에이콘 아카데미(강남) 권나은

Surface Crack Detection

1. Data set

   - 출처 : https://www.kaggle.com/datasets/arunrk7/surface-crack-detection
   - 구성 : Negative 20000개, Positive 20000개

2. Development

   - 목표

     1) CNN모델 정확도 90%이상

     2) 데이터 셋 생성 자동화

   - 프로젝트 소스 : Surface_Crack_Detection_권나은.ipynb

   - 개발 환경

     Google Colab

     모듈 Version

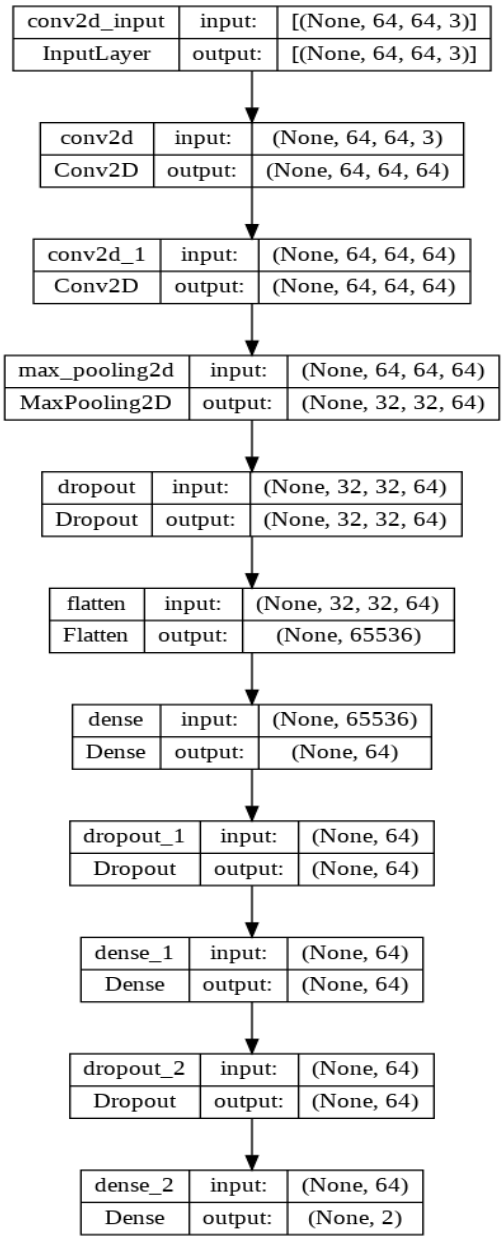
Package	Version
glob2	0.7
cv2	4.6.0
google-colab	1.0.0
keras	2.9.0
matplotlib	3.2.2
numpy	1.21.6
opencv-python	4.6.0.66
pandas	1.3.5
Python	3.8.16
pip	21.1.3
tensorflow	2.9.2

에이콘 아카데미(강남) 권나은

- 모델 아키텍처 및 선정 이유

<모델>

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 64)        1792

 conv2d_1 (Conv2D)           (None, 64, 64, 64)        36928

 max_pooling2d (MaxPooling2D  (None, 32, 32, 64)       0
 )

