ICV final project report

2016-12600

박기범

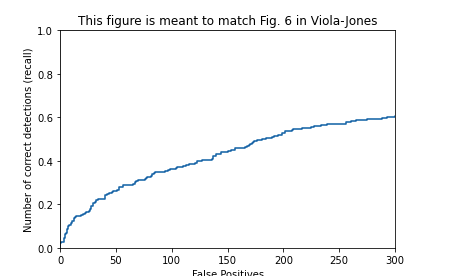
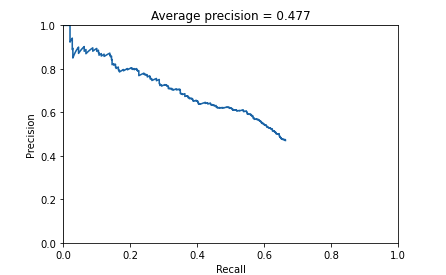
1.

주피터 노트북 p1에 있으며 그 안에 결과가 들어가 있습니다.

Sklearn의 라이브러리에서 hog를 제작 하였고, 이 과정에서 파지티브 이미지의 경우 이미지가 깨진 관계로 160 96에서 가져와서 크롭을 하였습니다. 네거티브 이미지의 경우 각 사진마다 100장의 128 64 랜덤 장소 크롭을 하였습니다.

이제 sklearn의 clf.fit으로 트레이닝을 진행 한 이후에 스케일을 원래 128\*64 부터 128의 k배가 이미지에 꽉 찰 때 까지 k를 1.1배 늘려가며 스케일 피미드를 제작 하였고 슬라이딩 윈도우로 모든 이미지 전반에 걸쳐 디텍션을 진행 하였습니다. nms의 경우 둘이 겹치되 0.3 이상 겹치거나 하나가 아예 포함 대는 경우에 전자의 경우 더 confidence가 큰 쪽을 후자의 경우 더 큰쪽을 채택 하였습니다.

그 결과



가 나왔습니다.



잘 안댔던 경우로는 위와 같이 자전거 바퀴등을 인식한 경우가 많았고 nms 과정에서 여러 사람을 합치는 경우도 많았습니다.

(자세한 사진들은 노트북에 있습니다.)

2.

초반부의 샘플링 까지는 1번과 같습니다.

이제 각 윈도별 로케이션과 사이즈를 정해 주어야 하는데 논문에서는 5031개를 썼지만 제가 재현한 바로는 4977개이상 구하지 못하였습니다. 구한 기준은 12~64중 4 혹은 6의 배수 선택, 만약 32 이상이고 8로 나누어진다면 8 을 스텝으로 선택 그 외에 6으로 나누어진다면 6 4로 나누어진다면 4로 정한 후 각 스텝별로 xmin,ymin,xmax,ymax등을 정해준 후 location\_list에 넣어 주었습니다.

이제 파지티브와 네거티브 샘플을 구해야 하는데 너무 시간이 오래 걸린 관계로 성능 향상을 위해 import threading을 이용하여 병렬 처리 하였습니다. negative\_1, positive\_1를 이용하여 각각 피처를 구하였습니다.

그리고 n개의 레이어에 k개의 클레시파이어 안에 3개의 클레시파이어 정보(클래시파이어 본체,웨이트, 로케이션 인덱스)를 가지고 있는 clf\_layer를 만듭니다.

이제 본격적으로 svm을 돌려야 하는데 마찬가지로 피팅도 병렬로 하기위해 하나의 svm트레이닝을 하는 svmfitter를 구현한 후에 multisvm을 사용하여 한번에 50개씩 svm을 트레이닝 한 후에

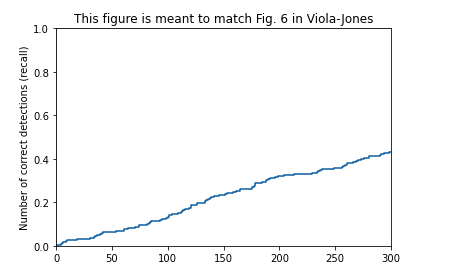
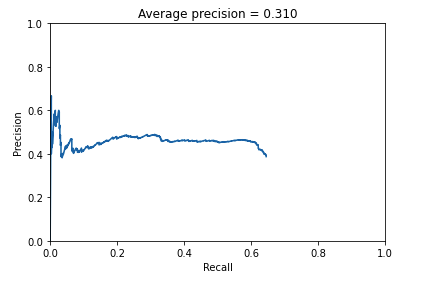
그 중 제일 성능이 좋은 클래시파이어를 골라서 그 클래시파이어의 location\_list 인덱스를 selectedlist에 추가해주고 clf\_layer의 해당하는층의 맨 마지막에 (클래시파이어 본체,웨이트, 로케이션 인덱스) 정보를 추가해 줍니다.

