인공지능 기말 프로젝트

<흉부 X-ray 촬영 사진을 CNN 학습하여 정상과 폐렴으로 분류하기>

산업융합학부 정보융합전공

2016054602

김나래

1. 요약

본 기말 프로젝트로 Kaggle에 공개되어 있는 흉부 X-ray 촬영 사진을 사용하여 정상인 폐와 폐렴을 가진 폐로 분류하는 학습을 진행하였다. Python Keras를 이용한 CNN을 구성, 이미지를 학습하여 정상/폐렴으로 이진 분류하였다.

2. 주제

2.1 주제 소개

Kaggle에 공개되어 있는 흉부 X-ray 촬영 사진을 정상 또는 폐렴(비정상) 으로 이진 분류한다. Python Keras를 이용, CNN을 구성하여 학습한다. 이미지의 사이즈를 두 가지(64x64, 128x128)로 나누어 예측 정확도와 성능에 차이가 있는지 확인하고, evalution metrics를 통해 전체적인 통계를 확인한다.

2.2 주제 선정 이유

분류에 사용될 이미지는 흉부 X-ray 촬영 사진으로, 컬러가 아닌 흑백의 이미지이다. 또한 X-ray 촬영 사진인 만큼, 그 형태가 매우 유사하다.(사진의 구도 등) CNN을 통해 흑백의 이미지는 얼마나 잘 분류해 낼 수 있는지를 테스트해보고자 한다.

3. 사용 데이터

Kaggle에 공개되어 있는 광저우의 병원에서 촬영된 정상, 폐렴 환자들의 흉부 X-ray 사진을 이용하였다. 학습에 사용된 이미지는 정상 1,342개/폐렴 3876개이고 모델 평가를 위해 사용된 테스트 이미지는 정상 234개/폐렴 390개를 사용했다.

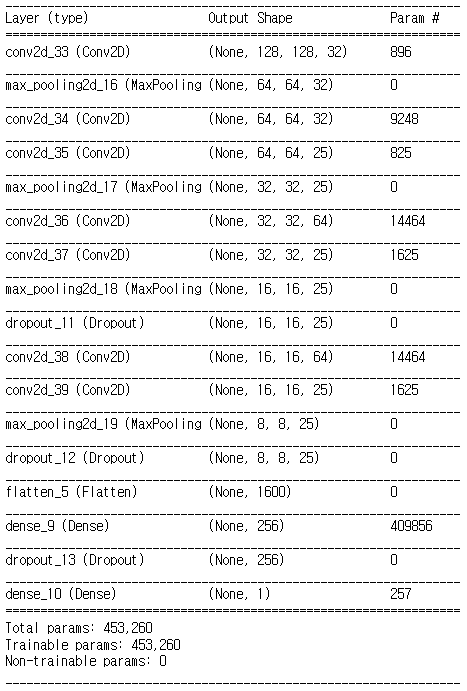
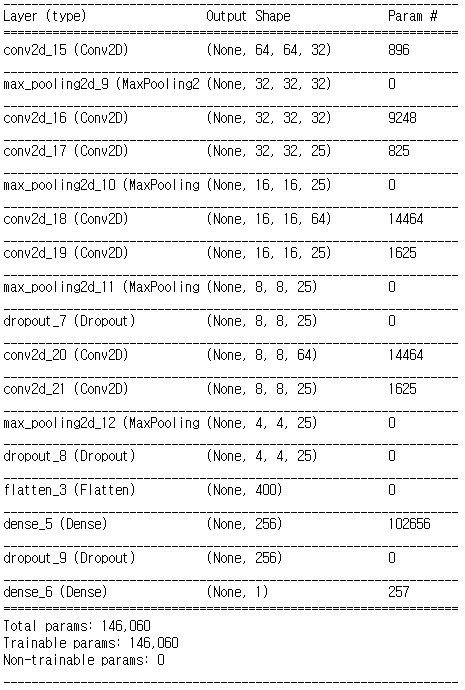
X-ray 사진의 특성 상 사진의 구도 등 형태가 정상과 폐렴의 경우 모두 유사하고, 흑백의 이미지이다. 정상과 폐렴의 경우를 카테고리화 하기 위해 폴더를 각각 두어 미리 분류되어 있다.

