Chapter 13. Convolutional Neural Networks

1. The Architecture of the Visual Cortex

1. Convolutional Layer

- Filters

- Stacking Multiple Feature Maps

- TensorFlow Implementation

- Memory Requirements

1. Pooling Layer

1. CNN Architectures

- LeNet-5

- AlexNet

- GoogLeNet

- ResNet

IBM의 딥 블루 슈퍼 컴퓨터는 1996 년 체스 세계 챔피언을 물리쳤지 만 최근에는 컴퓨터가 사진에서 강아지를 감지하거나 구어를 인식하는 것처럼 사소한 작업을 안정적으로 수행하지 못했습니다.

왜 이 일들이 우리 인간에게 그렇게 쉬운가? 해답은 뇌의 특수 시각, 청각 및 기타 감각 내에서 의식의 영역 밖에서 주로 인식이 일어난다는 사실에 있습니다. 감각 정보가 우리의 의식에 도달할 때까지 이미 높은 수준의 특징으로 꾸며져 있습니다. 예를 들어, 귀여운 강아지의 그림을 볼 때 강아지를 보지 않거나 귀여움을 느끼지 않도록 선택할 수는 없습니다. 또한 귀여운 강아지를 어떻게 인식하는지 설명할 수 없습니다. 그것은 당신에게 분명합니다. 따라서 우리는 주관적인 경험을 믿을 수 없습니다. 지각은 전혀 사소한 것이 아니며 이해하기 위해서는 감각이 어떻게 작용하는지를 봐야합니다.

CNN(Convolutional Neural Network)는 뇌의 시각 피질 연구에서 나왔으며 1980 년대부터 이미지 인식에 사용되어 왔습니다. 지난 몇 년 동안 계산 능력의 증가, 사용 가능한 학습 데이터의 양 및 제 11 장에서 심도있는 네트워크 교육을 위한 트릭 덕분에 CNN은 복잡한 시각적 작업에서 초인적인 성능을 얻을 수 있었습니다.

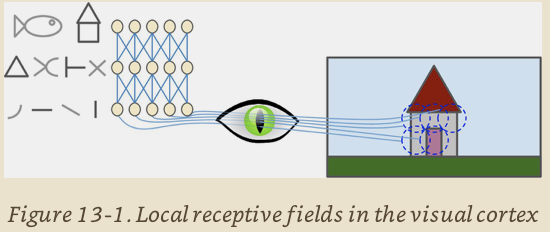
이미지 검색 서비스, 자동 운전 차량, 자동 비디오 분류 시스템 등을 지원합니다. 또한 CNN은 시각적 인식에 국한되지 않고 음성 인식이나 자연 언어 처리 (NLP)와 같은 다른 업무에도 성공적입니다. 그러나 지금은 시각적 응용 프로그램에 중점을 둘 것입니다.

이 장에서는 CNN이 어디서 왔는지, 빌딩 블록이 어떻게 보이는지, TensorFlow를 사용하여 CNN을 구현하는 방법에 대해 설명합니다. 그런 다음 우리는 최고의 CNN 구조 중 일부를 살펴볼 것입니다.

1. **The Architecture of the Visual Cortex**

David H. Hubel과 Torsten Wiesel은 1958년과 1959년 (그리고 몇 년 후 원숭이 3 마리) 고양이에 대한 일련의 실험을 수행하여 시각 피질(visual cortex)의 구조에 대한 결정적인 통찰력을 얻었습니다 .

특히 그들은 시각 피질의 많은 뉴런이 작은 수용영역을 가지고 있다는 것을 보여 주었는데, 이는 시각 영역의 제한된 지역에있는 시각적 자극에만 반응한다는 것을 의미한다 (그림 13-1 참조, 5 개 수용영역 뉴런은 파선으로 표시).



* 서로 다른 뉴런의 수용 영역은 겹칠 수 있으며, 함께 전체 시각 영역을 타일링합니다.
* 수평선의 이미지에만 반응하는 뉴런도 있고, 방향이 다른 라인에 대해서만 반응하는 뉴런 (두 개의 뉴런은 동일한 수용 필드를 가질 수 있지만 다른 라인 방향에 반응 할 수 있음)을 보여 주었다.
* 일부 뉴런은 더 큰 수용 영역을 가지고 있으며 더 낮은 수준의 패턴의 조합 인보다 복잡한 패턴에 반응한다는 것을 발견했다.

이러한 관찰은 더 높은 레벨의 뉴런이 인접한 하위 레벨 뉴런의 출력을 기반으로 한다는 아이디어를 도출하였습니다(그림 13-1에서 각 뉴런은 이전 레이어의 몇 뉴런에만 연결됨을 알 수 있음). 이 강력한 구조는 시각적 영역의 모든 영역에서 복잡한 패턴을 감지할 수 있습니다.

이러한 시각 피질에 대한 연구는 1980,4 년에 도입된 neocognitron에 영감을 주어 점차적으로 우리가 지금은 CNN이라고 부르는 것으로 발전했습니다. 중요한 이정표는 손으로 쓴 수표 번호를 인식하는데 널리 사용되는 유명한 LeNet-5 아키텍처를 도입한 Yann LeCun, Léon Bottou, Yoshua Bengio 및 Patrick Haffner의 1998 년 논문 이었습니다. 이 구조에는 완전히 연결된 레이어 및 Sigmoid 활성화 기능과 같이 이미 알고있는 빌딩 블록이 있지만 두 개의 새로운 빌딩 블록인 컨벌루션 레이어와 풀링 레이어도 있습니다.

이제 그것들을 살펴 봅시다.

[이미지 인식 작업을 위해 완전히 연결된 레이어가있는 일반적인 DNN을 사용하지 않는 이유는 무엇입니까? 불행히도 이 방법은 작은 이미지 (예 : MNIST)에 적합하지만 엄청난 양의 매개 변수가 필요하기 때문에 큰 이미지의 경우 분해됩니다. 예를 들어, 100 × 100 이미지는 10,000 픽셀을 가지며, 첫 번째 레이어의 뉴런이 1,000 개에 불과하면 (이미 다음 레이어로 전송되는 정보의 양이 크게 제한되어 있음) 총 1,000 만 개의 연결을 의미합니다. 그리고 이것은 단지 첫 번째 레이어일 뿐입니다. CNN은 부분적으로 연결된 레이어를 사용하여 이 문제를 해결합니다.]

