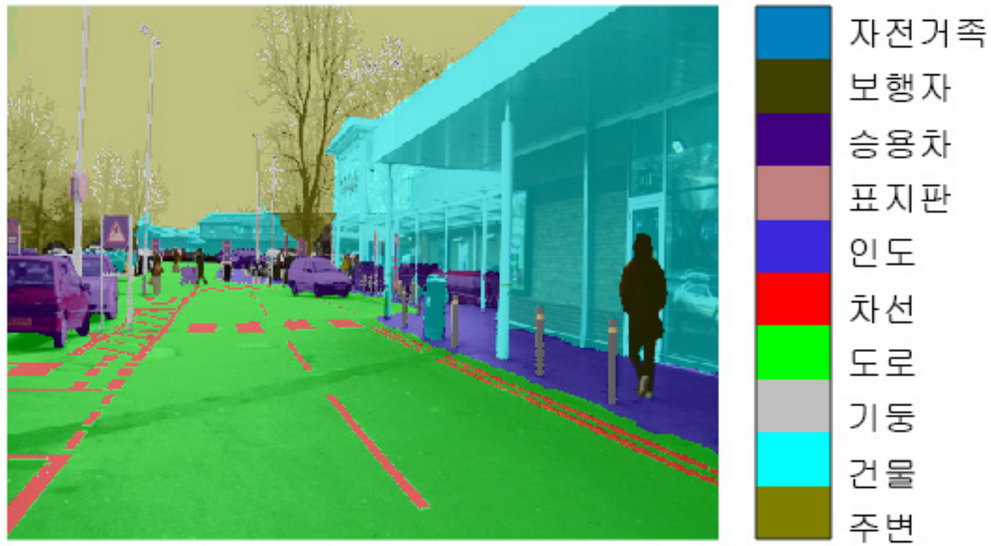
팁

식별하려는 범주에 상응하는 미리 레이블링된 데이터셋을 찾을 수 없으면, 스스로 데이터셋을 생성해야 합니다. 이미지 레이블러(Image Labeler)를 이용하면 이러한 번거로운 작업을 손쉽게 할 수 있습니다. 픽셀 그룹만 선택하면 앱이 상응하는 범주와 색상으로 자동 레이블링을 합니다.

이미지 데이터와 픽셀 레이블 데이터를 MATLAB에 가져왔다면 샘플 이미지를 취해서 원본 이미지와 픽셀 레이블이 조합된 합성 이미지를 살펴봅니다.

원본 이미지

합성 이미지




```
% Overlay segmentation results onto original image.
B = labeloverlay(I,C,'ColorMap',cmap);
```

데이터 증강(data augmentation) 기법을 활용하면 학습된 모델의 정확도를 높일 수 있습니다. 데이터 증강에서는 원본 이미지의 변경 버전을 추가함으로써 변화된 학습 이미지의 개수를 늘릴 수 있습니다. 가장 흔한 유형의 데이터 증강은 이미지 전환(회전, 평행이동, 스케일링)입니다.

이 예제에서 임의 평행이동을 도입해 봅니다.

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

예를 들어 원본 이미지를 좌측으로 10 픽셀 이동하여 새로운 이미지를 생성해 봅니다.

원본 이미지 대 평행이동 이미지

평행이동이 유발한 효과는 미미하지만, 네트워크가 작은 변화를 학습하고 이해하도록 강제함으로써 딥러닝 네트워크의 실력을 높일 수 있습니다. 실제 세계에서 활용하는 시스템에서는 대부분 이런 식으로 진행될 것입니다.

3. 분할 네트워크 설정하기

다시 상기해보면, 시맨틱 분할 네트워크는 이미지 분류 네트워크와 최종 픽셀 분류를 생성하는 업샘플링 부분으로 이루어져 있습니다.

MATLAB **segnetLayers()** 함수를 이용해서 자동으로 네트워크의 업샘플링 부분을 생성할 수 있습니다.

