프로젝트 주제

수업시간에는 주로 흑백 이미지를 다뤘기 때문에, 이번에는 컬러 이미지를 사용해보기로 했다. 첫 번째 기획은 한식 이미지 분류기를 주제로 삼았었지만, 데이터셋의 복잡도와 본인 컴퓨터 gpu의 성능한계로 인해 비교적 간단한 가위바위보 손 모양 분류기를 만들기로 최종 결정했다.

첨부된 파일 별 설명

모든 파일에 대한 좀 더 자세한 설명은 readme.txt를 함께 첨부했으며 여기서는 python 프로그램 파일에 대해서만 설명한다.

save_npdata.py는 미리 설치된 데이터셋을 정제하여 .npy로 저장한다.

이미지 한 개를 ImageDataGenerator를 활용하여 조금씩 변형된 10개로 늘려서 모델의 input size에 맞게 조절 후 저장한다. 전체 데이터를 8 : 1 : 1 의 비율로 train : valid : test 로 나누어 저장한다.

train.py는 생성해둔 train, valid 데이터를 사용하여 모델을 학습한다. 모델 구성은 이 후에 자세히 설명하겠다.

test.py는 테스트데이터 예측 및 모델 성능 지표를 확인한다.

gui.py는 모델을 활용해 직접 가위바위보를 할 수 있도록 하는 응용프로그램이다.

실행시킬 컴퓨터의 웹 캠이 동작하는 경우 정상적으로 사용 가능하다. 학습시킨 이미지 데이터를 보면 흰 배경에 손만 달랑 있는 이미지이기 때문에 실제 상황에서의 배경 등을 고려하지 않고 학습되었다. 그렇기 때문에 웹 캠에 흰색 종이를 배경삼고 그 위에 손만 보이게 하는 것이 그나마 정상적으로 분류된다. gui.py를 실행 시킨 후 게임시작 버튼을 누르면 안내창에 1.5초 간격으로 가위.. 바위.. 보!! 문구가 표기된다. 여기서 보!! 타이밍에 gui에서 출력 되고 있는 웹 캠의 현재 프레임을 input값으로 정제하여 학습시킨 모델로 예측한다. 그 결과와 컴퓨터의 랜

덤 값을 계산해 가위바위보 결과를 출력한다. 프로그램을 실행도중 게임 승률을 계산해서 보여주고, 게임결과는 이전에 실행한 결과들에 더해 로그로 누적한다.

데이터셋 소개

https://laurencemoroney.com/datasets.html의 Rock Paper Scissors Dataset을 사용했다.

가위 바위 보 각 label별로 흰색 배경안에 모양과 인종이 다른 손모양들이 975개씩 있다.

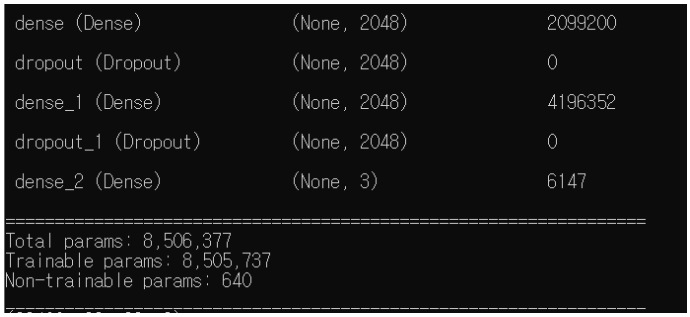
해당 데이터셋을 제공하는 페이지에서 train, valid, test 데이터를 따로 제공하지만, 수가 적기 때문에 imageDataGenerator를 활용한 python코드로 데이터를 불리기 수월하도록 한 폴더에 정리하였다. 데이터를 불려서 train data 23400개 valid data 2925개 test data 2940개가 되었다.

