# 파이썬 이미지 분류기 모델

확인 확인하기

## 목표

### 정확도 72%이상

## 기존

6. 모델 학습시키기

- epochs: 전체 학습 반복 수 정의

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

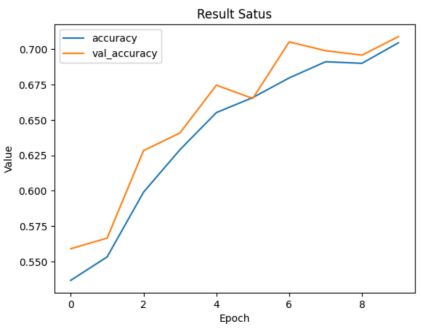
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can incl
  self._warn_if_super_not_called()
Epoch 1/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 69s 52ms/step - accuracy: 0.5213 - loss: 0.7072 - val_accuracy: 0.5590 - val_loss: 0.6834
Epoch 2/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 50ms/step - accuracy: 0.5456 - loss: 0.6869 - val_accuracy: 0.5665 - val_loss: 0.6766
Epoch 3/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 64s 50ms/step - accuracy: 0.6022 - loss: 0.6629 - val_accuracy: 0.6284 - val_loss: 0.6419
Epoch 4/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 84s 52ms/step - accuracy: 0.6312 - loss: 0.6387 - val_accuracy: 0.6408 - val_loss: 0.6413
Epoch 5/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 51ms/step - accuracy: 0.6487 - loss: 0.6268 - val_accuracy: 0.6746 - val_loss: 0.6062
Epoch 6/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 66s 51ms/step - accuracy: 0.6573 - loss: 0.6178 - val_accuracy: 0.6652 - val_loss: 0.5988
Epoch 7/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 64s 50ms/step - accuracy: 0.6786 - loss: 0.5976 - val_accuracy: 0.7052 - val_loss: 0.5714
Epoch 8/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 64s 50ms/step - accuracy: 0.6883 - loss: 0.5879 - val_accuracy: 0.6989 - val_loss: 0.5674
Epoch 9/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 51ms/step - accuracy: 0.6856 - loss: 0.5787 - val_accuracy: 0.6958 - val_loss: 0.5839
Epoch 10/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 64s 50ms/step - accuracy: 0.7068 - loss: 0.5627 - val_accuracy: 0.7089 - val_loss: 0.5598
```

기존 검증 정확도 약 70%

7. 학습 결과 시각화

- plt.plot: 학습 정확도 그래프 출력
- 두 그래프가 비슷하게 올라가면 일반화 성공/벌어지면 과적합 발생

```
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 Satus')
plt.legend()
plt.show()
```



8. 이미지 업로드 후 예측하기

- 예측 확률이 0.5 이상이면 강아지/이하면 고양이로 판단
- 직접 사진 넣어서 테스트 해보기!!

```
print("예측할 이미지를 업로드하세요.")
uploaded = files.upload()

for fn in uploaded.keys():
    # 이미지 로드
    img = load_img(fn, target_size=(64, 64))
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    # 예측
    prediction = model.predict(img_array)

    if prediction[0][0] > 0.5:
        print(f'{fn}: 강아지입니다.')
    else:
        print(f'{fn}: 고양이입니다.')
```

```
예측할 이미지를 업로드하세요.
Choose Files 잠낭이.jpg
- 잠낭이.jpg(image/jpeg) - 175375 bytes, last modified: 2025. 5. 27. - 100% done
Saving 잠낭이.jpg to 잠낭이.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
잠낭이.jpg: 고양이입니다.
```

8. 이미지 업로드 후 예측하기

- 예측 확률이 0.5 이상이면 강아지/이하면 고양이로 판단
- 직접 사진 넣어서 테스트 해보기!!

