Machine Learning

Report 3

2017. 6. 15.

2011722086

한도영

1. **제목**

Convolutional Neural Network Finger Classification

1. **목적**

1부터 5까지의 수를 표현하는 손가락 사진을 데이터 셋으로 사용한다. 목표는 Convolutional Neural Network를 이용하여 이 다섯 가지의 손 사진을 구별하는 프로그램을 작성하는 것이다.

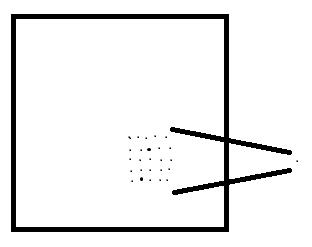
1. **원리 (배경지식)**

**3.1) Convolutional Neural Network**

이전 과제와 동일한 방법을 이용하므로 CNN의 설명을 복사하였다.

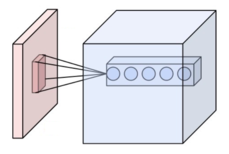
Machine Learning에서 CNN이란 동물의 시신경 구조에서 영감을 얻은 뉴런 사이의 관계 패턴에 관한 Feed-Forward 인공 지능의 한 형태라고 할 수 있다. 거의 항상 입력 데이터로 이미지를 받는다는 특징이 있으며 기존 방식에 비해 매개 변수의 수를 줄일 수 있다.

CNN의 주요 목적은 테두리, 선, 색 등 이미지의 시각적 특징이나 특성을 감지하는 것이다. 이는 입력 계층과 연결된 은닉 계층에서 처리된다. CNN에서는 입력 데이터 전체가 첫 번째 은닉 계층과 Fully-Connected되지는 않고, 입력 뉴런의 작은 일부 영역만이 첫 번째 은닉 계층의 한 뉴런과 연결된다. 그림으로 표현하면 다음과 같다.



은닉 계층의 각 뉴런은 입력 계층의 5x5 영역과 연결된다. 이런 식으로 전체 영역에 대해 뉴런을 연결한다. 이 방법을 수행하면 윈도우 크기가 다루지 못하는 영역이 잘리는 현상이 생기는데, 이를 막으려면 이미지의 양 옆을 0으로 채우는 제로 패딩 방법을 사용할 수도 있다.

입력 계층과 은닉 계층의 뉴런을 연결하기 위해서는 가중치 행렬 W와 바이어스 b가 필요하다. CNN의 핵심 특징은 W와 b를 은닉 계층의 모든 뉴런이 공유한다는 것이다. 이 공유 행렬 W와 b를 CNN에서는 **커널(kernel)** 또는 **필터(filter)**라고 한다. 이런 필터는 고유한 특징을 찾는 데 사용된다. 하나의 커널은 이미지에서 한 종류의 특징만을 감지한다.[[1]](#footnote-1) 그러므로 감지하고 싶은 각 특징에 한 개씩 여러 개의 커널을 사용하는 것이 좋다.



이렇게 첫 번째 은닉 계층은 여러 개의 특징 맵으로 구성된다. 이와 같이 특징 맵이 모여 있는 것을 **콘볼루션층(convolution layer)**이라고 한다. 그 다음엔 콘볼루션층의 출력을 단순하게 압축해주는 **풀링층(pooling layer)**이 있다. 이 예제에서는 2x2 Max Pooling 방식으로 풀링을 수행한다.

**3.2) CNN 성능 높이기**

CNN의 성능을 높이기 위해 생각할 수 있는 방법이 몇 가지 있다.

먼저 전처리에서는, 제공된 코드에서는 RGB dimension에서 컬러 값의 평균을 데이터로 사용한다. 이는 성능을 높이는 전처리로 보기는 어려우며, 성능을 높이기 위해서는 데이터에 Standardization을 수행할 수 있다. Standardization이란 말 그대로 데이터를 정규화하는 것이다. 따라서 원래 데이터를 정규화해줌으로써 원만한 등고선을 만들 수 있게 된다.