1. **Convolutional Layer**

컨볼루션 = 어떤 선형 시스템이든 입 출력관계를 나타내는 하나의 디스크립션

CNN의 가장 중요한 빌딩 블록은 convolutional layer 입니다. 첫 번째 컨볼루션 레이어의 뉴런은 이전 장에서처럼 입력 이미지의 모든 단일 픽셀에 연결되지 않고 수용 필드의 픽셀에만 연결됩니다 (참조 그림 13-2).

차례로, 두 번째 컨볼루션 레이어의 각 뉴런은 첫 번째 레이어의 작은 사각형 안에 있는 뉴런에만 연결됩니다. 이 구조는 네트워크가 첫 번째 히든 레이어의 하위 레벨 feature에 집중한 다음, 다음 히든 레이어의 상위 레벨 feature로 어셈블하는 등의 작업을 수행 할 수 있습니다.

이 계층적 구조는 CNN이 이미지 인식에 잘 작동하는 이유 중 하나인 real-world 이미지에서 일반적입니다.

Convolutional 
layer 2 
Convolutional 
layer 1 
Input layer 
Figure 13-2. CNN layers with rectangular local 
receptivefields 

[Note]

[지금까지 우리가 보았던 모든 다층 신경망에는 긴 줄의 뉴런으로 구성된 레이어가 있었으며 입력 이미지를 신경망에 공급하기 전에 입력 이미지를 1D로 병합해야 했습니다. 이제는 각 레이어가 2D로 표현되므로 뉴런과 해당 입력을 쉽게 비교할 수 있습니다.]

주어진 레이어의 (i,j)에 위치한 뉴런은 (i + fh-1, j + fw-1)에 위치한 이전 레이어의 뉴런의 출력에 연결됩니다. 여기서 fh 및 fw는 수용 필드의 높이와 너비 (그림 13-3 참조).

레이어가 이전 레이어와 동일한 높이와 너비를 가지려면 다이어그램에 표시된 것처럼 입력 주위에 0을 추가하는 것이 일반적입니다. 이를 제로 패딩이라고합니다.

Zero padding 
Figure 13-3. Connections between layers and zero 
padding 

그림 13-4와 같이 수용 필드를 간격을 띄워서 큰 입력 레이어를 더 작은 레이어에 연결할 수도 있습니다. 두 개의 연속적인 수용 필드 사이의 거리를 stride이라고 합니다. 다이어그램에서 3 × 3 수용 필드와 2의 stride를 사용하여 5 × 7 입력 레이어 (제로 패딩을 추가)가 3 × 4 레이어에 연결됩니다.

상위 레이어의 i 행, j 열에 위치한 뉴런은 i × sh에서 i × sh + fh - 1까지의 행에있는 이전 레이어 뉴런의 출력에 연결되며, 여기서 sh 및 sw은 수직 및 수평 stirde입니다.

Figure 13—4. Reducing dimensionality using astride 
of 2 

**2.1 Filters**

뉴런의 가중치는 수용 영역의 크기와 같은 작은 이미지로 표현될 수 있습니다.

예를 들어, 그림 13-5는 필터(또는 컨볼 루션 커널)라고하는 두 가지 가능한 가중치 세트를 보여줍니다.

Figure 13-5. Applying two differentfilters to get two 
feature maps 

In 
In [5]: 
fmap[ : , 
fmap[3, . 
1 
fmap[ : , : 
o, 
pit. show( ) 
plot _ image ( fmap[ : , : 
o, 
plt. show( ) 
from sklearn. datasets import load_sample_image 
china = load _ china. jpg" ) 
flower = load _ sample _ image( " flower . jpg" ) 
Image = 130:250] 
height, width, channels = 
image. shape 
image grayscale 
image . mean ( axis=2 ) . astype ( np. float32 ) 
images = 
image_grayscale. reshape( 1, height, width, 1) 
fmap = np.zeros(shape=(7, 7, 1, 2), dtype—np. float32) 
1]) 

첫 번째 것은 중간에 수직 흰색 선이있는 검정색 사각형으로 표시됩니다 (1을 나타내는 흰색의 중앙 열을 제외하고는 0으로 가득 찬 7x7 행렬입니다). 이 가중치를 사용하는 뉴런은 중앙 수직선을 제외하고 수용 영역에서 모든 것을 무시합니다 (모든 입력은 중앙 수직선에있는 것을 제외하고 0으로 곱해집니다).

두 번째 필터는 중간에 가로줄이 흰색으로 표시된 검은 색 사각형입니다. 이 가중치를 사용하는 뉴런은 중앙 수평선을 제외하고는 수용 영역의 모든 것을 무시합니다.

In [6] : 
In 
In [8] : 
reset _ graph ( ) 
X = tf .placeholder(tf .f10at32, shape=(None, 
feature_maps = tf. constant ( fmap) 
convolution = tf .nn. conv2d(X, feature maps, 
ME" ) 
with tf.Session() as sess: 
output = convolution. ( 
height, width, 1) ) 
1,1], padding="SA 
images ) 
plot_image ( images [ O , . 
0]) 
save fig( "china _ original" , tight layout=Fa1se) 
plt. show( ) 
Saving figure china_original 

In [9] : 
[10] : 
plot_image ( output [ O , : 
0]) 
save fig( 'I china vertical" , tight layout=Fa1se) 
plt. show( ) 
Saving figure china _ vertical 
plot_image ( output [ O , : 
1]) 
save fig( " china horizontal " , tight layout=Fa1se) 
plt. show( ) 
Saving figure china _ horizontal 

**2.2 Stacking Multiple Feature Maps**

지금까지 간단하게 각 컨볼루션 레이어를 얇은 2D 레이어로 나타내었지만 실제로는 크기가 동일한 몇 개의 feature맵으로 구성되어보다 정확하게 3D로 표현됩니다 (그림 13-6 참조).

하나의 feature맵 내에서 모든 뉴런은 동일한 매개 변수를 공유하지만, 다른 피쳐 맵은 다른 매개 변수를 가질 수 있습니다.