```
print("예측할 이미지를 업로드하세요.")
uploaded = files.upload()

for fn in uploaded.keys():
    # 이미지 로드
    img = load_img(fn, target_size=(64, 64))
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    # 예측
    prediction = model.predict(img_array)

    if prediction[0][0] > 0.5:
        print(f'{fn}: 강아지입니다.')
    else:
        print(f'{fn}: 고양이입니다.')
```

```
예측할 이미지를 업로드하세요
Choose Files 머멍이.jpg
- 머멍이.jpg(image/jpeg) - 64017 bytes, last modified: 2025. 6. 1. - 100% done
Saving 머멍이.jpg to 머멍이.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
머멍이.jpg: 고양이입니다.
```

# 1차 시도 - epochs 변경(10→15회)

6. 모델 학습시키기

- epochs: 전체 학습 반복 수 정의

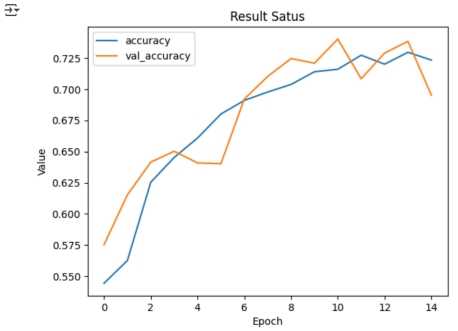
```
history = model.fit(
    train_generator,
    epochs=15,
    validation_data=validation_generator
)
```

```
Epoch 1/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 66s 52ms/step - accuracy: 0.7118 - loss: 0.5524 - val_accuracy: 0.6971 - val_loss: 0.5685
Epoch 2/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 51ms/step - accuracy: 0.7164 - loss: 0.5570 - val_accuracy: 0.7239 - val_loss: 0.5488
Epoch 3/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 66s 51ms/step - accuracy: 0.7139 - loss: 0.5494 - val_accuracy: 0.7027 - val_loss: 0.5757
Epoch 4/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 51ms/step - accuracy: 0.7161 - loss: 0.5465 - val_accuracy: 0.7064 - val_loss: 0.5494
Epoch 5/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 51ms/step - accuracy: 0.7331 - loss: 0.5327 - val_accuracy: 0.7183 - val_loss: 0.5494
Epoch 6/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 51ms/step - accuracy: 0.7295 - loss: 0.5334 - val_accuracy: 0.7245 - val_loss: 0.5484
Epoch 7/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 66s 51ms/step - accuracy: 0.7351 - loss: 0.5260 - val_accuracy: 0.7077 - val_loss: 0.5694
Epoch 8/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 51ms/step - accuracy: 0.7452 - loss: 0.5115 - val_accuracy: 0.6702 - val_loss: 0.6098
Epoch 9/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 64s 50ms/step - accuracy: 0.7321 - loss: 0.5270 - val_accuracy: 0.7389 - val_loss: 0.5380
Epoch 10/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 64s 50ms/step - accuracy: 0.7485 - loss: 0.5129 - val_accuracy: 0.7339 - val_loss: 0.5510
Epoch 11/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 51ms/step - accuracy: 0.7500 - loss: 0.4992 - val_accuracy: 0.7345 - val_loss: 0.5350
Epoch 12/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 64s 50ms/step - accuracy: 0.7603 - loss: 0.4930 - val_accuracy: 0.7483 - val_loss: 0.5103
Epoch 13/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 50ms/step - accuracy: 0.7541 - loss: 0.5013 - val_accuracy: 0.7452 - val_loss: 0.5057
Epoch 14/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 65s 50ms/step - accuracy: 0.7685 - loss: 0.4842 - val_accuracy: 0.7458 - val_loss: 0.5174
Epoch 15/15
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 82s 50ms/step - accuracy: 0.7482 - loss: 0.5049 - val_accuracy: 0.7651 - val_loss: 0.4897
```

7. 학습 결과 시각화

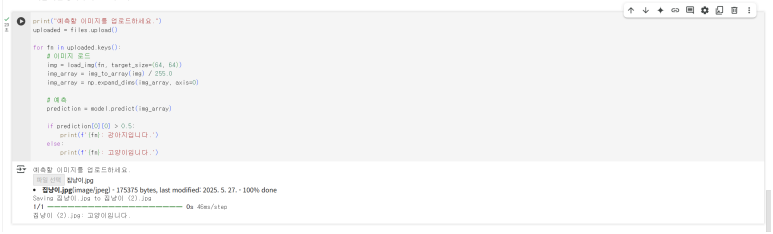
- plt.plot: 학습 정확도 그래프 출력
- 두 그래프가 비슷하게 올라가면 일반화 성공/벌어지면 과적합 발생

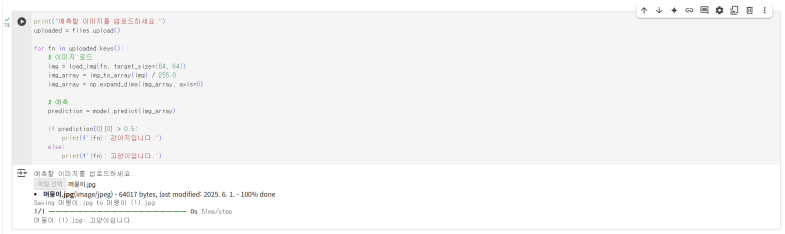
```
[7]  plt.plot(history.history['accuracy'], label='accuracy')
     plt.plot(history.history['val_accuracy'], label='val_accuracy')
     plt.xlabel('Epoch')
     plt.ylabel('Value')
     plt.title('Result Satus')
     plt.legend()
     plt.show()
```




8. 이미지 업로드 후 예측하기

- 예측 확률이 0.5 이상이면 강아지/이하면 고양이로 판단
- 적절 사진 넣어서 테스트 해보기!!





## 첨언

검증 정확도는 꾸준히 증가하여 최종적으로 76%까지 증가하였다. 하지만 에폭 15에서 과적합이 약간 보였다. 또한 여전히 강아지 사진을 고양이로 인식하는 모습을 볼 수 있다.

# 2차 시도 - batch_size 변경(5→15회)

## 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

