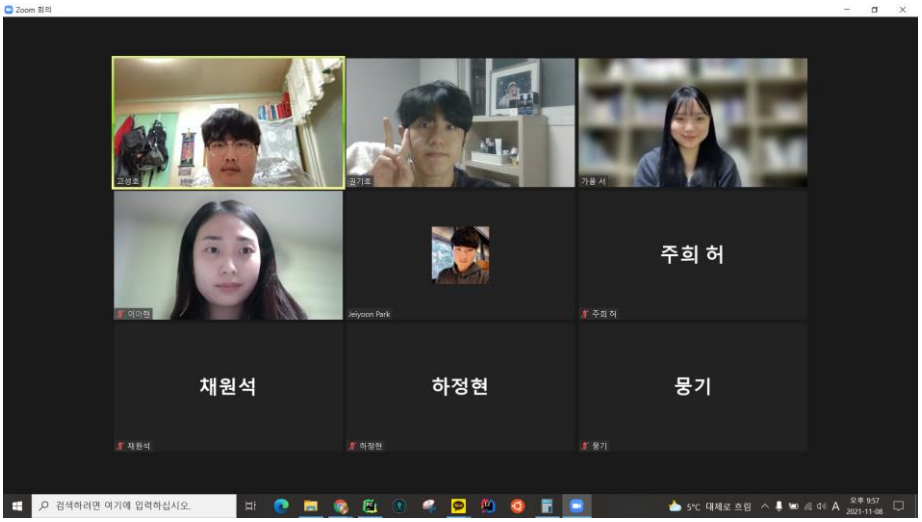
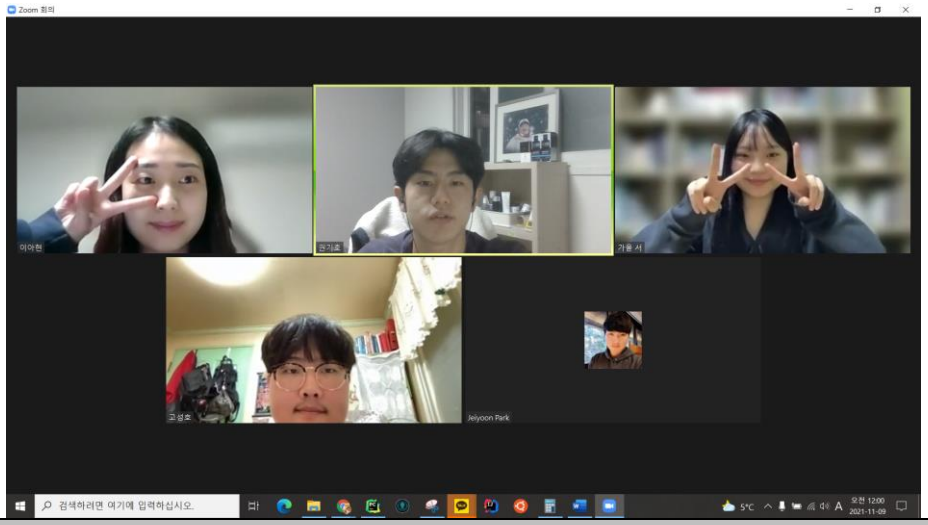
TAVE 서기

<table>
<tr><th colspan="4">서기 내용</th></tr>
<tr><td>서기
일자</td><td>21.11.08</td><td>서기</td><td>이아현</td></tr>
<tr><td>주제</td><td colspan="3">삼성 AI forum 및 딥러닝 추가 세션</td></tr>
<tr><td>시간</td><td>22:00~24:00</td><td>장소</td><td>Zoom 미팅 및 트위치</td></tr>
<tr><td>스터디
인원</td><td colspan="3">고성호, 권기호, 이아현, 서가을 : 시작
 

고성호, 권기호, 이아현, 서가을 : 종료
  </td></tr>
<tr><th colspan="4">내용</th></tr>
<tr><td>배운
내용</td><td colspan="3">Chapter 13. 텐서플로에서 데이터 적재와 전처리하기
13.1 데이터 API
Chapter 14. 합성곱 신경망을 사용한 컴퓨터 비전
14.2 합성층 곱

삼성 AI forum</td></tr>
</table>

- Day1 : The future of AI hardware
- Day1 : AI/ML in materials research and the laboratory of the future
- Day1 : Learning to see
- Day2 : Interpretability for skeptical minds