 dropout (Dropout)           (None, 32, 32, 64)        0

 flatten (Flatten)           (None, 65536)             0

 dense (Dense)               (None, 64)                4194368

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 64)                4160

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 2)                 130

=================================================================
Total params: 4,237,378
Trainable params: 4,237,378
Non-trainable params: 0
_________________________________________________________________
```



<선정 이유>

여러 번 실행을 통한 모델 설정

loss: 0.0021 - accuracy: 0.9995 - val_loss: 0.0348 - val_accuracy: 0.9937

reduce로 인한 최종 learning_rate : 1.2500e-05

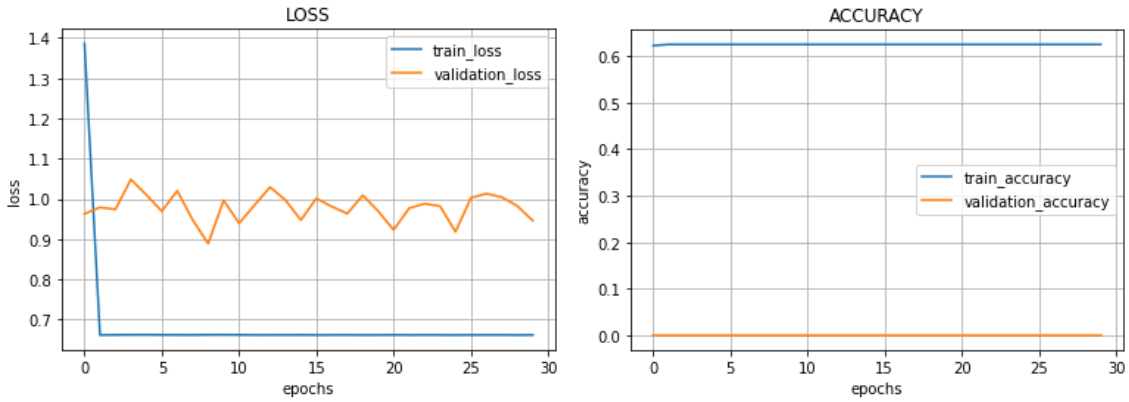
evaluate : 0.023939523845911026 / accuracy : 0.9955833554267883

에이콘 아카데미(강남) 권나은

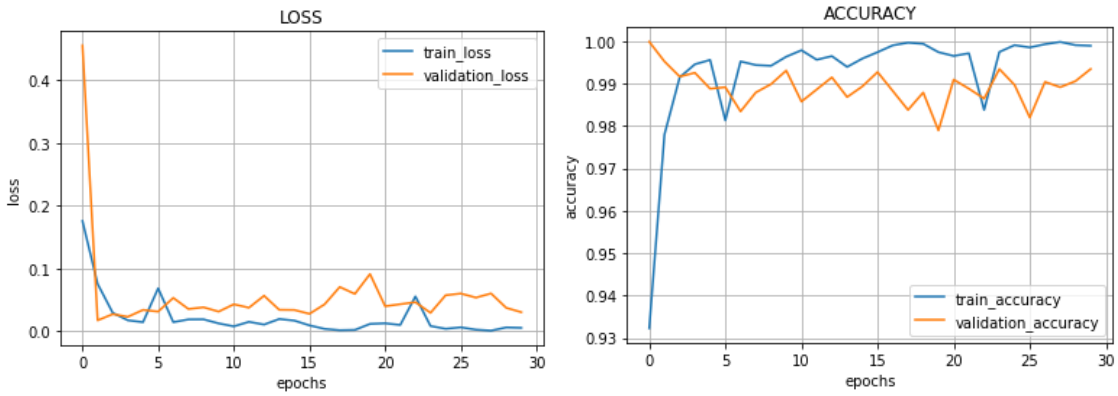
<모델 선정을 위한 실행 과정>

1. CCPFDD

   - Convolution filter 수 : 64 / Dense unit수 : 128 / Learning_rate : 0.01
   - Batch_size : 50 / Epochs : 30 / Validation_split : 0.2
   - Image size (64*64)



loss: 0.98 - accuracy: 0.6250 - val_loss: 0.9801 - val_accuracy: 0.0000+
데이터가 랜덤하게 뽑히는 과정에서 잘못 뽑힌 것 같아서 다시 데이터 뽑아서 진행



loss: 0.0051 - accuracy: 0.9989 - val_loss: 0.0301 - val_accuracy: 0.9934

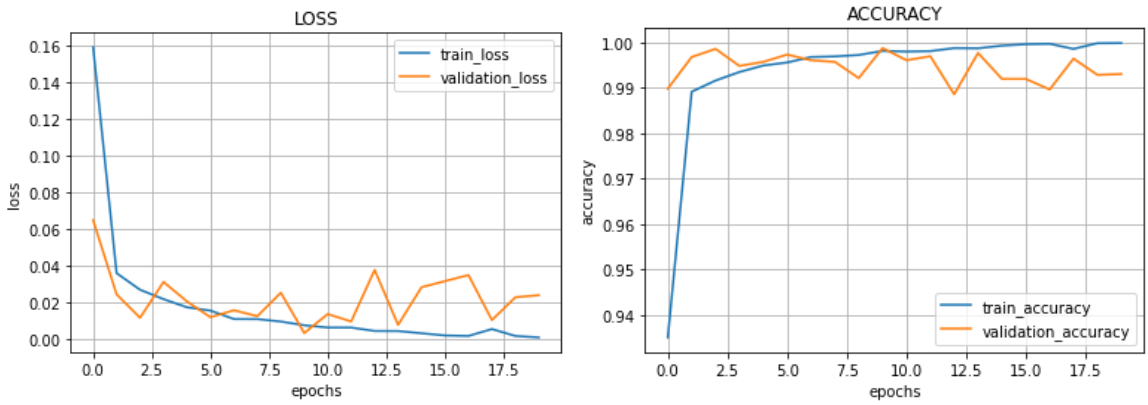
