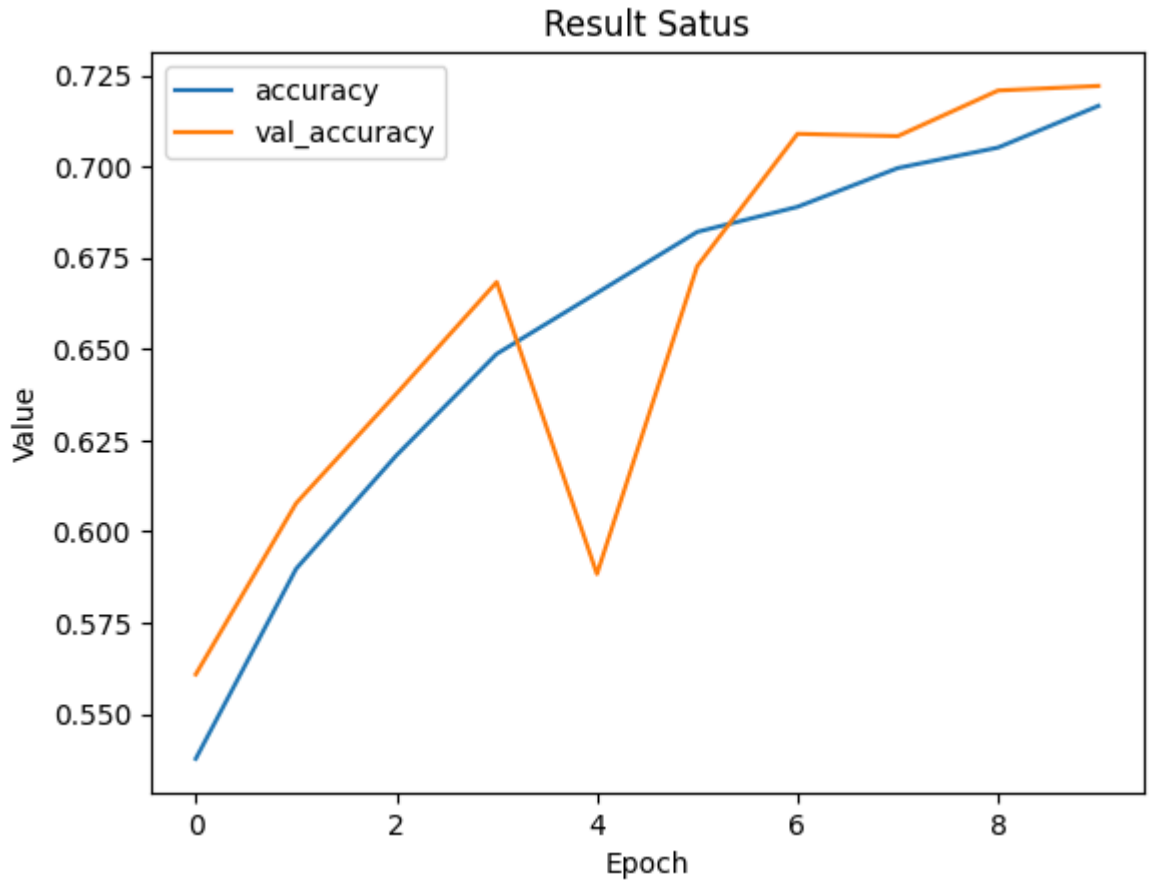
[멋사 13기] AI_윤지현

기존

- cnn 모델
- 에폭 10번



테스트 1

- **학습할 때 반복 수(Epoch) 늘리기**

Epoch(에폭) : 전체 학습 데이터를 모델이 **학습하는 과정**
Epoch을 늘리면 모델이 데이터를 더 많이, 더 깊이 학습할 수 있음
하지만, 너무
많이 학습시키면 **과적합 발생** 가능성

```
history = model.fit(
    train_generator,
    epochs=20,
    validation_data=validation_generator
)
```

→ 20으로 수정

- **데이터 증강**(Image Augmentation) **강화**

실제 이미지 **데이터는** 항상 **다양한 형태**(각도, 위치, 크기 등등)로 나타날 수 있음
데이터 증강은 이미지에
회전, 이동, 확대, 뒤집기 등을 적용해서 **새로운 데이터를 인위적으로 만들어내는 것**
→ 데이터 증강을 통해 다양한 이미지를 학습시킴으로써 정확도를 올리기

```
# 데이터 증강기 + 검증 세트 분리
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,          # 회전 각도 증가 (20 → 30)
    width_shift_range=0.3,      # 가로 이동 범위 증가 (0.2 → 0.3)
    height_shift_range=0.3,     # 세로 이동 범위 증가 (0.2 → 0.3)
    shear_range=0.2,            # 기울이기(shearing) 효과 추가
    zoom_range=0.2,             # 확대/축소 효과 추가
    horizontal_flip=True,
    fill_mode='nearest',        # 이동 시 빈 공간은 가장 가까운 픽셀로 채움
    validation_split=0.2
)
```