DAG(Directed Acyclic Graph) 네트워크를 생성하게 됩니다.

```
% segnetLayers returns SegNet network layers, lgraph, that is
preinitialized with layers and weights from a pretrained model.
lgraph = segnetLayers(imageSize,numClasses,'vgg16');
```

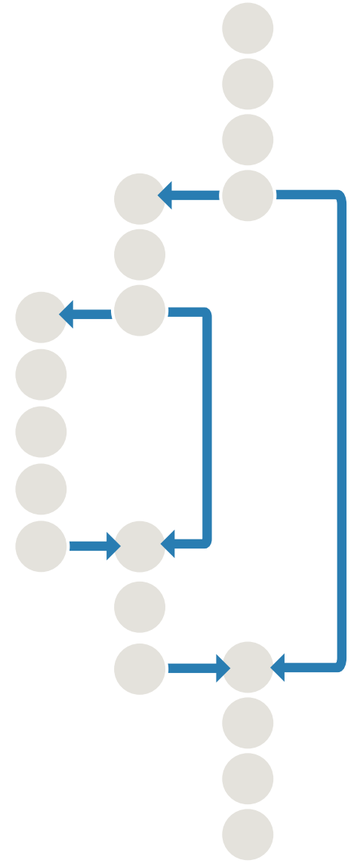
DAG 네트워크는 직렬 네트워크와는 달리, 다수의 계층과 입출력을 할 수 있습니다. DAG 네트워크는 계층 간에 더욱 복잡한 연결이 가능하고, 어려운 분류 작업에도 높은 정확도를 달성할 수 있습니다.

이 구조체의 분지 구조에 주목하시기 바랍니다. 하나의 입력 노드가 다수의 출력 결과를 낳을 수 있습니다.

아래 코드를 호출하면 모든 DAG 네트워크 구조를 표시할 수 있습니다.

```
plot(lgraph);
```

lgraph는 모든 계층 및 계층 간의 상호연결을 포함한 DAG 네트워크의 구조를 기술하는 계층 그래프입니다.



4. 네트워크 학습시키기

다른 예제와 마찬가지로 수많은 학습 옵션이 있습니다. 다음과 같은 옵션을 지정합니다.

- 최적화 알고리즘. SGDM(Stochastic Gradient Descent with Momentum)을 이용합니다. CNN 학습에 자주 사용하는 알고리즘입니다.
- 배치 사이즈. 학습 중의 메모리 사용량을 줄이기 위해 미니배치 사이즈를 4로 지정합니다. 배치 사이즈는 가용한 GPU 메모리 용량에 따라 늘리거나 줄일 수 있습니다.
- 프로세서. 이 네트워크는 NVIDIA™ Tesla K40c에서 학습하였습니다. 고급 하드웨어를 지정하면 학습 시간을 상당히 줄일 수 있습니다. 예를 들면 GPU가 하나인 데스크탑 대신에 멀티 GPU 클러스터를 활용할 수 있습니다.

팁

학습 속도를 높이는데 도움이 되는 많은 GPU가 있습니다. 알맞은 GPU를 선택하는 데에는 속도 요건이나 가격과 같은 다양한 인자가 영향을 미칩니다. MATLAB에 사용하는 최소 GPU 요건은 3.0 컴퓨트가 가능한 NVIDIA GPU입니다.

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

이러한 옵션으로 네트워크를 학습시키는 데에는 약 19시간이 걸립니다. 학습 시간을 줄이기 위해 매개 변수 몇 개를 조정할 수 있습니다. 예를 들면 다음과 같습니다.

- 반복 횟수. 반복 횟수를 20회 줄이면 학습에 약 10.5시간이 걸립니다.
- 미니배치 사이즈. 미니배치 사이즈를 늘리면 학습 시간이 줄어듭니다. GPU(또는 CPU)가 동시에 더 많은 데이터를 처리하기 때문입니다. 1을 늘리면 학습 시간이 19시간에서 12시간으로 줄어듭니다.
- 학습 속도. 학습 속도를 한 단위 내릴 때마다(0.1 → 0.01) 총 학습 시간이 약 30분 늘어납니다.

5. 네트워크 평가하기

테스트 데이터에 실행해 보고 메트릭를 컴파일해서 정량적으로 네트워크의 정확도를 평가하고 테스트 데이터 결과를 표시해서 정성적으로 네트워크의 정확도를 평가해 봅니다.

