2016312761 여혁수 인공지능프로젝트 Final project report

Explanation of the preprocessing the data (if any; e.g. data augmentation, validation split, etc.)

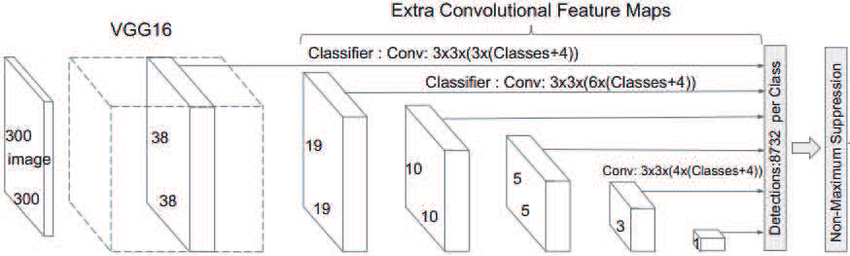
우선 이미지 데이터의 정보를 담고 있는 annotation 폴더의 각 이미지에 해당하는 파일을 파싱을 했습니다. 이미지에 있는 object의 이름을 담고 있는 <name>부분, object들이 이미지에서 어느 위치에 있는 지를 담고 있는 <bndbox>, 위치 좌표를 의미하는 <xmin>, <ymin>, <xmax>, <ymax>, 그리고 학습시키기 어려운 이미지를 의미하는 <difficult>까지 파싱하여 받아냅니다. 그래서 object가 존재하지 않는 이미지는 제외하고, train 데이터에 이미지와 정보들을 포함시킵니다.

Test 데이터도 같은 방식으로 preprocessing을 진행하는데, 이 때는 object가 존재하지 않는 이미지라도 데이터에 포함을 시킵니다. 또한 제가 test 데이터로 VOC2007 데이터셋을 사용하였는데, 일부는 train 데이터에도 있는 이미지여서 그것들은 test 데이터에서 제외시켰습니다.

Data augmentation은 별도로 진행하지 않았고, 모델에서 인풋 데이터로 300x300사이즈의 이미지를 받기 위해서 transform을 진행했습니다. Train, test데이터 모두 공통적으로 resize를 통해 사이즈를 300x300으로 해주었고, 빠르고 성능 좋은 학습을 위해 normalization을 진행했습니다. train데이터를 대상으로는 photometric\_distortion, expand, randomCrop, horizontalflip 등의 기술을 추가로 사용하였습니다. 데이터의 개수 증가는 없이 오래 학습을 하는 방향으로 진행하였습니다. Validation split은 할 필요가 없다고 느껴 하지 않고 모든 데이터를 train데이터로 학습에 넣었습니다.

Explanation of the model architecture

SSD300 모델과 같은 구조를 가진 모델을 만들었습니다.



위 그림과 같은 구조가 됩니다. 우선 처음에는 인풋 데이터가 VGG16과 같은 구조를 가진 convolutional layers를 지나게 됩니다. 여기서 여러 convolutional layer 중 conv4\_3 layer에서의 중간 결과를 받아내고 conv7 layer까지 진행한 마지막 결과도 받아냅니다. 2개의 결과를 따로 저장해놓고 conv7 layer까지의 결과를 다시 인풋으로 하여 여러 크게 4개의 convolutional layer를 더 지나게 됩니다. 여기서는 1개의 conv layer를 지날 때마다의 중간 결과를 모두 저장합니다. 이렇게 하면 각기 진행한 학습의 복잡도가 다른 6개의 결과가 생깁니다. 이제 이것들로 각각 object의 위치와 그 object가 맞을 확률을 예측합니다. 학습 구조는 위치를 예측하는 1개의 conv layer, 맞을 확률을 예측하는 1개의 conv layer로 구성되어 있습니다. 예측하는 layer까지 거치면 한 이미지에 8732개의 예측 결과가 생기게 됩니다. 이 때 동일한 object로 예측한 결과가 여러 개가 생기게 되는데, 그 중 가장 확률 값이 높은 결과만 남기고 나머지는 제거하는 것이 필요합니다. 그래서 non maximum suppression 기법을 참고하여 마지막 layer를 만들고 학습시켜 마무리합니다.