각 폴더를 카테고리화 하여 정상이면 0, 폐렴이면 1로 결과 레이블을 지정하였다. 학습 이미지는 64x64, 128x128로 조정한 후 255로 나누어 값을 normalization 하였다.

4. CNN 구성 및 학습

CNN의 구성은 모두 keras로 구성되었다. 학습에 사용한 CNN은 4개의 Conv Layer로 구성하고 오버피팅을 방지하기 위해 DropOut을 사용했다. activation은 마지막 출력 층을 제외하고 relu를 사용하였으며 출력층에서는 sigmoid를 사용하였다. 수행 중 연산량이 늘어나고 학습이 오래 걸려 성능 개선을 위한 출력 파라미터의 개수 감소를 위해 Conv Layer 다음 1x1의 Conv Layer를 연속하여 쌓는 Network In Network 방법을 적용하였다. 그에 따라 64x64를 예로, 초기 30만개에 달하던 파라미터의 개수가 약 15만개로 감소하였다. 모델은 64x64와 128x128을 따로 나누어 저장하여 사용했다. 정상/폐렴의 이진 분류이므로 binary\_crossentropy 사용, val\_loss 값이 2번 동안 변화가 없으면 학습을 중지하도록 EarlyStopping 옵션을 적용했다. 각 모델의 요약은 아래 [그림 1]과 같다.

**[그림 1. 64x64, 128x128 모델 요약]**



학습은 batch\_size는 128, epochs는 100마다 수행하였으며 각 회에 20%의 데이터는 검증용으로 사용하도록 validation\_split 값을 적용했다.

5. 결과 분석

각 모델의 학습에 소요된 시간과 test 셋을 이용해 각 모델을 평가한 정확도는 아래 [표 1]과 같다. 이하 input 이미지 형태에 따라 모델을 분류하여 64x64 모델은 64모델, 128x128 모델은 128 모델이라 명명한다.

[표 1. 모델별 학습 소요 시간과 정확도(acc)]

|  |  |  |
| --- | --- | --- |
|  | 학습 소요 시간 | 정확도 |
| 64모델 | 7분 5초 | 0.83 |
| 128모델 | 20분 | 0.75 |

각 모델 별 epoch에 따른 loss 값과 acc값의 변화를 그래프로 그렸을 때 아래 [그림 2]와 같다.

[그림 2. 모델 별 loss, acc 값 변화 그래프]

|  |  |
| --- | --- |
| 64모델 | 128모델 |
| C:\Users\Narae\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\34CC27B1.tmp | C:\Users\Narae\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\CEDB35E7.tmp |

이중 정확도(acc) 값만 따로 추출하여 그래프를 그렸을 때 아래 [그림 3]과 같다.

|  |  |
| --- | --- |
| 64모델 | 128모델 |
| C:\Users\Narae\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E1B6FF8D.tmp | C:\Users\Narae\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5796C623.tmp |

[그림 3]을 보면, 그래프는 전체적으로 128모델이 안정적이나, [표 1]에서 알 수 있듯이 정확도는 64모델이 높음을 알 수 있다. 128모델은 시간은 64모델에 비해 약 3배가 오래 걸렸지만 결과적으로 정확도는 더 낮았다. 또한 수행된 로그를 보면, EarlyStopping 옵션에 의해 epoch가 6에서 멈추었다. 좀 더 세부적인 학습 결과를 보기 위해 모델별로 test 데이터를 이용해 예측을 수행한 후, 결과 값을 정제하여 아래 [그림 4]와 같이 출력했다. 각 모델의 accuracy, precision, recall, f1 score, classification\_report, confusion\_matrix 순으로 출력한 결과이다.

