

[멋사 13기][AI_김서진]

1. 정확도를 높이기 위해 시도한 방법과 과정

1차 변경점

✓ 4. 이미지 데이터 전처리

- rescale: 픽셀값 정규화 (0~1)
- target size: (64,64)로 이미지 크기 통일(리사이징)
- batch_size: 한 번에 몇 번씩 학습할건지 정의
- class mode: (=binary) 이진 분류

```
[4] train_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        "/content/data",
        target_size=(64, 64),
        batch_size=10,
        class_mode='binary'
    )
```

- batch_size가 한 번에 몇 번씩 학습할 것인지 정의한다고 하였기에 batch_size를 5 → 10으로 변경하였습니다.

```

# 데이터 증강기 + 검증 세트 분리
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

# 학습용 제너레이터
train_generator = train_datagen.flow_from_directory(
    '/content/data/',
    target_size=(64, 64),
    batch_size=10,
    class_mode='binary',
    subset='training'
)

# 검증용 제너레이터
validation_generator = train_datagen.flow_from_directory(
    '/content/data/',
    target_size=(64, 64),
    batch_size=10,
    class_mode='binary',
    subset='validation'
)

# 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.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.01))
])

# 컴파일
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

```

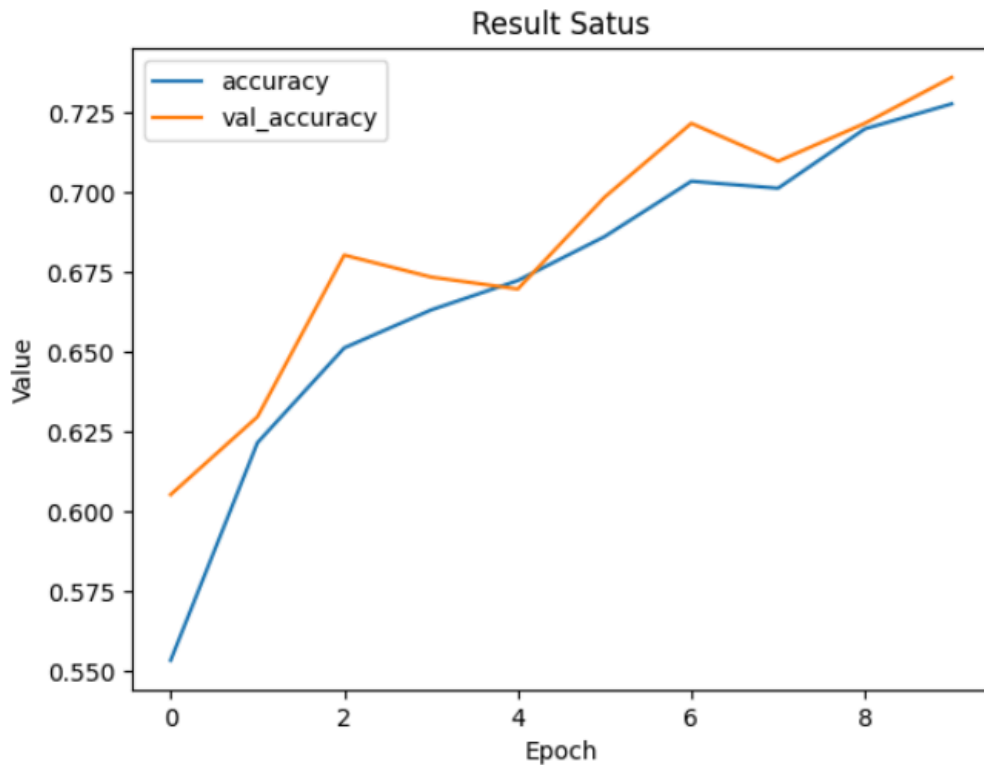
- 그 이하도 동일하게 학습용 제너레이터와 검증용 제너레이터 부분의 batch_size도 5 → 10으로 변경하였습니다.
- 추가로 파라미터 중에 kernel_regularizer라는 파라미터를 확인했습니다.
- kernel_regularizer 란?
 - 필터 가중치에 적용할 정규화 방법
 - ex) tf.keras.regularizers.l2(0.01) → L2 정규화, 가중치의 제곱 합에 패널티 부여
 - 과적합 방지에 유용
- 저는 이 파라미터를 가장 밑줄에 추가하여 batch_size를 늘린 만큼 학습 정확도와 검증 정확도가 일치하지 않는 것, 즉 과적합이 많이 발생할 것이라고 예상하여 과적합 방지를 위해 사용하였습니다.

```
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your "PyDataset" class should call "super().__init__(**kwargs)" in its constructor. "**kwargs" can include "workers", "use_multiprocessing", "max_queue_size". Do not pass these arguments to self._warn_if_super_not_called()
63s 59ste/step - accuracy: 0.5295 - loss: 0.7019 - val_accuracy: 0.6052 - val_loss: 0.6634
Epoch 2/10
641/641 ----- 59s 59ste/step - accuracy: 0.6157 - loss: 0.6541 - val_accuracy: 0.6266 - val_loss: 0.6451
Epoch 3/10
641/641 ----- 59s 59ste/step - accuracy: 0.6441 - loss: 0.6313 - val_accuracy: 0.6802 - val_loss: 0.6034
Epoch 4/10
641/641 ----- 59s 59ste/step - accuracy: 0.6633 - loss: 0.6122 - val_accuracy: 0.6793 - val_loss: 0.6101
Epoch 5/10
641/641 ----- 60s 59ste/step - accuracy: 0.6719 - loss: 0.5977 - val_accuracy: 0.6696 - val_loss: 0.6163
Epoch 6/10
641/641 ----- 63s 59ste/step - accuracy: 0.6859 - loss: 0.5772 - val_accuracy: 0.6983 - val_loss: 0.5665
Epoch 7/10
641/641 ----- 59s 59ste/step - accuracy: 0.7068 - loss: 0.5698 - val_accuracy: 0.7214 - val_loss: 0.5568
Epoch 8/10
641/641 ----- 60s 59ste/step - accuracy: 0.6986 - loss: 0.5762 - val_accuracy: 0.7096 - val_loss: 0.5597
Epoch 9/10
641/641 ----- 59s 59ste/step - accuracy: 0.7225 - loss: 0.5482 - val_accuracy: 0.7214 - val_loss: 0.5368
Epoch 10/10
641/641 ----- 62s 59ste/step - accuracy: 0.7300 - loss: 0.5403 - val_accuracy: 0.7358 - val_loss: 0.5323
```