evaluate : 0.036389075219631195 / accuracy : 0.9929166436195374

loss와 accuracy가 너무 일정하지 않아서 learning_rate 조절

epochs수도 20까지만 해도 충분한 듯하여 줄임

2. CCPFDD

- Convolution filter 수 : 64 / Dense unit수 : 128 / Learning_rate : 0.0001
- Batch_size : 50 / Epochs : 20 / Validation_split : 0.2
- Image size (64*64)



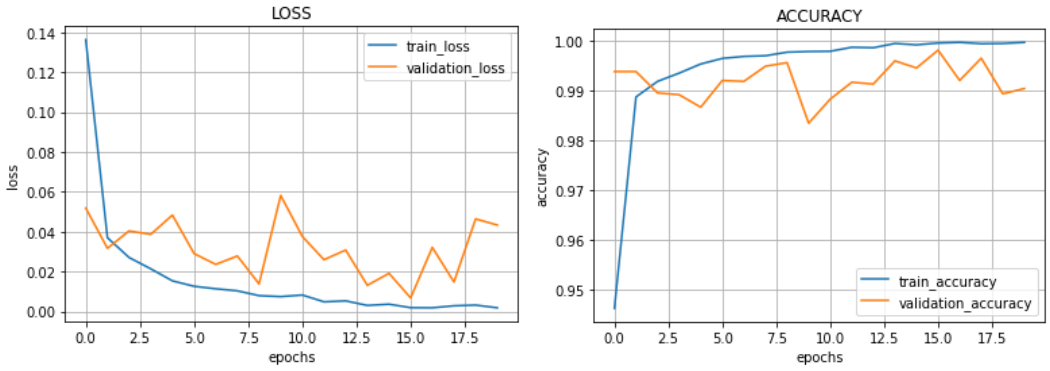
loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0240 - val_accuracy: 0.9930

evaluate : 0.0235510915517807 / accuracy : 0.9948333501815796

그래프의 높낮이가 어느정도 일정하게 바뀌긴했지만 과적합도 해결 필요

3. CCP(Dropout:0.2)FDD

- Convolution filter 수 : 64 / Dense unit수 : 128 / Learning_rate : 0.0001
- Batch_size : 50 / Epochs : 20 / Validation_split : 0.2
- Image size (64*64)



loss: 0.0018 - accuracy: 0.9996 - val_loss: 0.0433 - val_accuracy: 0.9904

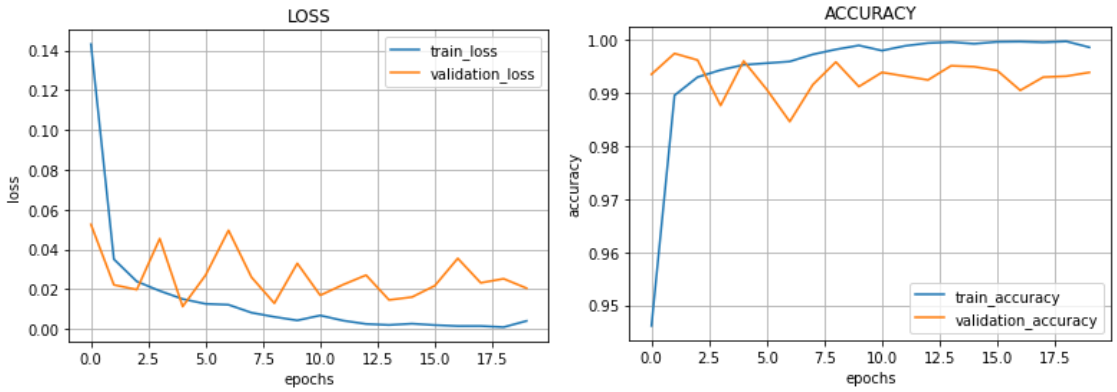
evaluate : 0.03235936909914017 / accuracy : 0.9922500252723694

에이콘 아카데미(강남) 권나은

CCP뒤에 dropout 추가 오히려 과적합이 더 심해짐 >> 한 번 더 시행

그래프도 더 변동이 많음

CCP(Dropout:0.2)FDD 한번 더 시행

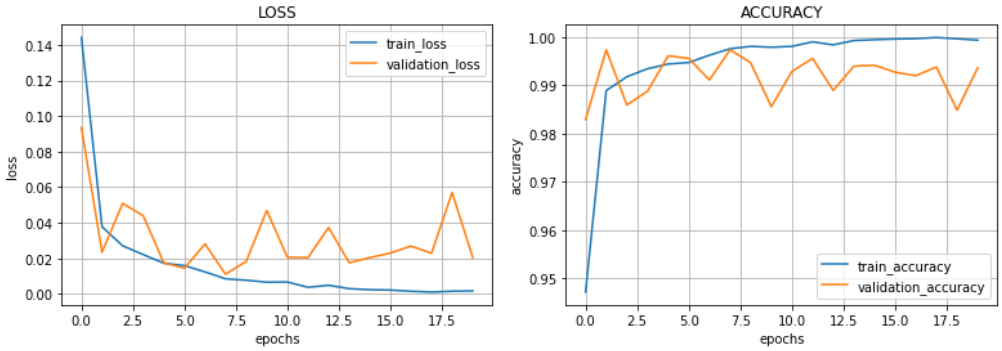


loss: 0.0040 - accuracy: 0.9987 - val_loss: 0.0205 - val_accuracy: 0.9939

evaluate : 0.02510133385658264 / accuracy : 0.9942499995231628

두 번 시행한 결과가 너무 달라서 은닉층에 가중치 초기화 실행

4. CCP(Dropout:0.2)FD(kernel_initializer="he_normal")D