학습 전에 남겨 둔 테스트 데이터를 활용해서 전체적인 정확도를 계산합니다 (클래스와 무관하게 정확히 분류한 픽셀 대 총 픽셀의 비율).

```
metrics.DataSetMetrics
```

```
ans =  1×5 table
```

	GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
1	0.9220	0.8976	0.6709	0.8511	0.7833

픽셀을 정확하게 레이블링하는 전체적인 정확도는 92%지만 개별 이미지 클래스는 어떻습니까? 만일 네트워크가 도로 표지판은 모두 정확하게 식별하지만 보행자를 오식별한다면 그래도 괜찮습니까?

네트워크 정확도를 나타내는 척도

- 평균 정확도(MeanAccuracy): 각 클래스에서 정확히 분류된 픽셀 대 총 픽셀의 비율. 모든 클래스에 대해 평균을 구합니다. 이 값은 **`ClassMetrics.Accuracy`**의 평균과 같습니다.
- 평균 겹침공간(MeanIoU): 모든 클래스의 평균 겹침공간(IoU)입니다. 이 값은 **`ClassMetrics.IoU`**의 평균과 같습니다.
- 가중 겹침공간(WeightedIoU): 모든 클래스의 평균 IoU로서 클래스에 있는 픽셀의 개수를 가중시킵니다.
- 평균 BF 스코어(MeanBFScore): 모든 이미지의 평균 경계 F1(BF) 스코어입니다. 이 값은 **`ImageMetrics.BFScore`**의 평균과 같습니다.

추가 정보

시맨틱 분할 메트릭

네트워크가 개별 이미지 클래스를 얼마나 정확히 식별하였는지 알아보기 위해 클래스 메트릭을 살펴볼 수 있습니다.

```
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

이 표를 보면 자동차, 주변, 도로 범주가 90% 이상의 정확도로 분류되었음을 알 수 있습니다. 기둥, 표지판, 자전거족의 분류 정확도는 90% 미만입니다.

응용 분야에 따라 이 정도로 충분할 수 있지만, 오식별된 클래스를 위주로 네트워크를 다시 학습시켜야 할 수도 있습니다.

오식별이 유발하는 비용이 어느 정도인가에 따라 요구되는 정확도도 달라집니다. 예를 들면 소형 로봇의 영상 시스템은 간혹 사람을 오식별할 수도 있지만, 자율주행 자동차가 행인을 오식별하는 것은 용납되어선 안 될 것입니다.

마지막으로, 학습된 네트워크의 출력물 옆에 수작업으로 레이블링 한 원본 이미지를 표시해 봅니다.