- accuracy
 - 학습 데이터에 대한 모델의 정확도
- val_accuracy
 - 검증 데이터에 대한 모델의 정확도
- 시작 (Epoch 1/10)
 - accuracy - 0.5295(52.95%)
 - val_accuracy - 0.6052(60.52%)
- 최종 (Epoch 10/10)
 - accuracy - 0.7300(73.00%)
 - val_accuracy - 0.7358(73.58%)
- 최고 정확도
 - accuracy - 0.7300(73.00%)
 - val_accuracy - 0.7358(73.58%)

결과는 이렇게 나왔으며 정확도는 **73%**로 확인되었습니다.



- 그래프를 확인했을 때 과적합이 크게 발생하진 않은 것을 확인했습니다.



예측할 이미지를 업로드하세요.

파일 선택 다운로드.jpg

• **다운로드.jpg**(image/jpeg) - 5440 bytes, last modified: 2025. 5. 28. - 100% done

Saving 다운로드.jpg to 다운로드 (1).jpg

1/1 ————— 0s 46ms/step

다운로드 (1).jpg: 고양이입니다.



예측할 이미지를 업로드하세요.

파일 선택 7170113c6a983.jpg

• **7170113c6a983.jpg**(image/jpeg) - 156760 bytes, last modified: 2025. 5. 28. - 100% done

Saving 7170113c6a983.jpg to 7170113c6a983 (2).jpg

1/1 ————— 0s 48ms/step

7170113c6a983 (2).jpg: 강아지입니다.

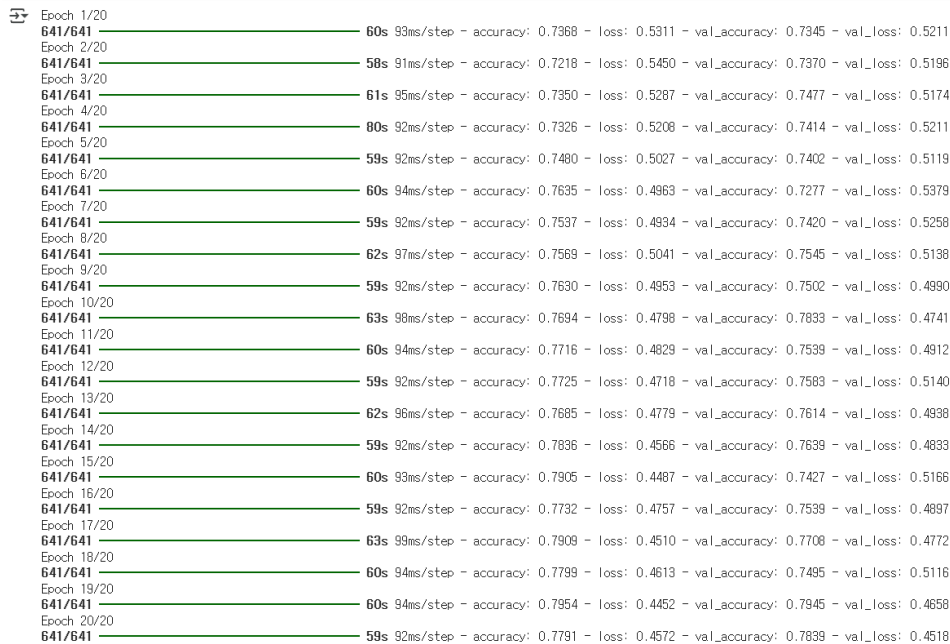
- 두 번의 이미지 예측 결과
 - 첫 번째 : 고양이 사진 → 고양이
 - 두 번째 : 고양이 사진 → 강아지

2차 변경점

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

- 2차 변경 점으로는 전체 학습 반복 수인 epochs를 10→20으로 변경하였습니다.