13.1 데이터 API

- 전체적인 데이터 API 의 중심에는 '데이터셋' 개념이 있음 => 연속된 데이터 샘플을 나타냄

```
import tensorflow as tf
X = tf.range(10) #0부터 9까지의 10개의 아이템을 가짐
dataset = tf.data.Dataset.from_tensor_slices(X)
dataset
```

- 데이터셋의 아이템을 순회할 수 있음

```
for item in dataset:
  print(item)
```

```
tf.Tensor(0, shape=(), dtype=int32)
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(2, shape=(), dtype=int32)
tf.Tensor(3, shape=(), dtype=int32)
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor(5, shape=(), dtype=int32)
tf.Tensor(6, shape=(), dtype=int32)
tf.Tensor(7, shape=(), dtype=int32)
tf.Tensor(8, shape=(), dtype=int32)
tf.Tensor(9, shape=(), dtype=int32)
```

13.1.1 연쇄변환

- 데이터셋이 준비되면 변환된 메서드를 호출하여 여러 종류의 변환을 수행할 수 있음
- 각 메서드는 새로운 데이터셋을 반환하므로 다음과 가 x 이 변환 메서드를 연결할 수 있음
- batch() 메서드를 drop_remainder=True 로 호출하면 길이가 모자란 마지막 배치를 버리고 모든 배치를 동일한 크기로 맞춤

```
dataset = dataset.repeat(3). batch(7)
for item in dataset:
  print(item)
```

```
tf.Tensor([0 1 2 3 4 5 6], shape=(7,), dtype=int32)
tf.Tensor([7 8 9 0 1 2 3], shape=(7,), dtype=int32)
tf.Tensor([4 5 6 7 8 9 0], shape=(7,), dtype=int32)
tf.Tensor([1 2 3 4 5 6 7], shape=(7,), dtype=int32)
tf.Tensor([8 9], shape=(2,), dtype=int32)
```

- map() 메서드를 호출하여 아이템을 변환할 수 있음

```
dataset = dataset.map(lambda x:x*2) #아이템:[0,2,4,6,8,10,12]
```

- apply() 메서드를 호출하여 데이터셋 전체에 변환을 적용할 수 있음

```
dataset = dataset.apply(tf.data.experimental.unbatch())
```

- filter() 메서드를 사용하여 데이터셋을 필터링할 수도 있음

```
dataset = dataset.filter(lambda x: x<10)
```

- 데이터셋에 있는 몇 개의 아이템을 볼 때는 take() 메서드를 사용함

```
for item in dataset.take(3):
  print(item)
```

13.1.2 데이터 셔플링

- shuffle() : 버퍼 크기를 지정해주어야 함
    1. 원본 데이터셋의 처음 아이템을 buffer_size 개수만큼 추출하여 버퍼에 채움
    2. 새로운 아이템이 요청되면 이 버퍼에서 랜덤하게 하나를 꺼내 반환함
    3. 원본 데이터셋에서 새로운 아이템을 추출하여 비워진 버퍼를 채움
    4. 이를 원본 데이터셋의 모든 아이템이 사용될 때까지 반복
    5. 버퍼가 비워질 때까지 계속하여 랜덤하게 아이템을 반환

해당 메소드는 버퍼 크기를 충분히 크게 해주는 것이 중요함(셔플링 효과가 감소될 수 있으므로, but 보유한 메모리 크기를 넘지 않아야 함, 데이터셋 크기 초과하면 안 됨)

```
dataset = tf. data. Dataset. range(10).repeat(3)
dataset = dataset.shuffle(buffer_size=5, seed=42, reshuffle_each_iteration=Fals
e).batch(7)
for item in dataset:
  print(item)
```

- 메모리 용량보다 큰 대규모 데이터셋은 버퍼가 데이터셋에 비해 작기 때문에 간단한 셔플링 버퍼 방식으로 충분하지 않음 -> 원본 데이터 자체를 섞으면 됨(리눅스에서는 shuf 명령어)
- 원본 데이터가 섞여 있더라도 일반적으로 에포크마다 한 번 더 섞음 why? 그렇지 않으면 에포크마다 동일한 순서가 반복되어 모델에 편향이 추가됨
- 샘플을 더 섞기 위해 많이 사용하는 방법은 원본 데이터를 여러 파일로 나눈 다음 훈련하는 동안 무작위로 읽는 것 but 동일한 파일에 있는 샘플은 여전히 함께 처리됨 -> 파일 여러 개를 무작위로 선택하고 파일에서 동시에 읽은 레코드를 돌아가면서 반환 후에 shuffle() 메서드를 사용해 그 위에 셔플링 버퍼를 추가할 수 있음

