TensorFlow를 이용한 이미지 분할
Image Segmentation with TensorFlow

TOPICS

- Lab Perspective
- 이미지 분할
- TensorFlow
- Lab
  - 랩 이해/ 개요
  - 랩 환경 론칭
  - 랩 리뷰

LAB PERSPECTIVE

여기서 우리는 무엇을 할 것인가?

- 딥러닝을 사용한 이미지 분할에 대한 이해 및 시뮬레이션
- CNN 학습 및 이미지 분할 워크플로우 평가용 TensorFlow 사용 실습

여기서 우리가 하지 않는 것은 무엇인가?

- 기계학습부터 처음으로 소개
- 복잡한 신경망의 불규칙한 수학 공식들을 설명
- TensorFlow의 모든 기능 및 옵션 조사

가정

- 여러분은 CNN (convolutional neural networks)에 이미 익숙하다
- 다음과 같은 경험이 있다면 도움이 될 것:
  - 이미지 인식 경험
  - TensorFlow 경험
  - Python 경험

TAKE AWAYS

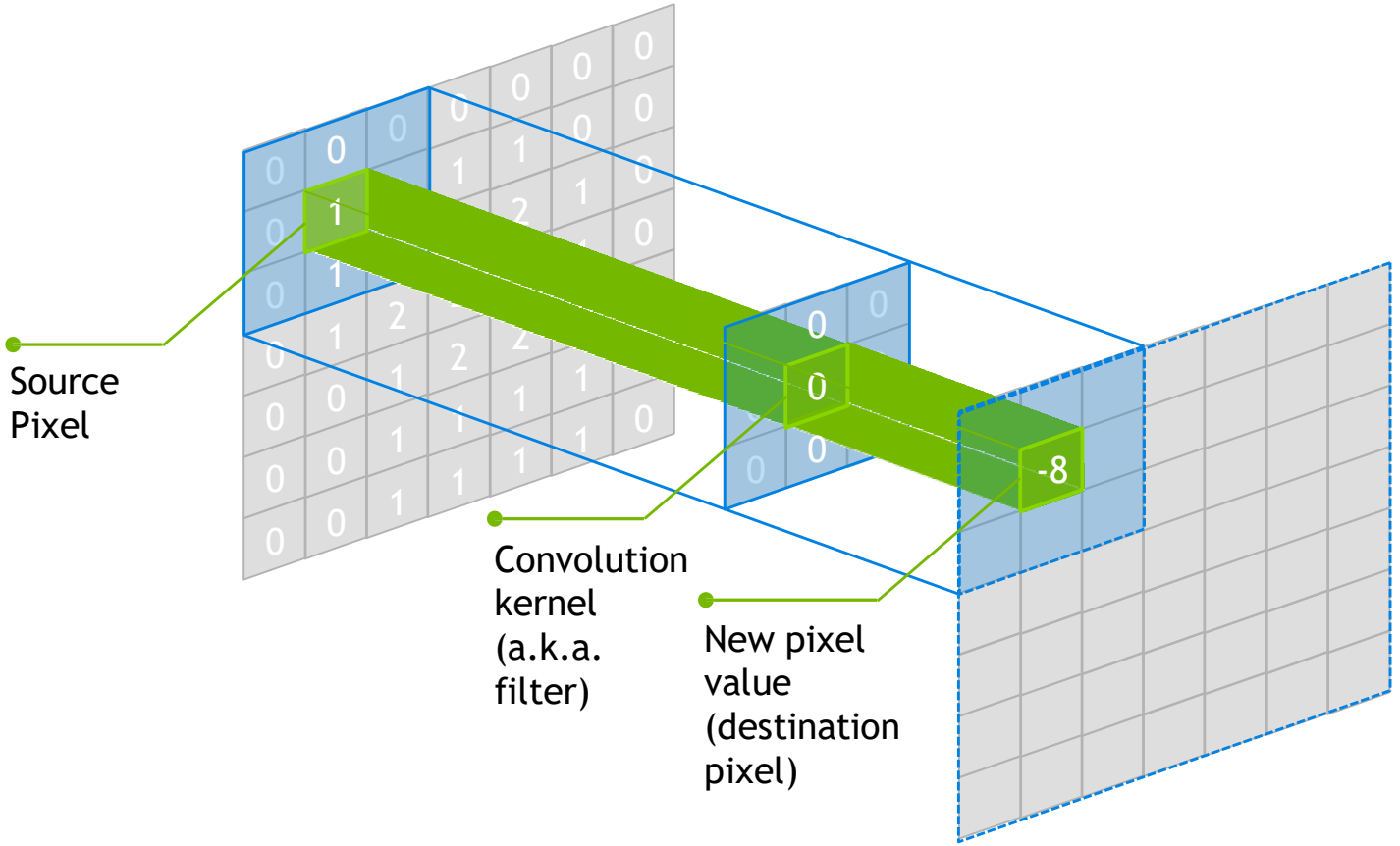
- TensorFlow에서 여러분 고유의 이미지 분할 워크플로우를 셋업 할 수 있게 됨
- 더 많은 정보를 얻기 위해 어디로 가야 할지 알게 됨
- TensorFlow에 익숙해 짐

