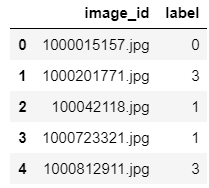
Cassava Leaf Disease Classification

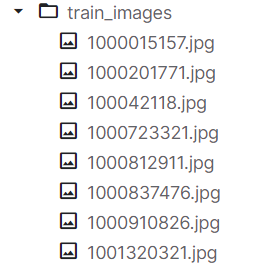
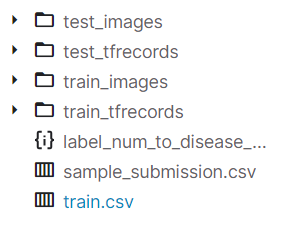
한현수

Problem

Your task is to classify each cassava image into four disease categories or a fifth category indicating a healthy leaf. With your help, farmers may be able to quickly identify diseased plants, potentially saving their crops before they inflict irreparable damage.

Data Fields

train.csv head

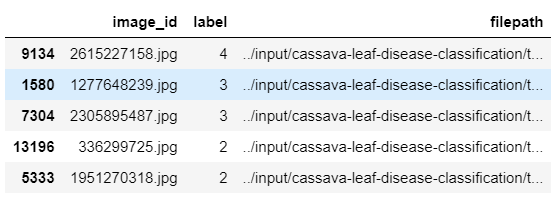
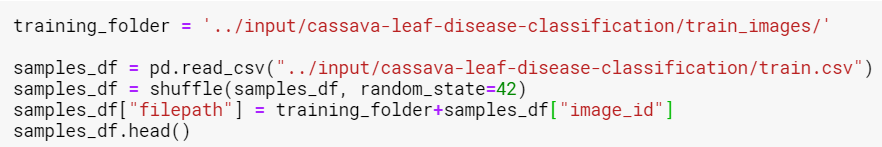


Data field는 이렇게 구성되어 있으며 train.csv 에는 image\_id와 label로 이루어져 있음을 알 수 있다. 그리고 필사한 노트북에서는 tfrecord 파일을 이용하지 않는다.

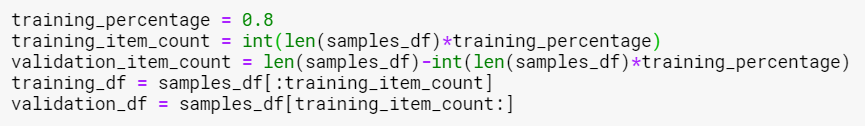
**EfficientNet+Aug (tf.keras) for Cassava Diseases**

\* google colab으로 작업하려 했으나 dataset을 Kaggle api로 다운받는 과정에서 image파일이 다운되지 않아 Kaggle notebook으로 진행하였다.

**import**

**Prepare the training and validation data generators**

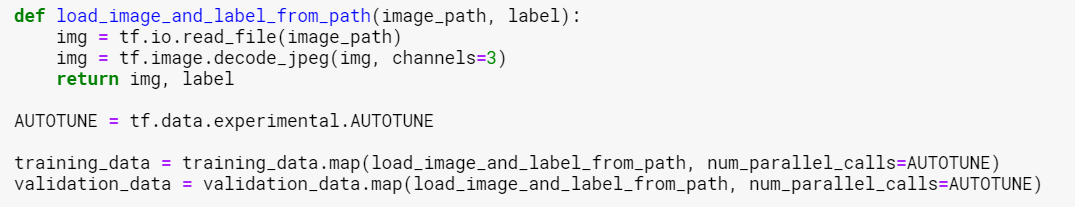
training folder에 있는 image filepath와 train.csv 를 결합한다.



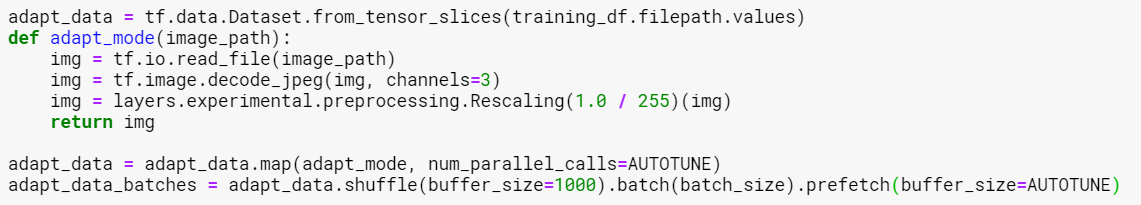
Training data를 80 : 20 비율로 train data와 validation data로 분할한다.

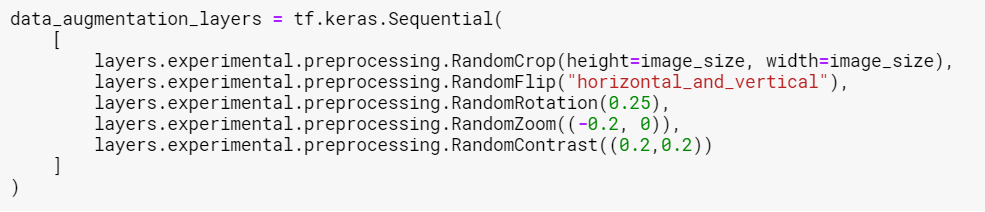
Batch size = 8, image shape = 512, 512, 3 dropout rate = 0.4 label = 0~4로 설정한다. 원래 image shape이 600,800,3 인데 efficientnet b3의 input 은 300,300,3으로 resolution scaling하는데, 노트북 작성자는 512,512,3으로 dimension을 설정하는 것이 좋은 결과가 나왔다고 한다.