뉴런의 수용 필드는 앞에서 설명한 것과 동일하지만 모든 이전 레이어의 feature 맵 전체로 확장됩니다. 간단히 말해서, convolutional layer는 입력에 여러 필터를 동시에 적용하여 입력의 여러 feature를 감지 할 수 있습니다.

Filters 
Feature 
Chan neb 
Con VOI al 
layer 2 
Convol utional 
layer I 
Input layer 
Figure 13-6. Convolution layers with multiplefeature 
maps, and images with three channels 

또한 입력 이미지는 여러 개의 하위 레이어로 구성됩니다. 일반적으로 빨강, 녹색 및 파랑 (RGB)의 세 가지가 있습니다. 그레이 스케일 이미지에는 채널이 하나 뿐이지만 일부 이미지에는 더 많은 빛이 포함될 수 있습니다 (예 : 적외선과 같은 추가 빛 주파수를 캡처하는 위성 이미지).

[Note]

[피쳐 맵의 모든 뉴런이 동일한 매개 변수를 공유한다는 사실은 모델의 매개 변수 수를 극적으로 줄이지만 가장 중요한 것은 CNN이 한 위치의 패턴을 인식하면 다른 위치에서도 패턴을 인식할 수 있다는 것입니다 . 반대로 일반 DNN이 한 위치의 패턴을 인식하면 해당 위치에서만 패턴을 인식 할 수 있습니다.]

같은 행 i와 열 j에 있지만 서로 다른 feature맵에 있는 모든 뉴런은 이전 레이어의 동일한 뉴런의 출력에 연결됩니다.

방정식 13-1은 위의 설명을 하나의 큰 수학 방정식으로 요약 한 것입니다. 컨볼루션 레이어에서 주어진 뉴런의 출력을 계산하는 방법을 보여줍니다. 모든 다른 지표로 인해 약간 복잡하지만 모든 입력의 가중치 합계와 바이어스 기간을 계산합니다.

Equation 13-1. Computing the output of a neuron in 
a convolutional layer 
i' = i X sh+u 

* zi, j, k는 컨벌루션 레이어 (레이어 l)의 특징 맵 k에서 행 i, 열 j에 위치한 뉴런의 출력
* bk는 (레이어 l에서) 피쳐 맵 k에 대한 바이어스 항
* fh와 fw는 수용영역의 높이와 너비이고 fn '은 이전 레이어 (레이어 l - 1)의 feature맵의 수
* x\_i ', j', k '는 레이어 l - 1, 행 i', 열 j ', feature 맵 k'에 있는 뉴런의 출력
* w\_u,v,k',k 입력 및 feature 맵 k '간의 연결 가중치입니다.
* sh와 sw는 수직 및 수평 stride

**2.3 TensorFlow Implementation**

TensorFlow에서 각 입력 이미지는 일반적으로 3D 텐서 형태 [높이, 너비, 채널]로 표시됩니다.

Mini-batch는 4D 텐서 형태 [미니 배치 크기, 높이, 너비, 채널]로 표현됩니다.

컨볼루션 레이어의 가중치는 4D 텐서 형태 [fh, fw, fn ', fn]로 표현됩니다.

컨볼루션 레이어의 바이어스 항은 단순히 1D 텐서 형태 [fn]로 표현됩니다.

간단한 예를 살펴 보겠습니다.

* 다음 코드는 Scikit-Learn의 load\_sample\_images ()(사원 중 하나와 꽃의 이미지를 로드)를 사용하여 두 개의 샘플 이미지를 로드합니다.
* 그런 다음 두 개의 7 × 7 필터를 만듭니다 (하나는 가운데에 세로 흰색 선이 있고 다른 하나는 가운데에 가로 흰색 선이 있음).
* TensorFlow의 tf.nn.conv2d() 함수(제로 패딩과 2의 stride)를 사용하여 컨벌루션 레이어를 사용하여 두 이미지에 적용합니다.
* 마지막으로, 결과 feature맵 중 하나를 플롯합니다.

In [11]: 
import numpy as np 
from datasets import 
# Load sample images 
china = china. jpg" ) 
flower = load _ sample _ image( " flower. jpg" ) 
dataset = np.array( [china, flower], dtype=np. float32) 
batch _ size, height, width, channels = dataset. shape 
# Create 2 filters 
filters = np. zeros ( , 7, channels, 2), dtype=np. float32) 
filters[:, 3, : , O] 
= 1 # vertical line 
filters [3, 
— 1 horizontal line 
# Create a graph with input X plus a convolutional layer applying the 2 fi 
1 ters 
X = tf .placeholder(tf. float32, shape= (None, height, width, channels) ) 
convolution = tf .nn.conv2d(X, filters, ) 
with tf.session() as sess: 
output = run (convolution, dataset } ) 
PI t. imshow( output [ O , : 
map 
plt. show( ) 
1], ) # plot 1st image 2nd feature 
25 
50 
75 
100 
125 
150 
175 
o 
50 
100 
150 
200 
250 
300 

이 코드의 대부분은 자명하지만 tf.nn.conv2d () 행은 약간의 설명이 필요합니다.

* X는 입력 미니 배치 (앞에서 설명한 4D 텐서)입니다.
* filters는 적용할 필터 집합입니다 (앞서 설명한 것처럼 4D 텐서도 포함).
* strides는 4 요소 1D 배열이며 두 개의 중심 요소는 수직 및 수평 스트라이드 (sh 및 sw)입니다. 첫 번째 요소와 마지막 요소는 현재 1과 같아야 합니다. batch stride(일부 인스턴스는 건너 뛰기) 및 channel stride (이전 계층의 feature맵 또는 채널 중 일부를 건너 뛰기)을 지정하는데 언젠가 사용할 수 있습니다.
* 패딩은 "VALID"또는 "SAME"중 하나여야 합니다.  
  "VALID"로 설정된 경우, 컨볼루션 레이어는 제로 패딩을 사용하지 않고 그림 13-7에서와 같이 보폭에 따라 입력 이미지의 하단과 오른쪽에있는 일부 행과 열을 무시할 수 있습니다.  
  "SAME"으로 설정된 경우, 필요하면 컨벌루션 레이어는 제로 패딩을 사용합니다. 이 경우, 출력 뉴런 수는 입력 뉴런의 수를 올림으로 나눈 값 (이 예제에서는 ceil (13 / 5) = 3)과 같습니다. 그런 다음 0을 가능한한 고르게 입력 주위에 추가합니다.

n VAL ID 
(i.e., Without padding) 
Ignored 
(i.e„ With zero padding) 
Figure 13-7. paddingoptions — input width: 13, 
filter width: 6, stride: 5 

이 간단한 예에서는 수동으로 필터를 만들었지만 실제 CNN에서는 교육 알고리즘이 최상의 필터를 자동으로 발견하도록 합니다.