한편, 레이어의 층이 깊을수록 정교하게 만들 수 있다. 이때 필터 크기를 너무 크게 하면 층을 옮길수록 이미지의 크기가 빨리 줄어들게 되는데, 이미지의 크기가 줄어드는 것은 속도 관점에서는 성능을 높이는 것이지만 그만큼 특징을 잃어버리므로 정보를 잃는다. 즉 적당한 선을 찾지 않으면 오히려 정확도가 떨어지는 결과가 나올 수 있기 때문에, 때때로 작은 크기의 필터를 사용해야 할 수도 있다. 이 예제 또한 작은 크기의 필터를 사용한다. 또한, 시그모이드 함수보다 더 우월한 함수로 알려진 ReLU 함수를 사용함으로써 성능을 높일 것이다.

1. **방법**

**4.1) 데이터 수집**

스마트폰을 이용하여 동영상을 촬영한 후, 팟플레이어 프로그램을 이용하여 프레임을 수집한다. 이때 트레이닝 셋이 너무 왜곡이 없으면 트레이닝 및 테스트 시에만 정확한 데이터가 나오기 때문에, 손을 움직이거나 배경을 바꾸는 등 이미지에 왜곡을 주어 특징이 명확하게 나타나도록 추출해야 한다.

**4.2) 구현**

각 모듈의 구현을 중심으로 설명하겠다. 이 과제는 2차 과제의 CNN을 거의 그대로 가져온 것이기 때문에, 변한 부분을 위주로 설명하고자 한다. 중요한 점만 한 번 더 강조하자면, 이 과제에서 padding을 인자로 받아야 하는데 인자 목록에 padding이 없는 함수가 있었기 때문에 padding 변수를 전역으로 선언했다. padding에 따른 결과를 보려면 utils.py 파일의 맨 위에 있는 padding 변수의 값을 바꾸면 된다.

|  |
| --- |
| **def** \_CNNModel(img\_width, img\_height, kernel\_info):**global** padding  x = tf.placeholder(tf.float32, [**None**, img\_width \* img\_height])  *# 제 1 콘볼루션 레이어입니다.* k\_width = kernel\_info[0][0]  k\_height = kernel\_info[0][1]  k\_outlayer\_count = kernel\_info[0][2]  k\_pool = kernel\_info[0][3]   x\_img = tf.reshape(x, [-1, img\_width, img\_height, 1])  W\_conv\_beg = weight\_variable([k\_width, k\_height, 1, k\_outlayer\_count])  b\_conv\_beg = bias\_variable([k\_outlayer\_count])  h\_conv\_beg = tf.nn.relu(Conv2d(x\_img, W\_conv\_beg, padding=padding) + b\_conv\_beg)  **if** k\_pool **is True**:  h\_pool\_beg = max\_pool\_2x2(h\_conv\_beg, padding=padding)**else**:  h\_pool\_beg = h\_conv\_beg  W\_conv\_list = [W\_conv\_beg]  b\_conv\_list = [b\_conv\_beg]  h\_conv\_list = [h\_conv\_beg]  h\_pool\_list = [h\_pool\_beg]  k\_outlayer\_count\_prev = k\_outlayer\_count  h\_pool\_prev = h\_pool\_beg  h\_pool = h\_pool\_beg  ... |

CNN 모델 함수는 이전엔 2개의 층뿐이었지만, 이제는 임의의 N개의 출력 층을 포함할 수 있다. 이 정보를 보관하는 것이 바로 kernel\_info 파라미터인데, 여기에는 [커널 너비, 커널 높이, 커널 출력 수, 풀링 여부]로 이루어진 리스트의 리스트이다. 즉 넘길 때는 다음과 같이 넘긴다.