이는 training\_df와 validation\_df를 tensorflow의 dataset으로 변환하는 과정이다. 이때 dataset은 filepath | label 로 구성된다.

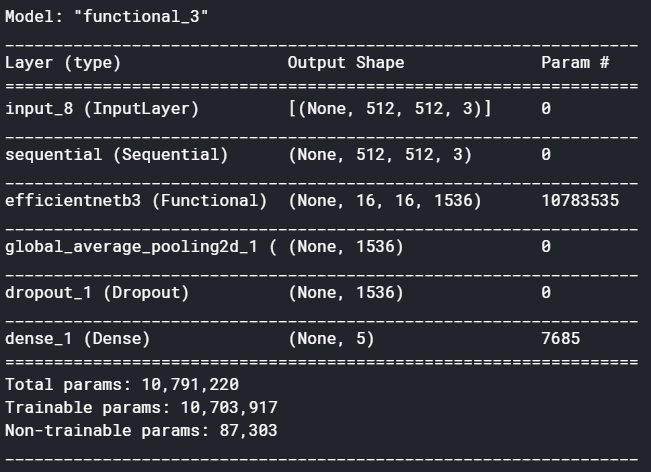
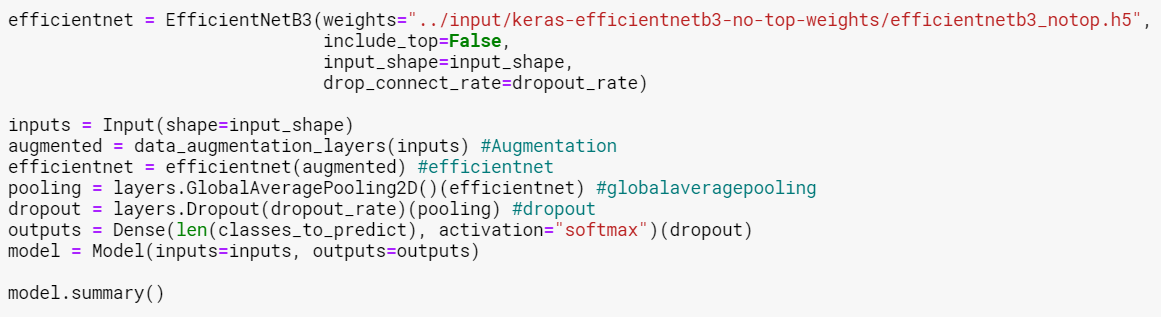
이는 filepath | label로 구성된 dataset을 Image | label 로 변환하는 과정이다. 여기서 num\_parallel\_calls = AUTOTUNE 은 dataset에서 training, loading 할 때 가져올 element의 수를 동적으로 조정한다는 매개변수이다.

Prefetch는 데이터 로딩과 학습이 병렬적으로 이루어져서 데이터 로딩에서 병목 현상이 일어나는 것을 방지할 수 있기에 사용되었다.

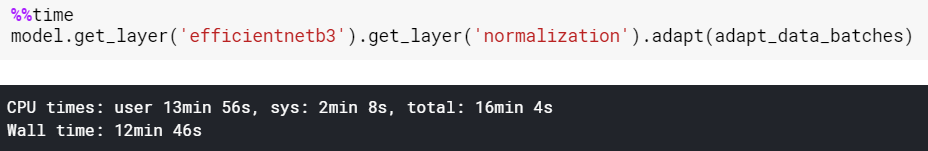
이는 efficient net이 imagenet 데이터셋에 적합한 normalization layer를 이 cassava leaf에 맞는 layer로 바꾸는 과정이다. 즉 image의 r,g,b feature를 0~255에서 0~1로 rescaling한다.

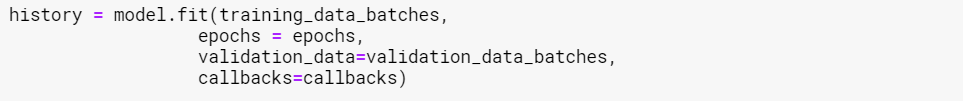
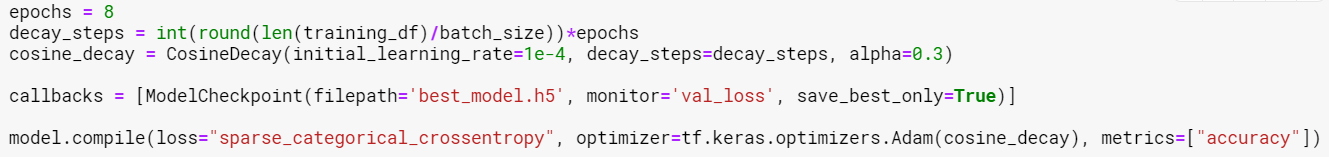
Data augmentation layer로 데이터를 회전, 뒤집기, 잘라내기를 이용하여 데이터 증강하는 layer이다. 이 augmentation은 training과정에서만 사용되고 inference에선 비활성화된다.