TensorFlow는 tf.layers.conv2d () 함수는 (kernel이라는) 필터 변수를 생성하고 무작위로 초기화합니다. 바이어스 변수 (bias라고도 함)를 만들고 0으로 초기화합니다. 예를 들어, 다음 코드는 2x2 스트라이드 (이 함수는 수직 스트라이드와 수평 스트라이드 만 예상 함)와 "SAME"패딩을 사용하여 2 개의 7x7 feature맵이있는 컨볼루션 레이어 다음에 입력 placeholder를 만듭니다.

In [13] : 
In [14]: 
In [15]: 
reset _ graph( ) 
X = tf .placeholder( height, width, channels), dtype=tf. float32 
conv = . layers.conv2d(X, filters=2, kernel_size=7, 
padding=" SAME " ) 
init = tf. global _ ) 
with tf.Session() as sess: 
init . run( ) 
output = sess run(conv, dataset } ) 
plt. : 
map 
plt. show( ) 
1], cmap= "gray") plot 1st image 's 2nd feature 
25 
50 
75 
125 
150 
175 
o 
50 
100 
150 
200 
250 300 

컨볼루션 레이어에는 몇 가지 하이퍼 매개 변수가 있습니다. 필터 수, 높이와 너비, 스트라이드 및 패딩 유형을 선택해야 합니다. 항상 그렇듯이 교차 유효성 검사를 사용하여 올바른 하이퍼 매개 변수 값을 찾을 수는 있지만 시간이 많이 걸립니다.

하이퍼 파라미터 값이 실제로 가장 잘 작동하는 지에 대한 아이디어를 얻기 위해 일반적인 CNN 구조에 대해 나중에 논의 할 것입니다.

**2.4 Memory Requirements**

CNN의 또 다른 문제점은 역전파의 역방향 패스가 순방향 패스 동안 계산된 모든 intermediate 값을 필요로하기 때문에 컨볼루션 레이어에 특히 트레이닝 중에 방대한 양의 RAM이 필요하다는 것입니다.

예를 들어 보폭 1 및 SAME 패딩을 사용하여 크기가 150 x 100 인 200 개의 기능 맵을 출력하는 5x5 필터가있는 컨볼루션 레이어를 고려해보십시오.

계산과정 생략하면 약 11.4 MB의 RAM을 차지합니다. 교육 배치에 인스턴스가 100 개 포함되어있는 경우 이 레이어는 1GB 이상의 RAM을 사용합니다.

새로운 인스턴스에 대한 예측을 할 때 한 레이어가 차지하는 RAM은 다음 레이어가 계산 되 자마자 해제 될 수 있으므로 두 개의 연속 된 레이어에 필요한 만큼의 RAM 만 필요합니다. 그러나 훈련 중에는 정방향 패스 중에 계산된 모든 것이 역 통과를 위해 보존되어야 하므로 필요한 RAM의 양은 적어도 모든 계층에서 필요한 총 RAM 용량 입니다.

[Note]

[메모리 부족 오류로 인해 교육에 장애가 발생하면 미니 배치 크기를 줄여보십시오. 또는 스트라이드를 사용하여 치수를 줄이거 나 몇 개의 레이어를 제거해보십시오. 또는 32 비트 부동 소수점 대신 16 비트 부동 소수점을 사용할 수도 있습니다. 또는 여러 장치에 CNN을 배포 할 수 있습니다.]

이제 CNN의 두 번째 공통 빌딩 블록 인 풀링 레이어를 살펴 보겠습니다.

1. **Pooling Layer**

컨벌루션 레이어가 작동하는 방식을 이해하면 풀링 레이어를 이해하기 쉽습니다.

이들의 목표는 계산 부하, 메모리 사용 및 파라미터의 수를 감소시키기 위해 입력 이미지를 서브 샘플링 (축소)함으로써 오버 피팅의 위험을 제한하는 것이다.

컨볼루션 레이어와 마찬가지로 풀링 레이어의 각 뉴런은 작은 사각형 수용 필드 내에 있는 이전 레이어의 제한된 수의 뉴런 출력에 연결됩니다. 이전과 마찬가지로 크기, 보폭 및 패딩 유형을 정의해야 합니다. 그러나 풀링 뉴런에는 가중치가 없습니다. 모든 작업은 최대 또는 평균과 같은 집계 함수를 사용하여 입력을 집계합니다.

그림 13-8은 풀링 레이어의 가장 일반적인 유형인 최대 풀링 레이어을 보여줍니다. 이 예에서는 2 × 2 풀링 커널, 2의 stride 및 패딩 없이 사용합니다. 각 커널의 최대 입력값만이 다음 레이어로 갑니다. 다른 입력은 삭제됩니다.

Figure 13-8. Max pooling layer (2 x 2 poolingkernel, 
stride 2, no padding) 

이것은 분명히 매우 파괴적인 종류의 레이어입니다. 작은 2x2 커널과 2의 stirde에서도 출력은 양방향으로 2 배 더 작습니다 (따라서 영역은 4 배 작아집니다).

풀링 레이어는 일반적으로 모든 입력 채널에서 독립적으로 작동하므로 출력 깊이는 입력 깊이와 동일합니다. 이미지의 공간 차수 (높이와 너비)는 변경되지 않지만 채널수가 줄어들 경우 다음에 보게 되듯이 깊이 차수를 넘을 수 있습니다.