```
train_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    "/content/data",
    target_size=(64, 64),
    batch_size=15,
    class_mode='binary'
)
```

```
Found 8005 images belonging to 2 classes.
```

## 6. 모델 학습시키기

- epochs: 전체 학습 반복 수 정의

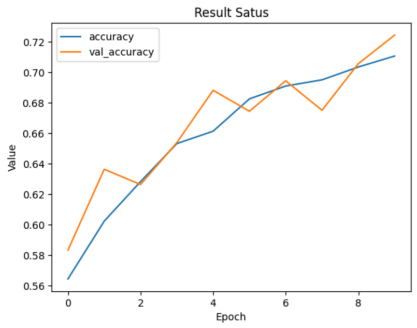
```
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```

```
/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 incl
  self._warn_if_super_not_called()
Epoch 1/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 78s 59ms/step - accuracy: 0.5479 - loss: 0.6868 - val_accuracy: 0.5834 - val_loss: 0.6643
Epoch 2/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 71s 55ms/step - accuracy: 0.5990 - loss: 0.6732 - val_accuracy: 0.6365 - val_loss: 0.6543
Epoch 3/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 71s 56ms/step - accuracy: 0.6136 - loss: 0.6480 - val_accuracy: 0.6265 - val_loss: 0.6528
Epoch 4/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 70s 55ms/step - accuracy: 0.6510 - loss: 0.6258 - val_accuracy: 0.6540 - val_loss: 0.6187
Epoch 5/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 74s 58ms/step - accuracy: 0.6666 - loss: 0.6199 - val_accuracy: 0.6883 - val_loss: 0.5795
Epoch 6/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 69s 54ms/step - accuracy: 0.6863 - loss: 0.5910 - val_accuracy: 0.6746 - val_loss: 0.5839
Epoch 7/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 73s 57ms/step - accuracy: 0.6845 - loss: 0.5862 - val_accuracy: 0.6946 - val_loss: 0.5705
Epoch 8/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 83s 58ms/step - accuracy: 0.6937 - loss: 0.5764 - val_accuracy: 0.6752 - val_loss: 0.5791
Epoch 9/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 71s 55ms/step - accuracy: 0.7009 - loss: 0.5670 - val_accuracy: 0.7058 - val_loss: 0.5504
Epoch 10/10
1281/1281 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - accuracy: 0.7111 - loss: 0.5581 - val_accuracy: 0.7245 - val_loss: 0.5360
```

## 7. 학습 결과 시각화

- plt.plot: 학습 정확도 그래프 출력
- 두 그래프가 비슷하게 올라가면 일반화 성공/벌어지면 과적합 발생

```
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 Satus')
plt.legend()
plt.show()
```