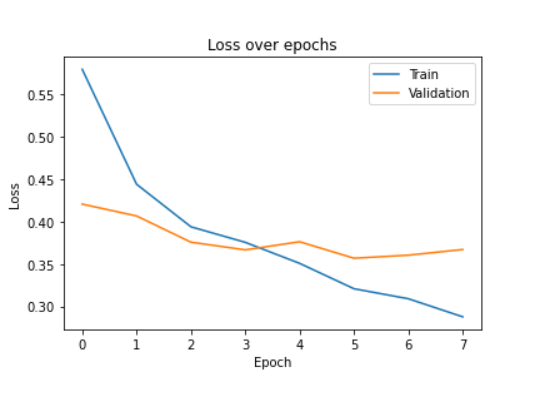
**Build the model**

****

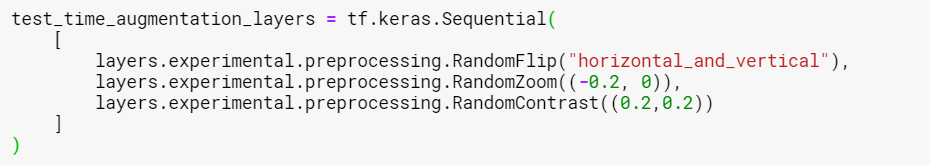
여기서 efficientnet의 weight를 가져와서 쓰지만 layer들을 freeze하지 않고 사용한다. 그 이유는 efficientnet이 cassava leaf dataset과 다른 imagenet dataset을 사용하여 학습했기 때문에 다시 학습시키는 것이 타당하다 생각해서 다시 학습시켰다고 한다.

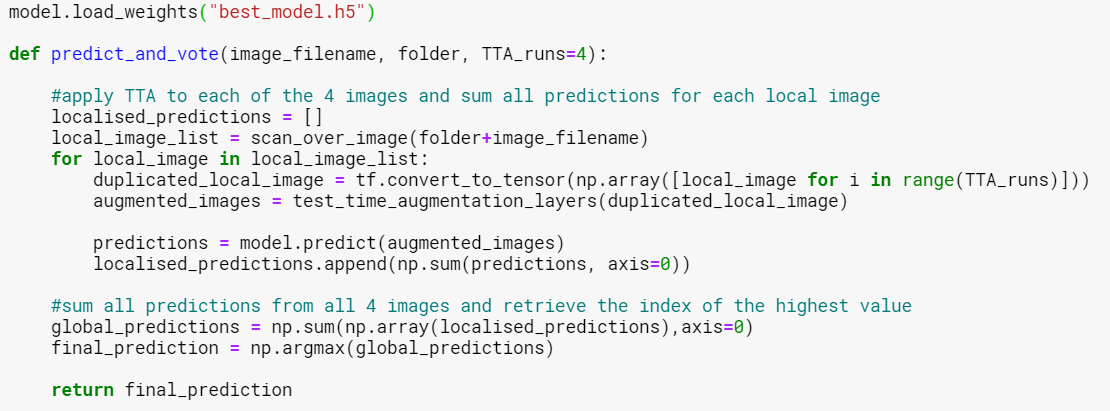
efficientnet의 3번째 layer는 normalization layer다. 이를 cassava leaf dataset에 적합한 normalization layer로 변환하는 과정이다.

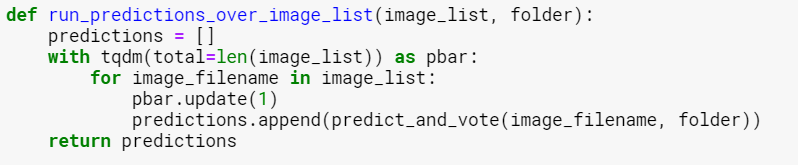
이는 epochs = 8 로 설정하고, cosin\_decay는 learning Rate를 cosine 함수를 따라서 변경하는 것으로 ReduceLROnPlateau와 같은 효과를 낸다. Callbacks는 val\_loss를 보고 최적의 model만 저장하게 설정하였다. 마지막으로 model을 compile 할 때 loss는 sparse\_categorical\_crossentropy, optimizer는 Adam으로 설정하였다.