   - Convolution filter 수 : 64 / Dense unit수 : 128 / Learning_rate : 0.0001
   - Batch_size : 50 / Epochs : 20 / Validation_split : 0.2
   - Image size (64*64)

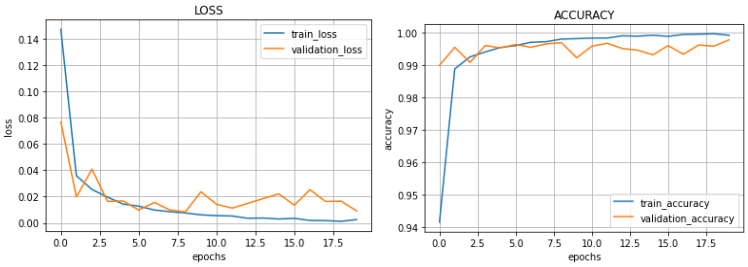


loss: 0.0017 - accuracy: 0.9993 - val_loss: 0.0205 - val_accuracy: 0.9936

evaluate : 0.02581479586660862 / accuracy : 0.9942499995231628

에이콘 아카데미(강남) 권나은

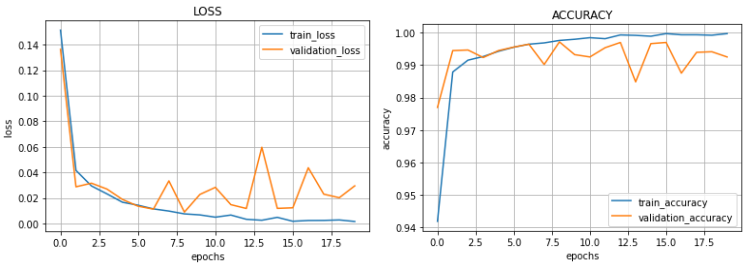
CCP(Dropout:0.2)FD(kernel_initializer="he_normal")D(두번째 실행)



loss: 0.0025 - accuracy: 0.9992 - val_loss: 0.0090 - val_accuracy: 0.9979

evaluate : 0.021811973303556442 / accuracy : 0.9940833449363708

CCP(Dropout:0.2)FD(kernel_initializer="he_normal")D(세번째 실행)



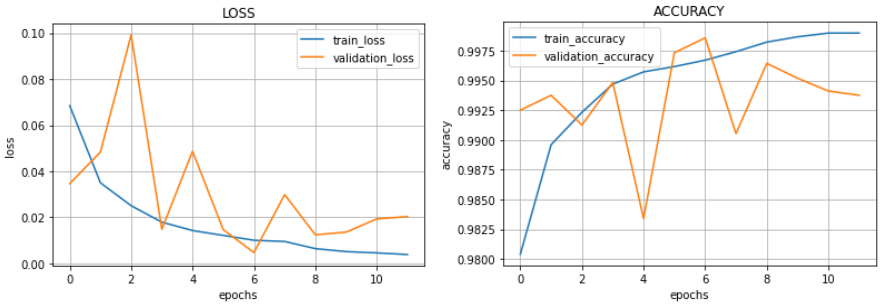
loss: 0.0016 - accuracy: 0.9997 - val_loss: 0.0295 - val_accuracy: 0.9925

evaluate : 0.026837319135665894 / accuracy : 0.9940000176429749

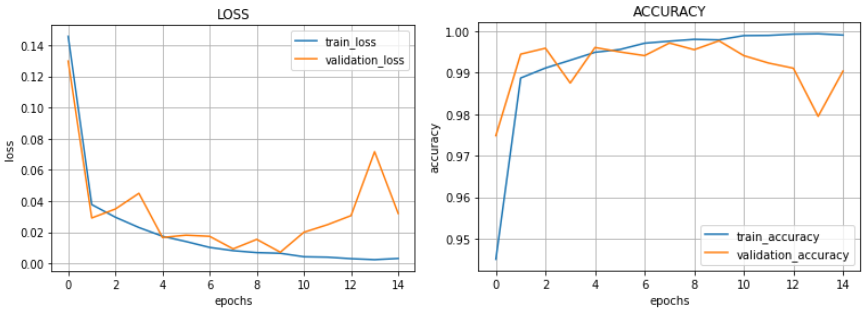
비슷한 수치가 계속 반복되는 것 같아서 call back함수 사용

5. CCP(Dropout:0.2)FD(kernel_initializer="he_normal")D
   - Convolution filter 수 : 64 / Dense unit수 : 128 / Learning_rate : 0.0001
   - Batch_size : 50 / Epochs : 20 / Validation_split : 0.2
   - Early Stopping 추가(val_loss 기준 patience=5)
   - Image size (64*64)

에이콘 아카데미(강남) 권나은



CCP(Dropout:0.2)FD(kernel_initializer="he_normal")D + EarlyStopping(두 번째 시행)



