[멋사 13기][AI_나한진].

이미지 분류기 정확도 올리기 위해(ai와 친해지기)

사용기술:

TensorFlow: 구글에서 만든 오픈소스 머신 러닝 라이브러리로 신경망 분야의 딥러닝 모델을 만들고 학습시키는데 주로 사용됨. 행렬 연산 같은 수학적 연산을 효율적으로 처리하며, GPU를 활용한 빠른 계산이 가능한 자동차 엔진 같은 존재.
이번 실습에선 CNN 모델을 만들고 학습시키는 핵심 엔진이다.

Keras: 운전대, 자동차 대시보드 같은 존재이며 TensorFlow 위에서 동작하는 High-level API로 딥러닝 모델을 쉽게 설계하고 학습시킬 수 있게 해주며 복잡한 코드를 간략화 해주는 도구
실습에선 CNN(합성곱 신경망) 모델의 구조를 층 단위로 쌓고, 학습 설정을 간단히 정의하는데 사용함

CNN(Convolutional Neural Network, 합성곱 신경망): 이미지를 보고 구분하는 똑똑한 눈 같은 존재이며 이미지, 비디오 데이터 처리 특화 딥러닝 모델이다. 특징을 자동 학습하여 분류하며, 주요 구성 요소로는 합성곱 층(Conv2D), 풀링 층(MaxPooling), 완결 연결 층(Dense)이다.

활용기술:

이미지 데이터 전처리(요리하는 과정): 모델이 이해 가능한 형태로 바꾸는 것이며 이미지 크기 조정, 픽셀값을 0~1로 정규화, 데이터 증강으로 다양성 늘리는 작업을 포함

모델학습: CNN 모델에 데이터를 넣어 학습시키는 과정으로 이미지를 모델에게 패턴을 학습시켜 분류할수 있게만들고 손실함수와 옵티마이저를 조정하여 점점 똑똑하게 만듦.

이번 실습에서는 에포크를 통해 반복 학습으로 분류 정확도를 개선할 수 있으며 학습데이터를 넣고 정답을 비교하여 오답률을 줄인다.

힌트

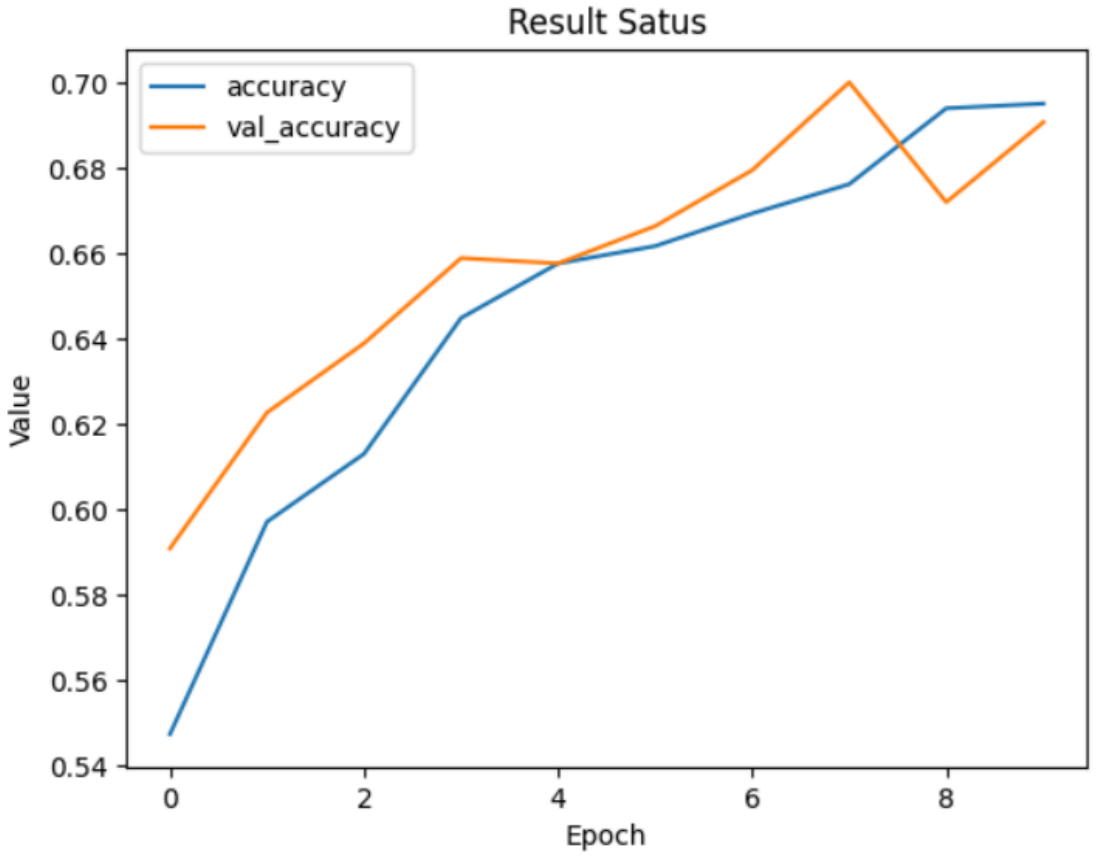
1. 모델 변경(MobileNet, VGG 등)
2. 폭(epoch) 변경
3. `EarlyStopping`, `ModelCheckpoint` 등을 이용해 학습 효율 향상 가능