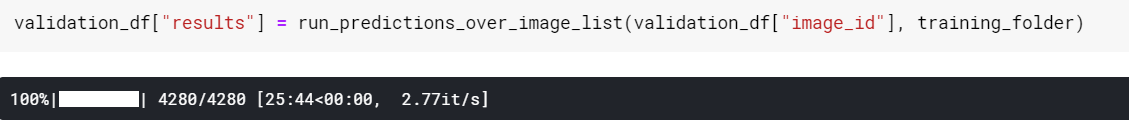


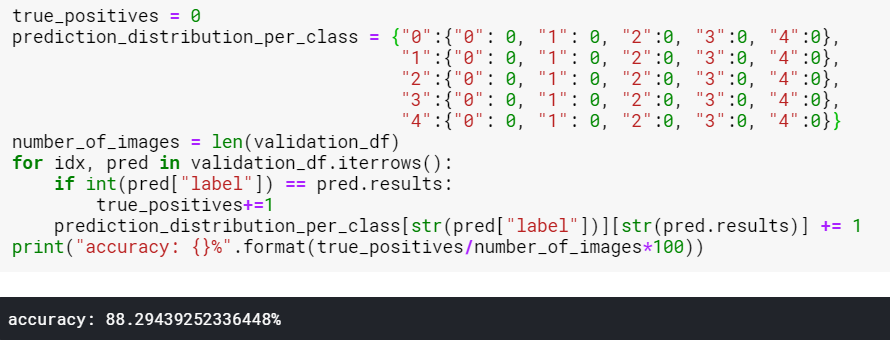
학습과정을 시각화 한 것으로 첫 epoch부터 val\_loss가 train\_loss보다 낮은 걸 확인할 수 있는데, 그 이유는 학습할 때 이미 학습된 network와 weight으로 학습을 시작했기 때문으로 보인다.

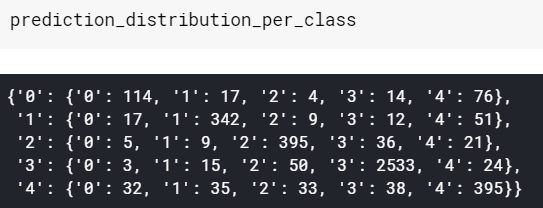
**Prediction on test images**이는 TTA를 위한 test data를 flip, zoom, contrast 하는 layer이다.

****TTA를 진행하고 inference 부분으로 test data를 바로 위의 layer를 통과시켜 4번 반복한 후, 이에 대한 predict를 진행한다. 이때 한 이미지당 4개를 predict하여 각 이미지에 대한 prediction을 더한 후, np.argmax를 이용하여 prediction을 return한다.

****이 함수는 각 이미지를 predict 하고 append하여 prediction을 리스트로 return한다.

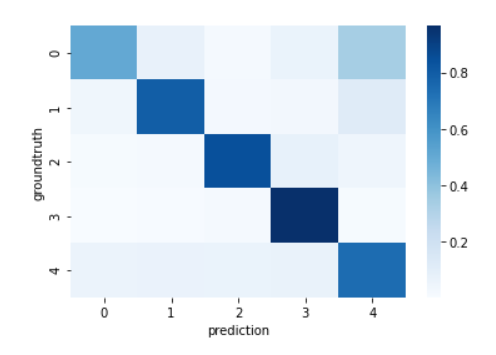
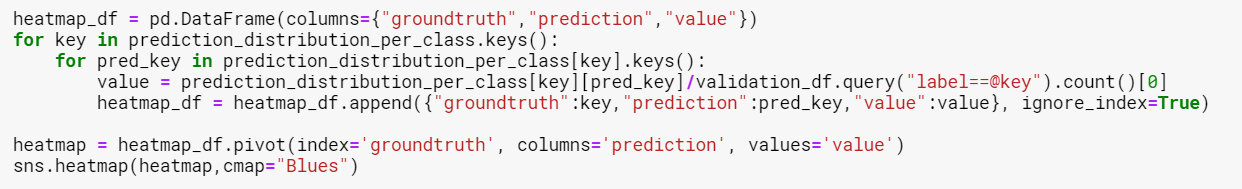
 이는 위의 함수들을 이용하여 validation data로 prediction을 만든다.

이는 validation data의 accuracy를 구하고 label과 prediction의 개수를 저장한다.



그 결과 각 label과 prediction의 결과 값들을 볼 수 있다. 각 label 별 accuracy는

0 = 50.7 , 1 = 80.0 , 2 = 84.8 , 3 = 96.5 , 4 = 74.1 로 확인 할 수 있다.



**Conclusion**

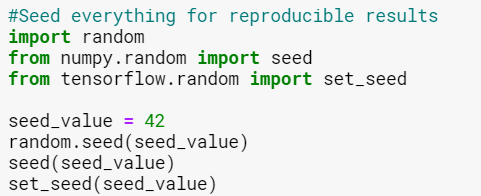
이를 heatmap을 그려 시각화 하였는데, 확인하니 3,2,1,4,0 순으로 accuracy가 높음을 확인할 수 있었는데, 이 차이는 각 label의 개수가 고르지 않고 각 개수가 3,4,2,1,0 순이기 때문임을 알 수 있다. 하지만 label 3을 제외한 다른 0,1,2의 경우 label 4로 predict한 경우가 많았기 때문에 이 문제를 해결할 방법을 알아 봐야 할 것 같다.