13.1.3 데이터 전처리

- 전처리를 수행하기 위한 간단한 함수 작성

```
X_mean, X_std = [0,1]
n_inputs = 8

def preprocess(line):
  defs =[0.] * n_inputs + [tf.constant([],dtype=tf.float32)]
  fields = tf.io.decode_csv(line,record_defaults=defs)
  x= tf.stack(fields[:-1])
  y= tf.stack(fields[-1:])
  return (x - X_mean) / X_std, y
```

X_mean 과 X_std 는 1 개씩 8 개의 실수를 가진 1D 텐서
파싱할 라인과 CSV 파일의 각 열에 대한 기본
decode_csv() 스칼라 텐서 리스트 반환
tf.stack 함수 (마지막 열 빼고) 스칼라 텐서 -> 하나의 값을 가진 1D 텐서

13.1.4 데이터 적재와 전처리를 합치기

- 재사용 가능한 코드를 만들기 위해 하나의 헬퍼 함수로 만듦
- CSV 파일에서 캘리포니아 주택 데이터셋 적재, 전처리, 셔플링, 반복, 배치를 적용한 데이터셋을 만들어 반환함

```
def csv_reader_dataset(filepaths,repeat=1, n_readers=5,
                       n_read_threads=None, shuffle_buffer_size= 10000,
                       n_parse_threads=5, batch_size = 32):
  dataset = tf.data.Dataset.list_files(filepaths).repeat(repeat)
  dataset = dataset.interleave(
      lambda filepath: tf.data.TextLineDataser(filepath).skip(1),
      cycle_length=n_readers, num_parallel_calls=n_read_threads)
  dataset = dataset.shuffle(shuffle_buffer_size)
  dataset = dataset.map(preprocess, num_parallel_calls=n_parse_threads)
  return dataset.batch(batch_size).prefetch(1)
```

13.1.5 프리페치

- prefetch(1) : 데이터셋은 항상 한 배치가 미리 준비되도록 최선을 다함
- 성능 향상: interleave()와 map() 메서드 호출할 때 num_parallel_calls 매개변수 지정
- CPU 와 GPU 를 동시에 사용: GPU 가 한 배치 처리할 때 CPU 가 그 다음 배치 준비 -> 훈련 속도가 더 빨라짐
- 자주 쓰는 데이터셋 메서드
    - concatenate()
    - zip()
    - window()
    - reduce()
    - shard()
    - flat_map()
    - padded_batch()

13.1.6 tf.keras 와 데이터셋 이용하기

- tf.keras 에서 반복을 처리하므로 반복을 지정할 필요가 없음

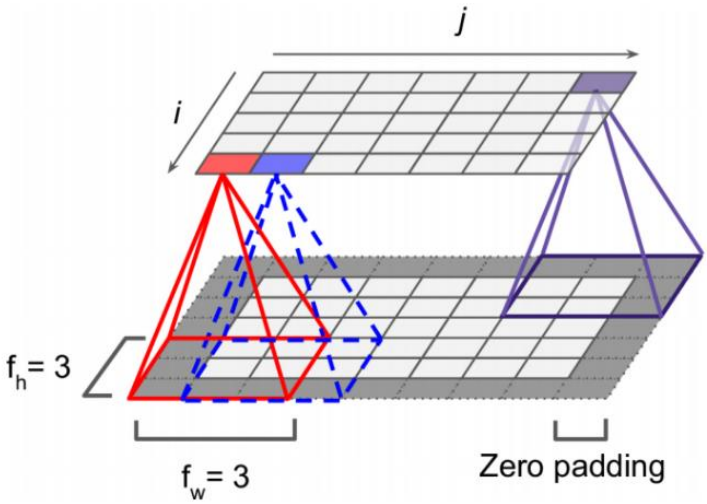
- 데이터셋(검증 세트, 테스트 세트) -> 케라스 모델 만들기 -> 훈련셋과 검증셋 전달