```
pic_num = 200;
I = readimage(imds, pic_num);
Ib = readimage(pxds, pic_num);
IB = labeloverlay(I, Ib, 'Colormap', cmap, 'Transparency',0.8);
figure
% Show the results of the semantic segmentation
C = semanticseg(I, net);
CB = labeloverlay(I, C, 'Colormap', cmap, 'Transparency',0.8);
figure
imshowpair(IB,CB,'montage')
HelperFunctions.pixelLabelColorbar(cmap, classes);
title('Ground Truth vs Predicted')
```

약간의 차이가 있습니다. 예를 들면 이미지 우측에 있는 기둥이 인도로 잘못 분류되어 있습니다.

최종 용도에 따라 이러한 네트워크의 정확도는 충분할 수 있지만, 더욱 정확하게 감지하려는 차이에 관하여 더 많은 이미지를 학습해야 할 수도 있습니다.

추가 정보

딥러닝 파헤치기: 시맨틱 분할과 배포 47:10
시맨틱 분할을 위한 학습 데이터 분석

이미지 이외의 것들

지금까지 살펴본 세 가지 예제에서는 주로 이미지 인식을 다루었습니다. 그렇지만 이미지 데이터가 아닌 신호 데이터를 활용하는 음성인식, 텍스트 분석 등의 많은 응용 분야에 점점 더 딥러닝을 적용하고 있습니다. 다음 섹션에서는 신호 데이터의 분류에 주로 사용되는 두 가지 기법을 간단히 살펴볼 것입니다.

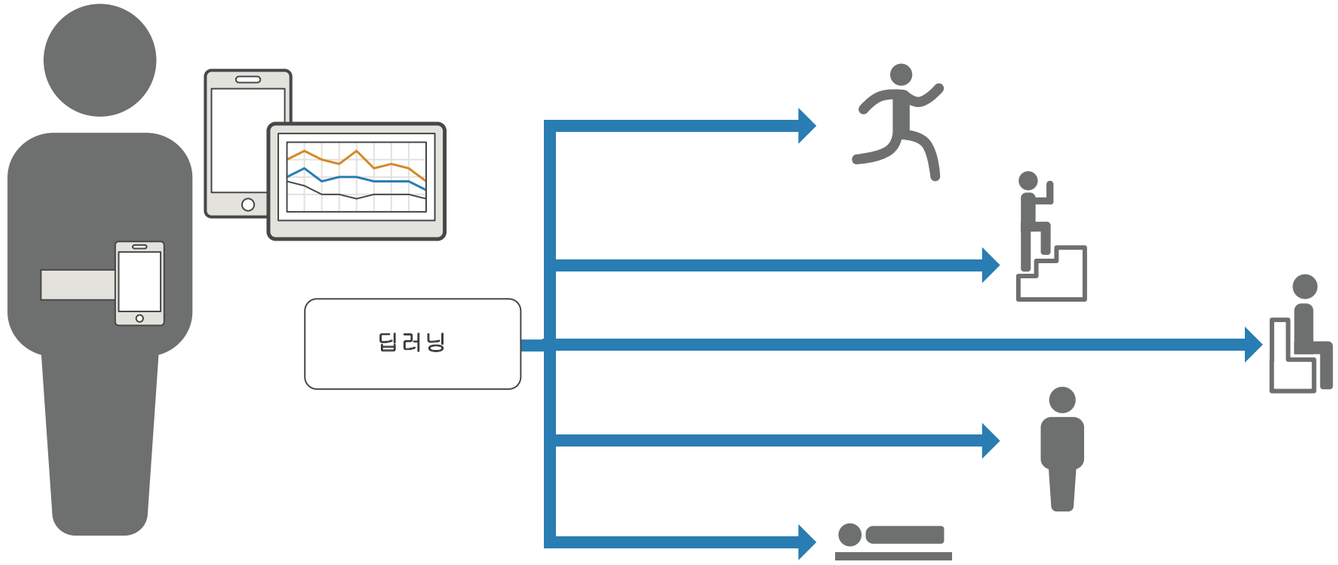
- 스마트폰에 캡쳐한 신호 데이터를 분류하기 위해 LSTM(Long Short-Term Memory)을 활용합니다.
- 오디오 파일에서 나온 데이터를 분류하기 위해 스펙트로그램을 활용합니다.