TensorFlow에서 최대 풀링 레이어를 구현하는 것은 매우 쉽습니다. 다음 코드는 2x2 커널, 보폭 2 및 패딩 없이 사용하여 최대 풀링 레이어를 만든 다음 데이터 세트의 모든 이미지에 적용합니다.

In [18]: 
In [19]: 
batch size, height, width, channels = dataset. shape 
filters = np. zeros ( 7, channels, 2), dtype=np. float32 ) 
filters [ : , 
filters[3, • 
O = 1 # vertical line 
— 1 # horizontal line 
X = tf .placeholder(tf. float32, shape=(None, height, width, channels) ) 
max_pool = tf , 
ALID" ) 
with tf.Session() as sess: 
sess run (max_pool, feed_dict={X: dataset} ) 
output = 
plt. . astype(np.uint8) ) # plot the output for the 1st imag 
e 
plt. show( ) 
25 
50 
75 
125 
150 
175 
o 
50 
100 
150 
200 
250 
300 

ksize 는 입력 텐서의 4 차원 모두를 따르는 커널 모양을 포함합니다 : [배치 크기, 높이, 너비, 채널].

TensorFlow는 현재 여러 인스턴스에서 풀링을 지원하지 않으므로 ksize의 첫 번째 요소는 1이어야 합니다. 또한 공간 차원 (높이와 너비)과 깊이 차원 모두에서 풀링을 지원하지 않으므로 ksize [1] ksize [2]는 모두 1과 같아야하며, 그렇지 않으면 ksize [3]은 1과 같아야 합니다.

평균 풀링 레이어를 만들려면 max\_pool () 대신 avg\_pool () 함수를 사용하면 됩니다. 이제 모든 구성 요소를 알고 길쌈 신경 네트워크를 만듭니다. 그들을 조립하는 방법을 보자.

1. **CNN Architectures**

일반적인 CNN 구조는 몇 개의 컨볼루션 레이어 (일반적으로 각각 하나는 ReLU 레이어가 뒤 따른다), 풀링 레이어, 그리고 또 다른 몇 개의 컨볼루션 레이어 (+ ReLU), 다른 풀링 레이어 등을 쌓아 놓습니다.

이미지는 네트워크를 통해 진행됨에 따라 작아지지만 컨볼루션 레이어 (그림 13-9 참조) 덕분에 일반적으로 더 깊어집니다 (즉, 더 많은 feature맵이 있음). 스택의 상단에는 몇 개의 완전히 연결된 레이어 (+ ReLUs)로 구성된 정기적인 피드 포워드 신경망이 추가되고 최종 레이어는 예측 (예 : 예상되는 클래스 확률을 산출하는 softmax 레이어)을 출력합니다.

Pod ng RAY 
Figure 13-9. Typical CNN architecture 

[Tip]

[일반적인 실수는 너무 큰 컨볼루션 커널을 사용하는 것입니다. 3x3 커널로 두 개의 레이어를 쌓아 9x9 커널로 하나의 콘볼루션 레이어와 동일한 효과를 얻을 수 있습니다. 계산 및 매개 변수가 훨씬 적습니다.]

수 년에 걸쳐 이 기본 아키텍처의 변형이 개발되어 이 분야에서 놀라운 발전을 이루었습니다. 이러한 발전의 좋은 척도는 국제영상인식대회 ILSVRC ImageNet 챌린지와 같은 경기에서의 오류율입니다. 이 경쟁에서 이미지 분류에 대한 상위 5 가지 오류율은 불과 6 년 만에 26 % 이상에서 3 % 미만으로 떨어졌습니다. 이러한 진화를 보는 것은 CNN이 어떻게 작동하는지 이해하는 좋은 방법입니다.

먼저 AlexNet (2012), GoogLeNet (2014) 및 ResNet (2015)과 같은 ILSVRC 도전 과제의 수상자 중 3인인 고전적인 LeNet-5 아키텍처 (1998)를 살펴 보겠습니다.

**4.1 LeNet-5**

LeNet-5 아키텍처는 아마도 가장 널리 알려진 CNN 구조일 것입니다.

앞서 언급했듯이, 1998 년 Yann LeCun에 의해 만들어졌으며 손으로 쓰는 숫자 인식 (MNIST)에 널리 사용되었습니다. 표 13-1에 표시된 계층으로 구성됩니다.

LeNet-5 architecture 
Layer Type 
Table 13-1. 
Maps 
Size 
10 
84 
5 
10 x 
14 x 
28 
Kernel size Stride Activation 
Out 
S4 
In 
Fully Connected — 
Fully Connected — 
Convolution 
120 
Avg Pooling 
16 
Convolution 
16 
Avg Pooling 
6 
Convolution 
6 
Input 
1 
10 
14 
28 
5 
2 
5 
2 
5 
2 
1 
2 
1 
RBF 
tan h 
tan h 
tan h 
tan h 
tan h 
tan h 
32 32 

주목할 몇 가지 추가 세부 정보가 있습니다.

* MNIST 이미지는 28 × 28 픽셀이지만 32 × 32 픽셀로 제로 패딩되고 네트워크에 공급되기 전에 정규화됩니다. 나머지 네트워크는 패딩을 사용하지 않으므로 이미지가 네트워크를 통해 진행됨에 따라 크기가 계속 축소됩니다.
* 평균 풀링 레이어는 평소보다 약간 더 복잡합니다. 각 뉴런은 입력의 평균을 계산한 다음 결과를 학습 가능한 계수(맵 당 하나)로 곱하고 학습할 수 있는 바이어스 항 (맵 당 하나)을 추가 한 다음 활성화함수에 적용합니다.
* C3 맵에서 대부분의 뉴런은 3 개의 또는 4 개의 S2 맵 (모든 6 개의 S2 맵 대신)에서 뉴런과 연결됩니다.
* 출력 레이어는 입력과 가중치 벡터의 내적을 계산하는 대신 각 뉴런은 입력 벡터와 가중치 벡터 사이의 유클리드 거리의 제곱을 출력합니다. 각 출력은 이미지가 특정 숫자 클래스에 속하는 정도를 측정합니다. 교차 엔트로피 비용 함수는 이제 나쁜 예측에 더 많은 불이익을 주며 더 큰 그라디언트를 생성하여 더 빠르게 수렴하므로 바람직합니다.