를 사용하기로 했다. 일단

7. 학습 결과 시각화

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

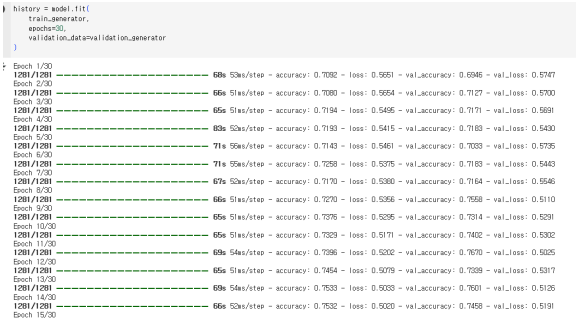
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Value')
plt.title('Result Satus')
plt.legend()
plt.show()
```

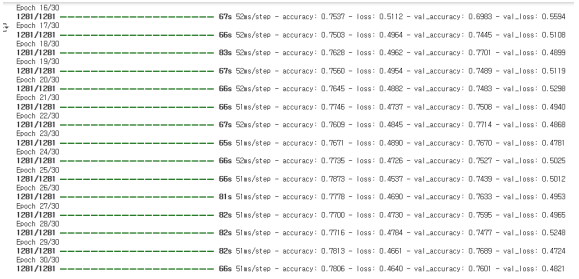


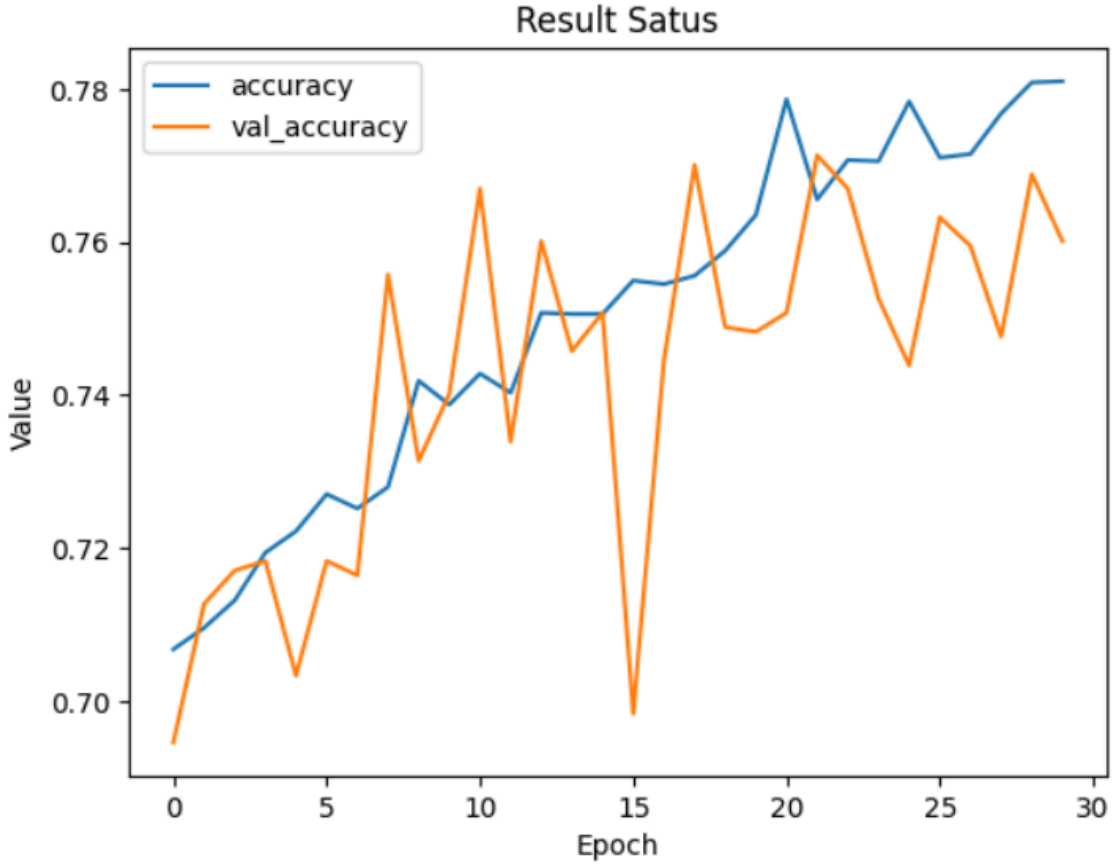
기본값으로 모델을 학습시키고 학습 과정을 시각화 하였을 때 해당 그래프처럼 accuracy와 val_accuary가 0.70까지 꾸준히 상승하다가 정체되며 과적합 정도는 크지 않게 나왔다. 다만 우리의 목표는 이미지 분류기의 정확도를 개선시키는 것이므로 우선적으로 에폭(Epoch)를 3배 늘린 30으로 설정하여

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

데이터를 통한 반복 학습을 강화하여 분류 정확도를 늘리고자 하였다.







에폭을 3배로 증가 시킨 결과 과적합이 에폭 15 부분에서 크게 발생하였지만 accuracy(학습 정확도)와 val_accuracy(검증 정확도)가 0.78 까지 상승함을 확인할 수 있다.

from tensorflow.keras.callbacks import EarlyStopping

#Early Stopping 설정

early_stopping = EarlyStopping(
monitor='val_loss', # 검증 손실을 모니터링
patience=5, # 5 에포크 동안 개선이 없으면 멈춤
restore_best_weights=True # 최적의 가중치로 복원
)

모델 학습 (Early Stopping 콜백 추가)

history = model.fit(
train_generator,
steps_per_epoch=100,
epochs=30,
validation_data=validation_generator,
validation_steps=50,

)
callbacks=[early_stopping] # 여기에 Early Stopping 추가

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

///

from tensorflow.keras.callbacks import EarlyStopping

early stopping을 함수를 추가하여 학습효율을 개선하고자 했으며 데이터 증강 (rotation_range=20→30) 그리고 batch_size를 5→10으로 변경하여 학습량 증가를 통해 정확도 개선 시도

```
# 데이터 증강기 + 검증 세트 분리
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30, #더 강한 회전
    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')
])
#Early Stopping 설정
early_stopping = EarlyStopping(
monitor='val_loss',  # 검증 손실을 모니터링
patience=5,  # 5 에포크 동안 개선이 없으면 멈춤
restore_best_weights=True  # 최적의 가중치로 복원
)

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

```
#모델 학습 (Early Stopping 콜백 추가)
history = model.fit(
train_generator,
steps_per_epoch=100,
epochs=20,
validation_data=validation_generator,
validation_steps=50,
callbacks=[early_stopping]  # 여기에 Early Stopping 추가
)
```