이미지 분할
IMAGE SEGMENTATION

CONVOLUTIONAL NEURAL NETWORK

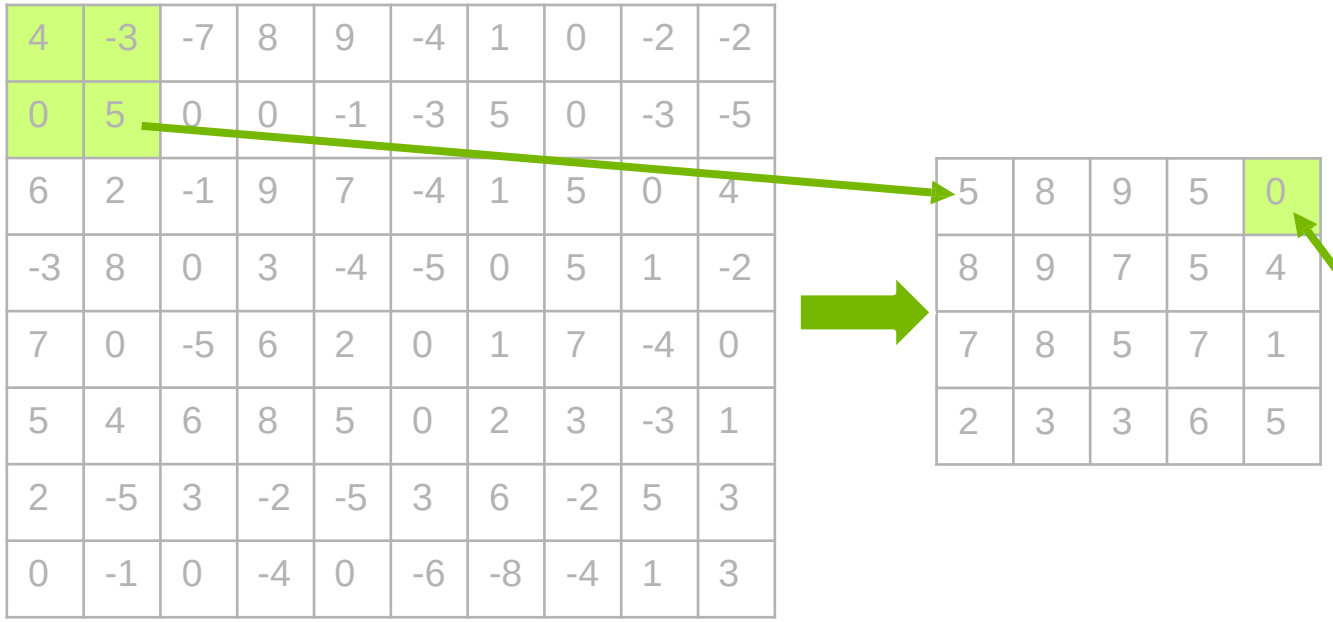
- Convolutions = kernels = filters
- Convolutions programmatically determine significant features
- Typical operations in CNN:
  - Convolution
  - Non-linearity / Activation function
  - Pooling
  - Classification

CONVOLUTIONS



- Kernel을 통해 영상특성을 계산해서 전달
- Kernel 이 작으면 더 작은 영역의 특성을 추출

POOLING



- Sliding window
  - (2 X 2 with stride of 2 in this example)
- Down-sampling technique
- 일부 플랫폼에서는 MaxPooling결과에 의해 음수가 0이 됨

COMPUTER VISION이 하는 일

이미지 분류

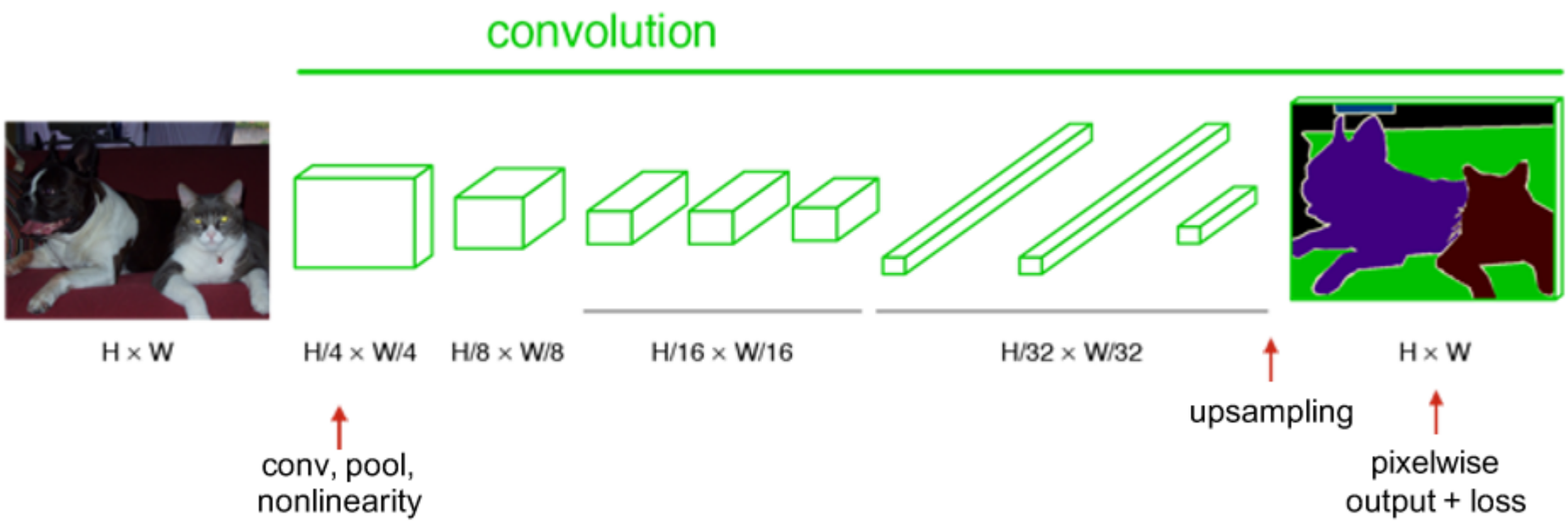
이미지 분류 + 위치

객체 인식

이미지 분할

(스탠포드 대학 cs231n 강의에서 사용된 한 슬라이드에서 영감을 받음)

영상분할을 위한 과정



이미지 분할
IMAGE SEGMENTATION

- 이미지를 “분류”하는 것과는 조금 다른 픽셀 분류로써의 “분할”
- 이번 랩에서는, 의미 분할(semantic segmentation)이 수행될 예정
  - 예, 한 이미지에서 각 픽셀은 여러 클래스 중 하나
- 어떤 의미에서 *각 픽셀이 하나의 클래스를 갖는 분류의 문제* vs. *각각의 이미지(픽셀의 조합)가 하나의 클래스를 갖는 이미지 인식의 문제*
- 특히, 우리는 의료용 이미지 데이터를 가지고, 좌심실(LV)가 어디 있는지를 찾을 예정
  - 예, 각 픽셀들이 LV의 일부인가 아닌가?