14.2 합성곱 층

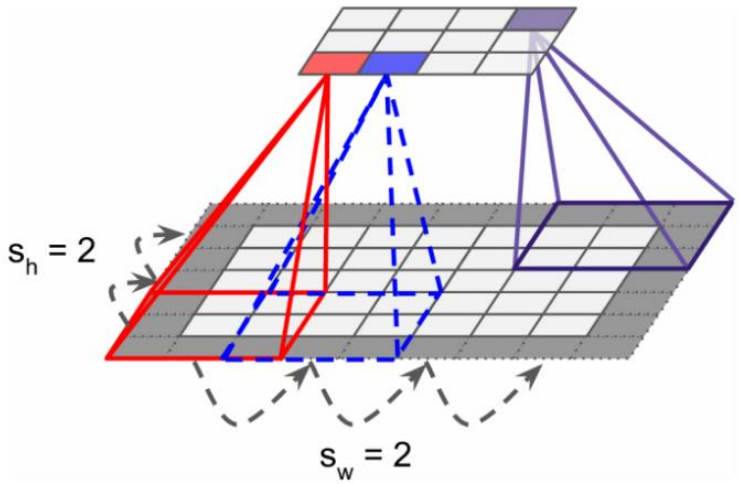
- CNN 의 가장 중요한 구성요소
- 첫 번째 합성곱 층의 뉴런은 입력 이미지의 모든 픽셀에 연결되는 것이 아니라 합성곱 층 뉴런의 수용장 안에 있는 픽셀에만 연결됨
- 두 번째 합성곱 층에 있는 각 뉴런은 첫 번째 층의 작은 사각 영역 안에 위치한 뉴런에 연결됨

=> 네트워크가 첫 번째 은닉층에서는 작은 저수준 특성에 집중하고, 그다음 은닉층에서는 더 큰 고수준 특성으로 조합해나가도록 도와줌

- 제로 패딩 : 높이와 너비를 이전 층과 같게 하기 위해 입력의 주위에 0 을 추가함



- 스트라이드 : 한 수용장과 다음 수용장 사이 간격



14.2.1 필터

- 하나의 필터는 하나의 특성 맵을 만듦 => 이 맵은 필터를 가장 크게 활성화 시키는 이미지의 영역을 강조, 훈련하는 동안 합성곱 층이 자동으로 해당 문제에 가장 유용한 필터를 찾고 상위 층을 이들을 연결하여 더 복잡한 패턴을 학습함

14.2.2 여러 가지 특성 맵 쌓기

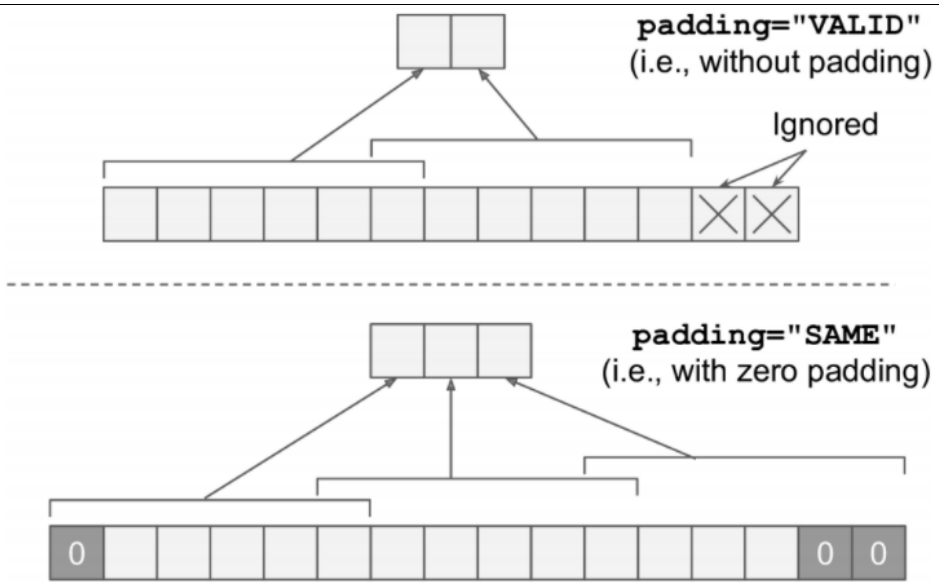
- 하나의 합성곱 층이 입력에 여러 필터를 동시에 적용하여 입력에 있는 여러 특성을 감지할 수 있음
- 한 특성 맵에 있는 모든 뉴런이 같은 파라미터를 공유한다는 사실은 모델의 전체 파라미터 수를 급격하게 줄어줌
- CNN 이 한 지점에서 패턴을 인식하도록 학습되었따면 다른 어느 위치에 있는 패턴도 인식할 수 있음. 반대로 일반적인 DNN 은 한 지점에 있는 패턴을 인식하도록 학습되었다면 오직 패턴이 그 위치에 있을 때만 감지할 수 있음