LSTM 네트워크를 활용한 인간 활동의 분류

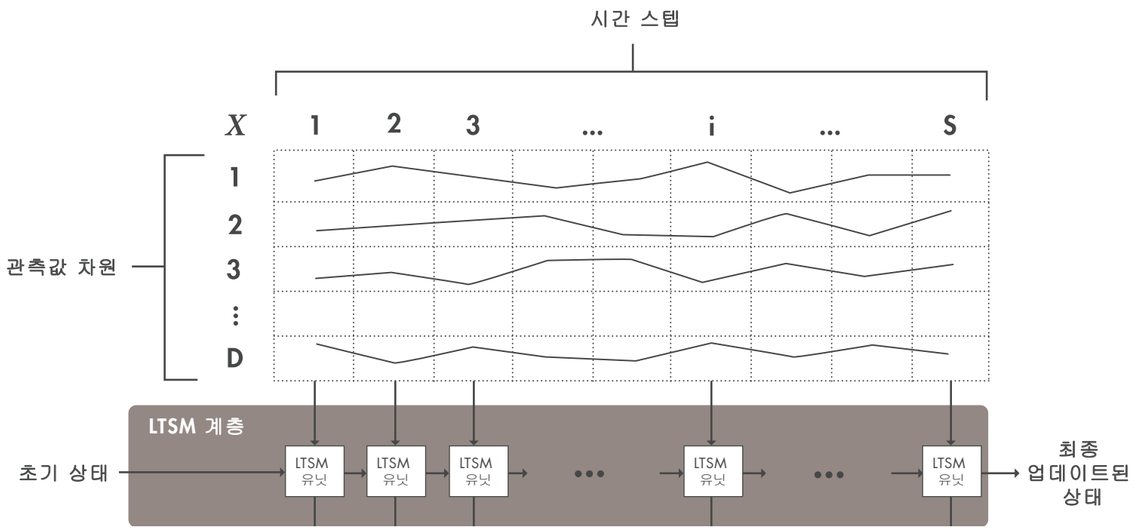
이 예에서는 스마트폰에서 캡쳐한 신호 데이터를 이용하여 여섯 가지 활동, 즉 평평한 땅에서 걷기, 계단 오르기, 계단 내려가기, 앉기, 서기, 눕기를 분류해 봅니다.

작업에 시퀀스 데이터가 수반되기 때문에 이러한 분류 작업에는 LSTM이 적합합니다. 시퀀스 데이터의 개별 시간 스텝을 기반으로 LSTM이 예측 작업을 수행합니다.

> LSTM 네트워크는 시퀀스 데이터의 시간 스텝들 사이의 장기적 의존성을 학습할 수 있는 일종의 RNN(Recurrent Neural Network)입니다. 기존의 CNN과 달리, LSTM은 예측 간에 네트워크의 상태를 기억할 수 있습니다.



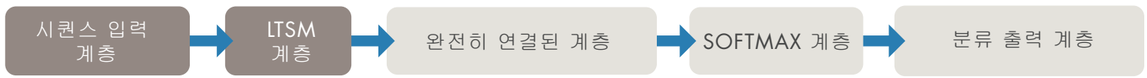
LSTM 아키텍처



LSTM 네트워크는 수집한 데이터 채널 각각에 상응하는 입력 계층 시퀀스로 정의합니다. 첫 번째 LSTM 유닛이 초기 네트워크 상태와 시퀀스의 첫 시간 스텝을 취하여 예측을 하고, 업데이트된 네트워크 상태를 다음 LSTM 유닛으로 보냅니다.

LSTM 네트워크의 핵심 구성요소는 시퀀스 입력 계층과 LSTM 계층입니다. 시퀀스 입력 계층은 시퀀스 또는 시계열 데이터를 네트워크에 입력합니다. LSTM 계층은 시퀀스 데이터의 시간 스텝 간 관계를 통해 장기적인 의존성을 학습합니다.

이 도표는 분류에 사용되는 간단한 LSTM 네트워크의 아키텍처를 나타냅니다.



네트워크는 시퀀스 입력 계층부터 시작하여 LSTM 계층으로 진행합니다. 나머지 계층은 앞의 예제에서 생성한 이미지 분류 모델과 동일합니다. (클래스 레이블을 예측하기 위해 네트워크는 완전히 연결된 계층, 소프트맥스 계층, 분류 출력 계층으로 끝납니다.)

