프로젝트 계획서

B팀

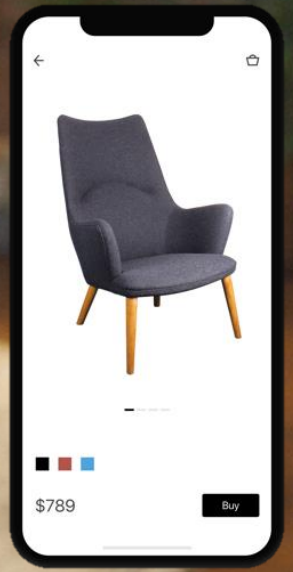
유승은 임소연 전주영

목차

프로젝트 개요

1. 가구 및 인테리어 소품 인식

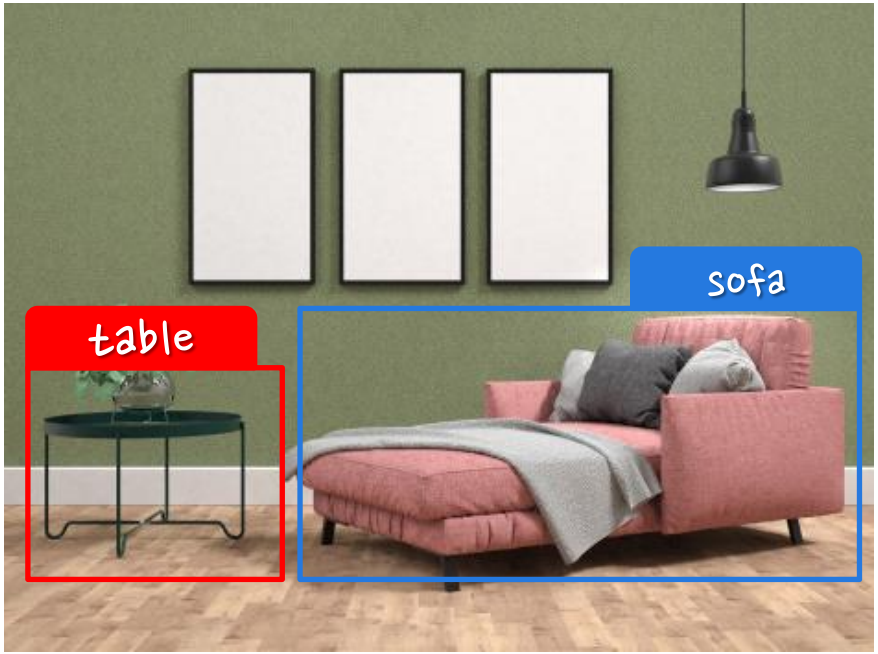
2. 비슷한 제품 검색



프로젝트 개요

Single Object Detection

Multi Object Detection



프로젝트 개요

Single Object Detection

Multi Object Detection

핵심 알고리즘

1. curriculum learning
2. Multi Object Detection
3. Localization

프로젝트 계획

1. Single Object Detection 모델 구현

- 10개의 가구(인테리어 소품)로 분류
- 기본 모델: VGG-16
- 개선 계획

  1) Data Augmentation: zoom, width, height, brightness

  2) hyperparameter 조절

  3) Input size 등 데이터 조정

  4) model customizing: 새로운 layer 추가 등

  5) GoogLeNet 모델 구현 및 적용

프로젝트 계획

2. Multi Object Detection 모델 구현 및 CAM 적용

- Multi-label dataset 구축 (weakly supervised learning 시도 가능)
- 최적화 시킨 Single Object Detection 모델에 학습
- 일정 수준 이상 성능 달성 시 CAM Test와 성능 개선 병행
- 성능 개선 방안

  1) SOD에서 조정한 내용들

  2) 성능에 치명적인 문제가 있는 경우 → Multi Object Detection 을 위한 새로운 모델 적용

     (RNN, SPP-Net 등)

Data Acquisition

Data preprocessing

Data Acquisition (1) and Preprocessing

1. Extreme Picture Finder 로 각 label에 해당하는 데이터 300여 장 수집
2. 1단계 학습 데이터(Single Object Detection)에 적절한 데이터만 골라내기
   > jpeg 파일만 학습 데이터로 사용 → 수작업
3. 데이터 전처리
   > 큰 용량 -> 데이터 리사이징 (PIL, Image 라이브러리 사용 - 500x500 이하)
4. 로컬 디렉토리에 사진 파일 unzip → 학습 속도 향상

Data Acquisition (2)

다양한 모델과 변수 조절을 통해 학습해도 정확도 개선 실패

기존 데이터의 한계가 정확도 부진의 문제가 된다고 판단 → 데이터 추가 수집

1. 각 class 당 1500장 이상 수집
2. 수집할 이미지 사이즈를 제한해 전처리 단계 생략
3. 이전 데이터로 학습시켜본 모델들 다시 train → 정확도 개선 큰 효과 없음

Single Object Detection

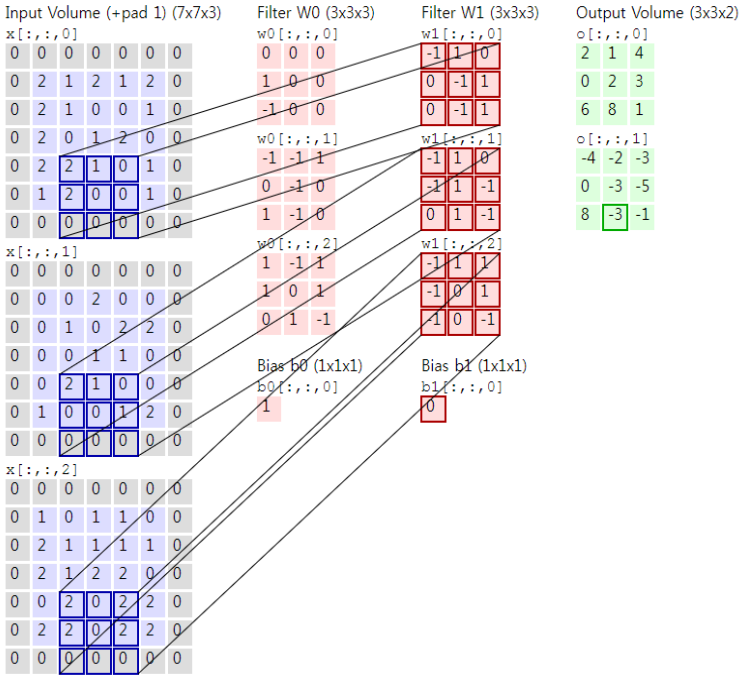
Model Design

1. VGG-16
2. SeyNet
3. GoogLeNet

CNN (Convolutional Neural Network, 합성곱 인공신경망)

일정 크기의 필터의 weight를 Input Data와 곱해서
더하는(convolutional) 과정을 거쳐
한 픽셀과 그 주변 픽셀의 관계를 통해 해당 픽셀의 특성을
비교적 명확히 나타낼 수 있어서 이미지데이터를 효과적으로
분석, 분류할 수 있는 신경망이다.

