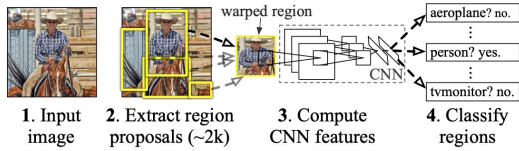


R-CNN



R-CNN: Regions with CNN features



- 전반적인 알고리즘

- ① **Selective search** 알고리즘을 통해 객체가 있을 법한 위치인 후보영역 (Region Proposal) 2000개를 추출하여 적당한 크기로 warp시킨다. (노름에서는 CNN에 넣기 위해 227, 227 사이즈로 resize 시킴)
- ② warp된 region proposal을 **AlexNet**에 입력하여 (2000 x 4096) 크기의 feature vector를 추출한다.
- ③ feature vector를 **linear SVM**과 **Bounding box regressor** 모델에 각각 입력
- 각각 confidence score와 bounding box 좌표를 얻는다.
- ④ 마지막으로 **Non-Maximum-Suppression** 알고리즘의 사용하여 최적의 bounding-box를 출력한다.

- 단점

- ① AlexNet을 그대로 쓰기 위해 이미지를 강제 변형
- ② warping에서 이미지가 왜곡, 손실됨
- ③ 2000개의 후보영역을 CNN에 집어넣어 시간이 오래걸린다.
- ④ selective search 나 SVM은 GPU가 힘들어 CPU로 하라보니 오래걸린다.
- ⑤ CNN, SVM, BoxRegression 모델 3개가 합쳐진거라서 end-to-end 훈련이 안된다. 3개의 모델 목적이 다르기때문
SVM, BoxRegression은 학습이 되어도 CNN은 안된다.

Selective search

- Bounding Box들을 찾아준다.

- hierarchical grouping algorithm (계층적 그룹화 알고리즘) 방식 사용

- 알고리즘 순서

- R: 선택된 region 후보들 $\{r_1, r_2, \dots\}$ → 초기에도 비슷할 것까지
- S: region들의 유사도 집합 $\{s(r_1, r_2), \dots\}$ 목표다.

초기화

① R 초기화

유사성이 '0' 일때까지 혹은 정한 임계 값이 될때까지 반복

- ① 유사성이 가장 높은 $s(r_i, r_j)$ 선택
- ② r_i 와 r_j 를 합친다. r_n 생성
- ③ r_i 와 r_j 가 포함된 $s(r_i, r_k), s(r_j, r_k)$ 삭제
- ④ 합쳐진 r_n 와 나머지 region들의 새로운 유사성 $s(r_n, r_k)$ 계산
- ⑤ 2번에서 만들어진 r_n 와 4번에서 만들어진 $s(r_n, r_k)$ 를 R 집합, S 집합에 포함시킨다.
- ⑥ 하나의 Region이 될 때까지 반복



- 유사성

- $[0, 1]$ 사이로 정규화된 4가지 요소 (color, Texture, size, Fill) 들의 가중치 합
- 식

$$S(r_i, r_j) = a_1 \cdot S_{color}(r_i, r_j) + a_2 \cdot S_{texture}(r_i, r_j) + a_3 \cdot S_{size}(r_i, r_j) + a_4 \cdot S_{fill}(r_i, r_j)$$
- 가중치 $A = \{a_1, a_2, a_3, a_4\}$ 값들은 조절 할 수 없지만 논문에서는 동일시켰다.
- $S_{color, texture}$ 등은 다음장에서 설명

• color

- 각 채널을 25개 bin 설정 (내 예상 이렇게 만들어준다 (예) 35억 $[1, 0.1, 0, 0, 0, 0, 0]$, 25억 $[1, 0, 0, \dots]$, ...)
- 각 region 마다 컬러 히스토그램 설정 $C_i = C_i^1 \dots C_i^h$
- 채널의 수는 $h = 75$ (3채널 $\times 25$)
- 정규화 진행
- 안정한 regions의 교집합을 유사도 측정
- $S_{color}(r_i, r_j) = \sum_{k=1}^{h=75} \text{Min}(C_i^k, C_j^k)$
- 새로 생성된 r_k 의 컬러 히스토그램 C_k 는 다음식을 따른다.
 - $$C_k = \frac{\text{size}(r_i) \times C_i + \text{size}(r_j) \times C_j}{\text{size}(r_i) + \text{size}(r_j)}$$
- r_k 의 사이즈는 간단히 $\text{size}(r_i) + \text{size}(r_j)$