Process and results of training (e.g. loss change, scheduler, early stopping, performance, optimization, experiments, hyperparameters, examples, metrics, etc.)

우선 이미지를 모델에 집어넣어 object의 위치와 object가 정답일 확률을 예측합니다. 그 후 실제 object의 위치에 대한 정보와 그 object의 이름 정보를 예측 결과와 비교하여 MultiBoxLoss를 구합니다. 그리고 나서 back propagation을 진행합니다. 이 때 사용되는 optimizer는 stochastic gradient descent로 하나의 batch마다 기울기를 변화 값을 계산합니다. 그래서 나온 결과로 모델의 매개변수 값들을 갱신합니다. 이 과정을 iteration 만큼 계속 반복하다가 전체 반복 횟수의 75%만큼 진행했을 때 learning rate를 10% 낮춥니다. Gradient exploding 을 막고 좀 더 안정적인 학습을 위해 75% 정도 진행했다면 거의 학습이 수렴을 했다고 보고 learning rate를 낮추는 것입니다. 85% 진행했을 때 한 번 더 10% 낮춥니다.

Result)

기본 hyperparameters 값입니다.

batch\_size = 32

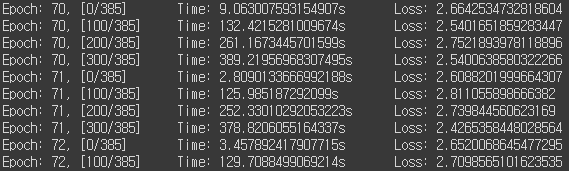
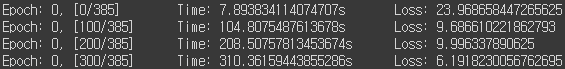
iterations = 60000

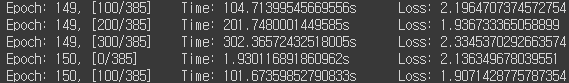
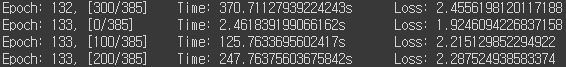
print\_every = 100

learning\_rate = 0.001

momentum = 0.9

weight\_decay = 5e-4

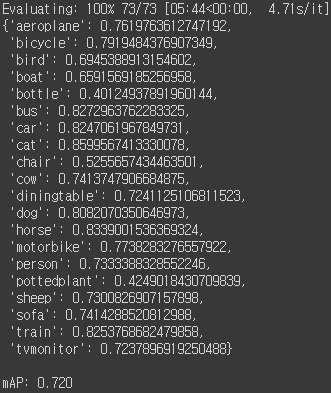
텍스트이(가) 표시된 사진

자동 생성된 설명

위 로그들은 현재 진행중인 epoch, 몇 개의 batch를 학습했는지, epoch별로 실행 시간, 그리고 loss value를 각각 나타냅니다.

진행도에 따라 Loss가 점점 낮아지는 것을 볼 수가 있습니다. Epoch을 늘려 학습을 더 진행했을때 더 좋은 성능을 낼 수 있지만 너무 오래 걸려서 iterations을 60000으로 설정하여 156epoch 만큼 학습시켰습니다.

테스트는 VOC2007 데이터로 train과 겹치지 않는 것만 뽑아 해보았습니다. 이미지 데이터의 수는 약 4600개 정도 되었습니다.



각 라벨 별로 AP 값이 다음과 같고, 평균을 낸 mAP값은 약 0.72 정도가 나왔습니다. Epoch을 원하는 만큼 돌리지는 못했지만 꽤 좋은 성능을 보여주는 것 같습니다.