첫번째 시행

loss: 0.0038 - accuracy: 0.9990 - val_loss: 0.0203 - val_accuracy: 0.9937

evaluate : 0.02145436592400074 / accuracy : 0.9940833449363708

두번째 시행

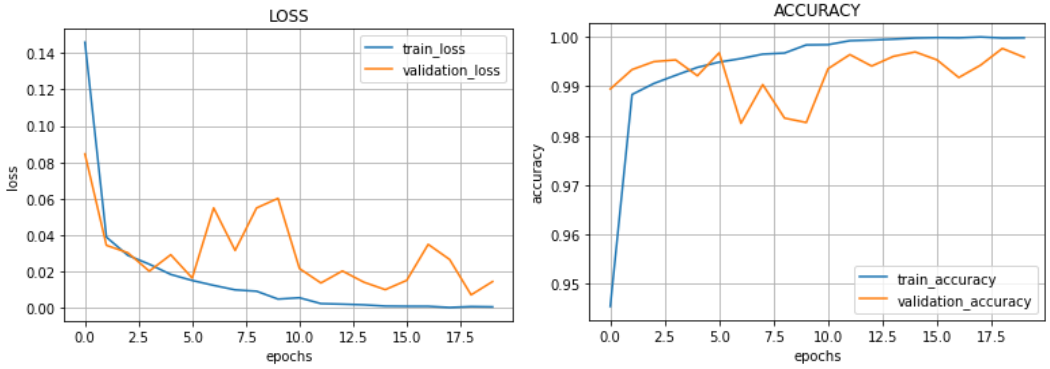
loss: 0.0032 - accuracy: 0.9991 - val_loss: 0.0321 - val_accuracy: 0.9904

evaluate : 0.026525963097810745 / accuracy : 0.9932500123977661

별로 효과가 없고 그래프 변동이 심해서 callback을 reduce로 변경

6. CCP(Dropout:0.2)FD(kernel_initializer="he_normal")D
   - Convolution filter 수 : 64 / Dense unit수 : 128 / Learning_rate : 0.0001
   - Batch_size : 50 / Epochs : 20 / Validation_split : 0.2
   - Reduce 추가(val_loss 기준 patience=5)
   - Image size (64*64)

에이콘 아카데미(강남) 권나은

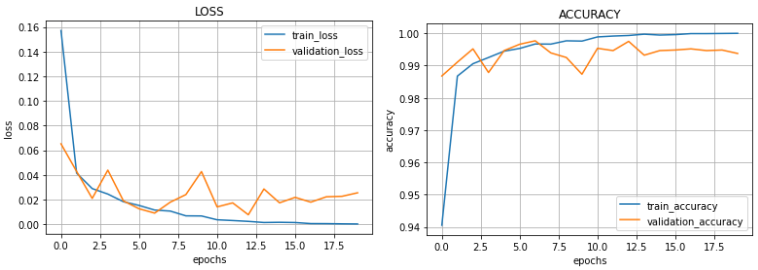


loss: 7.2548e-04 - accuracy: 0.9998 - val_loss: 0.0146 - val_accuracy: 0.9959

evaluate : 0.024821627885103226 / accuracy : 0.9950000047683716

그래프 변동이 심해서 patience = 3으로 줄여서 시행

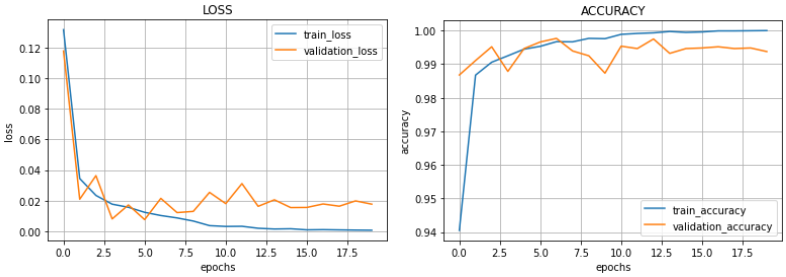
CCP(Dropout:0.2)FD(kernel_initializer="he_normal")D + Reduce(두 번째 시행: patience = 3)



loss: 2.8629e-04 - accuracy: 1.0000 - val_loss: 0.0256 - val_accuracy: 0.9937

evaluate : 0.017286857590079308 / accuracy : 0.9952499866485596

CCP(Dropout:0.2)FD(kernel_initializer="he_normal")D + Reduce(세 번째 시행: patience = 3)

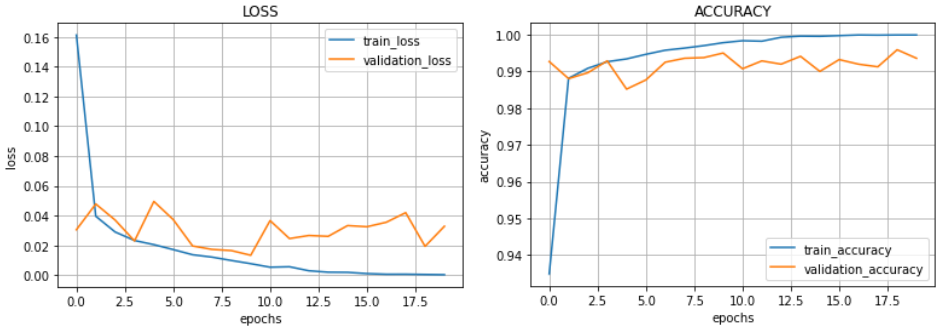