TENSORFLOW

TENSORFLOW란 무엇인가?

Google이 개발, tensorflow.org

- "기계 인텔리전스를 위한 오픈 소스 소프트웨어 라이브러리"
  - GitHub에서 가능
- 유연성– 여러분의 컴퓨터를 데이터 흐름 그래프(data flow graph)로 표현
  만약 TF syntax로 표현 가능하면, 실행도 가능
- 휴대성–CPUs 및 GPUs, 워크스테이션, 서버, 핸드폰
- 프로그래밍 언어 옵션–Python, C++
- 성능–CPUs와 GPUs의 성능향상에 맞춤
  - 각기 다른 하드웨어에 작업들을 할당
  - CUDNN 사용

TENSORFLOW 실행

- 그래프 구성—실제 연산들이 일어나기 전에 수행
  - 그래프로 신경망 구성
  - `Variables`(변수)- 시간에 따라 변할 수 있는 그래프의 특성들
    - 예: 학습된 가중치 (weights)
  - 운영—변수와 데이터를 결합한 컴퓨팅
    - 예: convolution, activation, matrix multiply 등
- `session` 실행
  - 하나의 그래프를 실행하기 위한 TF 용어
  - 이전에 생성된 그래프를 통해 데이터를 수집하고 실행

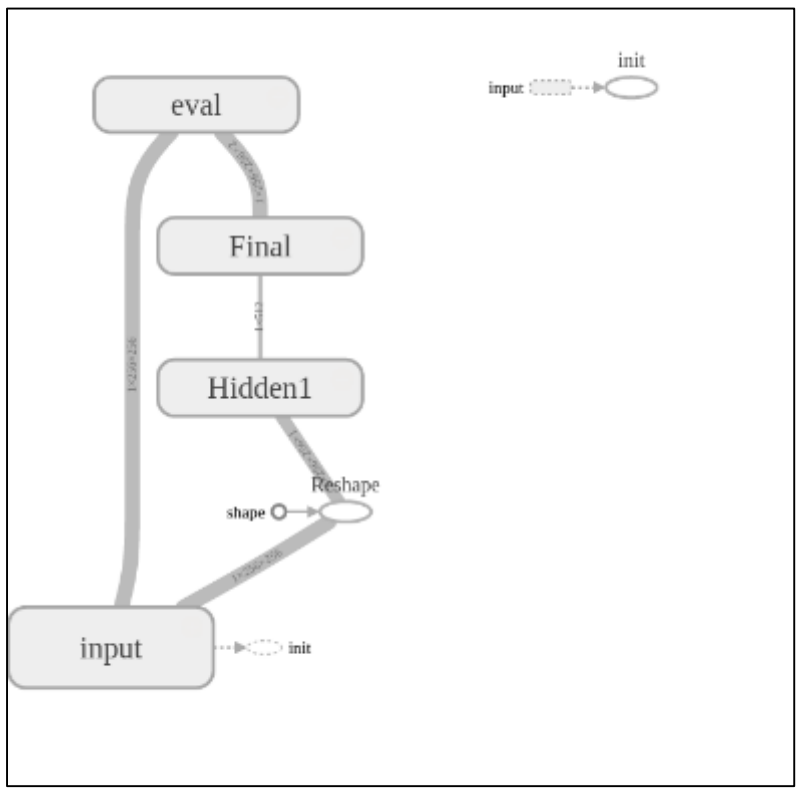
WORKFLOW의 예시

- 입력 데이터(input data) 준비
  - numpy arrays 사용가능, 매우 큰 데이터 셋일 경우 TFRecords 사용
- 컴퓨팅 그래프(computation graph) 구축
  - inference, loss, training nodes 생성
- 모델 학습
  - 입력 데이터를 해당 TF `session`의 그래프에 삽입하고, 입력데이터 반복 추가
  - Batch 크기, epoch 수, 학습량 등의 항목 지정
- 모델 평가
  - 그래프에서 추론을 실행한 다음 적절한 공식을 기반으로 정확도 평가

TENSORBOARD

- 학습 진행 상황을 시각화 하기 위한 TF 도구
  - 에포크마다의 loss, learning rate, accuracy의 시각화
  - 구성한 graph 시각화
- 실습에 TensorBoard 사용
  - 모델성능을 명확하게 분석하고, 학습 및 평가를 통합하는데 매우 유용

TENSORBOARD 그래프의 예



- 1개의 숨겨진 레이어(hidden layer)를 가진 신경망 평가 그래프
- 각각의 박스를 확대해 보려면 클릭
  - 각 사용자 정의 노드의 운영 및 변수를 보여준다

INFERENCE 그래프의 예

```
with tf.name_scope('Hidden1'):
    W_fc = tf.Variable(tf.truncated_normal( [256*256, 512],
                stddev=0.1, dtype=tf.float32), name='W_fc')
    flatten1_op = tf.reshape( images_re, [-1, 256*256])
    h_fc1 = tf.matmul( flatten1_op, W_fc )

with tf.name_scope('Final'):
    W_fc2 = tf.Variable(tf.truncated_normal( [512, 256*256*2],
                stddev=0.1, dtype=tf.float32), name='W_fc2' )
    h_fc2 = tf.matmul( h_fc1, W_fc2 )
    h_fc2_re = tf.reshape( h_fc2, [-1, 256, 256, 2] )

return h_fc2_re
```