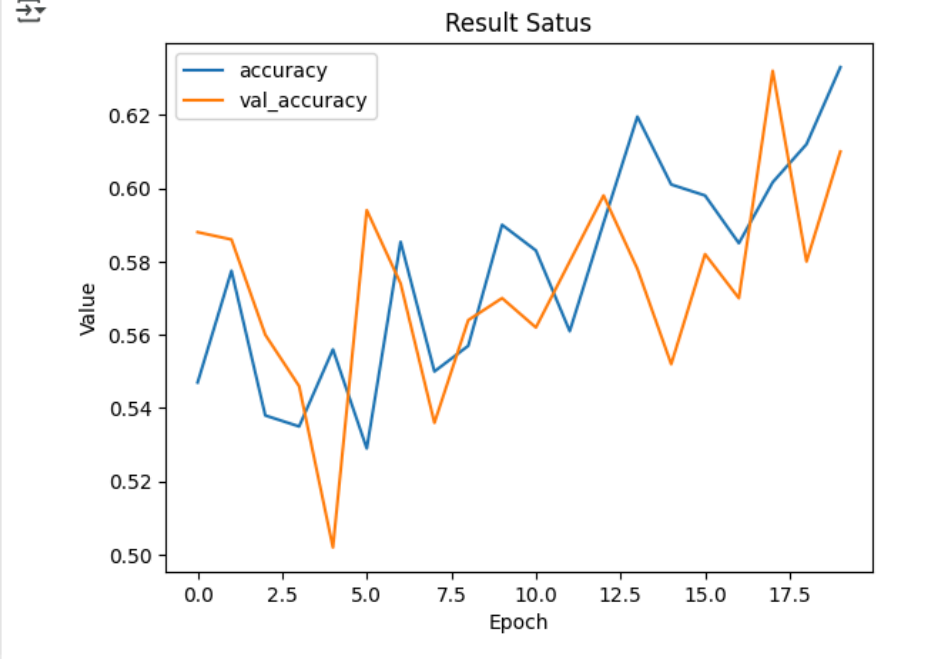
```python
#모델 학습 (Early Stopping 콜백 추가)
history = model.fit(
train_generator,
steps_per_epoch=100,
epochs=20,
validation_data=validation_generator,
validation_steps=50,
callbacks=[early_stopping]  # 여기에 Early Stopping 추가
)
```

```
Epoch 1/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.5416 - loss: 0.6893 - val_accuracy: 0.5880 - val_loss: 0.6886
Epoch 2/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.5746 - loss: 0.6820 - val_accuracy: 0.5860 - val_loss: 0.6801
Epoch 3/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 34ms/step - accuracy: 0.5523 - loss: 0.6842 - val_accuracy: 0.5600 - val_loss: 0.6839
Epoch 4/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.5467 - loss: 0.6845 - val_accuracy: 0.5460 - val_loss: 0.6850
Epoch 5/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 46ms/step - accuracy: 0.5800 - loss: 0.6840 - val_accuracy: 0.5020 - val_loss: 0.6897
Epoch 6/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - accuracy: 0.4876 - loss: 0.6934 - val_accuracy: 0.5940 - val_loss: 0.6746
Epoch 7/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - accuracy: 0.5890 - loss: 0.6837 - val_accuracy: 0.5740 - val_loss: 0.6822
Epoch 8/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 52ms/step - accuracy: 0.5375 - loss: 0.6846 - val_accuracy: 0.5360 - val_loss: 0.6920
Epoch 9/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 56ms/step - accuracy: 0.5615 - loss: 0.6800 - val_accuracy: 0.5640 - val_loss: 0.6867
Epoch 10/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - accuracy: 0.5955 - loss: 0.6763 - val_accuracy: 0.5700 - val_loss: 0.6663
Epoch 11/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - accuracy: 0.6184 - loss: 0.6632 - val_accuracy: 0.5620 - val_loss: 0.6900
Epoch 12/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 40ms/step - accuracy: 0.5598 - loss: 0.6802 - val_accuracy: 0.5800 - val_loss: 0.6755
Epoch 13/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 37ms/step - accuracy: 0.5837 - loss: 0.6755 - val_accuracy: 0.5980 - val_loss: 0.6463
Epoch 14/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - accuracy: 0.6322 - loss: 0.6559 - val_accuracy: 0.5780 - val_loss: 0.6710
Epoch 15/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 34ms/step - accuracy: 0.5757 - loss: 0.6771 - val_accuracy: 0.5520 - val_loss: 0.6844
Epoch 16/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 42ms/step - accuracy: 0.5875 - loss: 0.6826 - val_accuracy: 0.5820 - val_loss: 0.6834
Epoch 17/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - accuracy: 0.5619 - loss: 0.6785 - val_accuracy: 0.5700 - val_loss: 0.6864
Epoch 18/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - accuracy: 0.5736 - loss: 0.6722 - val_accuracy: 0.6320 - val_loss: 0.6386
Epoch 19/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 45ms/step - accuracy: 0.6053 - loss: 0.6560 - val_accuracy: 0.5800 - val_loss: 0.6822
Epoch 20/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - accuracy: 0.6495 - loss: 0.6589 - val_accuracy: 0.6100 - val_loss: 0.6641
```