변경 내용	이유
`rotation_range=30`	다양한 각도의 이미지를 학습시키기 위해서
`width_shift_range`, `height_shift_range` 증가	강아지·고양이의 위치 변화에 대한 적응력 향상
`shear_range=0.2` 추가	기울어진 이미지에도 잘 대응할 수 있도록

`zoom_range=0.2` 추가	확대된/축소된 이미지에 대한 일반화 성능 향상
`fill_mode='nearest'`	이동이나 회전 시 생기는 공백을 자연스럽게 보정하기 위해

- **모델 구조 수정**

CNN(합성곱 신경망) : **이미지 속 특징을 추출하는 역할**

간단한 구조는 **단순한 모양만 구분**하고, 깊은 구조는 **고양이의 눈 모양, 강아지의 귀 구조 같은 복잡한 특징**도 학습할 수 있음

```
# 모델 정의 (CNN 구조)
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(64, 64, 3)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),  # 세 번째 Conv2D 추가
    tf.keras.layers.MaxPooling2D(2, 2),                      # 세 번째 Pooling 추가
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),          # 노드 수 증가 (64 → 128)
    tf.keras.layers.Dropout(0.3),                           # 과적합 방지를 위한 Dropout 추가
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

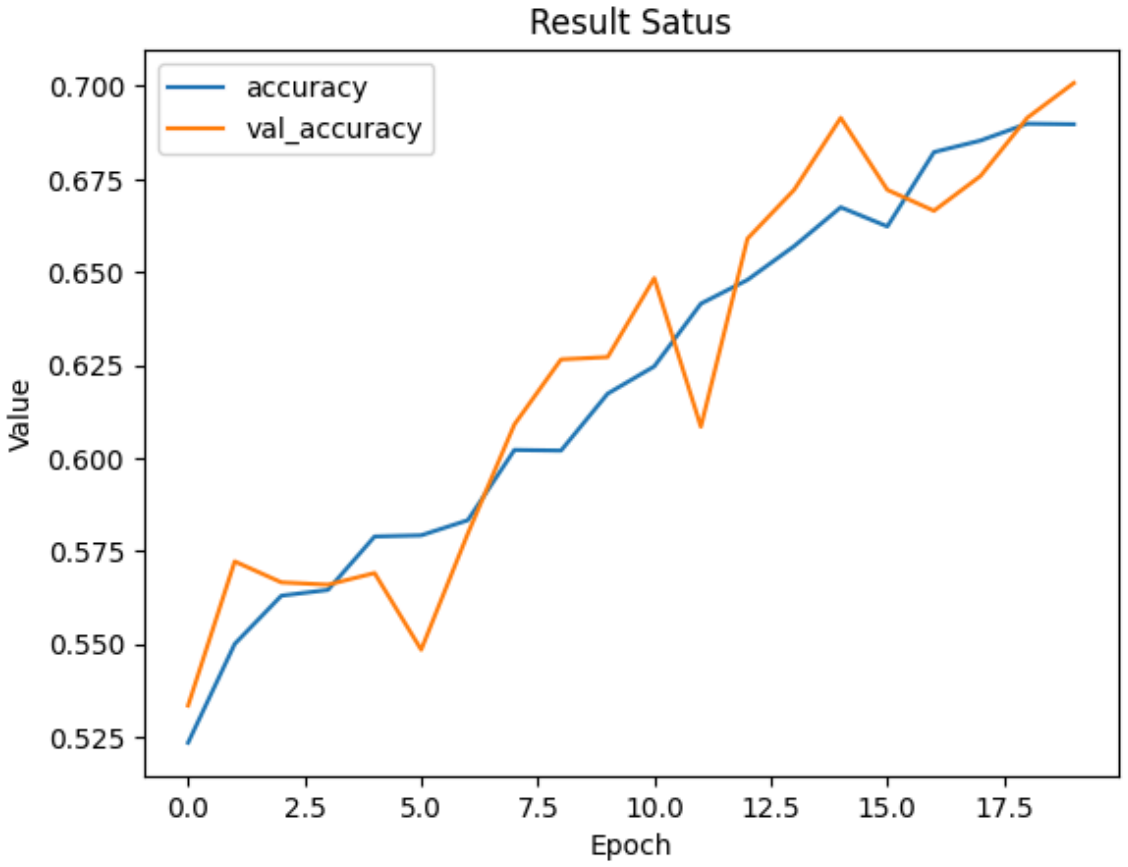
변경 내용	이유
`Conv2D(128)` 추가	더 복잡한 이미지 특징(예: 고양이 눈, 귀 등)까지 학습 가능하게
`Dense(128)` → (기존 64보다 큼)	더 많은 패턴을 표현할 수 있도록 표현력 강화
`Dropout(0.3)` 추가	과적합(overfitting) 방지하여 검증 정확도 향상 기대

***Conv2D** (Convolutional Layer) : 이미지 속 특징(패턴, 모양, 윤곽 등)을 찾아내는 역할, 반복될수록 이미지를 더 자세하게 학습함.

***MaxPooling2D** : 이미지 **크기를 줄이면서 중요한 정보만** 남기는 층.
역할: 계산량 줄이기, 중요 정보 유지 + 잡음 제거, 과적합 방지에도 도움

‼️ 결과