• Texture


- 주변 Pixel 값들의 변화량 [참고자료: HoG] (이것까진 못보겠다)
- $S_{texture}(r_i, r_j) = \sum_{k=1}^N \min(t_i^k, t_j^k)$

• size

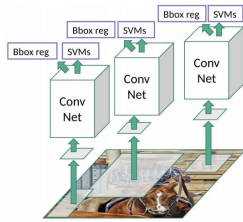
- 사이즈가 적을수록 유사도 크다
- $S_{size}(r_i, r_j) = 1 - \frac{\text{size}(r_i) + \text{size}(r_j)}{\text{size}(\text{원이미지})}$

• Fill

- candidate Bounding Box 와 Region들의 사건의 차이가 적을수록 유사도 높다.
- 식

$$S_{fill}(r_i, r_j) = 1 - \frac{\text{size}(BB_{ij}) - \text{size}(r_i) - \text{size}(r_j)}{\text{size}(\text{원이미지})}$$
- 예상 :  두개가 겹쳐서 경쟁구도 가설했을 때 경쟁사 - 자기 개인것 같다.

Alex Net

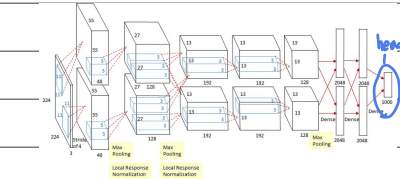


- 2000개의 후보영역을 Fine tune된 AlexNet에 입력하여 2000 x 4096 크기의 feature vector를 추출한다.

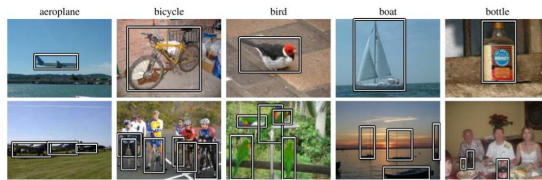
Fine tuning Pre-trained Alexnet

- 모델준비

• AlexNet의 구조 head 부분을 **찾으려는 객체 + 1개**로 수정한다.

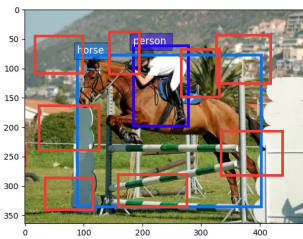


- 데이터 준비



① 위의 사진처럼 라벨링된 사진을 준비한다.

② selective search 알고리즘을 사용하면 아래처럼 될것이다 (예시라서 몇개안만들)



→ 성능특정에 설명해줌

③ bounding box 과 ground truth box 와의 IOU 값을 구한다.

④ IOU값이 0.5 이상이면 positive sample (객체)

⑤ IOU값이 0.5미만이면 negative sample (배경) 으로 저장한다.

⑥ 객체 사진 32개, 배경사진 96개 → 총 128개를 배치 '1'로 둔다.

- 준비된 데이터와 모델구조를 활용하여 Fine-tuning시킨다.

linear SVM

성능측정에 설명했다.

- (2000x4096) Feature vector를 입력 받아 class를 반환하고 confidence score를 반환한다.
- SVM는 이진분류기라서 배경을 포함한 (N+1)개의 독립적인 linear SVM 모델을 학습시켜야 한다.
- 학습이 끝나면 hard negative mining 기법을 적용하여 재학습한다.
- Soft max로 분류하면 map score가 54.2 → 50.9로 떨어진다. '성능측정' 파트

linear SVM 학습

- 학습 데이터

- 이전 AlexNet tuning할때 IOU를 기준으로 0.5 이상은 객체 미만은 배경으로웠다.

하지만 SVM 학습 데이터라고는 기준은 다르다.