Score = 0.888

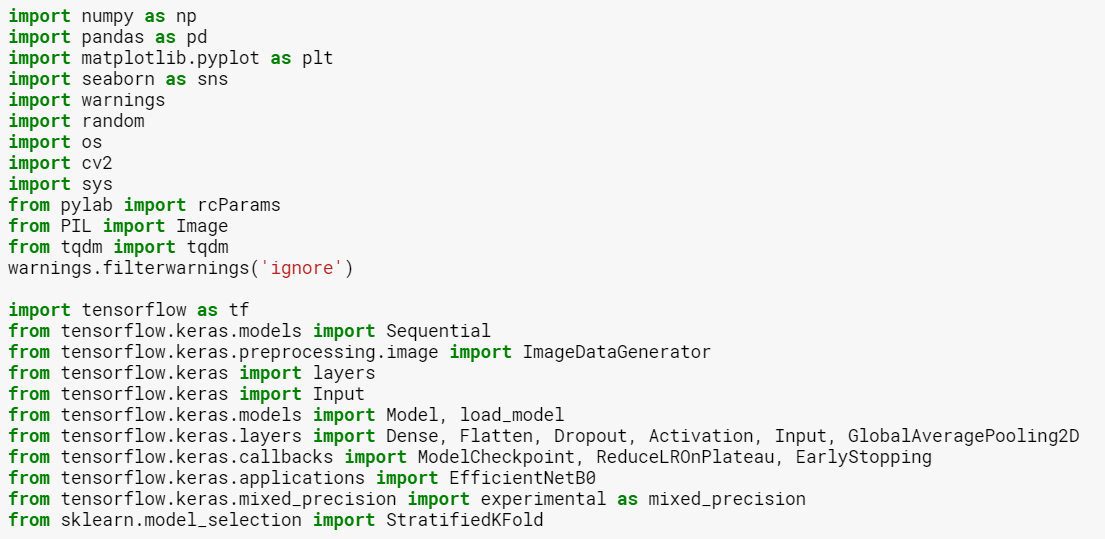
**Step-by-Step Guide to Denoising Your Labels**

Key = denoising label, efficient net b0, image augmentation, TTA(test time augmentation), mixed precision

**import**



Random seed 를 42 로 고정한다.



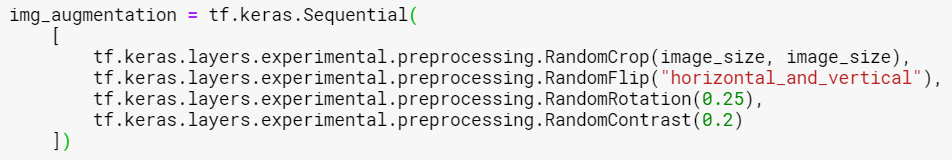
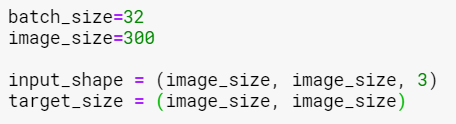
**Mixed precision**

****

Mixed precision은 모델 학습 시 FP16, FP32 부동 소수점 유형을 상황에 따라 유연하게 사용하여 학습을 더 빠르게 실행하고 메모리를 적게 사용하는 방법이다.

참고 : <https://brstar96.github.io/dl%20training%20tip/event&seminar/NVIDIAAIConf_session1/>

**Augmentation**

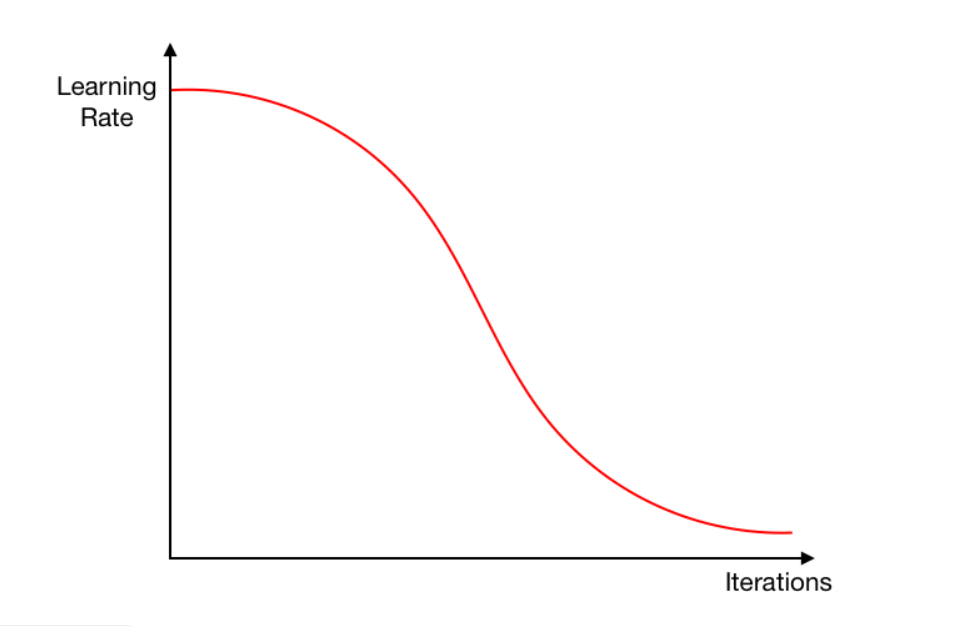
이는 Augmentation layer로 image data를 augmentation 하는 layer이다.