Yann LeCun의 웹 사이트 ( "LENET"섹션)에는 LeNet-5 숫자 분류 데모가 있습니다.

**4.2 AlexNet**

AlexNet CNN 구조는 2012 년 ImageNet ILSVRC 도전에서 큰 차이를 보았습니다.

17 %의 5 단계 오류율을 달성했지만 2 위는 26 %에 불과했습니다

그것은 LeNet-5와 매우 유사하고 더 크고 더 깊다. 각각의 컨볼루션 레이어 위에 풀링 레이어를 쌓는 대신에 서로의 위에 직접 컨볼루션 레이어를 쌓는 것이 처음이다.

표 13-2는이 아키텍처를 나타냅니다.

Table 13-2. AlexNet architecture 
Layer Type 
Maps 
Size 
1 ,ooo 
4,096 
4,096 
Kernel size Stride Padding Activation 
Out 
F9 
S4 
S2 
In 
Fully Connected - 
Fully Connected 
Fully Connected — 
Convolution 
Convolution 
Convolution 
Max Pooling 
Convolution 
Max Pooling 
Convolution 
Input 
256 
384 
384 
256 
256 
96 
96 
3 (RGB) 
13 
13 
13 
55 
27 x 27 
x 13 
x 13 
x 13 
x 13 
x 55 
3 
3 
3 
3 
5 
3 
3 
5 
3 
lixil 
X 
X 
1 
1 
2 
1 
2 
4 
SAME 
SAME 
SAME 
VALID 
SAME 
VALID 
SAME 
Softmax 
ReLU 
ReLU 
ReLU 
ReLU 
ReLU 
ReLU 
ReLU 
224 x 224 - 

오버 피팅을 줄이기 위해 AlexNet 개발자는 이전 장에서 설명한 두 가지 정규화 기법을 사용했습니다.

* 먼저 트레이닝 도중 F8 및 F9 레이어의 출력에 dropout(50 % 드롭 아웃 비율 사용)을 적용했습니다.
* 둘째, 다양한 오프셋으로 트레이닝 이미지를 무작위로 이동하고 가로로 뒤집어 조명 조건을 변경하여 데이터 확대를 수행했습니다.

AlexNet은 또한 local response nomalization라고 하는 레이어 C1 및 C3의 ReLU 단계 직후 경쟁 표준화 단계를 사용합니다. 이러한 형태의 정규화는 동일한 위치에서 뉴런을 가장 강력하게 활성화시키는 뉴런을 만든다(이러한 경쟁적인 활성화는 생물학적 뉴런에서 관찰되었다). 이렇게 하면 다양한 feature맵을 특수화하고, 이를 분리하여 더 광범위한 기능을 탐색하여 궁극적으로 일반화를 향상시킬 수 있습니다.

식 13-2는 LRN을 적용하는 방법을 보여줍니다.

Equation 13-2. Local response normalization 
jhigh = min (i + — l) 
bi = aÅk + a aj ) 
with 
jlow = max (0, i — f) 

* bi는 일부 행 u 및 열 v에서 피쳐 맵 i에있는 뉴런의 정규화된 출력입니다
* ai는 ReLU 단계 후에 정규화되기 전의 해당 뉴런의 활성화입니다.
* k, α, β 및 r은 하이퍼 파라미터이다. k를 bias r을 depth radius
* fn은 기능 맵의 수입니다.

예를 들어, r = 2이고 뉴런이 강한 활성화를 갖는 경우, 그것은 자신의 위와 아래의 feature맵에있는 뉴런의 활성화를 억제합니다.

AlexNet에서 하이퍼 매개 변수는 다음과 같이 설정됩니다.

r = 2, α = 0.00002, β = 0.75 및 k = 1

이 단계는 TensorFlow의 tf.nn.local\_response\_normalization ()함수를 사용하여 구현할 수 있다.

AlexNet의 변종인 ZF Net은 Matthew Zeiler와 Rob Fergus에 의해 개발되었으며 2013 ILSVRC 챌린지에서 우승했습니다. 몇 가지 조정된 하이퍼 매개 변수 (기능 맵 수, 커널 크기, 스트라이드 등)가있는 것은 본질적으로 AlexNet입니다.

**4.3 GoogLeNet**

GoogLeNet 구조는 Christian Szegedy 등이 개발했습니다. Google Research에서 10 점을 받았으며 상위 5 개 오류율을 7 % 미만으로 낮추어 ILSVRC 2014 도전 과제를 수상했습니다. 이 놀라운 성과는 네트워크가 이전의 CNN보다 훨씬 더 깊었기 때문에 큰 부분을 차지했습니다

이는 GoogLeNet이 이전 구조보다 훨씬 효율적으로 매개 변수를 사용할 수 있도록 하는 시작 모듈이라는 하위 네트워크로 가능합니다. GoogLeNet은 실제로 AlexNet보다 10 배 적은 매개 변수를 사용합니다 (약 6 천만 개가 아닌 6 백만 개).

Inception 
module 
Convolution 
IRI •IS 
Depth 
Cmvolution 
30 + 1 S 
Convolution 
Convolution 
Convolution 
Convolution 
IRI + -IS 
Max 
3x3•1 S 
Figure 13-10. Inception module 

그림 13-10은 시작 모듈의 구조를 보여줍니다.

표기법 "3 × 3 + 2 (S)"는 레이어가 3 × 3 커널, 스트라이드 2 및 SAME 패딩을 사용한다는 것

* 입력 신호는 먼저 복사되어 네 개의 다른 레이어로 공급됩니다. 모든 컨볼루션 계층은 ReLU 활성화 기능을 사용합니다.
* 두 번째 컨볼 루션 계층 집합은 서로 다른 커널 크기 (1x1, 3x3 및 5x5)를 사용하므로 서로 다른 크기로 패턴을 캡처 할 수 있습니다.
* 모든 단일 레이어는 1의 스트라이드와 SAME 패딩 (최대 풀링 레이어조차도)을 사용하므로 출력의 높이와 폭이 모두 입력과 ​​동일합니다. 이것은 최종 깊이 연결 층에서 깊이 치수를 따라 모든 출력을 연결하는 것을 가능하게한다
* 이 연결 레이어는 tf.concat () 작업을 사용하여 TensorFlow에서 축 = 3 (축 3은 깊이)으로 구현할 수 있습니다.