```
Epoch 1/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 91s 68ms/step - accuracy: 0.5193 - loss: 0.6954 - val_accuracy: 0.5334 - val_loss: 0.6895
Epoch 2/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 85s 66ms/step - accuracy: 0.5425 - loss: 0.6890 - val_accuracy: 0.5721 - val_loss: 0.6788
Epoch 3/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 85s 66ms/step - accuracy: 0.5676 - loss: 0.6798 - val_accuracy: 0.5665 - val_loss: 0.6765
Epoch 4/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 145s 69ms/step - accuracy: 0.5611 - loss: 0.6794 - val_accuracy: 0.5659 - val_loss: 0.6713
Epoch 5/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 85s 66ms/step - accuracy: 0.5817 - loss: 0.6691 - val_accuracy: 0.5690 - val_loss: 0.6680
Epoch 6/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 83s 65ms/step - accuracy: 0.5805 - loss: 0.6667 - val_accuracy: 0.5484 - val_loss: 0.6755
Epoch 7/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 85s 66ms/step - accuracy: 0.5744 - loss: 0.6645 - val_accuracy: 0.5796 - val_loss: 0.6659
Epoch 8/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 81s 63ms/step - accuracy: 0.6000 - loss: 0.6602 - val_accuracy: 0.6090 - val_loss: 0.6598
Epoch 9/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 81s 63ms/step - accuracy: 0.5877 - loss: 0.6607 - val_accuracy: 0.6265 - val_loss: 0.6412
Epoch 10/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 84s 66ms/step - accuracy: 0.6142 - loss: 0.6483 - val_accuracy: 0.6271 - val_loss: 0.6360
Epoch 11/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 84s 66ms/step - accuracy: 0.6172 - loss: 0.6411 - val_accuracy: 0.6483 - val_loss: 0.6281
Epoch 12/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 91s 71ms/step - accuracy: 0.6461 - loss: 0.6264 - val_accuracy: 0.6084 - val_loss: 0.6488
Epoch 13/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 86s 67ms/step - accuracy: 0.6448 - loss: 0.6252 - val_accuracy: 0.6590 - val_loss: 0.6137
Epoch 14/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 142s 67ms/step - accuracy: 0.6544 - loss: 0.6221 - val_accuracy: 0.6721 - val_loss: 0.6103
Epoch 15/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 83s 65ms/step - accuracy: 0.6619 - loss: 0.6200 - val_accuracy: 0.6914 - val_loss: 0.5849
Epoch 16/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 82s 64ms/step - accuracy: 0.6643 - loss: 0.6005 - val_accuracy: 0.6721 - val_loss: 0.5990
Epoch 17/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 83s 65ms/step - accuracy: 0.6872 - loss: 0.5988 - val_accuracy: 0.6665 - val_loss: 0.5961
Epoch 18/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 83s 65ms/step - accuracy: 0.6813 - loss: 0.5920 - val_accuracy: 0.6758 - val_loss: 0.6015
Epoch 19/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 84s 65ms/step - accuracy: 0.6963 - loss: 0.5808 - val_accuracy: 0.6914 - val_loss: 0.5990
Epoch 20/20
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 86s 67ms/step - accuracy: 0.6907 - loss: 0.5878 - val_accuracy: 0.7008 - val_loss: 0.5821
```



정확도가 떨어짐. **val_accuracy: 0.7008**

떨어진 이유 추측 :

- `rotation=30`, `shift=0.3` 등 너무 강하게 증강되어 **실제와 다른 이미지**가 학습됨 → 성능 저하
- 과적합 발생 가능성

개선된 부분:

- 그래프 곡선이 비교적 완만해짐
- val_accuracy의 급하락 폭이 줄어듦

테스트 2

- **데이터 증강 범위 완화 + 검증 데이터 불균형 해소**

증강 범위 완화:

rotation_range=20, zoom_range=0.1, shift=0.2

`val_accuracy`가 들쭉날쭉 → 데이터 수가 적거나 클래스 불균형일 수 있음

validation_split을 0.25 이상으로 늘리기 또는 class_weight로 가중치 보정