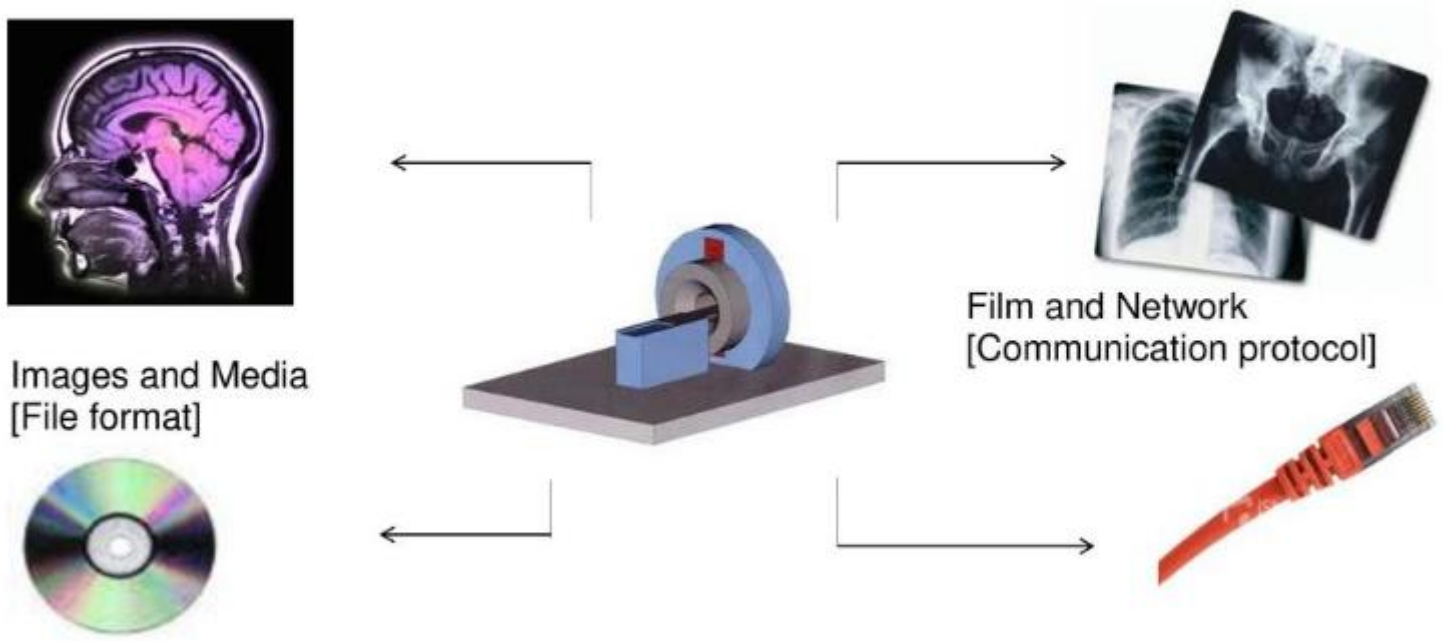
랩 이해 / 개요

DATASET

- 심장 MRI short-axis (SAX) 스캔본
  - 이전 경쟁 제품인 Sunnybrook의 심장 이미지
    http://smial.sri.utoronto.ca/LV_Challenge/Data.html
  - "Sunnybrook Cardiac MR Database" 는 위에 설명된 CC0 1.0 Universal license로 제공되며, 자세한 내용은 아래 참조:
    http://creativecommons.org/publicdomain/zero/1.0/
  - 출처:
    - Radau P, Lu Y, Connelly K, Paul G, Dick AJ, Wright GA. "Evaluation Framework for Algorithms Segmenting Short Axis Cardiac MRI." The MIDAS Journal -Cardiac MR Left Ventricle Segmentation Challenge, http://hdl.handle.net/10380/3070

DICOM 정의

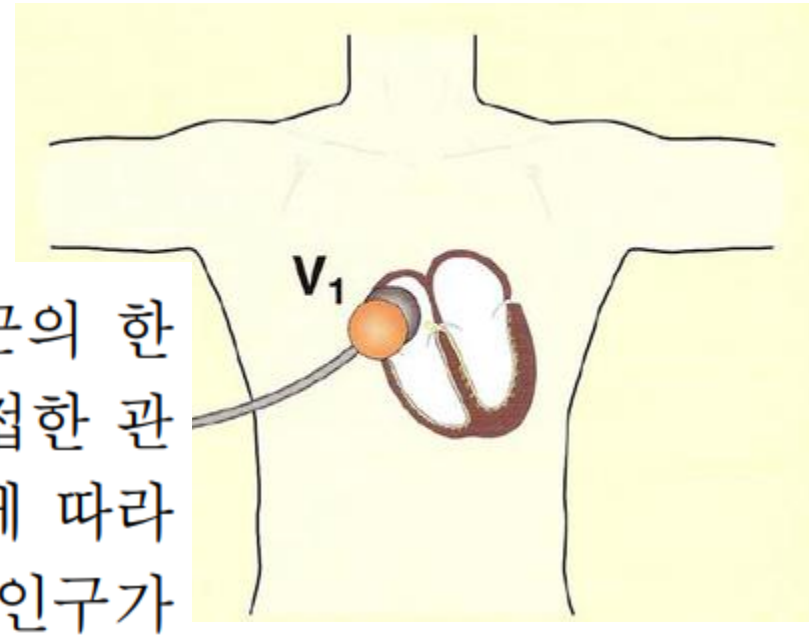
Digital **I**maging and **CO**mmunication in **M**edicine



