**Machine Learning Assignment\_3**

조 병 웅

2019312570

원전공 사학과/복수전공 인공지능 융합전공

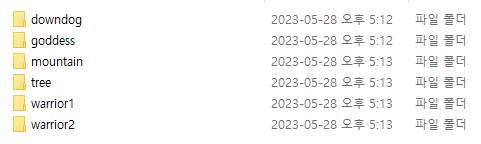
1. **Problem Definition**

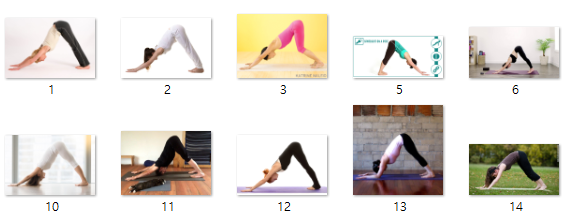
이번 프로젝트는 6가지의 요가 자세들의 이미지를 학습하여 test 데이터 이미지들이 들어왔을 때 그 이미지를 알맞은 요가 자세로 분류(classification)하는 문제이다. train데이터는 각 자세가 라벨링된 디렉터리 안에 각 자세의 이미지들로 분류되어 주어졌으며, 정답은 라벨 데이터가 없이 test디렉터리 안에서 섞여 있는 이미지들이다. 필자는 다중분류를 해야하는 이번 프로젝트의 특성상, softmax를 활용하여 다중분류를 수행하는 CNN 모델을 구축했다.

1. **Data**
   1. **Data Description**

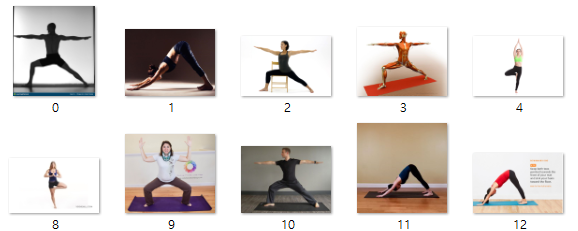
input data:

input 데이터셋 디렉터리 train은 6가지의 각 요가 자세 별 이미지 데이터들의 집합인 sub-folder 6개로 이루어져 있다. 각 sub-folder의 이름은 요가 자세의 label이고, 내부의 이미지가 label/class에 해당하는 이미지 데이터이다. 각 이미지 데이터들의 크기가 다양하며, 용량또한 다양하기 때문에, 이후 이미지 전처리 과정에서 이미지의 크기 조정이 필요함을 알 수 있다. 또한 ‘downdog’와 같이 라벨링된 값들을 0, 1과 같은 softmax의 값이 될 수 있는 라벨링 값으로 조정해야할 필요성이 있음을 알 수 있다.

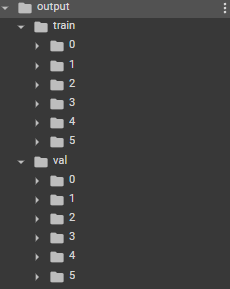




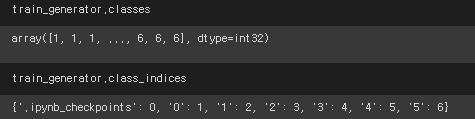
또 다른 input 데이터셋 디렉터리인 test는 train과 다르게 sub-folder로 라벨링되어있지 않다. 이 데이터셋은 각 데이터들이 모두 섞여 있는 모습을 보인다.



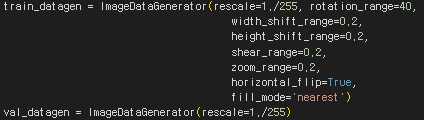
* 1. **Data Pre-processing**

먼저, 드라이브에 저장된 이미지 파일들을 환경(코랩)으로 불러들이기 위한 코드가 필요하다. 그러나 이러한 과정을 수행하기 전에, Train 데이터셋을 훈련한 뒤 이를 평가하기 위해 validation 셋을 먼저 나눌 필요가 있다. 따라서 필자는 먼저 드라이브 상에 있는 train셋의 서브 셋의 라벨링을 str에서 int형으로 바꾼 뒤, splitfolders 함수를 사용하여 8:2 비율로 train 데이터 셋과 validation 데이터 셋으로 나누었다.

그 후 flow\_from\_directory 함수를 활용해 해당 디렉터리 데이터 셋을 불러오는 geerator를 생성하였는데, 이때 적절한 이미지를 학습시키기 위해 배치 사이즈를 16으로 하는 224, 224 크기의 이미지들을 불러오도록 코드를 작성했다. train 셋 데이터를 불러올 generator는 미리 나누었던 /train디렉터리에서, val 셋 데이터는 /val 디렉터리에서 불러오도록 하였다. 이때 각 generator의 클래스의 분배는 다음과 같다.(체크포인트가 들어갔기 때문에, 실제 라벨링과 클래스는 차이가 있다. 후에 softmax를 거친후 submission할 때는 그 차이를 없어줘야 함을 알 수 있다.