```
# 데이터 증강기 + 검증 세트 분리
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,          # 30 → 20
    width_shift_range=0.2,      # 0.3 → 0.2
    height_shift_range=0.2,     # 0.3 → 0.2
    shear_range=0.2,
    zoom_range=0.1,             # 0.2 → 0.1
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.25       # 0.2 → 0.25
)
```

- **과적합 방지**

학습 정확도는 높고 검증 정확도는 들쭉날쭉함 → 일반화 실패 가능성

Dropout, BatchNormalization 추가 + EarlyStopping 도입

```
early_stop = EarlyStopping(
    monitor='val_loss',          # val_loss 기준으로
    patience=5,                  # 5번 참음
    restore_best_weights=True, # 가장 좋은 지점 weight로 복원
    verbose=1                    # 메시지 출력
)

history = model.fit(
    train_generator,
    epochs=25,  # 20 → 25
    validation_data=validation_generator,
    callbacks=[early_stop]  # 콜백 추가

)
```

콜백 : 학습 중 `val_loss`가 **5 epoch 연속 좋아지지 않으면 학습 자동 종료**시킴. (과적합 방지용)

```
# 검증용 제너레이터
validation_generator = train_datagen.flow_from_directory(
    '/content/data/',
    target_size=(128, 128),
    batch_size=16,
    class_mode='binary',
    subset='validation'
)

# 모델 정의 (CNN 구조)
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(128, 128, 3)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.BatchNormalization(),  # BatchNormalization 추가
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.BatchNormalization(),  # 추가
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.BatchNormalization(),  # 추가
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

`batch_size=5` → 16

batch_size : 한 번의 학습에 몇 장의 이미지를 동시에 처리할지를 정하는 값.
학습 안정성과 속도 사이 균형이 좋아짐

`target_size=(64, 64)` → (128, 128)

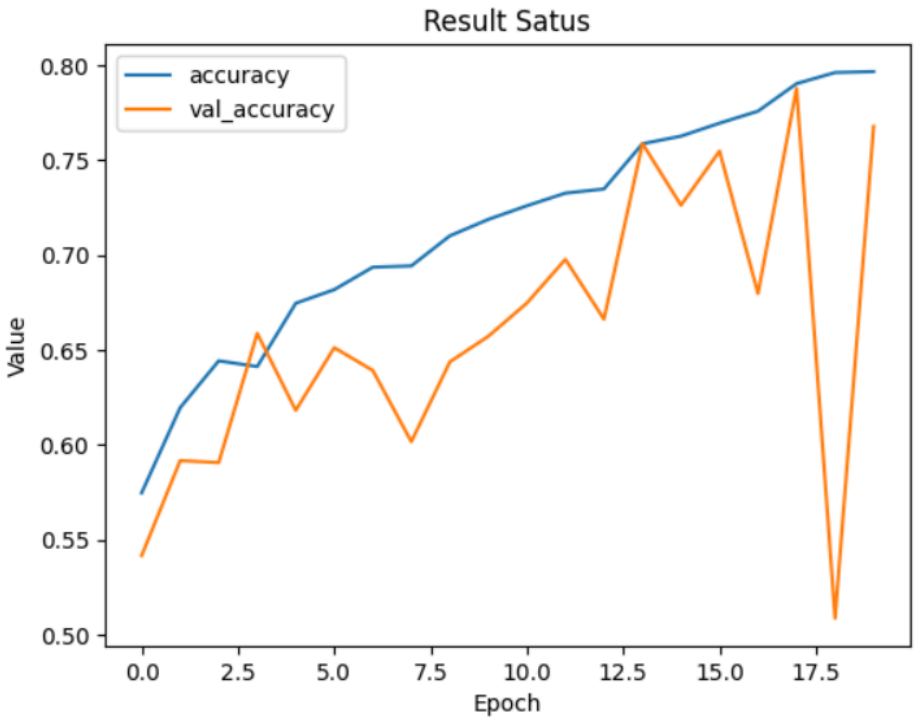
target_size : 이미지를 모델에 넣기 전에 크기를 얼마나 축소/확대해서 처리할지를 정하는 것