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

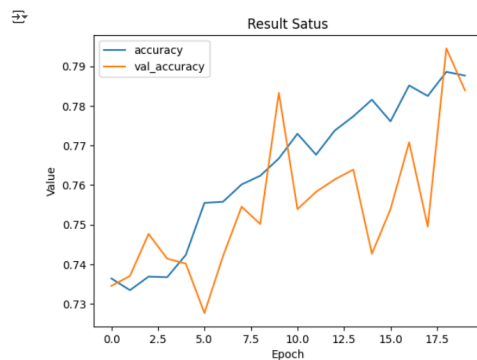


Epoch	Time/Step	accuracy	loss	val_accuracy	val_loss
Epoch 1/20	60s 93ms/step	0.7368	0.5311	0.7345	0.5211
Epoch 2/20	58s 91ms/step	0.7218	0.5450	0.7370	0.5196
Epoch 3/20	61s 95ms/step	0.7350	0.5287	0.7477	0.5174
Epoch 4/20	80s 92ms/step	0.7326	0.5208	0.7414	0.5211
Epoch 5/20	59s 92ms/step	0.7480	0.5027	0.7402	0.5119
Epoch 6/20	60s 94ms/step	0.7635	0.4963	0.7277	0.5379
Epoch 7/20	59s 92ms/step	0.7537	0.4934	0.7420	0.5258
Epoch 8/20	62s 97ms/step	0.7569	0.5041	0.7545	0.5198
Epoch 9/20	59s 92ms/step	0.7630	0.4953	0.7502	0.4990
Epoch 10/20	63s 98ms/step	0.7694	0.4798	0.7833	0.4741
Epoch 11/20	60s 94ms/step	0.7716	0.4829	0.7539	0.4912
Epoch 12/20	59s 92ms/step	0.7725	0.4718	0.7583	0.5140
Epoch 13/20	62s 96ms/step	0.7685	0.4779	0.7614	0.4998
Epoch 14/20	59s 92ms/step	0.7836	0.4566	0.7639	0.4893
Epoch 15/20	60s 93ms/step	0.7905	0.4487	0.7427	0.5166
Epoch 16/20	59s 92ms/step	0.7732	0.4757	0.7539	0.4897
Epoch 17/20	63s 99ms/step	0.7909	0.4510	0.7708	0.4772
Epoch 18/20	60s 94ms/step	0.7799	0.4613	0.7495	0.5116
Epoch 19/20	60s 94ms/step	0.7954	0.4452	0.7945	0.4658
Epoch 20/20	59s 92ms/step	0.7791	0.4572	0.7839	0.4518

- 시작 (Epoch 1/10)
 - accuracy - 0.7368(73.68%)
 - val_accuracy - 0.7345(73.45%)
- 최종 (Epoch 10/10)
 - accuracy - 0.7791(77.91%)

- val_accuracy - 0.7839(78.39%)
- 최고 정확도
 - accuracy - 0.7954(79.54%)
 - val_accuracy - 0.7945(79.45%)

2차 결과는 이렇게 나왔으며 정확도는 **77~79%**으로 확인되었습니다.



- 그래프를 확인했을 때 아무래도 전체 반복 수까지 늘리다 보니 1차 그래프보다 과적합이 많이 발생한 것을 확인 할 수 있습니다.

🔄 예측할 이미지를 업로드하세요.

파일 선택 2025040803041_0.jpg

- 2025040803041_0.jpg(image/jpeg) - 24778 bytes, last modified: 2025. 5. 28. - 100% done

Saving 2025040803041_0.jpg to 2025040803041_0.jpg

1/1 0s 369ms/step

2025040803041_0.jpg: 고양이입니다.

🔄 예측할 이미지를 업로드하세요.

파일 선택 강아지.jpg

- 강아지.jpg(image/jpeg) - 6159 bytes, last modified: 2025. 5. 29. - 100% done

Saving 강아지.jpg to 강아지 (1).jpg

1/1 0s 45ms/step

강아지 (1).jpg: 강아지입니다.

- 두 번의 이미지 예측 결과
 - 첫 번째 : 강아지 사진 → 고양이

- 두 번째 : 강아지 사진 → 강아지
-

느낀 점

- AI를 학습시켜 이미지 분석을 시킨다는 실습 자체가 흥미로워서 재미있게 했으며 코드를 수정할 때마다 결과값이 달라지는 것을 보고 추가적인 공부를 하게 유도 되는 것 같아 도움이 됐던 것 같습니다.