[[5, 5, 32, True], [3, 2, 64, True], [3, 2, 128, False]]

이에 따르면 제 1층은 5x5 커널 32개를 출력하고 결과는 풀링된다. 제 2층은 3x2 커널 64개를 출력하고 결과는 풀링된다. 제 3층은 3x2 커널 128개를 출력하고 결과는 풀링되지 않는다. 위에서 제시한 코드는 제 1층의 입력에 대한 처리이다.

|  |
| --- |
| ...  **for** i **in** range(1, len(kernel\_info)):  k\_width = kernel\_info[i][0]  k\_height = kernel\_info[i][1]  k\_outlayer\_count = kernel\_info[i][2]  k\_pool = kernel\_info[i][3]   W\_conv = weight\_variable([k\_width, k\_height, k\_outlayer\_count\_prev, k\_outlayer\_count])  b\_conv = bias\_variable([k\_outlayer\_count])  h\_conv = tf.nn.relu(Conv2d(h\_pool\_prev, W\_conv, padding=padding) + b\_conv)  **if** k\_pool **is True**:  h\_pool = max\_pool\_2x2(h\_conv, padding=padding)  *# h\_pool = tf.nn.dropout(h\_pool, keep\_prob=keep\_prob)* **else**:  h\_pool = h\_conv   k\_outlayer\_count\_prev = k\_outlayer\_count  h\_pool\_prev = h\_pool   W\_conv\_list.append(W\_conv)  b\_conv\_list.append(b\_conv)  h\_conv\_list.append(h\_conv)  h\_pool\_list.append(h\_pool)   *# 반환할 모델은 마지막 풀링층의 결과입니다.* model = h\_pool  print(**"%s ---> CNN Model was built"** %(model))  **return** model, x |

위는 제 1층 이후의 층에 대한 처리이다. 이것이 바뀐 것의 전부다. 이제 main 함수에 이전에 작성한 main\_app의 코드를 복사하면 된다.

|  |
| --- |
| img\_width = data\_shape[0] img\_height = data\_shape[1]  train\_count = 5000 test\_count = 490 TrainX = rawdata[:train\_count] *# mnist.train.images* TrainY = raw\_label[:train\_count] *# mnist.train.labels* testX = rawdata[train\_count:train\_count+test\_count] *# mnist.test.images* testY = raw\_label[train\_count:train\_count+test\_count] *# mnist.test.labels* |

이미지의 가로, 세로 길이와 train, test 셋의 수를 정한 후 데이터를 가져온다. 여기서는 트레이닝 셋의 수를 5000으로 잡았는데, 내가 테스트하는 경우에 이 값을 높여서 정확도를 높일 수 있었다.

|  |
| --- |
| CNNModel, x = ut.\_CNNModel(img\_width=img\_width, img\_height=img\_height,  kernel\_info=[  [3, 2, 32, **True**],  [3, 2, 64, **True**],  [3, 2, 128, **True**],  [3, 2, 64, **True**],  [3, 2, 128, **True**],  ]) |

CNN 모델의 파라미터가 바뀌었다. 인자를 보면 이것이 5개의 층으로 이루어진 CNN임을 짐작할 수 있다. 이전 MNIST 예제에서는 5x5 필터를 사용했지만, 이 예제에서는 3x2 크기의 작은 필터를 사용함으로써 성능을 높일 수 있었다.

참고로 여기서는 얼굴 인식에 대해서도 처리를 하고 있는데, 이를 확인하려면 다음 코드의 flag 변수의 값을 수정하면 된다.

|  |
| --- |
| *# flag이 0이면 finger, 1이면 face를 검사합니다.* flag = 1 **if** flag **is** 0:  class\_num = 5  additional\_path = **'\\finger\\' else**:  class\_num = 6  additional\_path = **'\\face\\'** |

변수 flag의 값이 0이면 손가락을, 1이면 얼굴을 검사한다. 기본 설정은 다음과 같다.

손가락 인식: train\_count=5000, 2 Convolutional Layers, 2 Pooling Layers

손가락 인식의 경우 train\_count를 2000 또는 5000에서, 레이어 수는 2층 또는 5층으로 테스트해보았다.