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()
```



과적합 발생은 개선하였지만 정확도는 오히려 악화되었다.

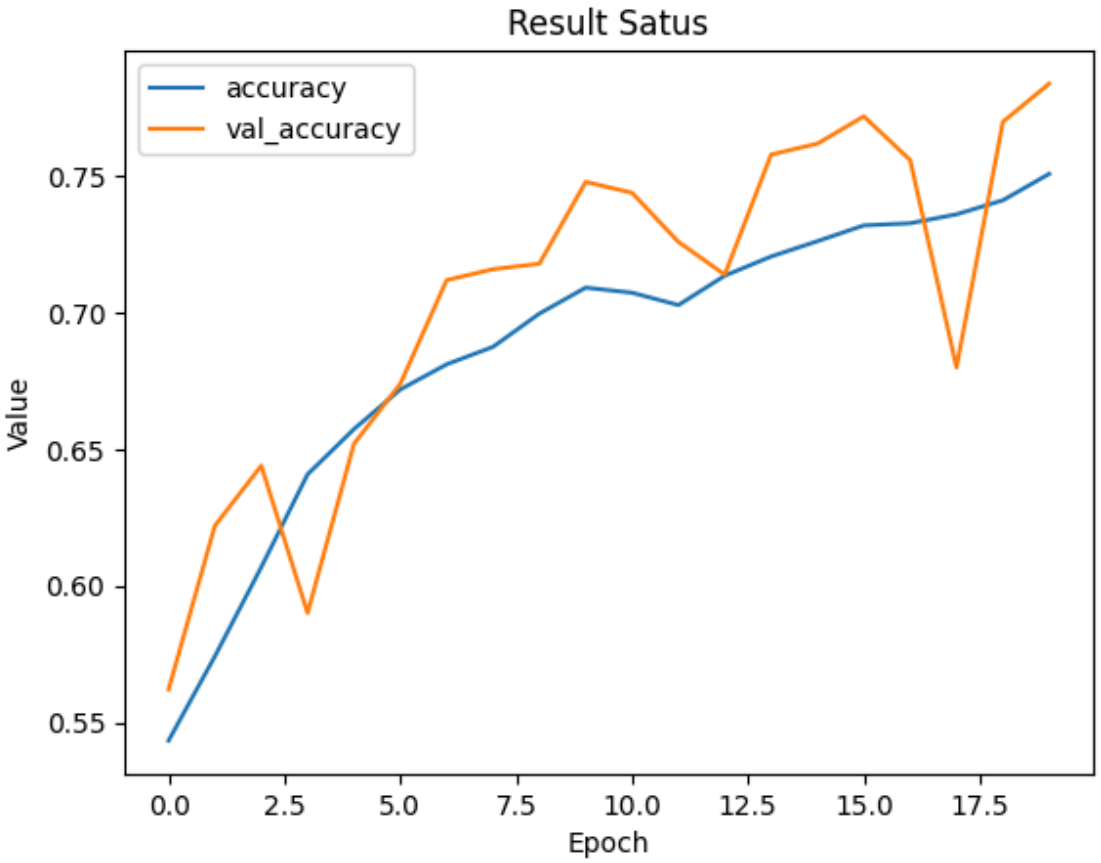
덕분에 내 patience가 한계에 도달했기 때문에 patience=5→10 으로 올려주어 정확도 개선 가능성 폭을 넓게 잡았다.