De-facto standard in medical imaging systems

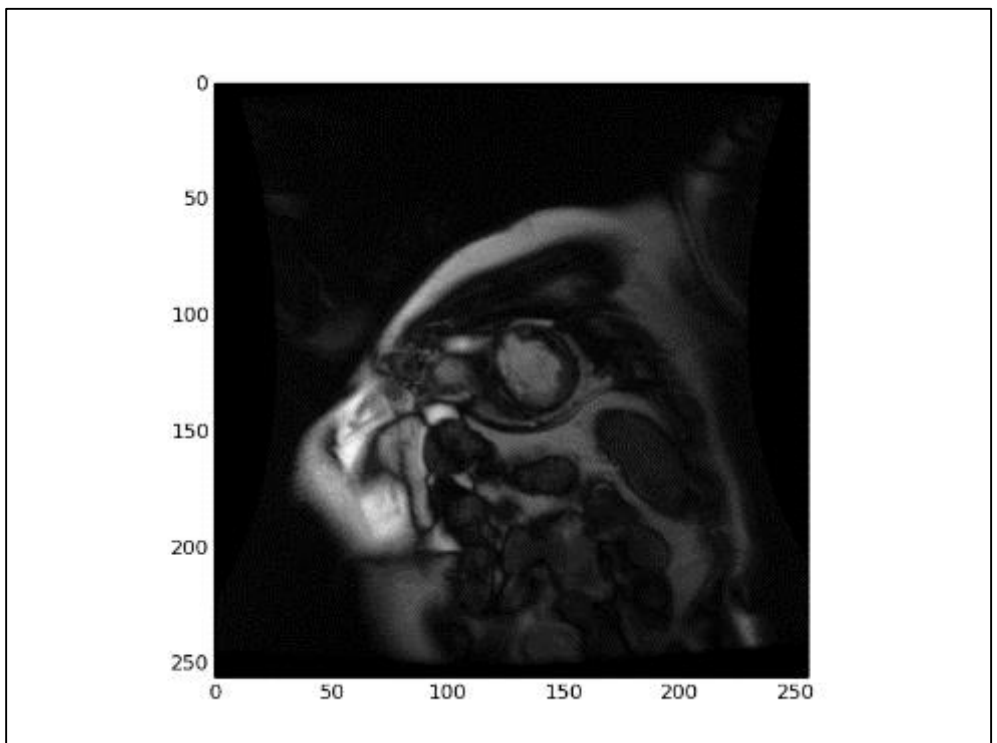
영상 분석을 통한 좌심실 비대 확인

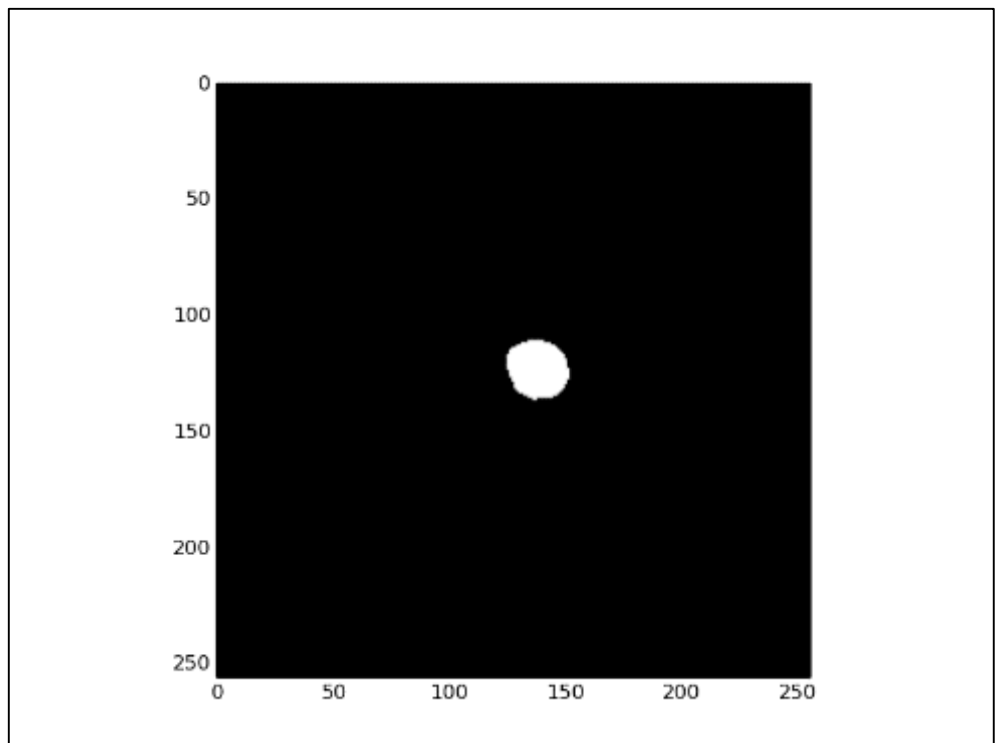
심방의 비대
-우심방 비대(Right atrial enlargement, RAE)
-좌심방 비대(Left atrial enlargement, LAE)
심실의 비대
-우심실 비대(Right ventricular hypertrophy, RVH)
-좌심실 비대(Lef ventricular hypertrophy, LVH)



좌심실 비대는 비정상적인 심부하에 대한 심근의 한 반응형태로 심장 혈관 질환의 발병 및 예후와 밀접한 관계가 있다.[1-4] 좌심실 비대는 평균 수명이 연장됨에 따라 노인 인구가 증가하고 생활양식의 변화로 비만 인구가 증가함에 따라 고혈압이 있는 사람에게뿐만 아니라 질병이 없는 사람에게도 나타나는 경우가 많으며 그 자체만으로도 돌연사, 심부전 및 허혈성 심질환, 부정맥 등의 빈도가 증가한다.[5-7]

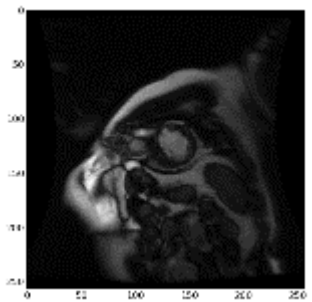
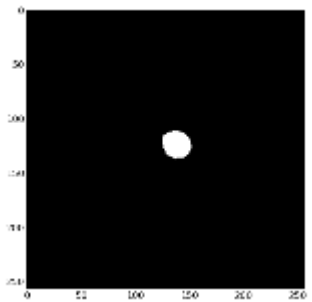
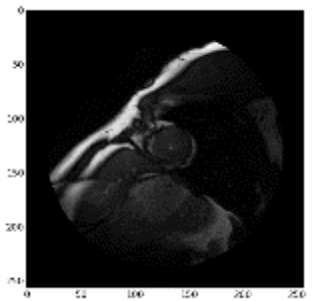
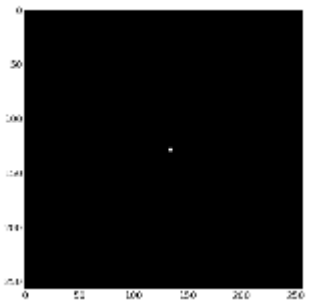
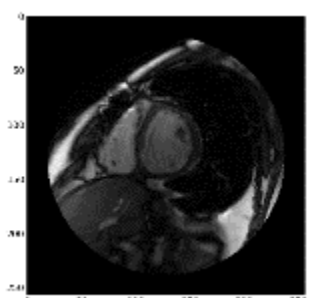
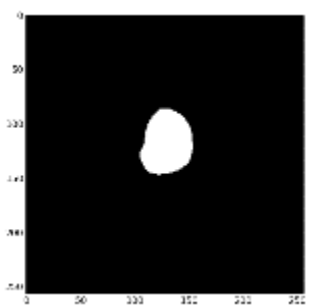
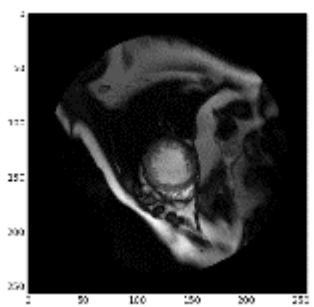
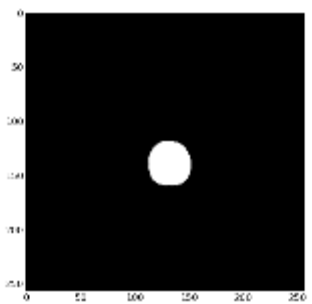
이미지의 예