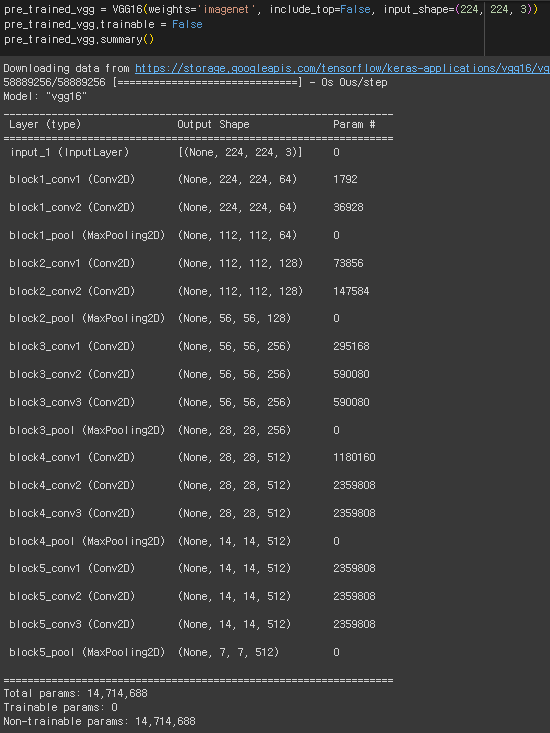
또한 이때 generator를 만들 때 ImageDataGenerator 클래스를 사용하였는데, 이는 수집한 데이터의 양이 몇천개의 단위로 큰 단위가 아니었기 때문에, 회전과 값 변경같은 이미지 변형을 통해 데이터를 증강시키고자 여러 파라미터를 주어 데이터를 형성하였다. 이를 통해 이미지 셋을 학습시킬 때 오버피팅을 막고 더 일반화된 훈련 모델을 형성할 수 있도록 만들었다.



보다시피, 줌, 가로로 뒤집기, shift등 다양한 파라미터를 통해 데이터 증강을 수행했으며, 또한 rescale을 통해 데이터의 스케일링을 수행하였다. train generator와 달리, val 데이터셋은 말 그대로 훈련을 위한 검증 데이터이므로 스케일링만을 수행했다.

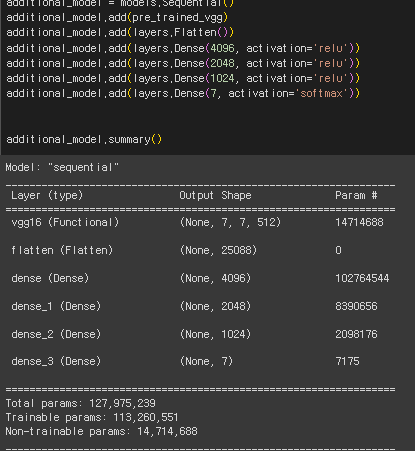
1. **Analysis**

이러한 과정들을 거치면서 데이터들을 학습에 알맞은 크기로 조장할 수 있었다. 특히, 데이터 증강을 수행했을 때와 수행하지 않았을 때의 validation loss의 변화가 매우 분명하게 나타났다. 데이터 증강을 수행하지 않았을 때는 모델이 90퍼센트의 정확도를 보였지만, epoch가 반복될 수록 매우 극심한 validation loss의 변폭을 보여주었다. 이는 모델이 다소 오버피팅되며 validation 검증에서 안 좋은 결과를 내었다고 생각한다. 때문에 오버피팅을 해소하고자 이미지 데이터를 변형하며 데이터 증강을 수행하였을 때, validation loss가 안정적으로 감소되는 모습을 보여주었고 또한 높은 변화는 아니지만 모델의 정확도 또한 92퍼센트로 증가하는 모습을 보여주었다.

1. **Model**
   1. **Classifier**

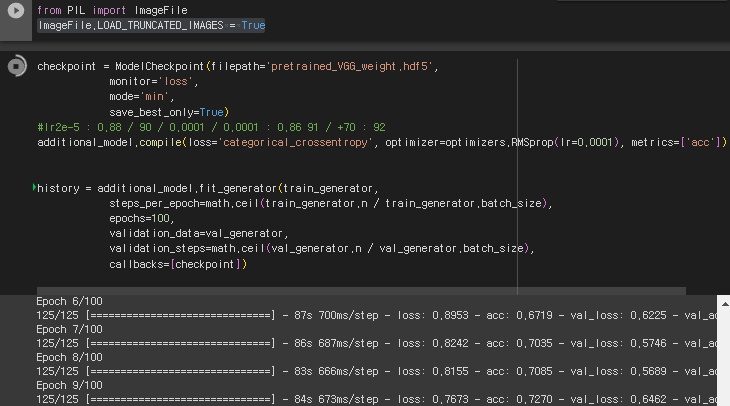
분류기는 수업 때 배웠던 keras, 그 중에서도 사전훈련된 네트워크 중 하나인 VGG 16을 활용하였다. 이 구조를 활용한 이유는 먼저 사용하기 쉽고 간단하며, 이미지 분류에 활용된 모델이기에 이미지 분류에 좋은 성능을 낼 수 있을 것이라고 기대했기 때문이다. 필자는 VGG16 이미지 넷 모델에, 필자의 fc 레이어 층을 추가하여 훈련시켜 모델을 만들었다.