더 섬세한 정보를 모델이 볼 수 있음. 즉, 정확도 개선 효과
하지만, 너무 키우면 학습 시간 증가함

‼️ 결과

```
Epoch 1/20
376/376 ——————————— 328s 860ms/step - accuracy: 0.5521 - loss: 1.7526 - val_accuracy: 0.5417 - val_loss: 0.7054
Epoch 2/20
376/376 ——————————— 325s 863ms/step - accuracy: 0.6011 - loss: 0.6606 - val_accuracy: 0.5917 - val_loss: 0.6689
Epoch 3/20
376/376 ——————————— 333s 887ms/step - accuracy: 0.6315 - loss: 0.6489 - val_accuracy: 0.5907 - val_loss: 0.6662
Epoch 4/20
376/376 ——————————— 315s 837ms/step - accuracy: 0.6346 - loss: 0.6615 - val_accuracy: 0.6587 - val_loss: 0.6278
Epoch 5/20
376/376 ——————————— 311s 827ms/step - accuracy: 0.6611 - loss: 0.6128 - val_accuracy: 0.6182 - val_loss: 0.6410
Epoch 6/20
376/376 ——————————— 312s 829ms/step - accuracy: 0.6857 - loss: 0.5919 - val_accuracy: 0.6512 - val_loss: 0.6216
```

```
Epoch 7/20
376/376 ——————————— 310s 825ms/step - accuracy: 0.6960 - loss: 0.5868 - val_accuracy: 0.6392 - val_loss: 0.6445
Epoch 8/20
376/376 ——————————— 319s 817ms/step - accuracy: 0.6883 - loss: 0.5805 - val_accuracy: 0.6017 - val_loss: 0.6471
Epoch 9/20
376/376 ——————————— 308s 820ms/step - accuracy: 0.7053 - loss: 0.5643 - val_accuracy: 0.6437 - val_loss: 0.6602
Epoch 10/20
376/376 ——————————— 307s 817ms/step - accuracy: 0.7182 - loss: 0.5586 - val_accuracy: 0.6572 - val_loss: 0.5978
Epoch 11/20
376/376 ——————————— 321s 815ms/step - accuracy: 0.7222 - loss: 0.5422 - val_accuracy: 0.6747 - val_loss: 0.5988
Epoch 12/20
376/376 ——————————— 305s 811ms/step - accuracy: 0.7342 - loss: 0.5288 - val_accuracy: 0.6977 - val_loss: 0.5905
Epoch 13/20
376/376 ——————————— 304s 808ms/step - accuracy: 0.7262 - loss: 0.5309 - val_accuracy: 0.6662 - val_loss: 0.5763
Epoch 14/20
376/376 ——————————— 303s 807ms/step - accuracy: 0.7533 - loss: 0.5196 - val_accuracy: 0.7586 - val_loss: 0.4957
Epoch 15/20
376/376 ——————————— 313s 833ms/step - accuracy: 0.7577 - loss: 0.4986 - val_accuracy: 0.7261 - val_loss: 0.5346
Epoch 16/20
376/376 ——————————— 317s 843ms/step - accuracy: 0.7673 - loss: 0.4818 - val_accuracy: 0.7546 - val_loss: 0.4977
Epoch 17/20
376/376 ——————————— 327s 870ms/step - accuracy: 0.7737 - loss: 0.4873 - val_accuracy: 0.6797 - val_loss: 0.5911
Epoch 18/20
376/376 ——————————— 316s 839ms/step - accuracy: 0.7904 - loss: 0.4601 - val_accuracy: 0.7876 - val_loss: 0.4650
Epoch 19/20
376/376 ——————————— 316s 841ms/step - accuracy: 0.8020 - loss: 0.4435 - val_accuracy: 0.5087 - val_loss: 3.4531
Epoch 20/20
376/376 ——————————— 319s 832ms/step - accuracy: 0.7995 - loss: 0.4349 - val_accuracy: 0.7676 - val_loss: 0.4758
Restoring model weights from the end of the best epoch: 18.
```



val_accuracy: 0.7876

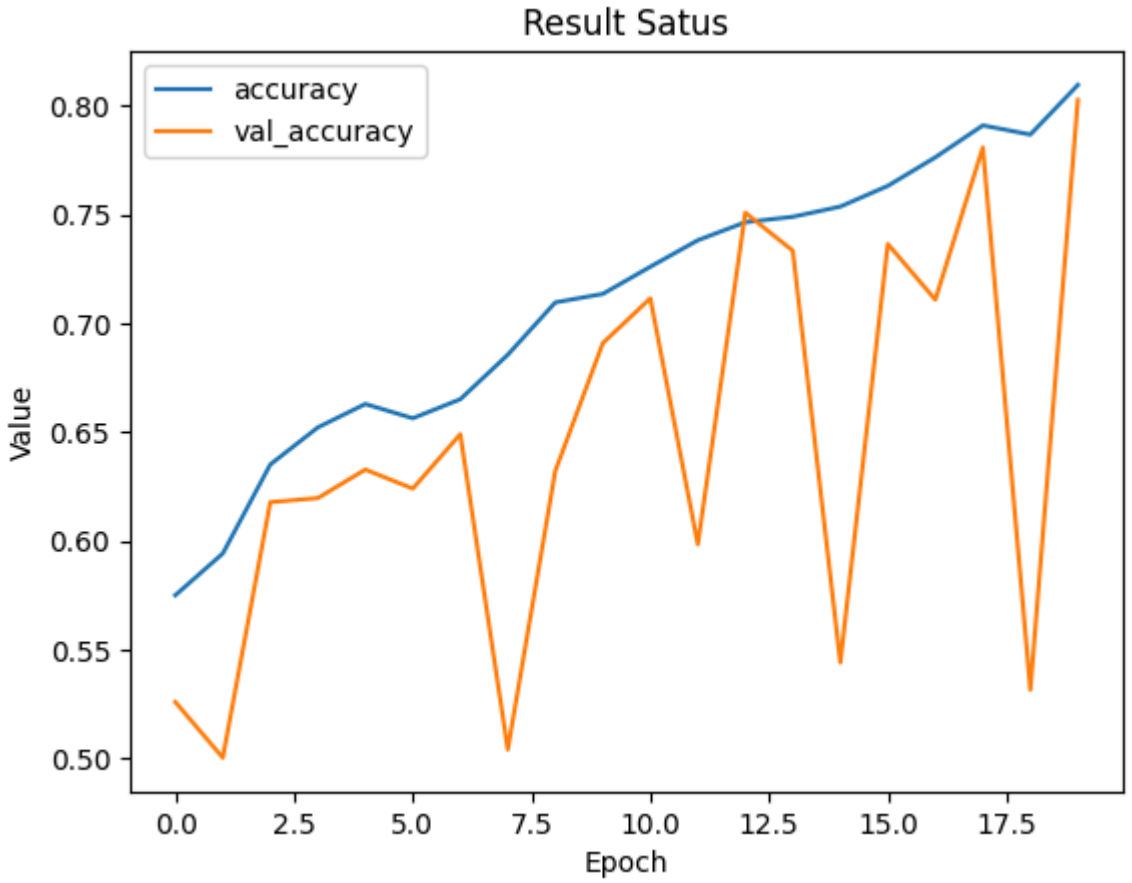
테스트 3

```python
# 데이터 증강기 + 검증 세트 분리
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2      # 0.25 → 0.2
)
```