모델 세부 구조

기본적으로 Sequantial 모델이고 layer한 개씩 순차적으로 쌓으면서, 한 iterate만 학습을 진행시켜 진행속도와 검증 에러율을 확인하며 결과가 좋은 것을 따라가는 방식으로 모델을 구성한 후 학습시켰다. 먼저 입력 layer는 image size가 96x96이고 컬러 이미지 이기 때문에 (96,96,3) 크기다. 컬러 이미지는 r,g,v 행렬 값을 정규화 하여 학습시키는 것이 효과적이기 때문에 LayerNormalization을 사용하여 input layer의 마지막 차원 기준으로 정규화 하여 학습에 사용한다. 이미지 학습에 효과적인 Convolution2D layer를 주로 사용하였으며, BatchNormalization layer를

통해 학습 중간중간 연산결과를 정규화 하여 모델의 일반화 성능을 높인다. Convolution2D layer를 사용할 때 중요 데이터만 추출하기 위해 MaxPooling2D layer를 사용하고, 추출된 데이터의 형태는 유지하기 위해 ZeroPadding2D layer를 사용한다. 몇 단계의 합성 곱 신경망이 끝나면 Dense Layer를 사용하여 출력층을 구현해야 하기 때문에 Flatten layer를 사용해 1차원 데이터로 변형한다. 한 층이 분류 문제에 필요한 일부 정보를 누락하면 그 다음 층에서 이를 복원할 방법이 없어 정보의 병목이 될 수 있기 때문에 중간 계층의 차원은 적지 않게 했다. 과적합 방지를 위해 Dropout layer도 몇 개 더해준 후 마지막으로 출력 label은 3개(가위,바위,보) 활성함수는 출력 값을 확률로 사용할 수 있는 softmax를 사용하였다.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer_normalization (LayerN  (None, 96, 96, 3)        6
ormalization)

conv2d (Conv2D)              (None, 22, 22, 64)        23296

batch_normalization (BatchN  (None, 22, 22, 64)        256
ormalization)

max_pooling2d (MaxPooling2D  (None, 11, 11, 64)        0
)

zero_padding2d (ZeroPadding  (None, 15, 15, 64)        0
2D)

conv2d_1 (Conv2D)            (None, 11, 11, 256)       409856

batch_normalization_1 (Batc  (None, 11, 11, 256)       1024
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 5, 5, 256)         0
2D)

zero_padding2d_1 (ZeroPaddi  (None, 7, 7, 256)         0
ng2D)

conv2d_2 (Conv2D)            (None, 5, 5, 256)         590080

zero_padding2d_2 (ZeroPaddi  (None, 7, 7, 256)         0
ng2D)

conv2d_3 (Conv2D)            (None, 5, 5, 256)         590080

zero_padding2d_3 (ZeroPaddi  (None, 7, 7, 256)         0
ng2D)

conv2d_4 (Conv2D)            (None, 5, 5, 256)         590080

max_pooling2d_2 (MaxPooling  (None, 2, 2, 256)         0
2D)