이렇게 하나씩 svm을 더할 때 마다 sum(svm출력\*w) 과 threshold를 비교하여 정답을 정답이라고 외치는 비율이 99.75%가 될때 까지 threshold를 낮춥니다. 이 조건을 만족할 때에 오답을 정답이라고 말하는 비율이 0.7 이하가 될 떄 까지 svm을 더해줍니다. (추가로 svm을 더해줄 때에 원래라면 샘플링할 확률을 조정하여 틀린애들을 더 강조해 주어야 하지만, 구현에 어려움을 겪던 와중에 sample\_weight라는 변수를 알게 되었는데 이 변수는 더 큰 값에 해당하는 인덱스의 변수를 더 강조해서 트레이닝하게 합니다. 이러면 전체 확률은 낮아질 지언정 합한 확률을 올라갈 것입니다. Error를 구하는 방법은 true\_value\*W1\*w2…wk 와 출력값을 비교한 후에 최대가 1 최소가 0으로 정규화 하여 넣어 주었습니다.)

한번 0.7이하가 댔다면 해당 레이어는 끝난 것이고 다음 레이어를 새로 만들어서 작업합니다. 여기서 중요한 점은 이번에 걸러진 네거티브 데이터는 다음 레이어로 가지 않습니다.

이제 이밸류에이션 파트 입니다. 1과 거의 같지만 대신에 detect\_cascade라는 함수를 만들어 각 층마다 threshold(threshold\_list에 있습니다)를 만족한다면 다음 레이어로 가는식으로 구현하였습니다.

결과는 오히려 1번에 비해 훨씬 느렸는데 이는 4번에서도 말하듯이 인테그랄 이미지 구현의 실패로 인한 것입니다.

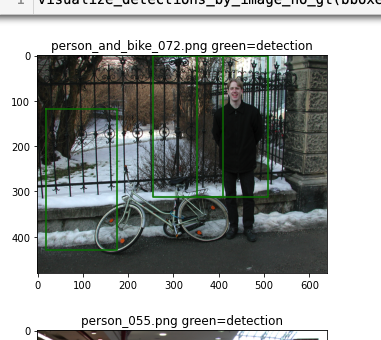
 

구현 오류로 워낙 느리게 진행되어 제대로 피팅을 못하고 처음 결과만을 올리자면 위와 같이 나옵니다. 여기서 보면 특이한 점은 0.310이라는 숫자와 정확도 대비 false positive 대비 recall값만 준수합니다.(비교적으로) 이는 각 레이어에서 웬만하면 yes를 외치기 때문으로 보입니다.

3. lbp의 경우 1의 코드 위에 그대로 피처 익스트랙션만 sklearn의 feature를 (local\_binary\_filter)사용하여 구현 하였습니다. 2와 똑같은 이유로 인테그랄 이미지 구현의 실패로 인한 것입니다.



결과적으로 큰 사진들의 경우 크게 문제가 없었지만,



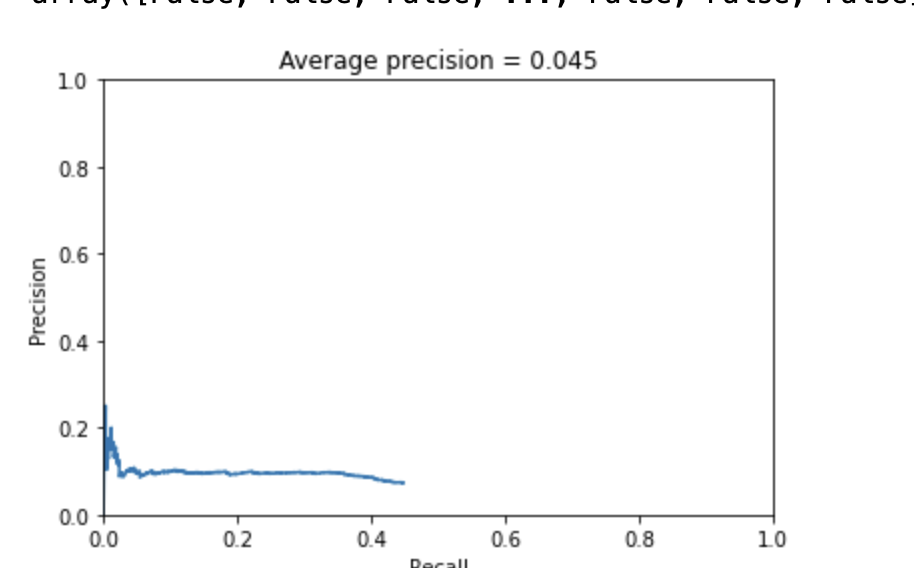
위와 같이 false positive가 많이 보였습니다.