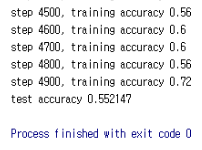
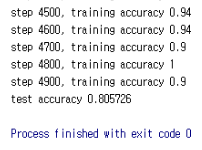
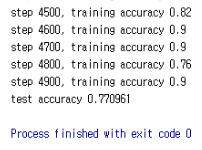
얼굴 인식: train\_count=5000, 2 Convolutional Layers, 2 Pooling Layers

얼굴 인식의 경우 train\_count를 2000에서, 레이어 수는 2층으로 테스트해보았다. 이외의 경우 train\_data의 수가 적거나 overfitting에 의해 accuracy가 NaN이 발생하는 경우가 있었다.

1. **결과**

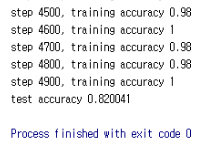
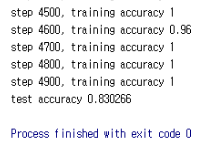
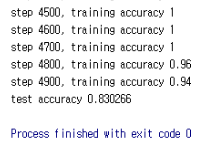
**5.1) Finger Classification**

5.1.1) #Train=2000, 2 Convolutional Layers, 2 Pooling Layers, 1 Softmax Layer, padding = 1



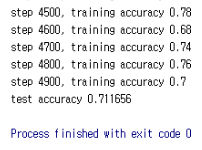
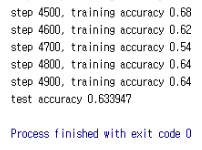
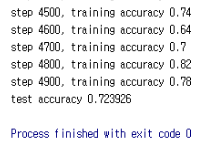
가장 낮은 경우 55%까지 내려갈 정도로 성능이 좋지 않았다.

5.1.2) #Train=2000, 5 Convolutional Layers, 5 Pooling Layers, 1 Softmax Layer, padding = 1



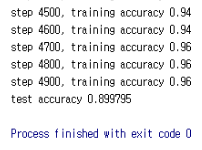
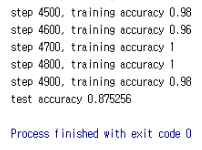
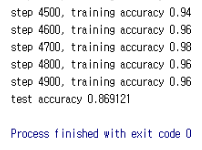
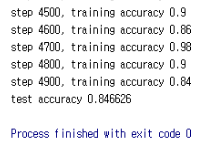
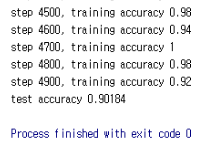
평균적으로 82~83%의 정확도를 보였다.

5.1.3) #Train=5000, 2 Convolutional Layers, 2 Pooling Layers, 1 Softmax Layer, padding = 1



Training 수를 늘렸지만 여전히 성능이 좋지 않았다. 65~70%의 성능을 보였다.

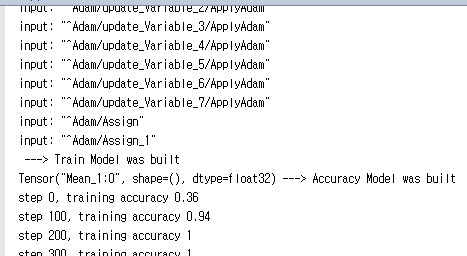
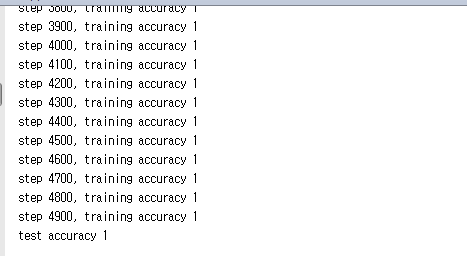
5.1.4) #Train=5000, 5 Convolutional Layers, 5 Pooling Layers, 1 Softmax Layer, padding = 1