[그림 4. 예측 후 세부 통계 비교]

|  |  |
| --- | --- |
| 64모델 | 128모델 |
|  |  |

평가 결과 정확도가 높게 나왔던 64모델이 전체적으로 값이 높음을 알 수 있다.

통계 값 결과와 소요 시간 등을 함께 비교하여 보았을 때, 64모델이 상대적으로 더 좋은 성능을 나타낸다고 판단할 수 있다. 흑백의 이미지를 잘 분류해낼 수 있는지에 대해서는 64모델의 정확도가 약 0.83이므로 낮지 않은 것으로 보인다. 다운받은 데이터셋에 테스트용으로 16개의 이미지가 별도로 제공되었고, 이를 64모델에 적용한 결과 6개의 이미지를 제외하고 모두 올바르게 분류해냈다. 그러나 폐렴인 경우에는 모두 폐렴으로 잘 분류했으나 정상인데 폐렴으로 분류한 경우가 6개이다. 위 [그림 4]에서도 알 수 있듯이 0(정상)인 경우에 대해서는 recall 값이 현저히 낮다.

본 프로젝트에서 학습 전, 128x128로 리사이징한 128모델이 상대적으로 나은 결과를 보일 것이라 예상했으나 그 반대였다. 위와 같은 결과가 도출된 이유에 대해서 몇 가지를 생각해보았다.

1) 흑백의 이미지이므로 크기가 커진다 해도 성능에 유효한 차이가 없다.

2) 성능 문제로 EarlyStopping 옵션을 2로 하여 128모델이 충분한 성능이 나오지 않았다.

3) 성능 문제로 연산을 줄이기 위해 1x1 Conv Layer를 적용한 것이 결과에 영향을 미쳤다.

6. 향후 개선방안

우선 CNN 모델의 개선이 필요하겠다. 지금은 성능 등의 문제로 간단한 모델을 구현했으나, 다른 CNN 구현체들(AlextNet, ResNet 등)을 적용해보고 비교해보도록 한다.

프로젝트 전 예측한 결과와 달리 64모델이 상대적으로 더 나은 성능을 보인 것에 대해서는 5번 마지막에 기술한 이유를 바탕으로 다시 모델을 구성하여 수행해 볼 필요가 있다. 64x64, 128x128, 256x256 등 이미지 크기에 영향이 있는지를 보기 위해 리사이징을 한 경우를 더 수행한다. EarlyStopping 옵션을 넉넉히 하여 모델이 충분한 성능을 발휘하도록 한다. 또한 성능 문제로 인해 연산을 줄이기 위해 사용했던 1x1 Conv Layer를 제거하여 수행해본다.

통계 값을 보았을 때, 정상인 경우에 대한 recall 값이 폐렴인 경우에 대한 값보다 현저히 낮다. 이 부분은 정상 경우인 데이터를 더 수집하여 학습을 수행해야 할 것으로 생각된다.

7. 느낀점

이미지를 분류하는 방법에 대해 궁금했는데, 실제로 프로젝트를 통해 구현을 해보면서 정보를 찾다 보니, 이미지 분류에 대한 무궁무진한 방법이 있음을 알게 되었다. CNN을 어떻게 구성하는 지에 따라 성능에 많은 영향이 있었으며 Keras를 통해 구현하였는데, 구현 시의 옵션 값들을 어떻게 적절히 조절하는 지도 매우 중요하다는 것을 느꼈다.

노트북의 성능 때문에 좀 더 다양한 시도를 하지 못한 것이 아쉽다. 처음에 이미지 사이즈를 256x256으로 리사이징하여 시도해보았으나 도저히 종료될 기미가 보이지 않아 포기하고 최대한으로 늘려본 것이 128x128 이었다.

이미 오픈되어 있는 많은 CNN 구현체들을 사용해보지 못한 것이 제일 아쉽게 느껴진다.

* 코드는 아래 GitHub에 업로드 함.