flatten (Flatten)            (None, 1024)              0
```



모델의 loss 함수는 다중분류에 자주 활용되는 categorical_crossentropy를 사용하였고, metrics는 label을 one-hot encoding한 후 categorical_accuracy를 사용했다.

배치 학습, 검증

Gpu의 성능 한계로 인해 모든 데이터를 한 번에 학습시킬 수 없었기 때문에 train 데이터를 한 iterate마다 인덱스를 랜덤 초기화 하여 256개씩 나누어 train_on_batch 하였다. 마찬가지도 valid 데이터 또한 256개씩 나누어 evaluate하였고 각 배치마다 loss와 accuracy를 모은 후 평균으로 전체 valid loss와 accuracy를 구해 valid 정확도를 계산했다.

최종 모델 성능 평가

최종 모델의 성능은 세 군데에서 파악했다.

먼저 모델을 학습시키는 도중에 학습 정확도와 검증 정확도를 파악할 수 있었다. 학습 정확도는 학습 데이터에 따라 모델을 학습시키기 때문에 모델 학습이 진행되고 있다는 정도로만 확인했고 거의 100% 까지도 올라갔지만, 검증 에러율이 5% 미만이 될 때 학습을 중단시켰더니, 학습 정확도는 98.82%, 검증 정확도는 95.34%에서 중단됐다. 해당 로그는 train_Log.txt에서 확인할 수 있다.

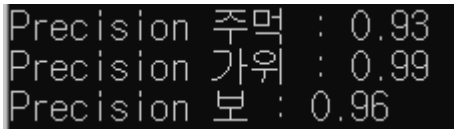
다음으로 test 데이터를 통해 모델의 성능을 검증하였다. 모델의 Accuracy, Precision, Recall, F1-score 4가지 기준을 통해 측정했다.

Accuracy(정확도)는 전체 데이터 중 예측이 맞은 데이터의 비율로 가장 직관적인

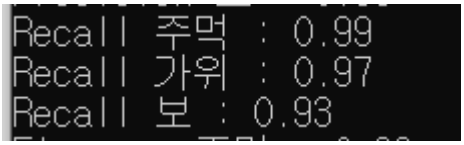
평가 지표이다. 최종 모델에서 전체 test 데이터에 대한 Accuracy는 96.19%이다.

나머지 세가지는 각 label별로 따로 계산했다.

Precision(정밀도)는 모델이 어떤 class라고 예측한 수 중 실제로 해당 클래스인 수의 비율이다.



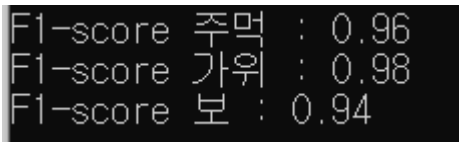
Recall(재현율)은 실제 class의 개수 중 해당 클래스라고 예측한 비율이다.



Precision과 Recall는 단일 지표로만 볼 경우 왜곡할 수 있고, 서로 상충하는 경우가 많기 때문에 두 값 중 하나라도 값이 낮아지면 크게 낮아지는 값을 구하기위해 조화평균을 구하여 사용한다.

$$F_\beta = (1 + \beta^2)\,(\mathrm{precision} \times \mathrm{recall}) \,/\, (\beta^2\,\mathrm{precision} + \mathrm{recall})$$

이렇게 계산된 값을 F-score라고 하는데 여기서 β가 1인 값을 F1-score라고 한다.



Test 데이터를 통한 성능 검증 또한 test_Log.txt에서 확인할 수 있다.

마지막으로 응용프로그램에서 모델이 예측을 잘하는지 확인해 보았다. 검증 오류율이 비슷한 모델이더라도 어떤 것은 현장에서 가위를 잘 구별하지 못했고, 어떤 것은 바위를 잘 구별하지 못했다. 최종적으로 선정한 모델은 본인이 낸 손모양을 100개중에 2~30개 정도만 제대로 분별했다. 학습시킨 데이터셋과 환경을 유사하도록 하얀 배경에서 손모양만 나온 것을 인식시키면 그나마 분류를 잘 해낸다.

모델의 성능을 더 높이기 위한 개선 방안

수집한 데이터셋 내에서의 검증, 테스트 정확도는 높은 편이기 때문에 만족하지만, 현장(응용프로그램)에서 손모양을 더 잘 구별하기 위해 어떻게 하면 좋을지 생각해 보았다. 데이터셋의 하얀 배경 부분에 랜덤으로 배경을 입히는 식으로 데이터셋을 추가하고, 데이터 셋이 복잡성이 올라갔으므로 모델 또한 더 깊게 구성하여 학습한다면 웹 캠 화면의 손모양도 분류를 잘 할 것이다. 그리고 이 데이터셋은 쥐다가 만 주먹이나 손가락을 세 개 편 가위 같은 모양이 많아서 내가 보기에도 주먹인지 보인지 헷갈리는 데이터들이 존재한다. 그래서 적절한 가위바위보 손 모양 데이터셋을 더 수집한다면 모델의 성능 또한 높아질 것이다.

결론

최종적으로 선정한 모델은 valid, test 데이터에 대한 예측 결과가 좋은 것을 선정하여 첨부하였지만, 응용프로그램에서는 아쉬운 분류 성능을 보인다. 특이한 점은 프로젝트를 진행하며 다양한 파라미터로 학습시킨 모델들 중 오히려 검증 정확도가 70~90%인 모델이 응용프로그램에서의 분류는 좀 더 잘한다는 점이였다. 학습시킨 데이터셋은 기본적으론 하얀 배경 (흰색천을 두고 찍은 것 같음)에 맨 손이 팔뚝부터 손끝까지 나와있는 이미지 들이다. 이것들을 조금씩 변형시켜 사용하긴 했지만, 크게 차이는 없을 것이기에, 응용프로그램을 실행한 우리집 배경과는 상황이 달라 분류를 제대로 하지 못한 것으로 보인다. 인공지능을 실제 사용할 곳의 데이터와 충분히 유사한 데이터 셋을 통해 학습시켜야 한다는 것을 알았다.