```
#Early Stopping 설정
early_stopping = EarlyStopping(
monitor='val_loss',  # 검증 손실을 모니터링
patience=10,  # 10 에포크 동안 개선이 없으면 멈춤
restore_best_weights=True  # 최적의 가중치로 복원
)
```

다시 모델을 학습 시킨 결과

```
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().
  self._warn_if_super_not_called()
641/641 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.5127 - loss: 0.7052 - val_accuracy: 0.5620 - val_loss: 0.6816
Epoch 2/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 20s 31ms/step - accuracy: 0.5578 - loss: 0.6847 - val_accuracy: 0.6220 - val_loss: 0.6511
Epoch 3/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 20s 31ms/step - accuracy: 0.6173 - loss: 0.6498 - val_accuracy: 0.6440 - val_loss: 0.6302
Epoch 4/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 18s 26ms/step - accuracy: 0.6386 - loss: 0.6275 - val_accuracy: 0.5900 - val_loss: 0.6447
Epoch 5/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 18s 27ms/step - accuracy: 0.6574 - loss: 0.6132 - val_accuracy: 0.6520 - val_loss: 0.6247
Epoch 6/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 21s 33ms/step - accuracy: 0.6688 - loss: 0.6055 - val_accuracy: 0.6740 - val_loss: 0.5942
Epoch 7/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 40s 32ms/step - accuracy: 0.6807 - loss: 0.5848 - val_accuracy: 0.7120 - val_loss: 0.5616
Epoch 8/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 17s 26ms/step - accuracy: 0.6845 - loss: 0.5873 - val_accuracy: 0.7160 - val_loss: 0.5743
Epoch 9/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 18s 28ms/step - accuracy: 0.6953 - loss: 0.5693 - val_accuracy: 0.7180 - val_loss: 0.5397
Epoch 10/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 21s 32ms/step - accuracy: 0.7071 - loss: 0.5632 - val_accuracy: 0.7480 - val_loss: 0.5295
Epoch 11/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 41s 32ms/step - accuracy: 0.7064 - loss: 0.5527 - val_accuracy: 0.7440 - val_loss: 0.5495
Epoch 12/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 17s 26ms/step - accuracy: 0.7052 - loss: 0.5552 - val_accuracy: 0.7260 - val_loss: 0.5296
Epoch 13/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 21s 26ms/step - accuracy: 0.7096 - loss: 0.5602 - val_accuracy: 0.7140 - val_loss: 0.5377
Epoch 14/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 20s 32ms/step - accuracy: 0.7164 - loss: 0.5379 - val_accuracy: 0.7580 - val_loss: 0.5542
Epoch 15/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 17s 26ms/step - accuracy: 0.7162 - loss: 0.5431 - val_accuracy: 0.7620 - val_loss: 0.5065
Epoch 16/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 41s 58ms/step - accuracy: 0.7454 - loss: 0.5161 - val_accuracy: 0.7720 - val_loss: 0.4820
Epoch 17/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 21s 32ms/step - accuracy: 0.7385 - loss: 0.5214 - val_accuracy: 0.7560 - val_loss: 0.5182
Epoch 18/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 38s 27ms/step - accuracy: 0.7407 - loss: 0.5255 - val_accuracy: 0.6800 - val_loss: 0.5690
Epoch 19/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 20s 27ms/step - accuracy: 0.7413 - loss: 0.5164 - val_accuracy: 0.7700 - val_loss: 0.5004
Epoch 20/20
641/641 ━━━━━━━━━━━━━━━━━━━━ 20s 32ms/step - accuracy: 0.7529 - loss: 0.5049 - val_accuracy: 0.7840 - val_loss: 0.4881
```

정확도가 0.5127에서 0.7529까지 꾸준히 우상향했으며 과적합도 크게 발생하지 않았음을 확인했다.



이미지 업로드 후 예측하기

강아지로 판단함

강아지로 판단함

느낀점: 코드로 이미지 분류기를 만들고 이후 코드 수정과 추가를 통해 정확도를 개선하는 과정을 통해 ai라는 친구에 대해 조금 알게 됬습니다. 아직은 어사이긴 하지만 앞으로 관련 공부를 통해 친해지는 시간을 가질 수 있기를 희망합니다.