새로운 계층 2개(시퀀스 계층과 LSTM 계층)를 추가함으로써, 새로운 활동 신호를 분류할 수 있는 모델을 신호 데이터를 이용하여 학습시킬 수 있습니다.

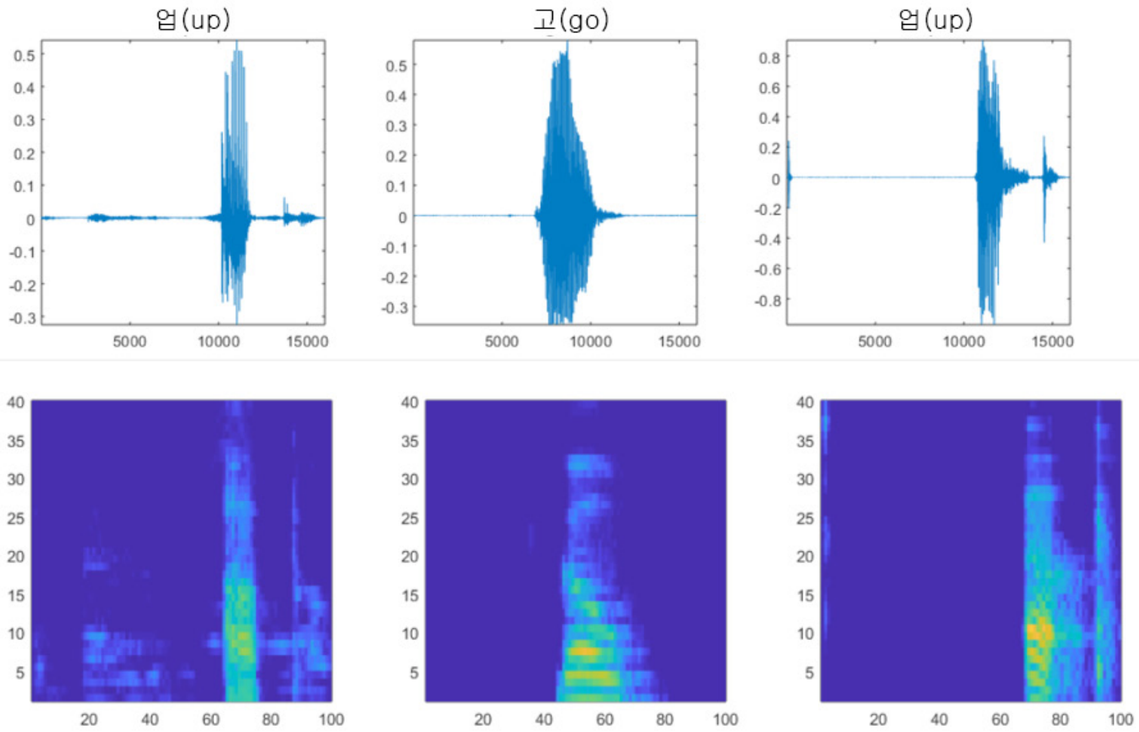
학습된 네트워크를 새로운 데이터에 실행하면 정확도 95%를 달성합니다. 활동 추적용 응용 프로그램에 적합한 결과입니다.

추가 정보

LSTM(Long Short-Term Memory) 네트워크
LSTM 네트워크를 사용한 시퀀스 데이터 분류
LSTM 네트워크를 사용한 텍스트 데이터 분류

스펙트로그램을 이용한 음성인식

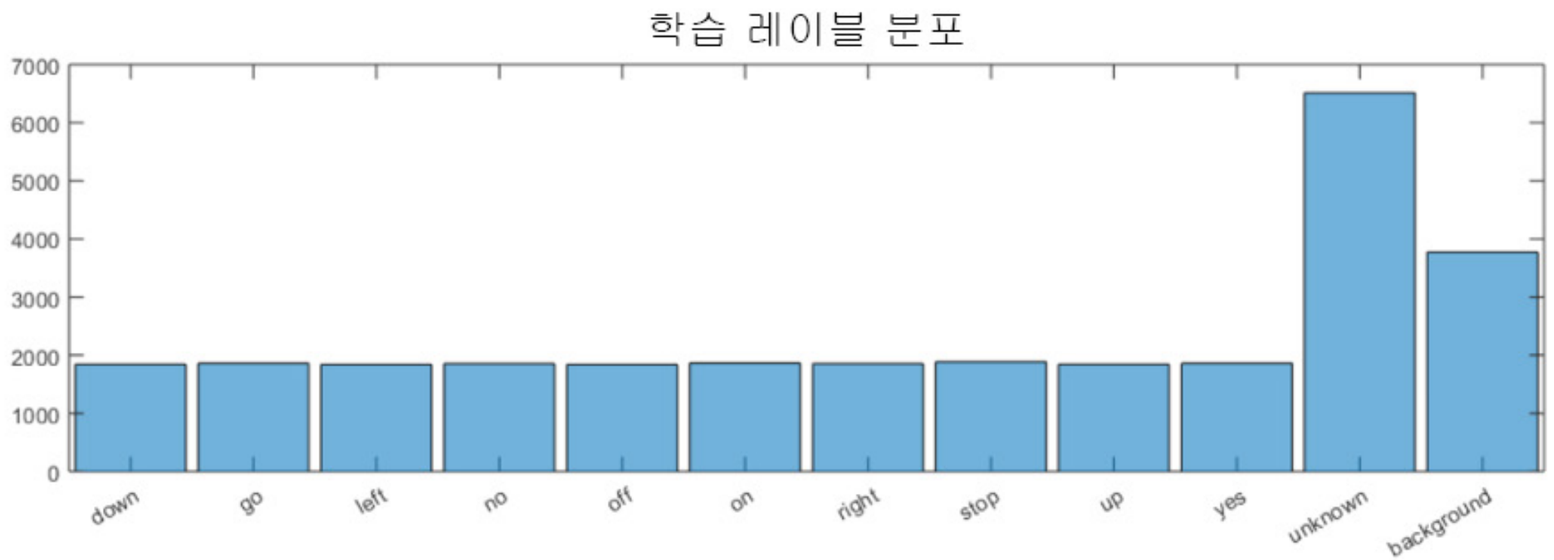
이 예에서는 음성 오디오 파일을 그에 상응하는 단어 클래스로 분류해 봅니다. 스펙트로그램을 이용하여 1차원 오디오 파일을 2차원 이미지로 변환하고, 이를 기존의 CNN에 입력물로 사용할 수 있습니다.



위: 원본 오디오 신호 아래: 신호에 상응하는 스펙트로그램

`spectrogram()` 명령을 이용하여 오디오 파일을 그에 상응하는 시간대별 주파수로 간단히 변환할 수 있습니다. 그렇지만 음성은 특수한 형태의 오디오 처리로서, 중요한 특징이 특정 주파수에 몰려 있습니다. CNN이 그러한 주파수에 집중하기를 원하기 때문에, 음성과 관련성이 가장 높은 주파수 영역을 대상으로 개발된 MFCC(mel-frequency cepstal coefficient)를 사용합니다.

분류하고자 하는 단어 클래스에 고르게 학습 데이터를 분산시킵니다.



긍정 오류를 줄이기 위해, 의도한 범주와 혼동될 수 있는 단어 범주를 포함시킵니다. 예를 들어 의도한 단어가 “on”인 경우, “on”과 발음이 비슷하거나 쉽게 혼동될 수 있는 단어인 “mom”, “dawn”, “won” 등을 “unknown” 범주에 넣습니다.

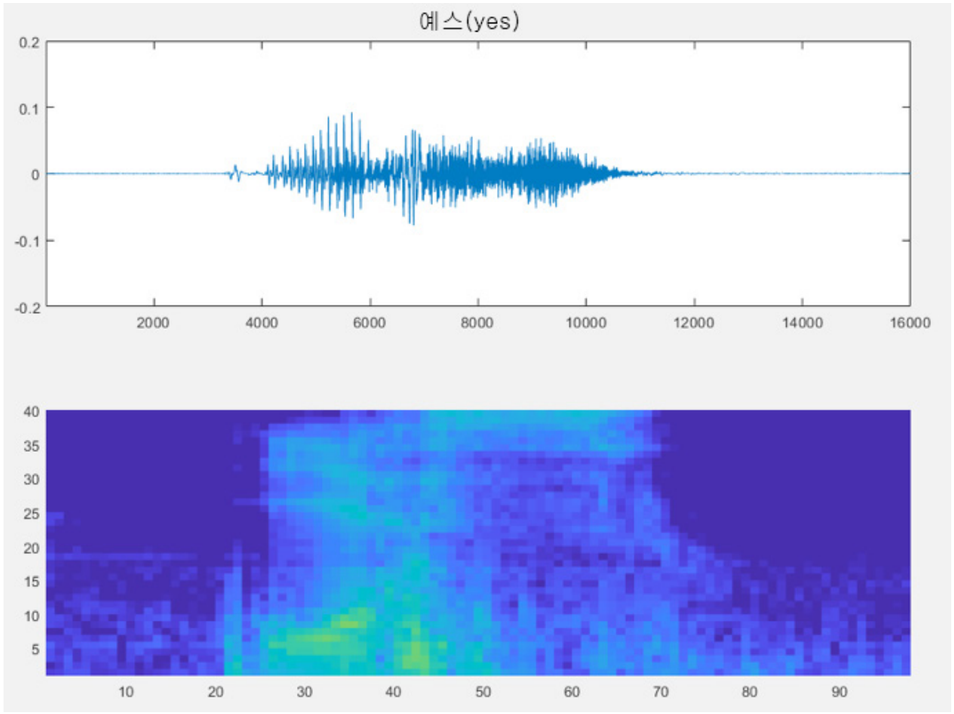