왜 시작 모듈이 1x1 커널을 가진 컨벌루션 레이어를 가지고 있는지 궁금할 것입니다.

실제로 이러한 계층은 두 가지 용도로 사용됩니다.

* 첫째, 입력보다 훨씬 적은 수의 feature맵을 출력하도록 구성되므로 병목 현상이 발생하여 차원성이 저하됩니다. 이는 3 × 3 및 5 × 5 컨볼루션 이전에 특히 유용합니다. 이는 이들이 매우 계산적으로 값 비싼 계층이기 때문입니다.
* 둘째, 컨볼루션 레이어 ([1 × 1, 3 × 3] 및 [1 × 1, 5 × 5])의 각 쌍은 보다 복잡한 패턴을 캡처 할 수 있는 단일의 강력한 컨볼루션 레이어와 같은 역할을 합니다. 사실 이미지 전체에 간단한 선형 분류기를 sweep하는 대신 (단일 컨볼 루션 레이어처럼) 이 컨볼루션 레이어 쌍은 이미지 전체에 걸쳐 2-레이어 신경망을 sweep합니다.

즉, 전체 시작 모듈을 스테로이드의 컨볼루션 레이어로 생각할 수 있습니다. 다양한 스케일로 복잡한 패턴을 캡처하는 feature맵을 출력할 수 있습니다.

이제 GoogLeNet CNN의 구조를 살펴 보겠습니다 (그림 13-11 참조).

너무 깊기 때문에 3 열로 표현해야했지만 GoogLeNet은 실제로는 각각 3 개의 레이어가 포함 된 9 개의 시작 모듈 (회전식 상단이있는 상자)을 포함하여 하나의 큰 스택입니다. 각 컨볼루션 레이어와 각 풀링 레이어에서 출력되는 feature맵의 수는 커널 크기 앞에 표시됩니다.

Inception 모듈의 여섯 개의 숫자는 모듈의 각 컨볼루션 레이어에 의해 출력되는 feature맵의 수를 나타냅니다 (그림 13-10에서와 같은 순서로). 모든 컨볼 루션 계층은 ReLU 활성화 기능을 사용합니다.

이 네트워크를 살펴 보겠습니다.

Max pool 
192. s 
144 32 
Local Response 
Norm 
128 24 
Convolution 
192, 3x3•1S 
112 24 
Convolution 
48 64 
96 16 
Local Response 
Norm 
64, + 2(S) 
Convolution 
64, m +2 S 
Input 
Figure 13-11. 
112 
12B 
160 
192 
288 
256 
224 
64 
64 
64 
64 
64 
64 
s 
Max Pool 
480, 
128 192 96 64 
12B 32 
64 128 32 32 
96 16 
Softmax 
Fully Connected 
1000 units 
Dropout 
Avg pool 
1024. 7x7*1 
384384 128 128 
192 48 
256 320 128 128 
160 32 
Max Pool 
832, •2 
56 320 128 128 
160 32 
inception module 
GoogLeNet architecture 

* 처음 두 레이어는 이미지의 높이와 너비를 4로 나눕니다 (따라서 영역을 16으로 나눕니다). 이렇게하면 계산량이 줄어 듭니다.
* 그런 다음 로컬 응답 정규화 레이어는 이전 레이어가 다양한 feature를 학습하도록 합니다.
* 두 개의 컨벌루션 레이어가 이어지며, 첫 번째 레이어는 bottleneck 레이어와 유사합니다. 설명한 것처럼 이 쌍을 하나의 똑똑한 컨볼루션 레이어로 생각할 수 있습니다.
* 다시 말하면, 로컬 응답 정규화 레이어는 이전 레이어가 다양한 패턴을 포착하도록합니다.
* 다음 최대 풀링 레이어는 이미지 높이와 너비를 2 씩 줄여 다시 계산 속도를 높입니다.
* 그런 다음 9 개의 inception 모듈로 이루어진 스택이 나오고, 최대 풀링 레이어가 몇 개 삽입되어 치수를 줄이고 네트워크를 가속화합니다.
* 다음으로 평균 풀링 레이어는 "VALID"패딩이있는 feature맵 크기의 커널을 사용하여 1 × 1 기능 맵을 출력합니다. 이를 글로벌 평균 풀링이라고 합니다. 따라서 CNN 상단에 여러 개의 완전히 연결된 레이어 (예 : AlexNet)가 있을 필요가 없어 네트워크의 매개 변수 수가 크게 줄어들고 overfitting의 위험이 줄어듭니다.
* 마지막 계층은 정규화를 위한 dropout, 그리고 softmax 활성화 함수가 있는 완전히 연결된 계층으로 추정 클래스 확률을 출력합니다.

이 다이어그램은 약간 단순화 되었습니다. 원래 GoogLeNet 구조에는 세 번째 및 여섯 번째 시작 모듈 위에 두 개의 보조 분류자가 포함되었습니다. 그들은 모두 하나의 평균 풀링 레이어, 하나의 컨볼루션 레이어, 완전히 연결된 두 개의 레이어 및 softmax 활성화 레이어로 구성되었습니다.

훈련 중 손실 (70 % 축소)이 전반적인 손실에 추가되었습니다. 목표는 vanishing gradients 문제를 해결하고 네트워크를 표준화하는 것이었습니다. 그러나 그 영향은 상대적으로 미미한 것으로 나타났습니다.

**4.4 ResNet**

ILSVRC 2015 도전 과제의 우승자는 Kaiming He 외 12 명이 개발한 Residual Network입니다.

이는 3.6 % 미만의 놀라운 top5 오류율을 제공했으며, 152 레이어로 구성된 깊은 네트워크를 사용하였다. 이러한 심층망을 훈련 할 수있는 핵심은 skip connection(바로 가기 연결)을 사용하는 것입니다. 레이어로 전달되는 신호는 스택 위로 조금 더 높은 위치에 있는 레이어의 출력에도 추가됩니다. 이것이 왜 유용한지 알아 보겠습니다.

신경망을 트레이닝할 때 목표는 모델을 목표 함수 h (x)로 만드는 것입니다.