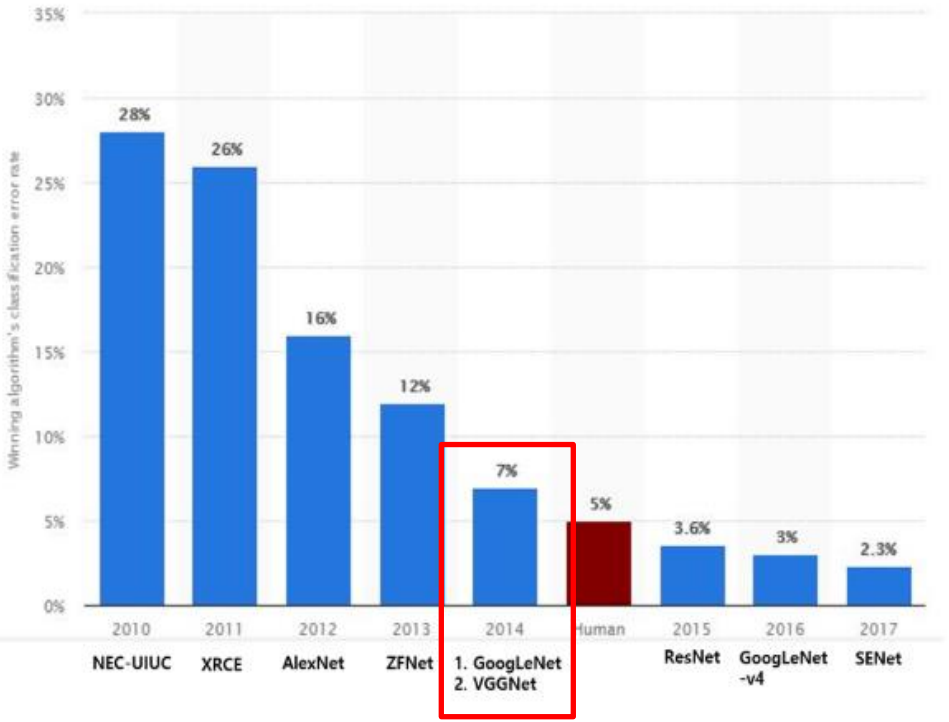
연산 결과로 나오는 데이터는 해당 layer에서 Input으로부터
추출해낸 이미지의 특성을 보여주는 Feature Map이 된다.



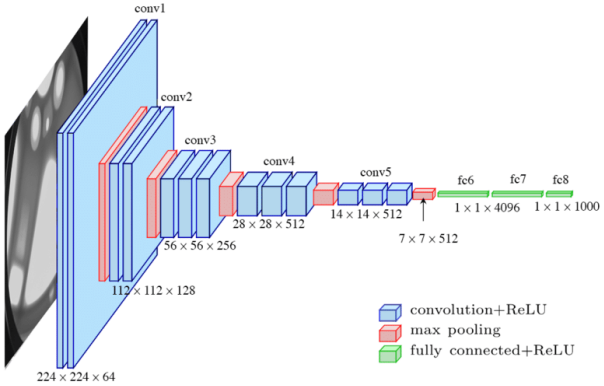
ILSVRC(ImageNet Large Scale Visual Recognition challenge)

대용량의 이미지셋을 받아 분류하는
알고리즘 성능 평가 대회로,
2010년에 시작하였다.

이전 대회들에 비해 훨씬 인간에
가까워진 획기적인 error rate을 기록한
VGG16와 GoogLeNet을 활용



1. VGG-16



작은 필터를 이용하고 많은 층을 쌓아 높은 정확성 확보 가능

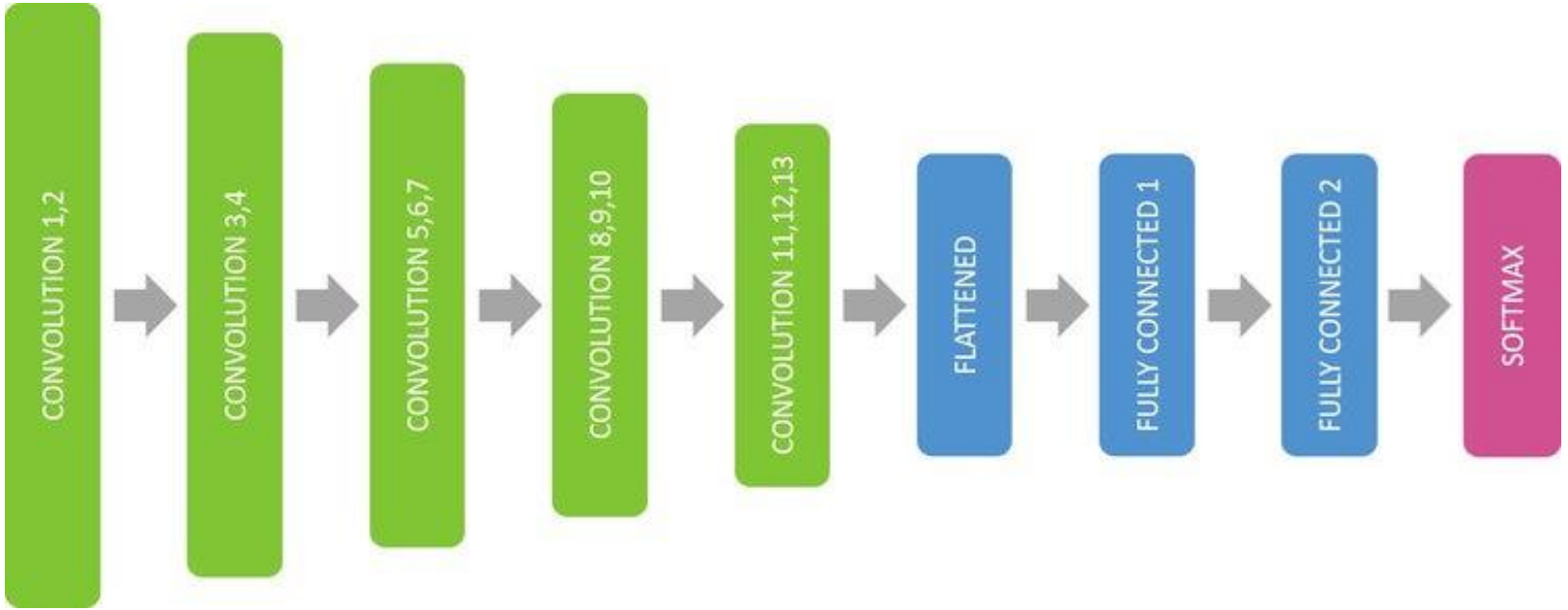
1주차 competition session에서 비교적 좋은 성능을 보여줌