loss: 2.8629e-04 - accuracy: 1.0000 - val_loss: 0.0256 - val_accuracy: 0.9937

evaluate : 0.016093676909804344 / accuracy : 0.996666669845581

과적합 더 해결 필요

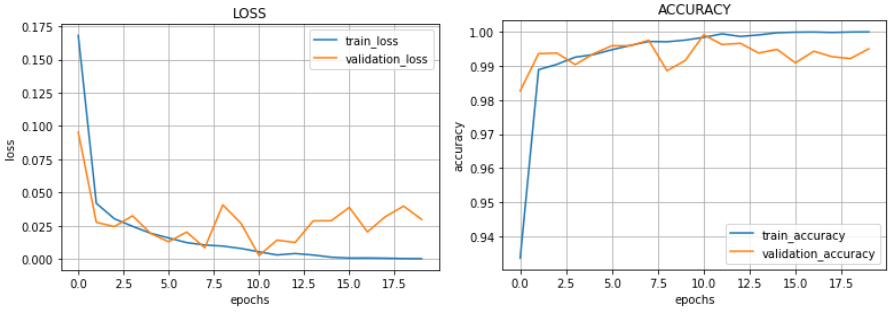
7. CCP(Dropout:0.2)FD(kernel_initializer="he_normal")(Dropout:0.2)D
   - Convolution filter 수 : 64 / Dense unit수 : 128 / Learning_rate : 0.0001
   - Batch_size : 50 / Epochs : 20 / Validation_split : 0.2
   - Reduce (val_loss 기준 patience=3)
   - Image size (64*64)



loss: 3.1778e-04 - accuracy: 0.9999 - val_loss: 0.0328 - val_accuracy: 0.9936

evaluate : 0.024807779118418694 / accuracy : 0.9954166412353516

CCP(Dropout:0.2)FD(kernel_initializer="he_normal")(Dropout:0.2)D(두 번째 시행)



loss: 3.0182e-04 - accuracy: 1.0000 - val_loss: 0.0298 - val_accuracy: 0.9950

evaluate : 0.023869367316365242 / accuracy : 0.9957500100135803

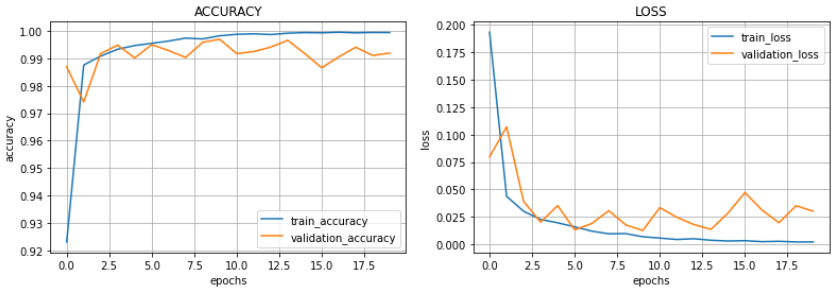
8. 어느 정도 안정되었다고 생각해서 이미지 크기 증가

1) (7번과 같은 모델) + Image size (128*128) : Colab에서 시행 안 됨(가능한 RAM모두 사용)

2) (7번과 같은 모델) + Image size (100*100) : evaluate불가(가능한 RAM모두 사용)

   loss: 0.0012 - accuracy: 0.9999 - val_loss: 0.0277 - val_accuracy: 0.992

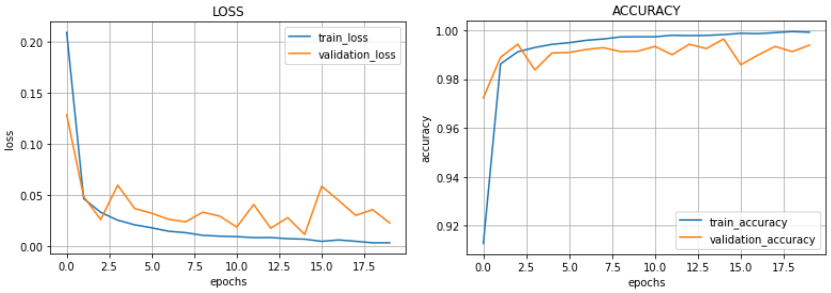
3) (7번과 같은 모델) + Image size(80*80)



loss: 0.0023 - accuracy: 0.9994 - val_loss: 0.0303 - val_accuracy: 0.9920 - lr: 6.2500e-06