네트워크의 출력에 입력 x를 추가하면 (skip connection을 추가하면) 네트워크는 h (x) 대신 f (x) = h (x) -x를 강제로 모델링하게됩니다. 이를 residual 학습이라고합니다.

h(x) 
Layer 2 
Layer I 
Input 
Layer 2 
Layer I 
Input 
Figure 13-12. Residual learning 

일반 신경망을 초기화하면 가중치가 0에 가까워 지므로 네트워크가 0에 가까운 값을 출력합니다. 스킵 연결을 추가하면 결과 네트워크는 입력의 사본을 출력합니다. 즉, 초기에 identitiy 함수를 모델링합니다. 대상 함수가 (종종 경우에 따라) identitiy 함수에 상당히 가깝다면, 이것은 트레이닝을 상당히 가속화할 것입니다.

또한 여러 개의 스킵 연결을 추가하면 여러 레이어가 아직 학습을 시작하지 않아도 네트워크가 진행될 수 있습니다 (그림 13-13 참조). 연결을 건너 뛰기 때문에 신호가 전체 네트워크에 쉽게 전달 될 수 있습니다. 깊은 residual 네트워크는 residual units의 스택으로 볼 수 있습니다. 여기서 각 잔여 유닛는 스킵 연결이 있는 작은 신경 네트워크입니다.

x 
x 
x 
Layer close to 
its initial State 
Resdual 
layers that cm•tput close 
X = to zero and block 
backpropagaticy-l 
Figure 13-13. Regular deep neural network (left) 
and deep residual network (right) 

이제 ResNet의 아키텍처를 살펴 보겠습니다 (그림 13-14 참조).

GoogLeNet과 완전히 동일하게 시작하고 끝나며 단순한 residual 단위의 스택입니다. 각각의 resdual 단위는 3 × 3 커널을 사용하고 공간 차원 (stride 1, SAME 패딩)을 유지하면서 일괄 정규화 (BN) 및 ReLU 활성화와 함께 2 개의 컨벌루션 레이어로 구성됩니다.

128, 3-3, 1 
units 
128, 3-3. 1 
Avg 
1024, 7-7 • 1 
128, 3-3. 1 
128, 3-3 • 
Deep ! 
64, 3-3+1 s 
64, 3-3, 
64. 
64. 3-3 •IS 
64, 30-1 
64.30+1 
Figure 13-14. ResNet architecture 

feature 맵의 수는 몇 개의 residual 유닛이 있을때마다 두 배가되며, 높이와 너비가 반으로 줄어들 때 (stride 2 인 컨볼루션 레이어를 사용) 동시에 증가합니다. 이 경우 입력은 동일한 모양을 가지지 않으므로 나머지 유닛의 출력에 직접 추가할 수 없습니다 (이 문제는 그림 13-14의 점선으로 표시된 연결 건너 뛰기에 영향을 미침).

이 문제를 해결하기 위해 입력은 stirde 2의 1x1 컨볼루션 레이어와 올바른 출력 feature맵 수를 통과합니다 (그림 13-15 참조).

Convolution 
128, 1 x 1 + S 
Convolution 
128, 30 + 
Convolution 
128, 3x3 +2 s 
Figure 13-15. Skip connection when changing 
feature map size and depth 

*[[ResNet-34는 64 개의 feature맵을 출력하는 3 개의 residual 유닛, (128 개의 맵이있는 4 개의 RU), (256 개의 맵이있는 6 개의 RU), (512 개의 맵이있는 3 개의 RU)를 포함하는 34 개의 레이어 (단지 컨볼 루션 레이어와 완전히 연결된 레이어만 계산)가 있는 ResNet입니다.*

*ResNet-152와 같이 더 깊은 ResNets는 약간 다른 residual 단위를 사용합니다.*

*두 개의 3 × 3 컨볼루션 레이어 대신 256 개의 피쳐 맵을 사용하며, 3 개의 컨볼루션 레이어를 사용합니다.*

*먼저 bottleneck 레이어처럼 작동하는 64 개의 feature 맵 (4 배 적은)이 있는 1 × 1 컨볼루션 레이어가 있고 , 64 개의 feature 맵을 가진 3 x 3 레이어, 그리고 원래의 깊이를 복원하는 256 개의 feature 맵(64 x 4)을 가진 또 다른 1 x 1 컨볼루션 레이어가 있다.*

*ResNet-152에는 (256 개의 맵을 출력하는 3 개의 RU), (512 개의 맵이있는 8 개의 RU), (1,024 개의 맵이있는 36 개의 RU), 그리고 (2,048 개의 맵이있는 3 개의 RU)가 있습니다.]]*

보시다시피,이 분야는 매년 변하는 모든 종류의 아키텍처로 빠르게 움직이고 있습니다.

한 가지 확실한 추세는 CNN이 계속 더 깊고 깊어지고 있다는 것입니다. 그들은 또한 더 가볍고, 더 적은 수의 매개 변수를 필요로합니다. 현재 ResNet 구조는 가장 강력하고 가장 단순한 아키텍처이기 때문에 논란의 여지가 있지만 매년 사용해야 할 ILSVRC 과제를 계속 살펴보십시오. 2016 년 우승자는 중국의 Trimps-Soushen 팀으로 2.99 %의 놀라운 오류율을 기록했습니다. 이를 달성하기 위해 이전 모델의 조합을 훈련하고 앙상블에 합류했습니다.

특히 VGGNet13 (ILSVRC 2014 챌린지의 준우승)과 Inception-v414 (GoogLeNet 및 ResNet의 아이디어를 병합하고 3 %의 상위 5 개 오류에 근접)를 살펴볼 수있는 몇 가지 아키텍처가 있습니다. ImageNet 분류 기준).

방금 논의한 다양한 CNN 아키텍처를 구현하는 데 특별한 것이 없습니다. 이전에 모든 개별 빌딩 블록을 작성하는 방법을 보았으므로 원하는 아키텍처를 만들기 위해 어셈블링하는 것만으로도 충분합니다. 다가오는 연습에서 완전한 CNN을 만들 것이며 Jupyter 노트북에서 전체 작업 코드를 찾을 수 있습니다.