기본 모델 사용

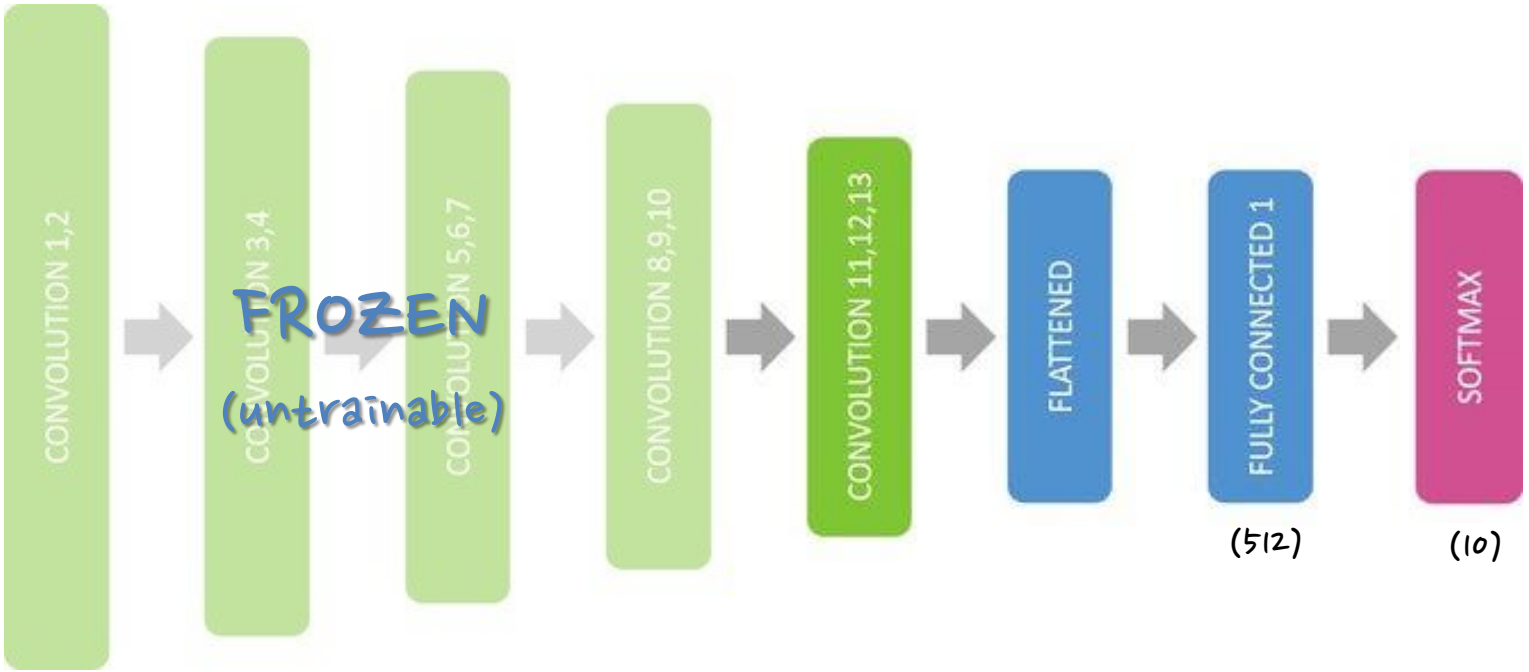
Refining variation

- dense layer (1024, 512, none)
- data augmentation
- batch size
- learning rate schedule

1. VGG-16


CONVOLUTION 1,2
CONVOLUTION 3,4
CONVOLUTION 5,6,7
CONVOLUTION 8,9,10
CONVOLUTION 11,12,13
FLATTENED
FULLY CONNECTED 1
FULLY CONNECTED 2
SOFTMAX

1. VGG-16


CONVOLUTION 1,2
FROZEN
(untrainable)
CONVOLUTION 8,9,10
CONVOLUTION 11,12,13
FLATTENED
FULLY CONNECTED 1
(512)
SOFTMAX
(10)

VGG-16 Model Refining

Data Augmentation
- Shift : 0.3
- brightness : 0.8-1.2

방 사진마다 가구의 위치가 달라지고
방의 밝기가 다를 수 있기 때문에 효과적이라고 판단
→ 실제 overfitting 문제 해결에 효과적

Learning Rate Decaying
- False

비율, 조건 등의 옵션 설정에 어려움을 겪음
adam 외의 optimizer(SGD, RMSProp)로 변경하려 사용했지만
모두 좋지 않은 결과

VGG-16 Model Refining

Freeze Layer

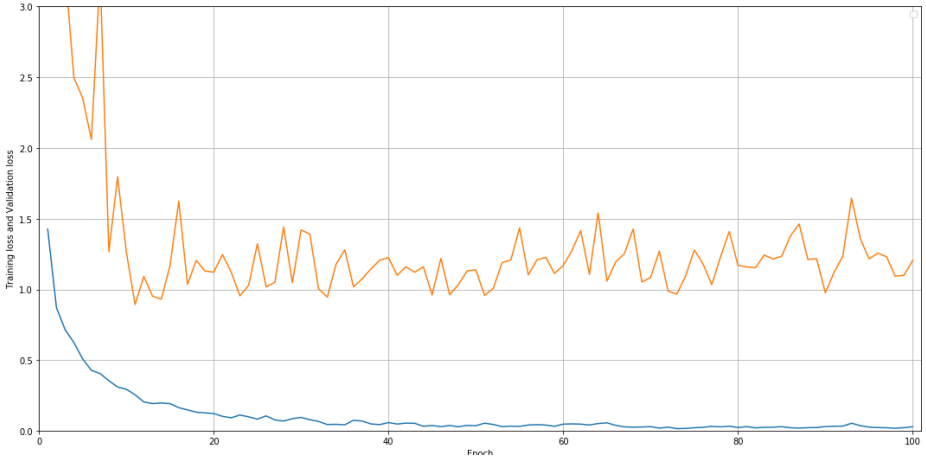
- layers[:-4] trainable True
  batch_n = 128, 256

1주차 competition에서 좋은 성능을 보여주었던 setting
→ 이미 잘 학습된 모델이기 때문에 데이터셋에 맞춰
적당한 trainable layer 설정이 필요