네트워크가 이러한 단어들을 알 필요는 없고, 인식하려는 단어가 ‘아니라는 것’만 알면 됩니다.

이어서 CNN을 정의합니다. 1차원 신호를 2차원으로 표현한 스펙트로그램을 입력물로 활용하기 때문에 CNN의 구조는 이미지 처리에 사용한 구조와 아주 비슷할 수 있습니다.

팁

원래의 학습 세트와 특징이 다르다면 전이 학습이 잘 되지 않습니다. 사전에 이미지를 학습한 AlexNet 또는 GoogLeNet과 같은 네트워크는 스펙트로그램으로 잘 전이되지 않음을 의미합니다.

모델이 학습을 완료하면 입력 이미지(스펙트로그램)를 취하여 적절한 범주로 분류할 것입니다. 유효성 검사의 정확도는 약 96%입니다.



마지막 모델은 Audio System Toolbox™의 **`audioDeviceReader`**를 이용하여 마이크에서 나온 연속적인 라이브 신호에 맞춰 연속적으로 실행할 수 있습니다.

추가 정보

딥러닝 음성인식 모델의 학습
시계열 및 시퀀스 데이터 예제를 이용한 딥러닝

딥러닝 네트워크의 설치

정확도 목표에 맞춰 네트워크를 학습시켰다면 이제 응용 프로그램으로서 네트워크를 설치할 준비가 되었습니다. 딥러닝 모델은 현장 또는 클라우드에 있는 프로덕션 시스템, 데스크탑, NVIDIA Tegra GPU, Intel® 또는 ARM® 프로세서와 같은 내장형 장치에 설치할 수 있습니다.

설치 옵션은 물론 개발하는 응용 프로그램의 종류에 따라 달라집니다. 딥러닝 모델을 설치하는 가장 흔한 방법 네 가지를 소개합니다.

모델을 데스크탑 응용 프로그램으로서 설치

MATLAB Compiler™를 이용하여 최종 사용자가 로컬 머신에서 실행할 수 있는 독립된 응용 프로그램으로서 모델을 패키징합니다.

서버나 클라우드에 설치

MATLAB Production Server™ 를 이용하여 C, C++, Java®, .NET, 또는 Python®에서 호출할 수 있는 API 형태로 모델을 설치합니다.