* 1. **Design Consideration**

모델을 사용하면서, 따로 freeze를 하지는 않았고, fc레이어를 따로 구현했기 때문에 include\_top을 False로 주었다. 또한 파라미터들을 이미지넷 학습된 값에서 사용하기 위해 trainable 속성에 False****를 주었다.

이후 fc레이어를 추가하면서 flatten, dense레이어를 쌓고, 마지막 activation function을 다중 준류를 위해 softmax를 사용하였다. 앞에서 봤다시피 원래 classs는 0부터 5까지 6개지만, 코드를 작성하면서 checkpoint가 들어갔기 때문에 분류를 7개로 분류했다.

이후 모델을 알맞은 setting에 맞추어 컴파일하고, fit을 하였다. 이때 history 변수들과 checkpoint를 추가하여 train\_acc와 val\_acc를 추적할 수 있도록 코드를 작성했다. 그런데 이때 코드를 돌릴 때 이미지 파일의 크기 때문인지 오류가 발생했다. git을 찾아보니 코랩의 메모리 용량과 관련된 문제라고 하여 ImageFile.LOAD\_TRUNCATED\_IMAGES = True라는 코드를 또한 추가하여 문제를 해결했다.



1. **Experiments**
   1. **Setting**

모델을 컴파일 할때, 다중븐류이기 때문에 loss 모델로 categorical\_crossentropy를 선택했다. 또한 옵티마이저는 0.00001, 0.00002, 0.0001 등 다양한 값들을 시도해보았을 때, 비슷한 모습을 보이지만 val\_loss가 가장 적은 0.0001로 lr을 선택하였다. epoch값은 lr이 작을 때는 50일 때 높은 모습을 보였지만, lr이 클 때는 70정도가 높은 정답률을 보였다.

* 1. **Performance Metrics**

평가지표는 compile단계에서는 keras 모델에서 지원하는 ‘acc’ 방법을 택하였다. 이때, history 함수를 통해 train acc, loss 그리고 val\_acc와 loss를 추적하여 visualization하여 직접 눈으로 보면서 평가 할 수 있었다.

* 1. **Results**

그 결과 valiation과 train셋을 통해 검증했을 때는 약 88퍼센트의 정답률을 보였다. 하지만 오버피팅을 피하기 위해 데이터 증강을 적용햇을 때 비슷한 정답률을 보여줬지만 실제 submossion에서는 일반화된 모델의 영향때문인지 약 92퍼센트의 높은 정답률을 보여주었다. 아마 모델에 적절한 피팅이 되어 오버피팅을 피한 영향이라고 생각된다.

1. **Discussion and Limitation**

데이터 증강, VGG16의 사전 학습 등 다양한 방법을 통해 모델을 구축하니 92퍼센트라는 높은 정확도가 나왔던 것 같다. train데이터를 바로 활용하지 않고, 데이터 셋을 validation 데이터 셋과 나누어 검증하면서 모델을 구축했던 것도 모델의 높은 정확도에 도움을 준 것으로 예상된다. 한가지 아쉬운 것은 VGG 16을 활용한 모델을 구축할 때, 모델 적합성을 위해 freezed와 같은 다양한 모델의 구축을 시간 관계상 시도해보지 못했던 것이 아쉽다. fc레이어층의 dense 레이어 또한 dropout 등 오버피팅을 줄일 수 있는 시도들을 구상하기는 하였으나, 역시나 시간관계상의 문제도 있고, 또한 사용했던 환경인 colab의 환경이 다소 시간에 제약이 있어 많은 시도를 하지 못했던 부분도 있다. 하지만 모델 자체가 92퍼센트의 정확도를 보여준 것을 봤을 때, 실제 데이터의 분류에도 충분히 좋은 적용이 될 것이라고 생각한다.

1. **References**

-코랩에서 작업하였고, api코드들은 캐글 문서에서 가져왔다.

- VGG16 모델의 사전 학습 부분은 <https://bkshin.tistory.com/entry/%EC%BB%B4%ED%93%A8%ED%84%B0-%EB%B9%84%EC%A0%84-5-%EC%96%BC%EA%B5%B4-%EC%9D%B4%EB%AF%B8%EC%A7%80%EC%97%90%EC%84%9C-%EA%B0%90%EC%A0%95-%EB%B6%84%EB%A5%98Emotion-Classification> 해당 글에서 많은 레퍼런스를 얻었다.

- 이미지 모델을 구축하며 생긴 오류를 해결하기 위해 <https://deep-deep-deep.tistory.com/34> 를 참고하여 오류를 해결하였다.