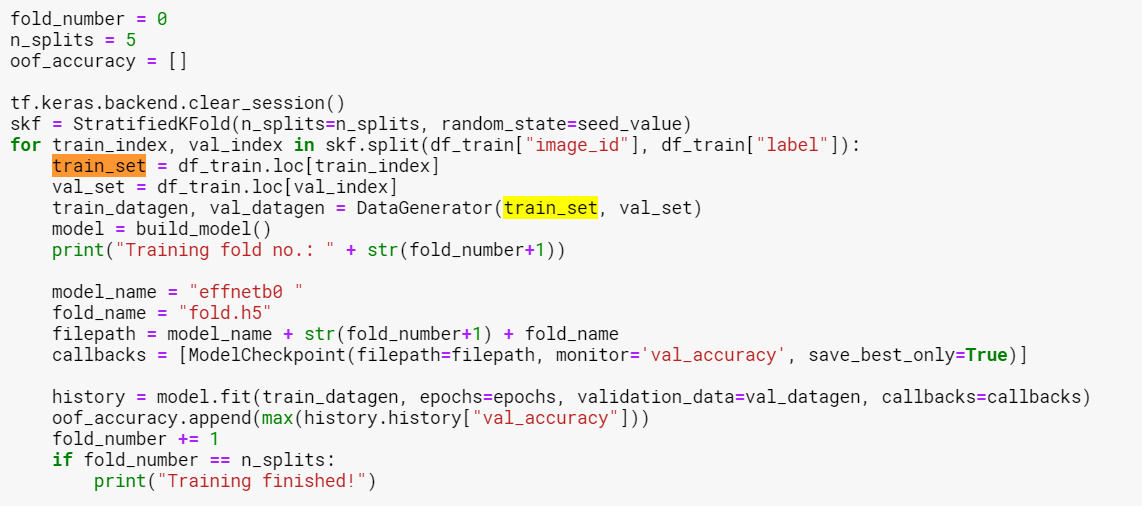
**Data Generator**

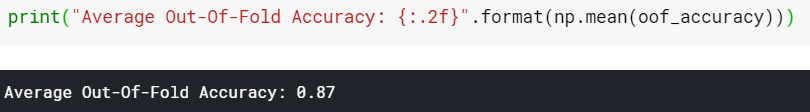


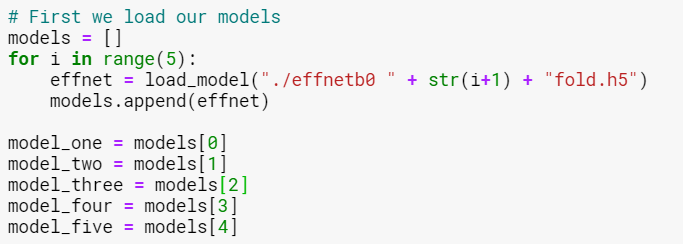
**Build model**

이 모델에서 기본 base model은 efficient net b0를 사용하였고, 특이한 점은 outputs layer에서dtype을 float32로 설정하는데, 이 이유는 mixed precision을 사용하여 원활한 학습을 위해서이다. 그리고 learning rate는 cosine decay , loss는 sparse\_categorical\_crossentropy로 모델을 컴파일한다.

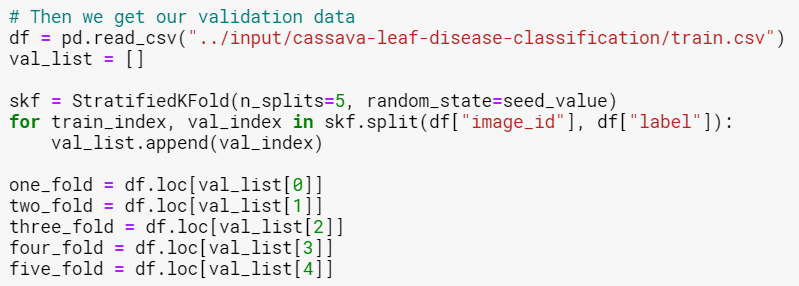
**Cosine Decay**Cosine Decay란 learning rate를 위와 같은 cosine 함수에 맞춰 줄여 나가는 learning rate설정 방법이다.

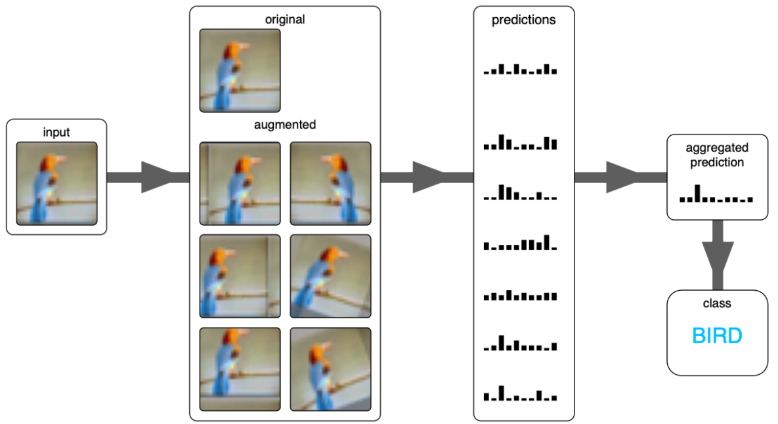
**Stratified 5-Fold Cross Validation**학습시에, 학습 데이터를 5개로 나누어 학습하고 각 fold별로 모델을 저장한다.