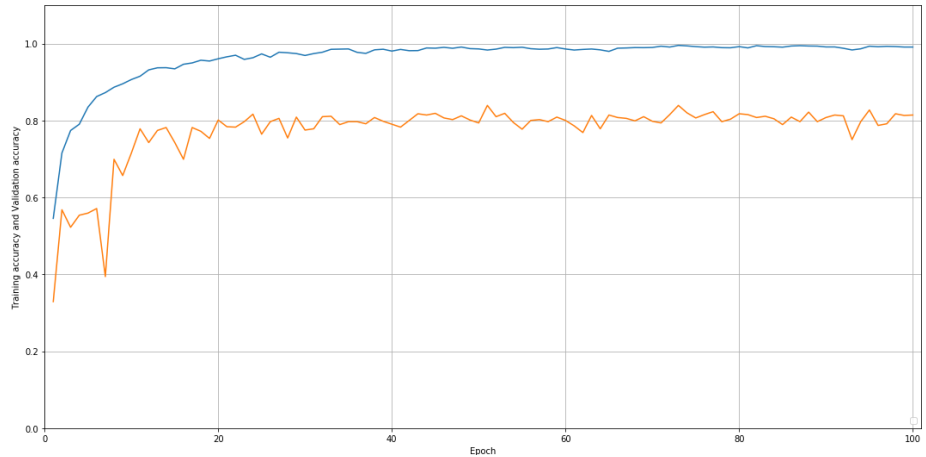
Dense Layer

- 512
- none

둘이 비슷한 성능
→ VGG는 parameter가 많기 때문에 DenseLayer를 많이 쌓거나
weight 갯수를 늘리면 parameter가 너무 많아져서 gradient
vanishing problem이 발생할 수 있음



loss - val_loss



acc - val_acc

Max Acc : 83.93%

Min Loss : 0.9671

>> 기대에 미치지 못한 성능

2. SeyNet(customized Model)

VGG와 비슷한 구조로 직접 설계한 모델

VGG16보다 나은 accuracy를 위해 SeyNet을 시도

다른 조에서 좋은 성능을 보임

데이터와 class가 VGG16 쓰기에 단순해서 gradient vanishing이 발생할 가능성이 있다고 판단

Layer Design

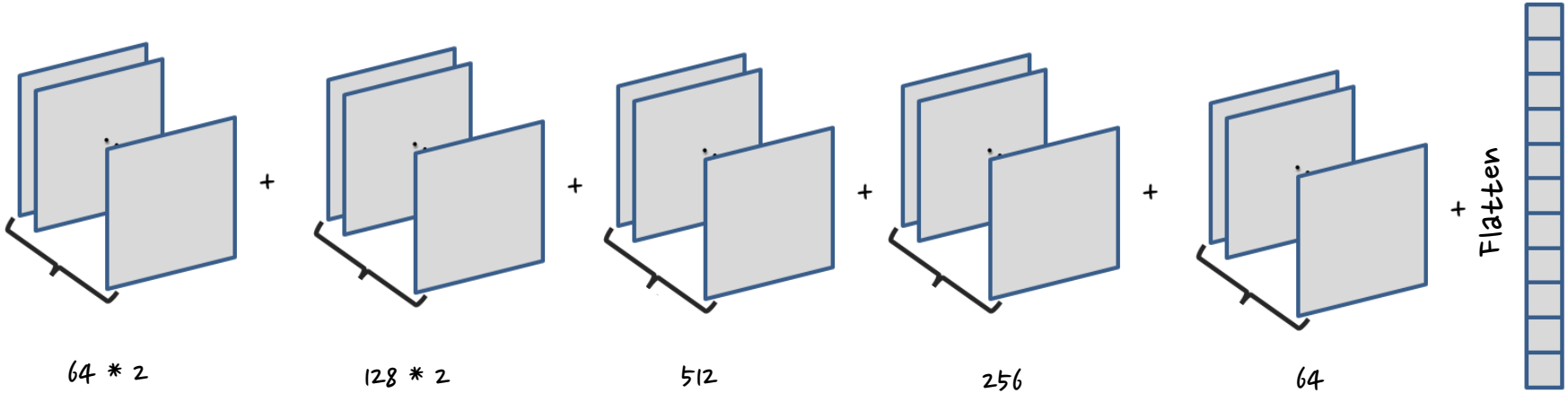
Data Augmentation
- Shift : 0.1
- brightness : 0.8-1.2

Batch Size : 200

2. SeyNet - Model Design

```
# conv 0
model.add(tf.keras.layers.Conv2D(64,3,padding='same',activation='relu',input_shape=(112,112,3)))
model.add(tf.keras.layers.Conv2D(64,3,padding='same',activation='relu'))
model.add(tf.keras.layers.BatchNormalization(momentum=0.85))
model.add(tf.keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))
# conv 2
model.add(tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization(momentum=0.85))
model.add(tf.keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))
# conv 3
model.add(tf.keras.layers.Conv2D(512, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization(momentum=0.85))
model.add(tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)))
# conv 4
model.add(tf.keras.layers.Conv2D(256, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization(momentum=0.85))
model.add(tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)))
# conv 5
model.add(tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization(momentum=0.85))
model.add(tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)))
# dense layers
model.add(tf.keras.layers.Flatten(name='flatten'))
model.add(tf.keras.layers.Dense(len(class_name), activation='softmax', name='dense_10'))
```

2. SeyNet - Model Design



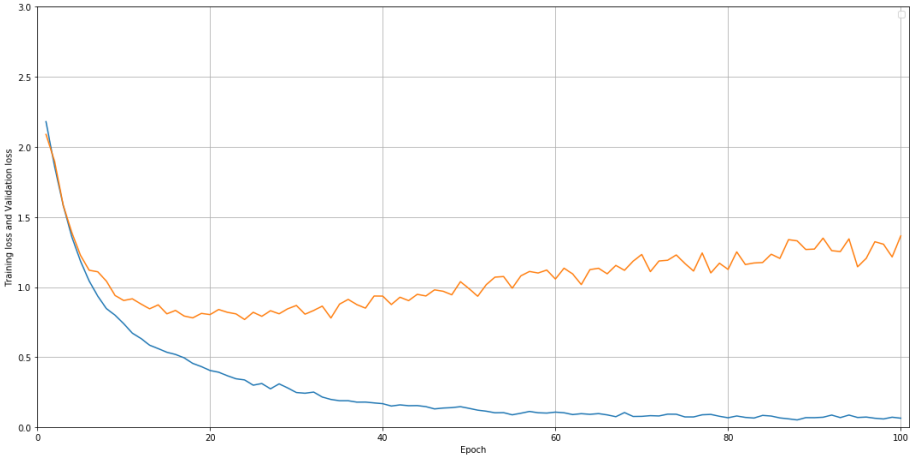
64 * 2
conv2d layer
BatchNormalize
MaxPooling

128 * 2
conv2d layer
BatchNormalize
MaxPooling

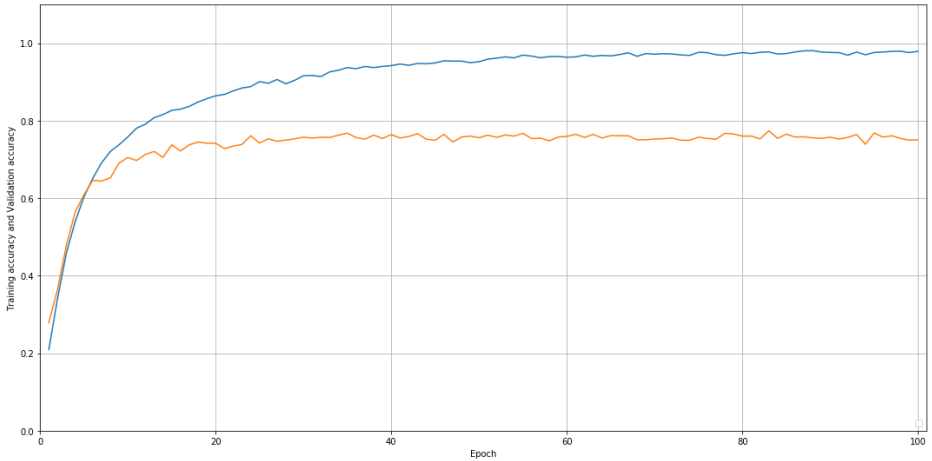
512
conv2d layer
BatchNormalize
MaxPooling