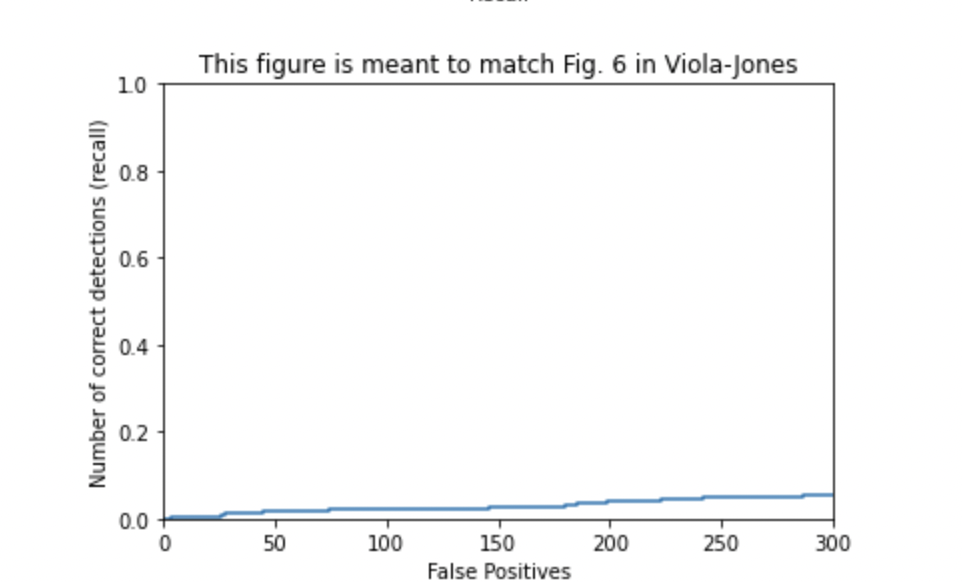
이를 해결해 보려고 threshold 를 조절해 봤지만

텍스트, 실외, 자전거, 남자이(가) 표시된 사진

자동 생성된 설명

이 문제를 해결 하려면 lbp를 adaboost method로 진행해야 햇었던것 같습니다.

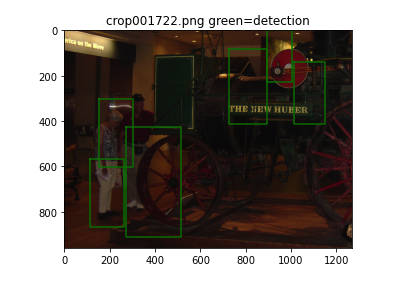




4.

틀린 에러들을 보면서 알 수 있는것이 있었는데 비교적 작은 애들이 훨씬 많은 false detection을 만들어 냅니다. 이는 여러 이유가 있겠지만 작은애들이 훨씬 수가 많기 때문 인 것으로 판단하였습니다.

따라서 탐지 사각형의 크기에 따라 다른 쓰레쉬 홀드를 적용하자 1번의 accuracy가 0.2 =>0.3으로 띄었고 네거티브 데이터의 더 효율적인 샘플링을 위해 랜덤 사이즈 만큼 네거티브 데이터를 확대하여 샘플을 만들어 낸후에



이렇게 밝기에 따라 다른 사진을 잘 감지 하지 못하였기에 사진을의 normalization을 이용하여 조정을 하였습니다.

마지막으로 2,3의 경우 1보다 훨씬 느리게 결과값이 나왔는데 이는 제가 이미지 인테그랄까지 구현하지 못하였기에 이런 일이 일어난 것으로 보입니다.

추가로 1번에 엣지를 이용한 피처도 넣어서 다시한번 트레이닝을 해보고 싶습니다.