0
50
100
150
200
250
0
50
100
150
200
250


0
50
100
150
200
250
0
50
100
150
200
250

이미지의 예

완전한 이미지와 전문적으로 라벨링 된 좌심실 윤곽선

DATA 개요

- 원본 이미지는 256 x 256 grayscale DICOM 형식
- 출력은 256 x 256 x 2 사이즈의 tensor
  - 각각의 픽셀은 두개 클래스 중 하나에 속함
- 학습용 세트는 234개 이미지로 구성
- 검증용 세트는 26개 이미지로 구성

데이터 셋업 배경 설명

- 데이터를 셋업/ 추출(setup/extract) 하는 다양한 방법은 아래 링크에서:
  - https://www.kaggle.com/c/second-annual-data-science-bowl/details/deep-learning-tutorial
- 원본 데이터에서 이미지와 윤곽선이 추출되어 TensorFlow에서 수집할 수 있도록 패키징 됨
  - 데이터 추출 코드(Data extraction code)가 포함되었지만 시연하지 않을 예정
- TensorFlow data records는 제공되지만, 원본은 이번 랩에 제공되지 않음
  - 관심 있다면, 직접 다운로드 받을 수 있음

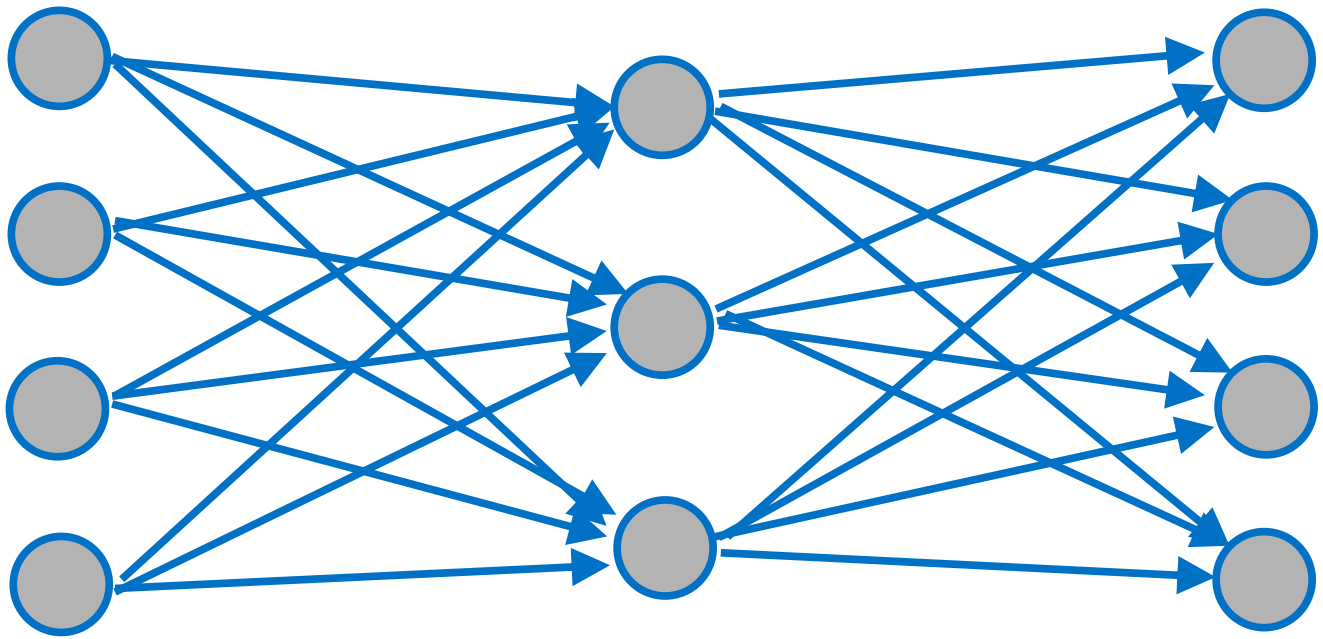
실습 1

모든 것이 잘 되고 있는지 확인할 것!

- 하나의 숨겨진 레이어를 가진 완전히 연결된 신경망을 학습하고 테스트
  - 이 신경망의 시각적 표현은 다음 페이지에
- 손실 계산을 위해 빌트인 함수 사용
  `sparse_softmax_cross_entropy_with_logits`
  - 추론 값(inference output)의 softmax를 계산한 다음, 올바른 라벨에 대한 교차 엔트로피(cross entropy) 계산