256
conv2d layer
BatchNormalize
MaxPooling

64
conv2d layer
BatchNormalize
MaxPooling



loss - val_loss



acc - val_acc

Max Acc : 78.28%

Min Loss : 0.7809

기대에 미치지 못한 성능

- 이미지의 크기가 크고
- context가 복잡

3. GoogLeNet

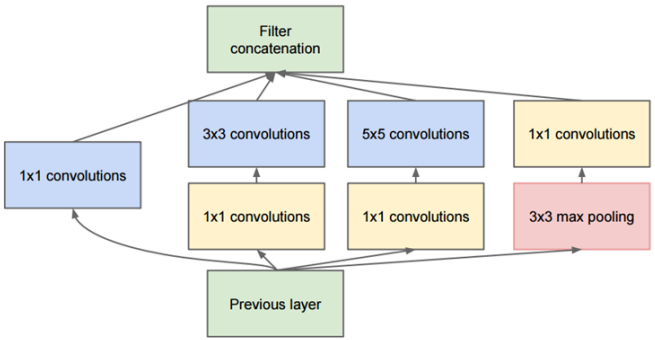
GoogLeNet InceptionV3

22 layers DNN

망이 깊고 넓을수록 연산량, 과적합, vanishing 문제 ↑

깊고 넓은 망의 한계를 Inception module로 극복

이미지 학습, 예측에 탁월한 성능



Full Inception module

기본 모델 사용

- 모든 layer trainable 설정

→ 86%에 가까운 좋은 성능

but, CAM TEST 적용 불가능

GoogLeNet customizing 필요!

GoogLeNet Model customizing

1. 모델 수정 개요

> 기존의 정확도를 살리기 위해 최대한 GoogLeNet 모델에서 많은 layer를 취하고자 함

> Inception module 특성상 mixed layer를 단위로 모델 분리

> **CAM TEST를 위해 적당한 사이즈의 conv layer를 추가**해야 함

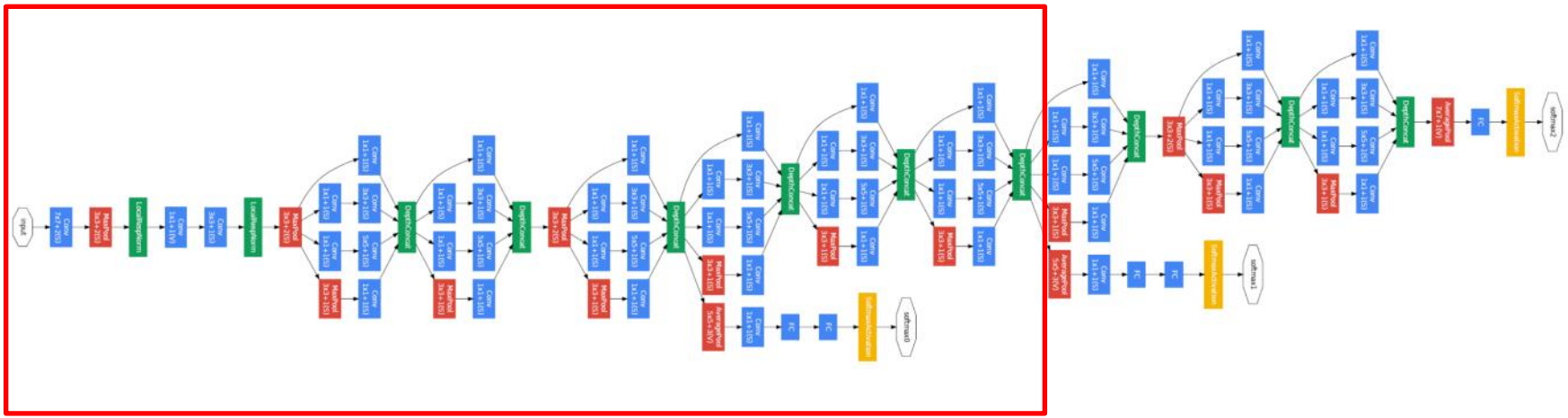
>> (224, 224) image → (12, 12)의 weight를 추출할 수 있도록 mixed 7 layer 선택

2. 실험 시 고려할 요소

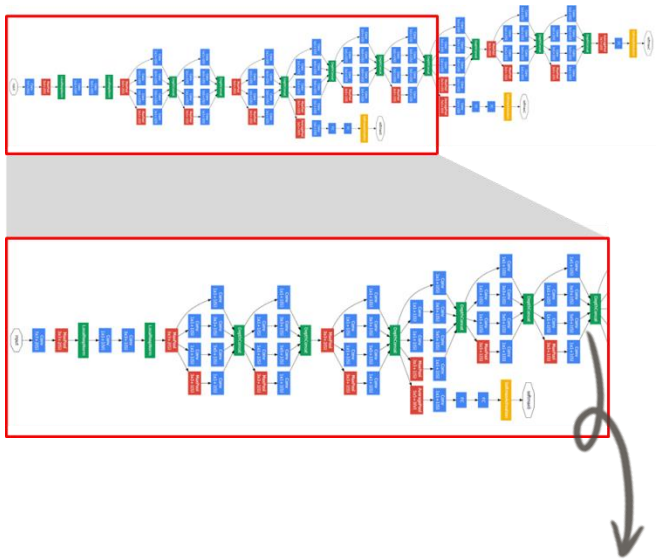
> 기존 GoogLeNet 보다 확연히 적은 layer → custom layer의 형태와 개수