evaluate : 0.02371836267411709 / accuracy : 0.9940833449363708

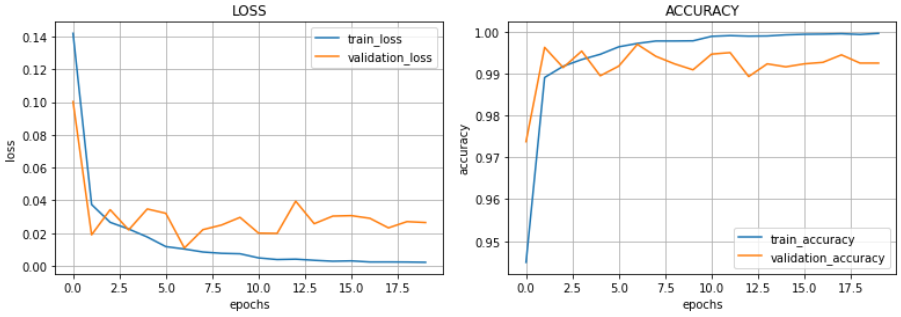
(7번과 같은 모델) + Image size(80*80)(두 번째 시행)



loss: 0.0035 - accuracy: 0.9992 - val_loss: 0.0230 - val_accuracy: 0.9939 - lr: 2.5000e-05

evaluate : 0.021589506417512894 / accuracy : 0.9948333501815796

(7번과 같은 모델) + Image size(80*80)(세번째 시행)



loss: 0.0022 - accuracy: 0.9996 - val_loss: 0.0265 - val_accuracy: 0.9925 - lr: 3.1250e-06

evaluate : 0.019367728382349014 / accuracy : 0.9945833086967468

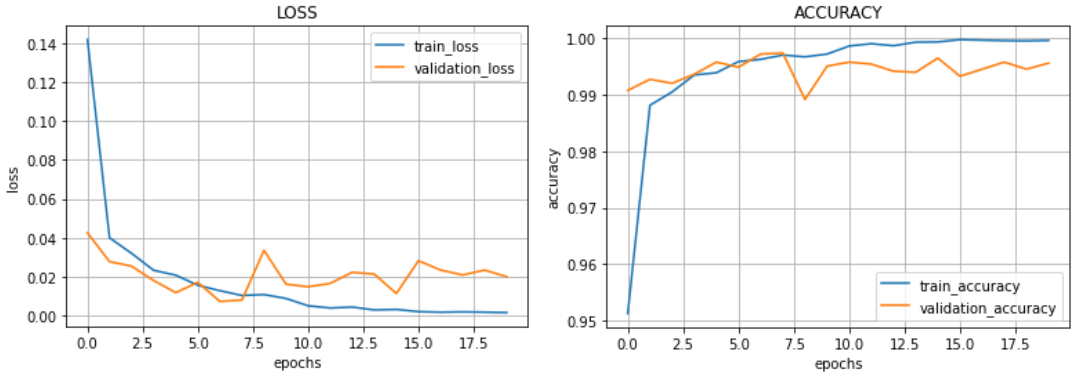
80*80으로 했을 때 그래프가 더 안정적으로 보이지만 Google Colab에서 시행하기에 자꾸 다운됨 >> 다시 64*64로 시행

과적합 더 줄이기 위해 Dropout 0.2 -> 0.25로 변경

9. CCP(Dropout:0.25)FD(kernel_initializer="he_normal")(Dropout:0.25)D

   - Convolution filter 수 : 64 / Dense unit수 : 64 / Learning_rate : 0.0001
   - Batch_size : 32 / Epochs : 20 / Validation_split : 0.2
   - Reduce (val_loss 기준 patience=3)
   - Image size (64*64)

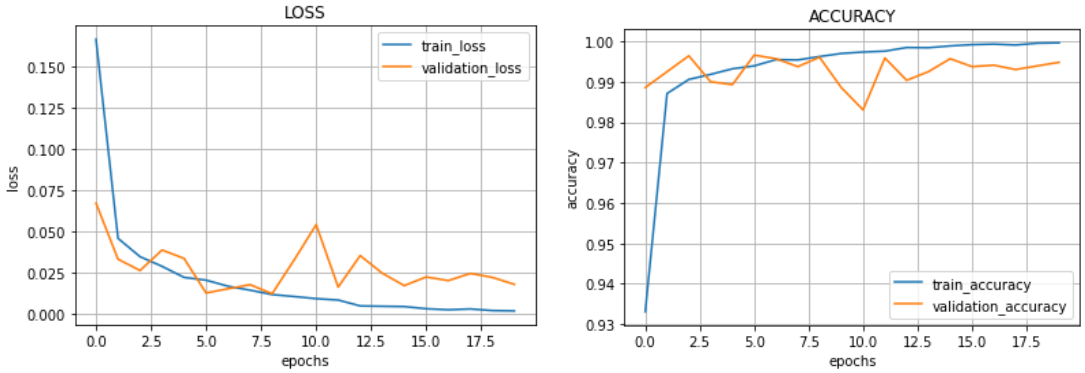
Dense unit수도 128까지 필요 없을 것 같아서 64로 진행, 대신 batch_size 32로 줄임



loss: 0.0016 - accuracy: 0.9996 - val_loss: 0.0200 - val_accuracy: 0.9955 - lr: 6.2500e-06

evaluate : 0.020839113742113113 / accuracy : 0.9956666827201843

CCP(Dropout:0.25)FD(kernel_initializer="he_normal")(Dropout:0.25)D(두 번째 시행)