- IOU가 0.3 미만인 것을 배경
- ground truth Boxes 인것만 객체

▶ 한마이크 사진에 보면 라벨링 된것만 쓴다 (overfitting 될것같지만 이때 당시 데이터가 많지 않았다. 그래서 성능을 올릴꺼면 이렇게 해야했다.)

hard negative mining

- 왜 쓰는가?

- negative (배경) 데이터가 많은 클래스 불균형으로 False Positive가 많이 발생한다.

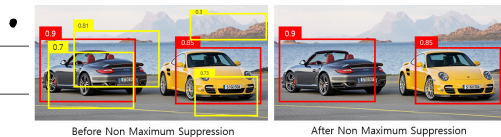
- 어떻게 쓰는가?

- False Positive 데이터를 모아서 학습과정에서 추가하여 재학습시킨다.

Non Maximum Suppression

- bounding Box에서 비슷한 위치의 Box 삭제해서 가장 적합한 Box 찾는 알고리즘

- 알고리즘

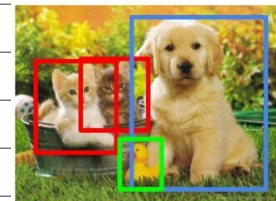


① threshold 0.5로 그 아래 것들은 삭제

① 가장 큰 값고른다. 0.9

② 0.9와 IOU를 측정해서 0인것은 뱉고 threshold 0.5보다 큰값은 삭제한다.

③ 남은 0.85도 같은 방식으로 줄여나간다.



이건 사진 때문에
threshold를 기준으로
삭제할지 말지 결정한다.

Bounding box Regressor

- selective search 알고리즘을 통해 얻은 객체의 위치는 부정확 할수있다.

- 그래서 bounding box 좌표를 변환하여 객체의 위치를 세밀하게 조정해주는 Bounding box regressor 모델 사용한다.

자세한 설명

- 정의

regressor 안에 들어가는 값 (Predicted Box) $P^i = (p_x^i, p_y^i, p_w^i, p_h^i) = (x좌표, y좌표, width, height)$

맞았으면 하는 정답 값 (ground truth box) $G = (G_x, G_y, G_w, G_h) = (위장 같다)$

- 목표

- Predicted box 가 ground truth box 와 유사하도록 학습한다.

- transformation 함수

- Predicted box를 옮기는 함수로

$$d_x(P) = W_x^T \phi_5(P)$$

- 적용하면 다음과 같다.

이렇게 이해하면 식이 쉬어진다.

$$\hat{G}_x = P_w \cdot d_x(P) + P_x \quad \Rightarrow \text{원래 위치 } P_x \text{로부터 얼마나 움직일지 } [P_w \cdot d_x(P)] \text{ 결정한다.}$$

$$\hat{G}_y = P_h \cdot d_y(P) + P_y \quad \Rightarrow \quad "$$

$$\hat{G}_w = P_w \cdot \exp(d_w(P)) \quad \Rightarrow \text{기준의 넓이 } P_w \text{를 몇배 } d_w(P) \text{만큼 키울지 결정한다}$$

$$\hat{G}_h = P_h \cdot \exp(d_h(P)) \quad \Rightarrow \quad "$$

- 실제 값과 예측 값의 차이 (±)

- 위에 \hat{G} 값은 예측이다 Loss 를 구하기 위해서 정답 값이 필요하다

- 이는 예측한 위치에서 실제 위치까지 실제 얼마나 차이가 났는지를 계산한 것이다.

$$t_x = (G_x - P_x) / P_w$$

$$t_y = (G_y - P_y) / P_h$$

$$t_w = \log_2(G_w / P_w)$$

$$t_h = \log_2(G_h / P_h)$$

- Loss 계산

$$Loss = \underset{\hat{W}}{\text{argmin}} \sum_i^N (t_i - \hat{W}^T \phi_5(P^i))^2 + \lambda ||\hat{W}||^2$$

argmin 이유 예: 4개의 x, y, w, h 값이 연관되어있어



정답

이렇게 예측하고 argmin을 하면 Loss를 굳이 4개 할 필요 없이 x 위치, y 위치를 조금씩 수정 할수있다?