14.2.3 텐서플로 구현

- 입력 이미지는 보통 [높이, 너비, 채널] 형태의 3D 텐서로 표현
- 하나의 미니배치는 [미니배치 크기, 높이, 너비, 채널] 형태의 4D 텐서로 표현
- 합성곱 층의 가중치는 $[f_h, f_w, f_{n'}, f_n]$ 형태의 4D 텐서로 표현
- 합성곱 층의 편향은 간단하게 $[f_n]$ 형태의 1D 텐서로 나타냄

```
import numpy as np
from sklearn.datasets import load_sample_image

# 샘플 이미지를 로드합니다.
china = load_sample_image("china.jpg") / 255
flower = load_sample_image("flower.jpg") / 255
images = np.array([china, flower])
batch_size, height, width, channels = images.shape

# 2개의 필터를 만듭니다.
filters = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters[:, 3, :, 0] = 1  # 수직선
filters[3, :, :, 1] = 1  # 수평선

outputs = tf.nn.conv2d(images, filters, strides=1, padding="SAME")

plt.imshow(outputs[0, :, :, 1], cmap="gray") # 첫 번째 이미지의 두 번째 특성맵을 그립니다.
plt.axis("off") # 책에는 없습니다.
plt.show()
```

- 텐서플로 저수준 딥러닝 API 중 하나인 tf.nn.conv2d() 함수를 사용함
- padding 은 “VALID”와 “SAME” 중 하나를 지정
VALID 를 지정하면 합성곱 층에 제로 패딩을 사용하지 않음



- 변수를 직접 만드는 것보다 keras.layers.Conv2D 층을 사용함

```
conv = keras.layesrs.Conv2D(filters = 32,
       kernel_size = 3, stride = 1, padding='same' activation = 'relu')
```

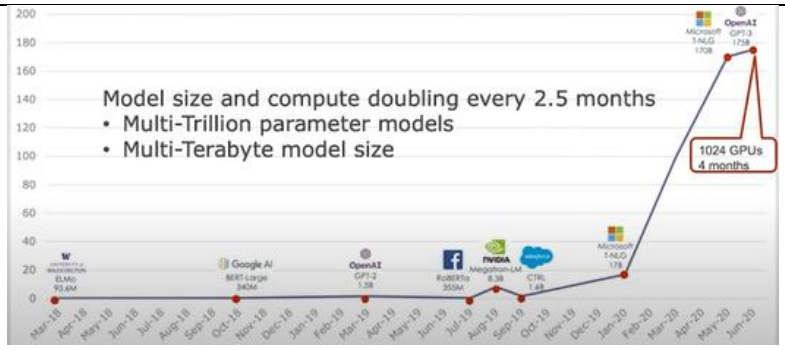
14.2.4 메모리 요구 사항

- CNN 에 관련된 문제중 하나는 많은 RAM 을 필요로 함
- 훈련하는 동안에 역전파 알고리즘이 역방향 계산을 할 때 정방향에서 계산했던 모든 중간값을 필요로 하기 때문
- 추론을 할 때 하나의 층이 점유하고 있는 RAM 은 다음 층의 계산이 완료되자마자 해제될 수 있음. 따라서 연속된 두 개의 층에서 필요로 하는 만큼의 RAM 을 가지고 있으면 됨. But 훈련하는 동안에는 정방향에서 계산했던 모든 값이 역방향을 위해 보존 되어야 함. 이에 각 층에서 필요한 RAM 양의 전체 합만큼 필요함
- 메모리 부족으로 훈련이 실패?
  - 미니 배치크기를 줄여봄
  - 스트라이드를 사용해 차원을 줄이거나 몇 개 층의 제거할 수 있음.
  - 32 비트는 부동소수 대신 16 바트 부동소수를 사용
  - 여러 장치에 CNN 을 분산시킬 수 있음