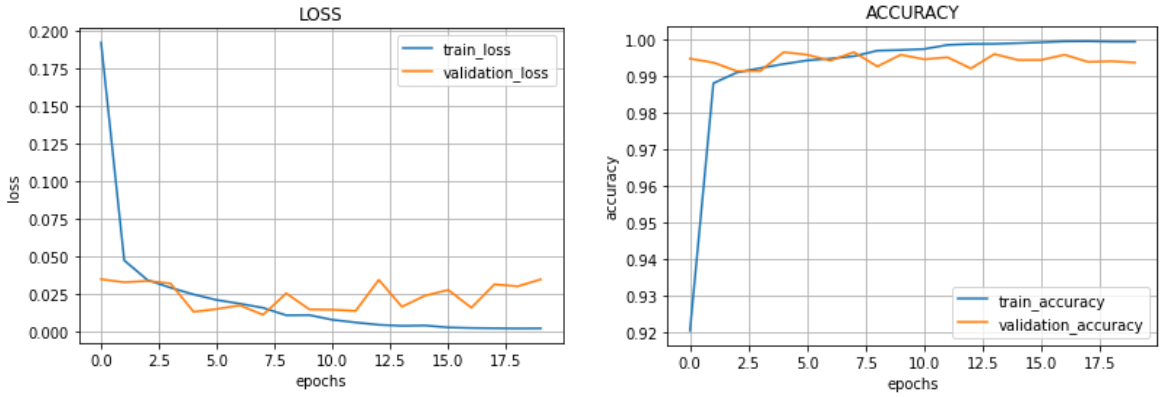


loss: 0.0018 - accuracy: 0.9996 - val_loss: 0.0179 - val_accuracy: 0.9948 - lr: 1.2500e-05

evaluate : 0.019511252641677856 / accuracy : 0.9958333373069763

10. CCP(Dropout:0.25)FD(kernel_initializer="he_normal")(Dropout:0.25)D(Dropout:0.25)

- Convolution filter 수 : 64 / Dense unit수 : 64 / Learning_rate : 0.0001
- Batch_size : 32 / Epochs : 20 / Validation_split : 0.2
- Reduce (val_loss 기준 patience=3)
- Image size (64*64)



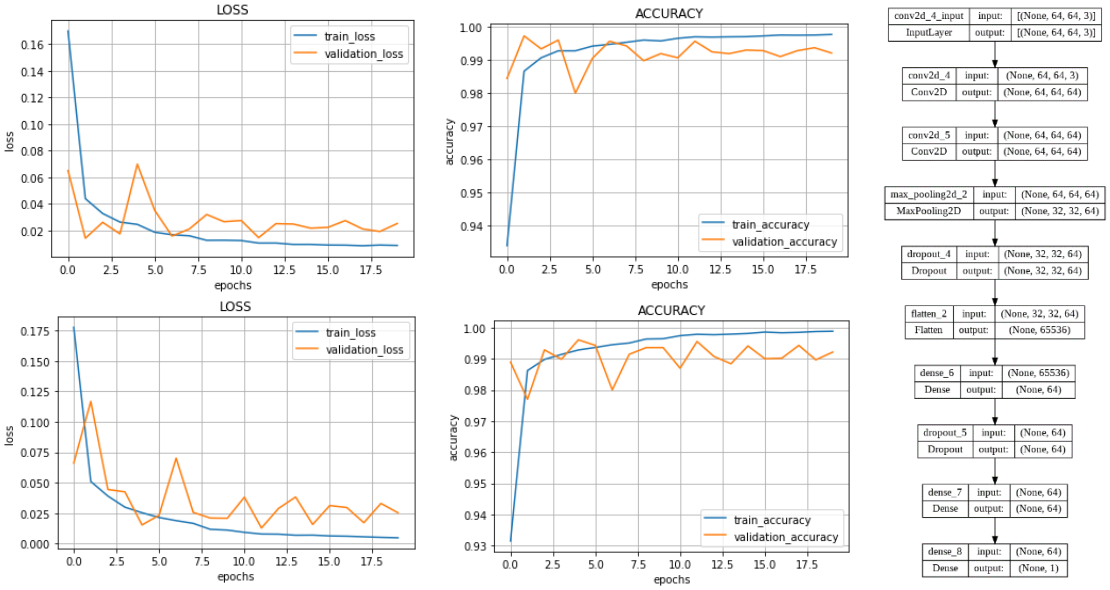
loss: 0.0028 - accuracy: 0.9994 - val_loss: 0.0304 - val_accuracy: 0.9927 - lr: 6.2500e-06

loss: 0.0021 - accuracy: 0.9995 - val_loss: 0.0348 - val_accuracy: 0.9937 - lr: 1.2500e-05

evaluate : 0.023939523845911026 / accuracy : 0.9955833554267883

그래프의 변동이 크지 않고 과적합도 많이 줄고 성능도 나쁘지 않아서 이 모델을 최종적으로 선택

추가 : 출력층을 sigmoid로 (binary_crossentropy)



에이콘 아카데미(강남) 권나은

- 프로젝트 진행 시 시행착오 및 문제점

  train, test분리 및 데이터를 가져올 때 속도가 너무 느리고 진행이 잘 되지 않아 train:test = 8:2에서 7:3으로 변경

  이미지 데이터 가져올 때 for문 대신 코드 2번 진행

  learning_rate = 0.01에서 시행할 때 validation_accuracy가 0이나오는 경우가 있었음 데이터를 다시 불러와서 실행

  추가로 learning_rate를 0.0001로 변경한 후 validation_accuracy가 0이 나오지는 않음 현재 colab의 GPU를 사용하는 경우 64*64의 크기로 이미지를 학습시키는 것이 안정적(128*128, 100*100, 80*80 : 불안정_80*80은 한번 실행 후 RAM 초과)