> 학습의 정확도를 개선시키기 위한 hyperparameter

GoogLeNet Model customizing



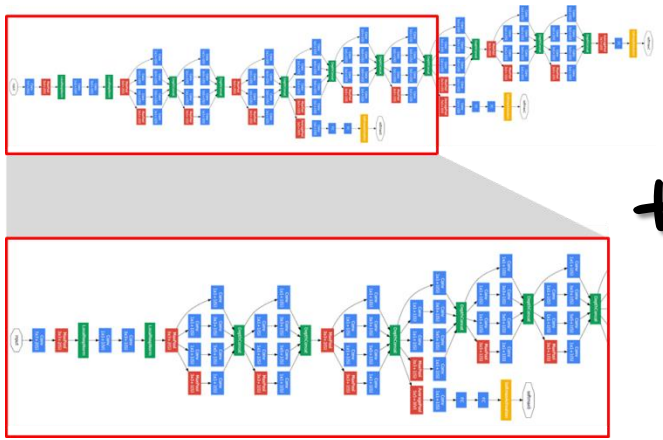
GoogLeNet Model customizing



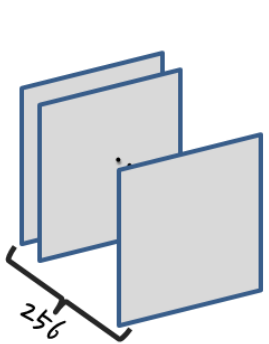
Mixed7 layer

(12, 12, 768)

GoogLeNet Model customizing



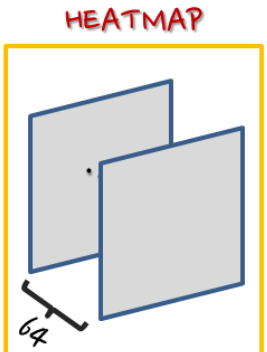
+



conv2d layer
(12, 12, 256)

+

HEATMAP



conv2d layer
(12, 12, 64)

+ Batch Normalization
momentum=0.85

+

Flatten
Dense layer(10)

GoogLeNet Model Refining

custom layer의 형태와 개수

> cam을 위해 (12, 12) conv layer를 꺼내와야 함 → 사진 사이즈 유지 불가피

> 최종 mixed7 layer의 output이 (12, 12, 768)이므로 바로 다음 conv layer에서 256 이상의 필터개수를 사용할 때 80% 이상의 정확도가 보장됨

> 세 개 이상의 conv layer를 쌓아도 정확도 개선 x → conv layer 2개 사용

>> 최종: conv(256, 3) + conv(64, 3) + batch normalization + flatten + dense(10)

GoogLeNet Model Refining

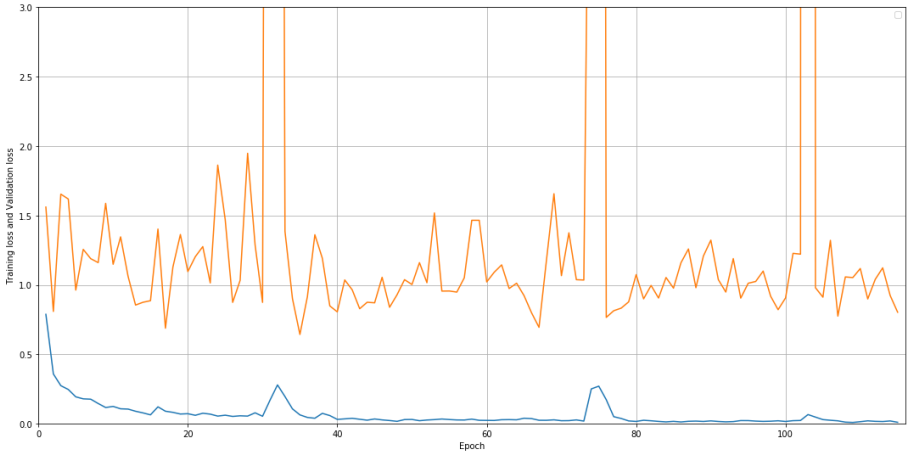
hyperparameter

> batch size, epoch 조절

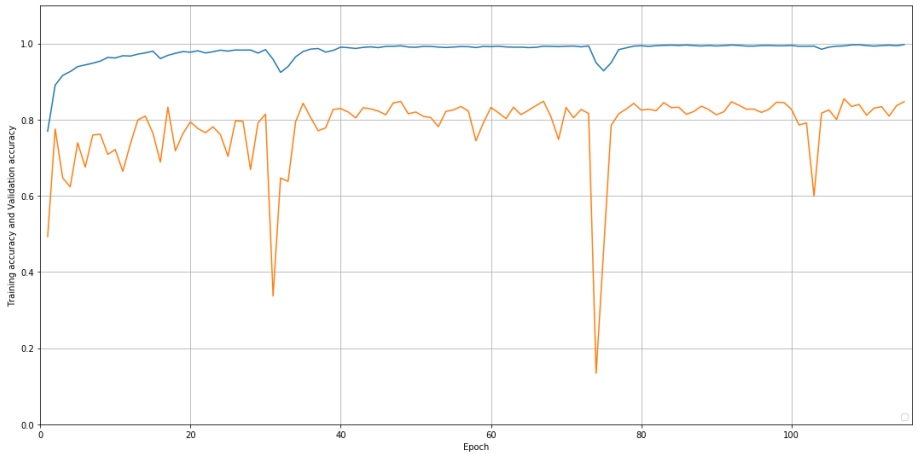
> batch_size : OOM(Out of Memory) Error로 128, 150, 180만 실험 가능

train 이미지 갯수(15299 장)에 비해 batch_size 비율 차가 크지 않아 학습 정도에 큰 차이가 없었음

> epoch : 대부분의 모델이 3~40회 부터 일정 수준 이상의 accuracy를 달성하지 못하고 비슷한 수준에서 수렴하였음



loss - val_loss



acc - val_acc

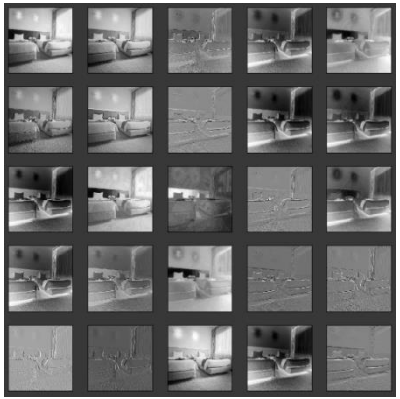
Max accuracy = 86%

GoogLeNet의 특성상 그래프가 불안정하지만

일정 수준으로 수렴하는 것을 볼 수 있음

Feature Map

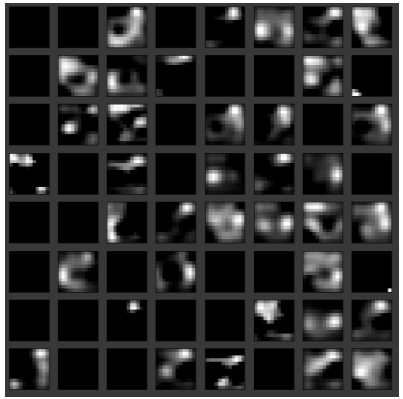
각각의 층에서 추출한 feature map



Layer[1]

Layer[12]

Layer[21]