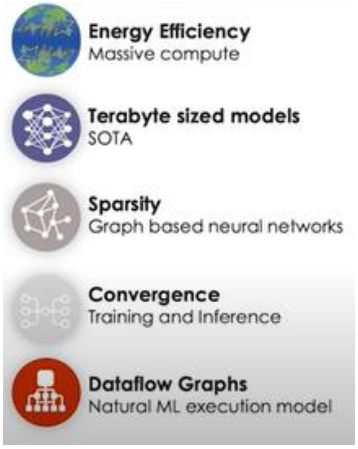
삼성 AI forum

Day1 : The future of AI hardware

- 무어 법칙의 둔화, 데나드 스케일링의 종식 -> 머신러닝의 성공
- 머신러닝이 가져다 주는 막대한 규모의 영향력 ex) 자율 주행 자동차와 개인 맞춤 의료 실현
- 머신러닝 알고리즘을 위한 향상된 성능을 제공하는 것이 목표 -> 전력 효율과 와트당 성능을 향상 시킴
- 머신러닝의 출현과 발전은 연산 능력과 밀접한 연관이 있음
- 신경망의 발전 : 모델 복잡성, 규모의 증가, 데이터의 증가
- 머신러닝 모델의 동향



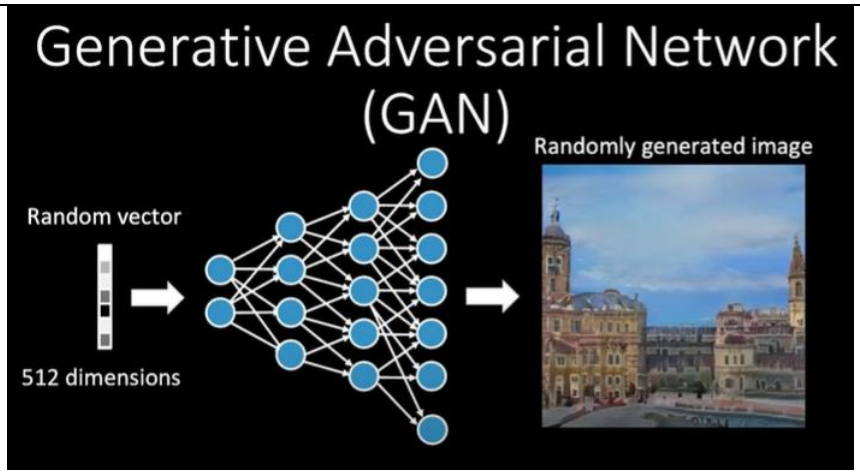
- 머신러닝 : 학습과 추론의 통합
- 다음 세대를 위한 머신러닝 하드웨어의 필요조건



- RDU 인터페이스 : PCU 라고 부르는 연산 특화 장치가 있음
- RDU 장점 : 매우 큰 모델 학습 가능, 전체 이미지 분석 및 전체 이미지 병리학을 실현
- **인공지능 하드웨어의 미래는 SambaNava Systems 의 기본 SN10 가변 구조형 데이터 흐름 장치에서 찾을 수 있음**

Day1 : AI/ML in materials research and the laboratory of the future
- 많은 응용 분야에서 소재의 중요성에 대해 인지하고 있음
- 소재 프로젝트 예시
- 소재와 같은 분야는 데이터가 적음, 관리 힘듦 -> 기계학습이 힘든 분야임 but 성공
- 자연어 처리와 화학 : 단어연상

Day1 : Learning to see
- GAN : 현실적으로 보여야 할 이미지를 재현하는 방법을 학습하면서 시각 세계에 대해 학습함 -> 식별 가능한 이미지를 생성하거나 이미지가 가짜인지 진짜인지를 판별할 수 있도록 시스템을 학습시킴

<table>
<tr><td></td><td> 

- Iamgenet100 performance : 가상 세계에서의 활용 등

Day2 : Interpretability for skeptical minds
- post-training interpretability methods : 누군가가 제공했거나 우리가 학습시킨 모델을 가지고 있는 것으로 이 모델을 변경하지 않을 것임
- 무작위 네트워크 등의 예제
- 어떻게 해야 사용자 중심적인 맥락 최적화를 통해 해석을 제공할 수 있을까?
- TCAV : Testing with Concept Activation Vectors
- 얼룩말 분류를 네트워크 상에서? 임베딩 공간에서 벡터로 나타냄, 선형 분류기 학습, 민감도
- 벡터가 유효하다는 사실을 알아내는 방법 중 하나는 양적 검증
- 세션 목표 : 회의적인 태도를 갖기 -> 더 조심스럽게 접하기</td></tr>
<tr><td>과제할당</td><td>없음</td></tr>
<tr><td>특이사항</td><td>없음</td></tr>
<tr><td>비고</td><td>없음</td></tr>
</table>