실습 1 - 신경망

한개의 hidden layer만 가능



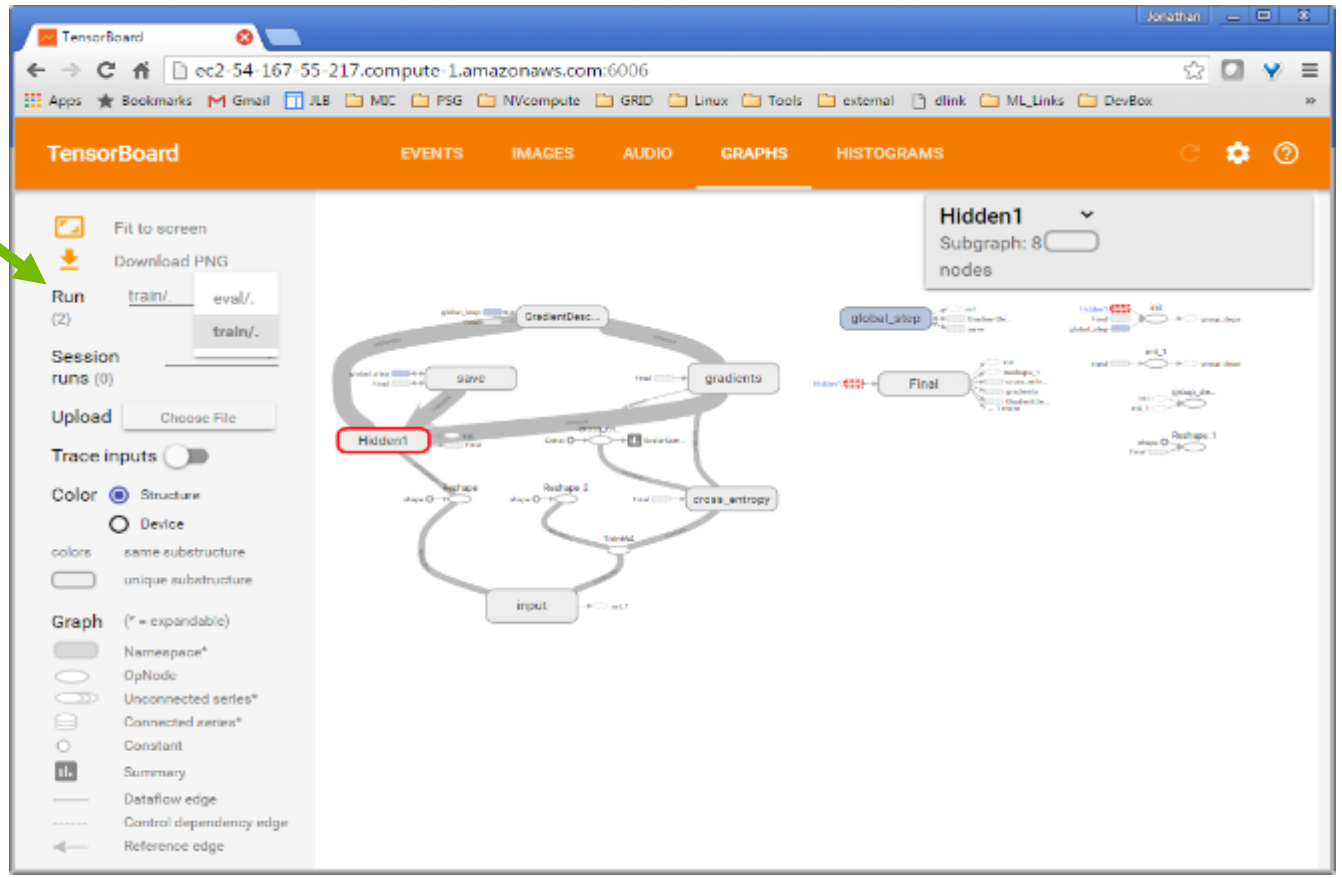
Input layer
65536 (256 x 256) pixels

hidden layer
512

output layer
256 x 256 x 2

실습 1 - TENSORBOARD

- train 또는 eval graph 형식을 선택가능
- 이것은 학습용 그래프
- 더 자세한 내용을 보기 원하면 각 노드를 클릭
- 더 많은 training 또는 eval을 실행하고자 하면, 브라우저를 새로 고침 하면, 통계가 업데이트 됨



실습 2 - 추가 LAYERS

- Convolution layers
  - 각 입력 픽셀에 초점을 맞춘 이전 예제
  - 기능이 여러 개의 입력 픽셀을 포함하는 경우에 사용
  - 더 큰 receptive fields를 캡처하는 데 사용가능
- Pooling layers
  - 기본적으로 컴퓨팅 복잡성을 제거하는 동시에 정보를 유지하는 다운샘플링 (down-sampling) 방법

실습 2 - FULLY CONVOLUTIONAL NETWORK (FCN)

- Image classification layers—Convolutions, pooling, activations, fully connected
  - 출력은 N-차원 벡터이며 여기서N == Number_of_classes
- 이 네트워크를 활용하여 분할할 수 있습니까? YES!
- 이 문제를 픽셀 분류로 재인식
  - i.e., 각 픽셀은 특정 클래스에 속한다
- 이미지 분류 네트워크의 대부분을 재사용
- fully connected layer(s)를 deconvolution (교체된 convolution)으로 교체
  - 출력은 256 x 256 x N tensor이며, 여기서 N == Number_of_classes
  - 이 실습에서 N == 2

실습 2 - 추가 LAYER

- Deconvolution (transpose convolution) layer
  - 최종 픽셀 분류를 위해 더 작은 이미지 데이터셋을 원래 크기로 되돌리는 업샘플링(Up-sampling) 방법
- Long et al (CVPR2015)의 논문: 분할을 위한 FCN
  - AlexNet및 기타 정식 네트워크에서 FCN 생성
- Zeiler et al (CVPR2010)의 논문은 deconvolution을 설명
- 우리가 사용할 네트워크는 아래 Vu Tran의 Kaggle 예와 매우 유사: https://www.kaggle.com/c/second-annual-data-science-bowl/details/deep-learning-tutorial

실습 3 - 학습 파라미터

파라미터 공간 검색 시 중요

- learning_rate: 초기 학습 속도
- decay_rate: 초기 학습 속도를 감소시키는 비율
  - e.g., 1.0은 줄지않게 하며, 0.5는 감소 속도를 절반으로 줄이는 것을 의미
- decay_steps: 학습 속도를 변경하기 전에 실행할 단계 수
- num_epochs: 입력 데이터를 반복하는 횟수
- batch_size: 현재는 1로 유지
- **learning_rate, decay_rate, decay_steps, num_epoch**으로 실험
- 최상의 Dice 점수를 제공하는 파라미터를 기록