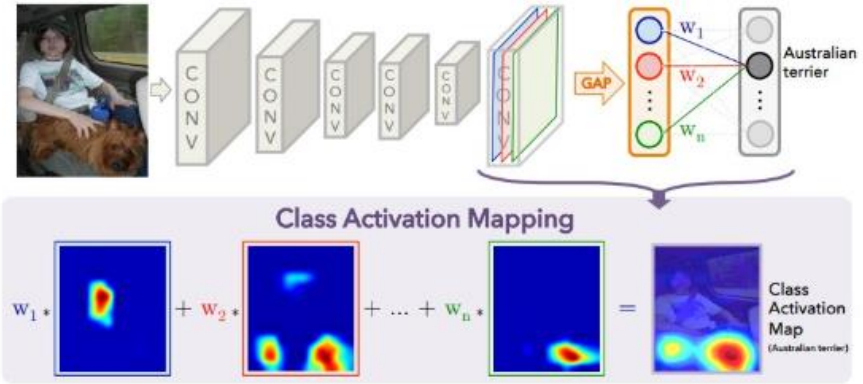
Layer[-4]

cAM (class Activation Map)

1. Single Object Detection
2. Multi Object Detection

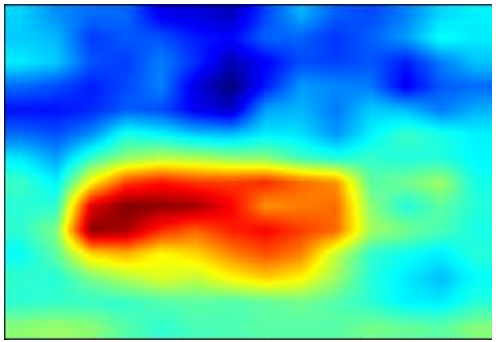
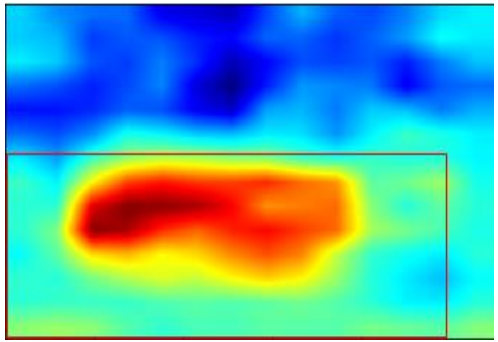
class activation map

- CNN을 통해서 프로그램이 학습한 내용을 해석하고자 함
- 마지막 conv layer에서 특정 class로 분류되는데 영향을 끼치는 weight들의 feature map을 꺼냄
- classification 하고 싶은 사진에 꺼낸 값을 곱해 heatmap으로 표현 가능



VGG-16 (Single) cAM Test

- Heatmap 뽑아내기 : 각각의 Object의 heatmap 추출
- Heatmap을 기반으로 Object Localization 구현

>> 최종 모델로 'customized GoogLeNet' 선택

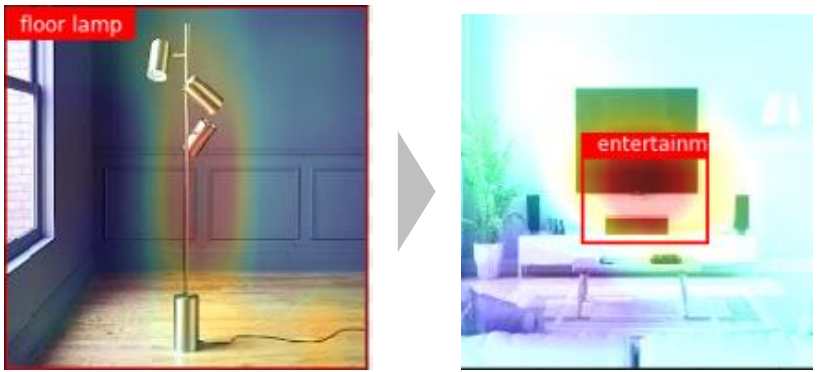
GoogLeNet (Single) cAM Test

VGG와 동일한 방식으로 Bounding box 그리기

- 특정 RGB 값 범위에 들어가는 모든 좌표의 min, max 값으로 box 범위 지정

→ 80% 중반 이상의 정확도를 보여 heatmap에서 사물의 형태를 잘 잡아내는 것처럼 보이지만 정교하지 못함

단순 최대, 최소가 아닌 특정 좌표를 중심으로 box 그림 수정



Multi Object Detection

기존 모델 Multi classification 시도

1위 TV장식장 (99%)

2위 테이블 (0.02%)

1위 침대 (99%)

2위 소파 (0.48%)

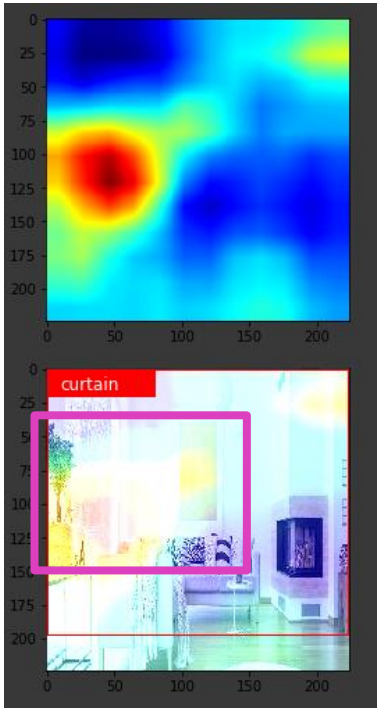
기존 모델 Multi classification 시도

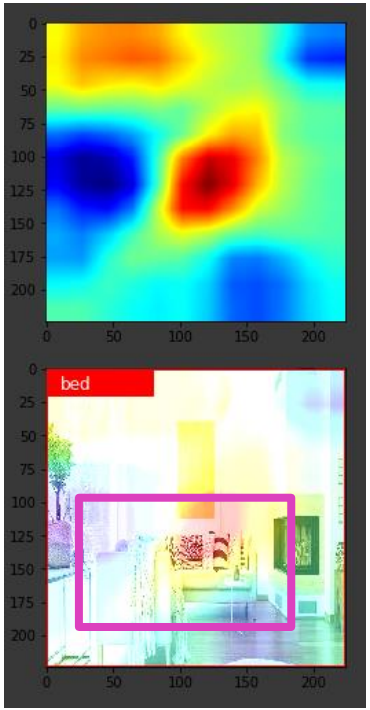
사진 속의 다양한 물체들을
정확하게 감지하는 것으로 보임

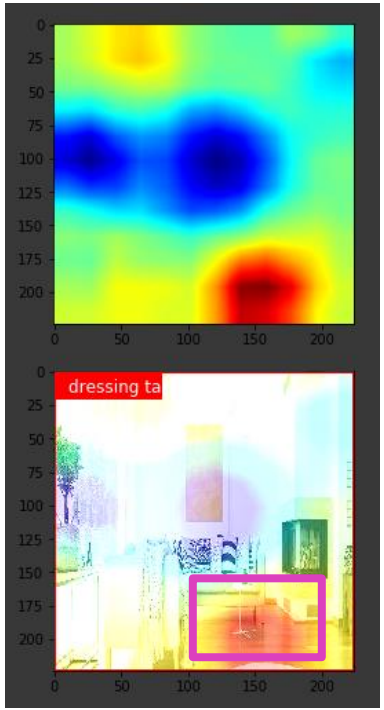
1위 TV장식장 (99%)
2위 테이블 (0.02%)