이부분만 수정

‼️ 결과

```
Epoch 1/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 314s 773ms/step - accuracy: 0.5630 - loss: 1.5296 - val_accuracy: 0.5259 - val_loss: 0.7870
Epoch 2/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 313s 780ms/step - accuracy: 0.5878 - loss: 0.6691 - val_accuracy: 0.5003 - val_loss: 0.6905
Epoch 3/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 316s 788ms/step - accuracy: 0.6302 - loss: 0.6473 - val_accuracy: 0.6177 - val_loss: 0.6537
Epoch 4/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 312s 779ms/step - accuracy: 0.6528 - loss: 0.6343 - val_accuracy: 0.6196 - val_loss: 0.8394
Epoch 5/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 312s 779ms/step - accuracy: 0.6523 - loss: 0.6196 - val_accuracy: 0.6327 - val_loss: 0.6376
Epoch 6/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 307s 766ms/step - accuracy: 0.6570 - loss: 0.6094 - val_accuracy: 0.6240 - val_loss: 0.6632
Epoch 7/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 305s 760ms/step - accuracy: 0.6547 - loss: 0.6131 - val_accuracy: 0.6490 - val_loss: 0.6232
Epoch 8/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 306s 762ms/step - accuracy: 0.6733 - loss: 0.5893 - val_accuracy: 0.5041 - val_loss: 1.3966
Epoch 9/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 304s 758ms/step - accuracy: 0.7079 - loss: 0.5654 - val_accuracy: 0.6321 - val_loss: 0.5860
Epoch 10/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 308s 767ms/step - accuracy: 0.7013 - loss: 0.5697 - val_accuracy: 0.6908 - val_loss: 0.5873
Epoch 11/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 305s 760ms/step - accuracy: 0.7208 - loss: 0.5461 - val_accuracy: 0.7114 - val_loss: 0.5565
Epoch 12/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 306s 763ms/step - accuracy: 0.7375 - loss: 0.5190 - val_accuracy: 0.5984 - val_loss: 0.7540
Epoch 13/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 305s 760ms/step - accuracy: 0.7500 - loss: 0.5269 - val_accuracy: 0.7508 - val_loss: 0.5143
Epoch 14/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 306s 763ms/step - accuracy: 0.7449 - loss: 0.5139 - val_accuracy: 0.7333 - val_loss: 0.5480
Epoch 15/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 302s 752ms/step - accuracy: 0.7556 - loss: 0.4987 - val_accuracy: 0.5440 - val_loss: 0.9089
Epoch 16/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 324s 757ms/step - accuracy: 0.7459 - loss: 0.5016 - val_accuracy: 0.7364 - val_loss: 0.5136
Epoch 17/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 303s 756ms/step - accuracy: 0.7826 - loss: 0.4636 - val_accuracy: 0.7108 - val_loss: 0.6070
Epoch 18/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 308s 767ms/step - accuracy: 0.7894 - loss: 0.4559 - val_accuracy: 0.7808 - val_loss: 0.4611
Epoch 19/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 306s 764ms/step - accuracy: 0.7910 - loss: 0.4603 - val_accuracy: 0.5315 - val_loss: 1.4797
Epoch 20/20
401/401 ━━━━━━━━━━━━━━━━━━━━ 326s 812ms/step - accuracy: 0.8062 - loss: 0.4245 - val_accuracy: 0.8026 - val_loss: 0.4258
Restoring model weights from the end of the best epoch: 20.
```



val_accuracy: 0.8026는 더 높게 나왔지만 그래프는 더 들쭉날쭉해짐.