처리 속도 향상을 위해 데스크탑 GPU를 활용

GPU Coder™를 이용하여 학습과 예측에 사용되는 CUDA 코드를 생성합니다.

내장형 장치에 설치

GPU Coder를 이용하여 MATLAB 외부에서 실행할 수 있는 최적화된 CUDA 코드를 생성합니다.

추가 정보

MATLAB 응용 프로그램의 공유와 설치 26:15

딥러닝을 위한 편리한 툴

이 ebook에 있는 예제와 같이 MATLAB을 이용하면 전문가가 아니어도 딥러닝 모델을 구축할 수 있습니다. MATLAB은 데이터를 관리하고 레이블링하며 학습 진전상황을 모니터링하고 결과를 보여주는 툴을 통해, 시간이 많이 소요되고 번잡한 딥러닝 작업을 간소화합니다. 예제에서 사용한 툴들을 간략히 소개합니다.

툴 또는 함수	설명
`ImageDataStore`	딥러닝 모델을 위한 많은 학습용 이미지와 테스트 이미지를 관리합니다. 커스텀 읽기 함수를 생성하여 다수의 이미지에 대한 전처리 과정을 자동화합니다.
`ImageLabeler`	관심 대상인 객체 주위에 경계 박스를 그립니다. 픽셀 수준에서 이미지를 빠르게 레이블링을 하여 시맨틱 분할에 사용합니다.
`imageDataAugmenter`	기존 이미지에 대한 자동 평행이동, 회전, 스케일링을 통해 더 많은 테스트 이미지를 생성함으로써 학습 데이터셋을 확장합니다.
`heatmap`	단순한 이 정오분류표를 활용하여, 학습된 모델에 있는 각 범주의 정확도를 표시합니다.
`deepDreamImage` / `activations`	모델 계층을 표시하고, 계층을 통과한 이미지 출력물을 표시합니다.
`spectrogram`	신호 데이터를 다룰 때, 오디오 파일을 그에 상응하는 시간대별 주파수로 손쉽게 변환해 줍니다.

추가 정보

MATLAB을 활용한 딥러닝을 위한 툴과 리소스에 대한 가이드