1위 침대 (99%)
2위 소파 (0.48%)

Multi Object Detection with cAM





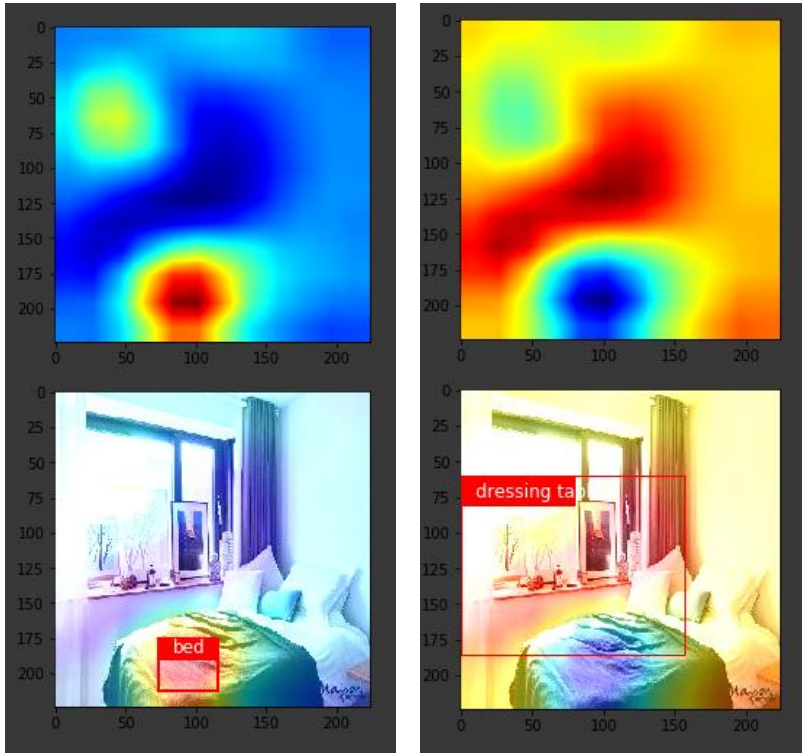


1~3번째로 높게 예측된 사물

- curtain, bed, dressing table

Heatmap에서 각각 사물의 위치가 그럴듯함

Multi Object Detection Test with cAM



Multi Object Detection 문제점 파악

1위 침대

2위 화장대

3위 소파

4위 커튼

>> 3개 이상의 물체를 예측할 시 정밀한 위치 감지가 어려움

Multi Object Detection 문제점 파악

1위로 예측한 개체 외의 Heatmap이 불분명한 경우가 많음

원인)

- 특정 개체에 확률 쏠림 현상
  → 단일 개체 예측을 위해 학습시켰기 때문

해결 방안 1)

- 두 가지 개체의 학습 데이터 이미지를 합쳐서 multi prediction model 학습시키기
- Multi label 이므로 새로운 loss function(binary cross entropy) 필요

해결 방안 2)

- 첫 번째로 확인된 물체를 blur처리하고 다시 single object prediction 실행하기

해결방안 1) 적용 시도

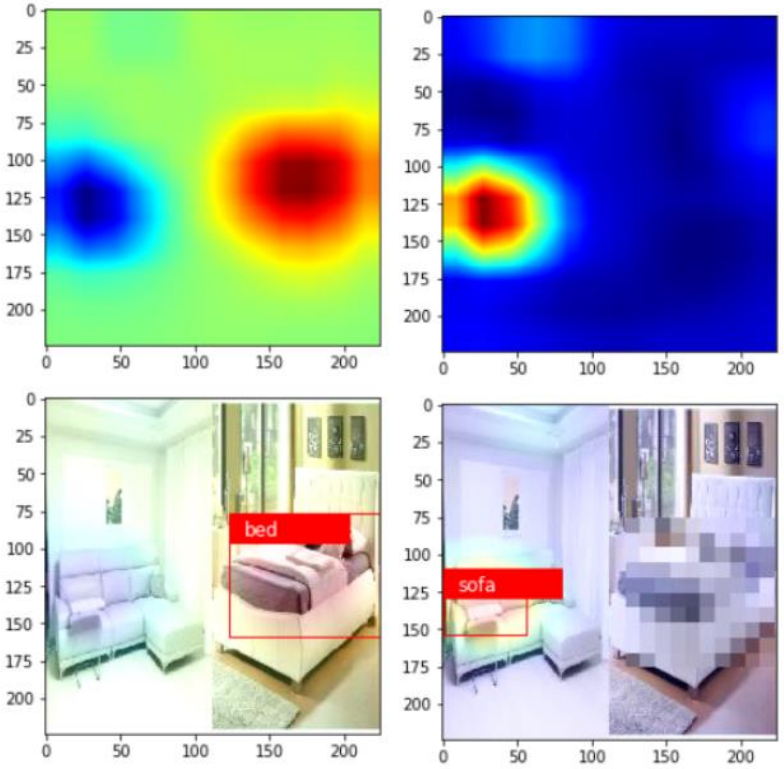
- 두 가지 개체의 학습 데이터 이미지를 합쳐서 multi prediction model 학습시키기
- Multi label 이므로 새로운 loss function(binary cross entropy) 필요

진행 상황)

- 이미지 합성 모듈 완성
- but 시간상의 제약으로 이론적 근거와 모델 수정이 어려워 시도해 보지 못함

>> 추후 multi object dataset을 넣어 학습시킨다면 더 개선될 것으로 기대함

해결방안 2) 적용



1위로 예측된 개체를 정확하게 예측한 경우
2위로 예측된 개체도 정확하게 detect함

한계) 여러 물체를 좀 더 정확하게 예측하지만
Single object detection을 두 번한 것에 불과함

conclusion

의의와 한계

1. Single Object Detection 및 Localization 정확도가 높음
2. GoogLeNet 기반 CAM Test 적용 성공
3. Multi Object Detection 정확도 높음

1. Single Object Detection 기반으로 multi object detection 시
   가장 두드러진 물체에 weight이 편중되는 현상을 보임 → localization 한계
2. Multi Object Detection과 localization에 적합한 dataset을
   구축해서 그에 맞는 모델로 학습했다면 더 좋은 결과를 기대할 수 있었을 것

느낀 점

1. 이미 구축되어 있는 모델을 custom할 때 컨트롤하기 어려워서 많은 시간을 씀
2. Hyperparameter 조정을 좀 더 세심하게 계획해 모델을 개선해 나가야 함
3. 모듈화를 통해 팀 프로젝트를 수월하게 진행하였음
4. 프로젝트를 하면서 추상적으로 느껴졌던 개념을 확실하게 이해할 수 있었음

감사합니다