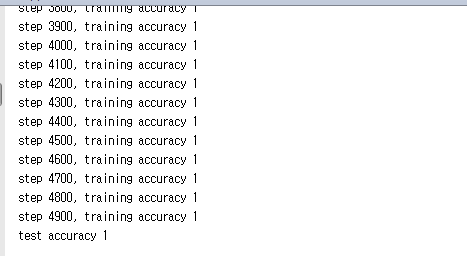
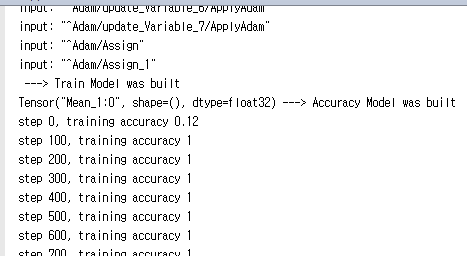


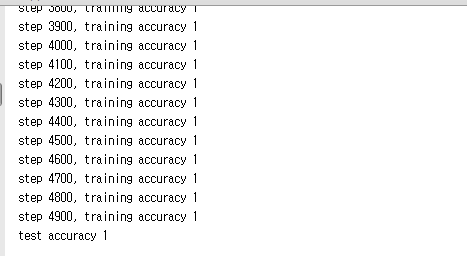
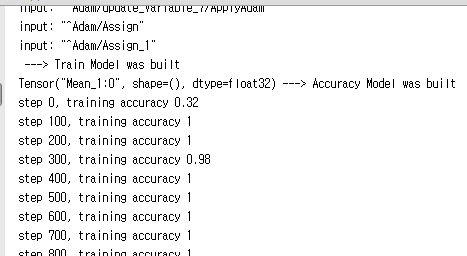
전체적으로 성능이 다른 예제에 비해 우월했다. 85~90% 정도로, 바로 이전 예제가 65~70%의 성능을 보였던 것에 비해 대단히 우수하다고 할 수 있다.

**5.2) Face Classification**

얼굴 인식의 경우도 사실 손가락 인식과 메커니즘이 온전히 같기 때문에 이 프로그램을 그대로 적용할 수 있다. 실험 전에는 얼굴 인식이 손가락 인식보다 더 잘 안 될 것이라고 생각했는데, 실제 실험 결과는 놀랍게도 얼굴 인식 쪽의 정확도가 훨씬 높았다. 6명 중 4명은 촬영 환경조차 동일했기 때문에, 얼굴 인식이 손가락 인식보다 정확도를 높이기 위해 필요한 레이어의 수가 적다는 결론을 내렸다. 다음은 얼굴 인식의 결과이다. 트레이닝 데이터 수는 2000, 사용한 레이어의 수는 콘볼루션층 2개, 풀링층 2개, 소프트맥스층 1개이다.





세 번 모두 test accuracy가 1이 되는 놀라운 결과를 보였다.

1. **고찰**

손가락을 구별할 수 있는 예제를 구현해봄으로써, MNIST 예제만 돌리는 것보다 더 많은 응용을 손댈 수 있게 되었다. 또한, 손가락을 구별하는 것과 얼굴을 구별하는 것이 서로 완전히 같은 메커니즘을 사용하고 있음을 파악함으로써, 얼굴 인식 또한 성공적으로 구현할 수 있었다. Deep Learning은 막연한 것이라고 생각했는데, 막상 예제를 통해 돌려보니 보다 구체적으로 인공신경망을 이해할 수 있게 되었다. CNN을 이용한 Image Classification 문제는 사실 배경지식에서 언급했던 것처럼, 개선할 수 있는 여지가 아직 더 있다. 이후에는 이 모듈을, 그러한 개념을 활용하여 개선해보고자 한다.

End of report.

한 학기 동안 수고하셨습니다.

1. 이를 ‘커널로 특징 맵(feature map)을 만든다’라고도 한다. [↑](#footnote-ref-1)