테스트 4

그래프 안정성을 높여야할듯.

- `ReduceLROnPlateau` **콜백 추가**

모델이 **성능 개선 없이 정체되면**, **학습률(learning rate)을 자동으로 줄이는 콜백**

val_accuracy의 흔들림 줄이기 위해서 추가함

```
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True,
    verbose=1
)

lr_scheduler = ReduceLROnPlateau(           # 수정: ReduceLROnPlateau 추가
    monitor='val_loss',
    factor=0.5,
    patience=3,
    min_lr=1e-6,
    verbose=1
)
```

- `batch_size=32` **로 통일**

한 번에 모델이 학습하는 이미지 수를 늘리는 것

```
# 학습용 제너레이터
train_generator = train_datagen.flow_from_directory(
    '/content/data/',
    target_size=(128, 128),
    batch_size=32,  # 수정: batch_size 16 → 32
    class_mode='binary',
    subset='training'
)
```

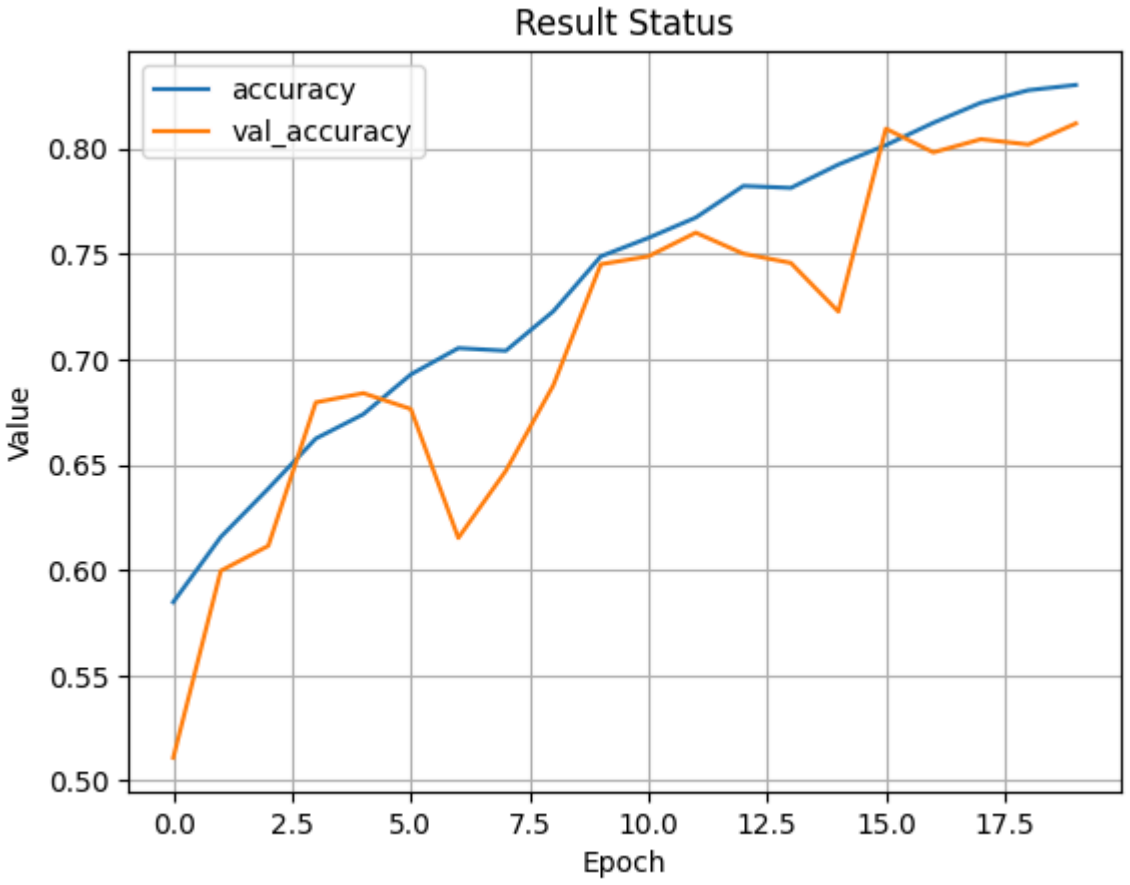
- `plt.grid(True)` 추가

그래프에 격자선을 표시

```
[ ]  plt.plot(history.history['accuracy'], label='accuracy')
     plt.plot(history.history['val_accuracy'], label='val_accuracy
     plt.xlabel('Epoch')
     plt.ylabel('Value')
     plt.title('Result Status')
     plt.legend()
     plt.grid(True)
     plt.show()
```