이렇게 학습시에 각 fold당 average accuracy는 0.87로 나왔다.

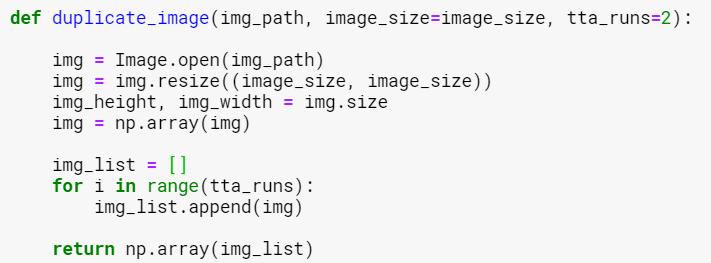
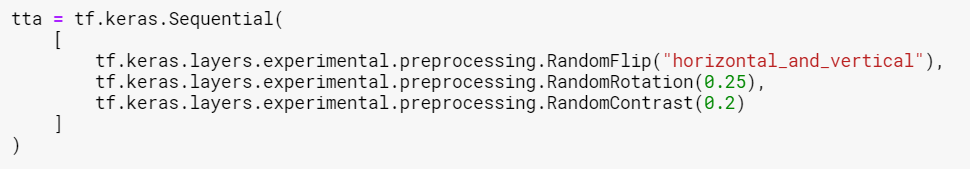
**Load model**

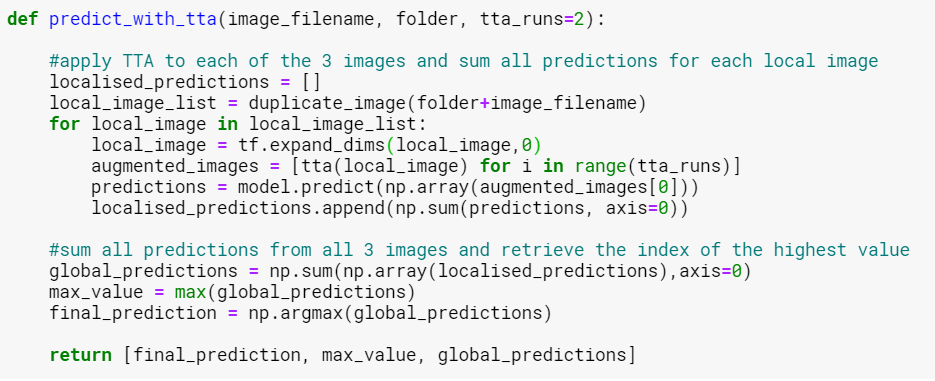
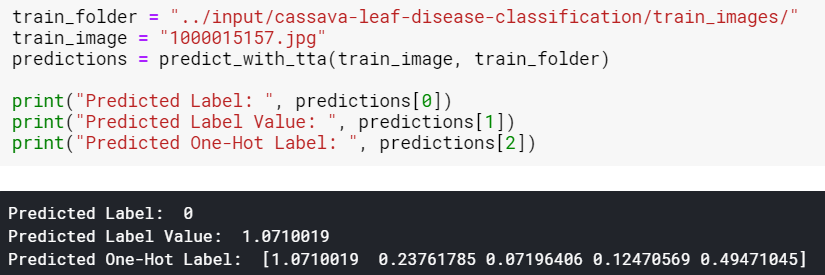
각 fold별로 model을 load한다.

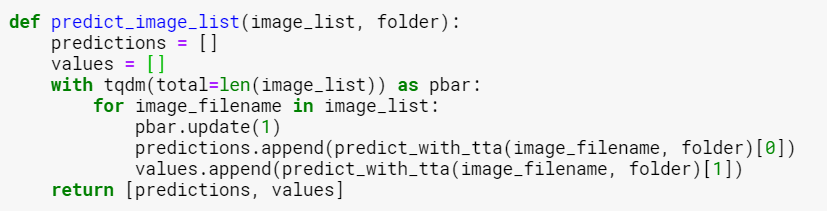
각 fold별로 val\_data를 형성한다.