LAB REVIEW

LAB 요약

- 이미지 분할 소개
  - 픽셀 분류 vs. 이미지 분류
- 분할을 위해 이미지 인식 네트워크를 FCN으로 변환
- TensorFlow를 프레임워크로 사용하여 FCN에 대한 다양한 최적화 탐색
- 새로운 정확도 공식(Dice Metric)을 통해 더 나은 인식 정확도 획득

실습 1 - 학습 결과물

!python exercises/simple/runTraining.py --data_dir /data

Output:

```
OUTPUT: Step 0: loss = 2.621 (0.169 sec)
OUTPUT: Step 100: loss = 4.958 (0.047 sec)
OUTPUT: Step 200: loss = 4.234 (0.047 sec)
OUTPUT: Done training for 1 epochs, 231 steps.
```

Lots of messages printed to the screen - look for “OUTPUT”

실습 1 - 평가

```
!python exercises/simple/runEval.py --data_dir /data
```

Output:

```
OUTPUT: 2016-08-23 15:37:26.752794: accuracy = 0.504

OUTPUT: 26 images evaluated from file
/tmp/sunny_data/val_images.tfrecords
```

- 출력은 예측의 정확도와 활용된 데이터를 보여줌
  - 1.0은 NN이 모든 데이터를 라벨과 동일하게 분류한 것을 의미. 즉, 100% 정확하다.

실습 1 솔루션

```
with tf.name_scope('Hidden1'):
    W_fc = tf.Variable(tf.truncated_normal( [256*256, 512],
                  stddev=0.1, dtype=tf.float32), name='W_fc')
    flatten1_op = tf.reshape( images_re, [-1, 256*256])
    h_fc1 = tf.matmul( flatten1_op, W_fc )

with tf.name_scope('Final'):
    W_fc2 = tf.Variable(tf.truncated_normal( [512, 256*256*2],
                stddev=0.1, dtype=tf.float32), name='W_fc2' )
    h_fc2 = tf.matmul( h_fc1, W_fc2 )
    h_fc2_re = tf.reshape( h_fc2, [-1, 256, 256, 2] )

return h_fc2_re
```

실습 2

exercises/tf/segmentation/cnn/neuralnetwork.py

- CNN을 정상 동작시키기 위해, “FIXME” 교체
  - vi / vim 파일명 /FIXME (변화가 생긴 곳을 확인하기 위함 또는 Ctrl+F 로 검색)
  - Dimensions 확인 필요
- Convolution1, 5x5 kernel, stride 2; Maxpooling1, 2x2 window, stride 2
- Convolution2, 5x5 kernel, stride 2; Maxpooling2, 2x2 window, stride 2
- Convolution3, 3x3 kernel, stride 1; Convolution4, 3x3 kernel, stride 1
- Score_classes, 1x1 kernel, stride 1; Upscore (DeConv), 31x31 kernel, stride 16
- 시간이 남으면 num_epochs을 가지고 실험

실습 2 - 부분 솔루션

```
with tf.name_scope('Conv1'):
        W_conv1 = tf.Variable(tf.truncated_normal([5,5,1,100],stddev=0.1,
                        dtype=tf.float32),name='W_conv1' )
        print_tensor_shape( W_conv1, 'W_conv1 shape')
        conv1_op = tf.nn.conv2d( images_re, W_conv1, strides=[1,2,2,1],
                        padding="SAME", name='conv1_op' )
        print_tensor_shape( conv1_op, 'conv1_op shape')

with tf.name_scope('Pool1'):
        pool1_op = tf.nn.max_pool(relu1_op, ksize=[1,2,2,1],
                                  strides=[1,2,2,1], padding='SAME')
        print_tensor_shape( pool1_op, 'pool1_op shape')
```

실습 2 - 평가 결과

- 1 epoch 학습

```
OUTPUT: 2016-08-26 20:44:55.012370: precision = 0.571
```

- 30 epochs 학습

```
OUTPUT: 2016-08-26 20:48:16.593103: precision = 0.985
```

- 98.5% 정확도!
  - 매우 훌륭한 정확도, 이제 충분한가?

실습 2 - 정확도

- 정확도는 어떻게 정의되는가
  - 라벨의 픽셀 값을 CNN에서 계산한 값과 비교
    - 그래서, 98.5% 정확하게 픽셀을 예측하고 있음
- 그러나, 전체 이미지 클래스 불균형 문제에 비해 윤곽선의 사이즈가 비교적 작음
- 만약 우리가 모든 픽셀에 대해 단순히 “notLV(좌심실이 아니다)” 클래스의 출력 값을 도출했다면, 95%이상의 정확도를 가졌을 것
  - 하지만, 이것이 우리가 원하는 것은 분명히 아님

실습 3 - 평가 결과

- 이전 예에서의 결과는:
  - 1 epoch: `precision = 0.501`
  - 30 epochs: `precision = 0.985`
- Dice metric 사용시 (1.0이 완벽한 정확도를 의미)
  - 1 epoch: Dice metric = 0.033
  - 30 epochs: Dice metric = 0.579
- 우리가 원래 생각했던 것만큼 좋지 않음

실습 3 - RESULT

One possible result

--learning_rate=0.03

--decay_rate=0.75

--num_epochs=100

--decay_steps=10000

OUTPUT: 2016-08-26 21:22:15.590642: Dice metric = 0.861

정확도가 훨씬 좋아짐!

WHAT ELSE?

- 더 오래 학습
  - 시연을 목적으로, 우리는 진짜 짧게 학습시간을 가졌음
  - 더 많은 epoch 필요
- 더 많은 학습 데이터
  - 학습 데이터로 단 236개의 이미지만 사용했음
  - 더 많은 데이터 수집 필요
  - 회전 및 반전 등을 통해 더 많은 이미지 획득 필요
    - TF 는 자동으로 flip/rotate/transpose 하는 기능이 있음
- 더 크고 복잡한 네트워크

WHAT'S NEXT

- 배운 내용을 사용해 보고 실습하세요.
- DNN의 실제 활용 애플리케이션에 대해 동료들과 토론해 보십시오.
- NVIDIA와 Deep Learning Institute에 컨택 하십시오.

www.nvidia.com/dli