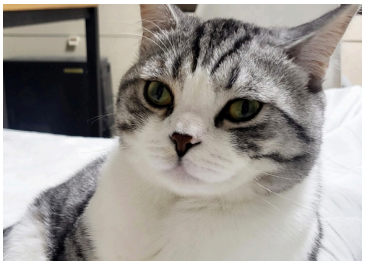


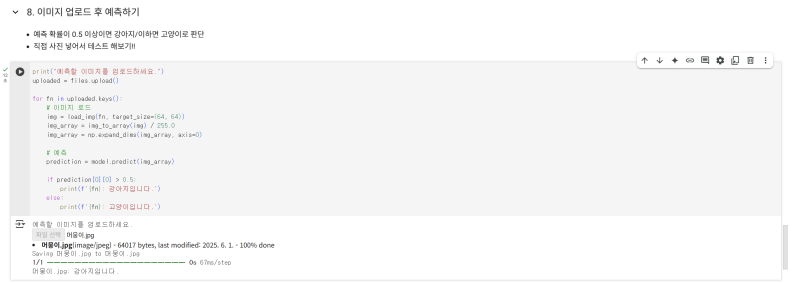

## 8. 이미지 업로드 후 예측하기

- 예측 확률이 0.5 이상이면 강아지/이하면 고양이로 판단
- 직접 사진 넣어서 테스트 해보기!!

```
print("예측할 이미지를 업로드하세요.")
uploaded = files.upload()

for fn in uploaded.keys():
    # 이미지 로드
    img = load_img(fn, target_size=(64, 64))
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    # 예측
    prediction = model.predict(img_array)

    if prediction[0][0] > 0.5:
        print(f'{fn} : 강아지입니다.')
    else:
        print(f'{fn} : 고양이입니다.')
```

```
예측할 이미지를 업로드하세요.
Choose Files 집냥이.jpg
  집냥이.jpg(image/jpeg) - 175375 bytes, last modified: 2025. 5. 27. - 100% done
Saving 집냥이.jpg to 집냥이.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step
집냥이.jpg : 강아지입니다.
```






## 첨언

이번엔 강아지는 제대로 나왔으나 오히려 고양이가 강아지로... 나온 모습을 볼 수 있다.. 어째서지.. 물론 검증 정확도도 73%를 넘기지 못한 모습을 볼 수 있다. epochs와 batch_size를 모두 변경하면 둘 다 잘 나오지 않을까? 하는 생각을 해본다.

# 느낀 점

모델을 바꿔서도 시도해보고 싶었지만 너무 많은 것들을 변경해야 할 것 같아서 건들지 못했다. 그래서 현 모델에서 할 수 있는 것들을 이용해 정확도를 높여보았다. 정확도를 높이기 위해 필요한 요소들을 찾아보며 나오는 다양한 개념도 간단하게 보았다. 단순히 tensorflow 학습도 조절로 찾아보니 단순 이용 방법만 많이 나와서 과제 파일 속 개념들을 이용하여 몇 가지를 변경해보는 식으로 실습을 진행해보았다. 먼저 에포크를 조절해보았는데 같은 데이터만 너무 많이 학습하면 해당 데이터셋에만 모델이 맞춰져 오히려 검증 정확도가 내려가는 것이다. 때문에 여러가지 요소를 고려하여 적절히 값을 줘야 했고 과적합으로 인해 검증 정확도가 계속 하락할 때의 상황도 따로 추가하여 학습 방법을 예외상황일 때 바꿔보는 식으로 하면 좋았을 것 같다.

### 참고

과적합 또는 과대적합: 기계 학습에서 학습 데이터를 과하게 학습하는 것. 일반적으로 학습 데이터는 실제 데이터의 부분 집합이므로 학습 데이터에 대해서는 오차가 감소하지만 실제 데이터에 대해서는 오차가 증가하게 된다.

https://velog.io/@richeberry/Tensorflowpython-%EC%8B%A0%EA%B2%BD%EB%A7%9D-%EB%AA%A8%EB%8D%B8-%EC%A1%B0%EC%A0%88%ED%95%98%EA%B8%B0