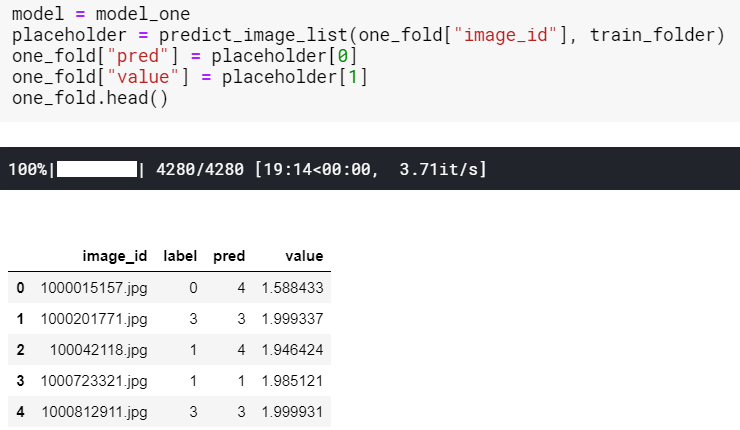
**Test Time augmentation**

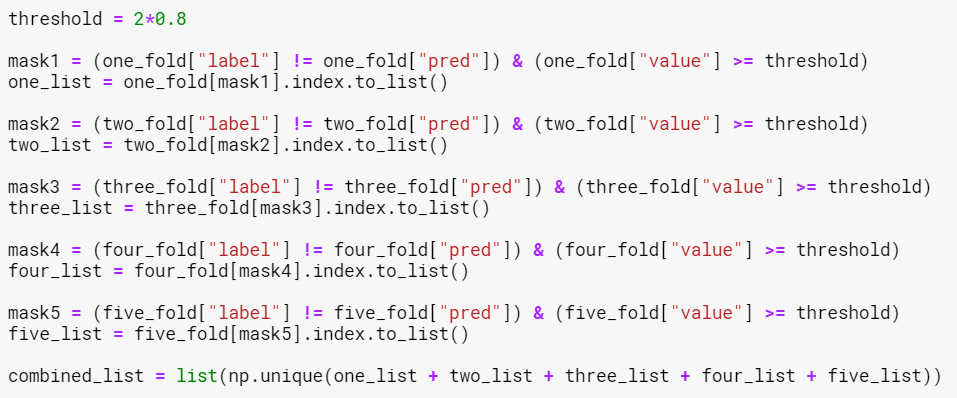
TTA는 test data를 augmentation하여 predict를 진행하는 것이다.

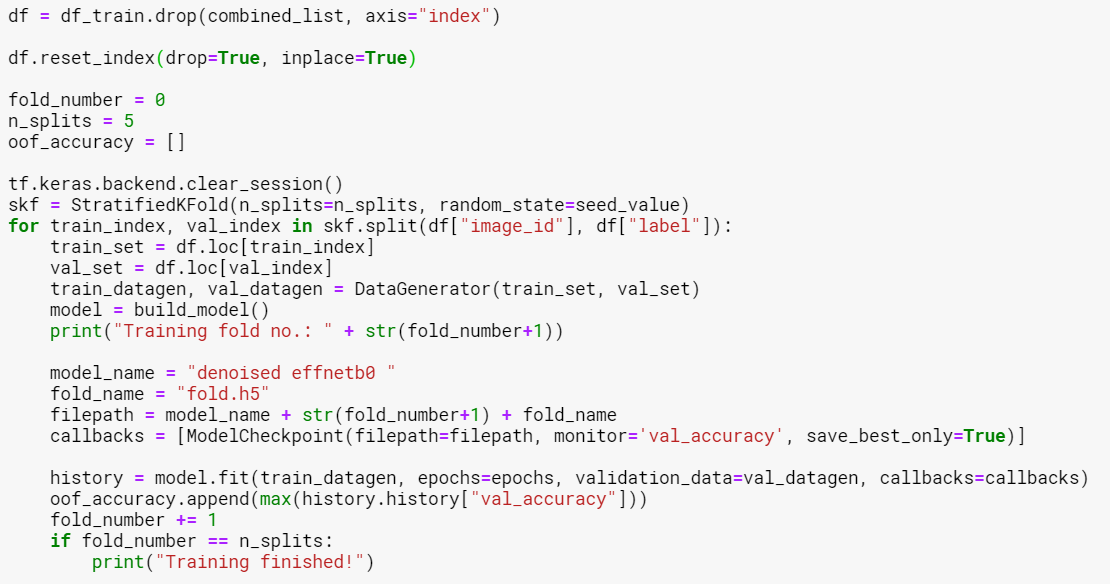


TTA layer를 만들고 TTA predict를 진행하기 위해 image를 np.array로 return하는 함수이다.TTA predict 하는 함수로, image2개를 TTA layer에 넣어 predict 값을 더하여 argmax로 final\_predcition을 결정한다. max\_value는 이후 label denoising을 위한 값이다. 위 함수에 임의의 data를 넣어 사용했을 때 output 결과이다.

이 함수는 predict\_with\_tta가 image 1개를 predict하지만 image list를 predict하는 함수다.

위 함수의 결과는 이런 형식으로 각 fold에 저장한다.

**Denoising Data**여기서 threshold는 label이 다를 때 2개의 이미지의 prediction value중 가장 높은 값이 threshold보다 높을 때 그 이미지를 label이 잘못 설정 되었다고 하여 train data에서 삭제한다.

마지막으로 학습하는 부분으로, 위의 기준으로 잘못 labeling된 data를 지우고 새로 학습을 한다.

**Conclusion**

이 노트북에선 labeling이 잘못된 data를 이런 방식으로도 추출할 수 있는 방법도 알게 되었고, 간단하지만 이런 방식으로도 acc를 높일 수 있다는 것을 알게 되었다.