‼ 결과

```
Epoch 1/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 387s 2s/step - accuracy: 0.5736 - loss: 1.3372 - val_accuracy: 0.5109 - val_loss: 0.9484 - learning_rate: 0.0010
Epoch 2/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 363s 2s/step - accuracy: 0.6116 - loss: 0.7024 - val_accuracy: 0.5996 - val_loss: 0.6757 - learning_rate: 0.0010
Epoch 3/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 378s 2s/step - accuracy: 0.6458 - loss: 0.6319 - val_accuracy: 0.6115 - val_loss: 0.6425 - learning_rate: 0.0010
Epoch 4/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 386s 2s/step - accuracy: 0.6606 - loss: 0.6478 - val_accuracy: 0.6796 - val_loss: 0.5930 - learning_rate: 0.0010
Epoch 5/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 367s 2s/step - accuracy: 0.6624 - loss: 0.6146 - val_accuracy: 0.6839 - val_loss: 0.5851 - learning_rate: 0.0010
Epoch 6/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 371s 2s/step - accuracy: 0.6790 - loss: 0.5827 - val_accuracy: 0.6765 - val_loss: 0.5787 - learning_rate: 0.0010
Epoch 7/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 369s 2s/step - accuracy: 0.7140 - loss: 0.5611 - val_accuracy: 0.6152 - val_loss: 0.6760 - learning_rate: 0.0010
Epoch 8/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 373s 2s/step - accuracy: 0.7029 - loss: 0.5660 - val_accuracy: 0.6471 - val_loss: 0.8834 - learning_rate: 0.0010
Epoch 9/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7152 - loss: 0.5514
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
201/201 ━━━━━━━━━━━━━━━━━━━━ 383s 2s/step - accuracy: 0.7152 - loss: 0.5514 - val_accuracy: 0.6877 - val_loss: 0.6328 - learning_rate: 0.0010
Epoch 10/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 365s 2s/step - accuracy: 0.7406 - loss: 0.5178 - val_accuracy: 0.7452 - val_loss: 0.5116 - learning_rate: 5.0000e-04
Epoch 11/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 374s 2s/step - accuracy: 0.7556 - loss: 0.4988 - val_accuracy: 0.7489 - val_loss: 0.5131 - learning_rate: 5.0000e-04
Epoch 12/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 375s 2s/step - accuracy: 0.7634 - loss: 0.4911 - val_accuracy: 0.7601 - val_loss: 0.4878 - learning_rate: 5.0000e-04
Epoch 13/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 391s 2s/step - accuracy: 0.7782 - loss: 0.4673 - val_accuracy: 0.7502 - val_loss: 0.5054 - learning_rate: 5.0000e-04
Epoch 14/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 378s 2s/step - accuracy: 0.7804 - loss: 0.4638 - val_accuracy: 0.7458 - val_loss: 0.5115 - learning_rate: 5.0000e-04
Epoch 15/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7979 - loss: 0.4415
Epoch 15: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
201/201 ━━━━━━━━━━━━━━━━━━━━ 390s 2s/step - accuracy: 0.7979 - loss: 0.4416 - val_accuracy: 0.7227 - val_loss: 0.5607 - learning_rate: 5.0000e-04
Epoch 16/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 372s 2s/step - accuracy: 0.7961 - loss: 0.4254 - val_accuracy: 0.8095 - val_loss: 0.4244 - learning_rate: 2.5000e-04
Epoch 17/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 362s 2s/step - accuracy: 0.8185 - loss: 0.3997 - val_accuracy: 0.7983 - val_loss: 0.4176 - learning_rate: 2.5000e-04
Epoch 18/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 382s 2s/step - accuracy: 0.8241 - loss: 0.3972 - val_accuracy: 0.8045 - val_loss: 0.4232 - learning_rate: 2.5000e-04
Epoch 19/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 379s 2s/step - accuracy: 0.8236 - loss: 0.3930 - val_accuracy: 0.8020 - val_loss: 0.4671 - learning_rate: 2.5000e-04
Epoch 20/20
201/201 ━━━━━━━━━━━━━━━━━━━━ 366s 2s/step - accuracy: 0.8278 - loss: 0.3810 - val_accuracy: 0.8120 - val_loss: 0.3867 - learning_rate: 2.5000e-04
Restoring model weights from the end of the best epoch: 20.
```



정확도도 높아지고 이전보다 그래프가 안정됨.

`val_accuracy : 0.8120`

느낀점

: ai모델을 처음 다뤄봤는데 성과가 나올때마다 성취감있고 재미있었다. 하지만 학습하는 시간이 너무 오래걸려서 기다리기 힘들기도 했다.

데이터 전처리, 하이퍼파라미터 조정, 콜백 설정 같은 사소하다고 생각했던 설정들이 생각보다 모델 성능에 큰 영향을 미친다는 것을 체감했다.