



[멋사 13기][AI_이지원]

초기 모델 정확도

약 68% (0.68)

batch_size=5는 안전하지만 느림. 정확도 향상을 위해선 16~32로 늘리는 걸 추천

2. Batch size

- model.fit() 함수 안에 batch_size라는 인자가 추가된 것을 볼 수 있다.
- batch_size는 전체 데이터셋을 한 번에 학습시키자니, 데이터가 너무 크기 때문에 메모리 과부하와 속도 저하 문제가 발생하므로, 데이터를 Batch size만큼 쪼개서 학습을 시키는 것을 의미한다.
- 학습 단위에 대한 보다 자세한 설명은 다음 포스팅을 참고하기 바란다.
- Batch size는 전체 데이터셋을 Batch size로 나눴을 때, 나머지가 생기지 않은 크기로 만드는 것이 좋다.
- Batch size를 너무 크게 하면, 메모리 과부하가 발생하기 쉬우나, 더 많은 데이터를 보고 파라미터를 결정하므로 학습이 안정적으로 진행된다.
- Batch size를 너무 작게 하면, 자주 파라미터 갱신이 발생하므로 학습이 불안정해진다.
- Batch size를 얼마나 잡느냐에 대해선 정답이 없다고 할 수 있는데, 어떤 사람들은 자신의 머신이 가진 메모리를 넘지 않는 선에서 Batch size를 최대로 잡는 것이 좋다고 하고, 또 다른 사람은 32보다 큰 미니배치를 사용해선 절대 안된다고도 했다.
- 양쪽 다 주장이 꽤 탄탄하므로, 미니 배치를 크게도 해보고 작게도 해보며, 결과를 비교해보도록 하자.

2.1. Batch size의 효과

- Batch size는 학습에 걸리는 시간과 학습에 소모되는 메모리에 큰 영향을 미친다.
- 얼마나 차이가 나는지 확인하기 위해 Batch size를 설정하지 않은 학습과 설정하지 않은 학습을 비교해보도록 하자.

<https://gooopy.tistory.com/93>

target_size = 작을수록 연산 빠르지만, 정보 손실 커짐 (정확도 내려감)

epochs= 너무 적으면 학습 부족, 너무 많으면 과적합

빠른 학습을 위한 런타임 유형 변경

런타임 유형 변경

런타임 유형

Python 3

하드웨어 가속기 ?

☐ CPU ☒ T4 GPU ☐ A100 GPU ☐ L4 GPU
☐ v2-8 TPU ☐ v6e-1 TPU ☐ v5e-1 TPU

프리미엄 GPU를 이용하실까요? [추가 컴퓨팅 단위 구매](#)

취소

저장

1차 시도에서 감이 안와 AI를 이용해서 이것저것 넣어보고 코드를 돌려보았는데 정확도가 50을 넘지 못함

2차 시도 (정확도 73%)

문제점: 한번에 모르는 코드를 이것저것 다 넣어본 것이 문제였던 것 같다. 그래서 2차 시도에서는 코드는 너무 많이 수정하지않고 기존코드에서 숫자만 간단하게 수정하였다.

- batch_size=10 (전처리, 학습, 검증 데이터 다 수정)
- `tf.keras.layers.Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.01))` 추가
- epochs=20

```
# 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.reg
    ])

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

```

```

Epoch 12/20
641/641 ————— 21s 32ms/step - accuracy: 0.7176 - loss: 0.5475 - val_accuracy: 0.7171 - val_loss: 0.5580
Epoch 13/20
641/641 ————— 22s 35ms/step - accuracy: 0.7248 - loss: 0.5380 - val_accuracy: 0.7339 - val_loss: 0.5332
Epoch 14/20
641/641 ————— 21s 33ms/step - accuracy: 0.7369 - loss: 0.5289 - val_accuracy: 0.7308 - val_loss: 0.5285
Epoch 15/20
641/641 ————— 21s 32ms/step - accuracy: 0.7499 - loss: 0.5178 - val_accuracy: 0.7402 - val_loss: 0.5234
Epoch 16/20
641/641 ————— 42s 34ms/step - accuracy: 0.7332 - loss: 0.5268 - val_accuracy: 0.7458 - val_loss: 0.5453
Epoch 17/20
641/641 ————— 21s 33ms/step - accuracy: 0.7385 - loss: 0.5194 - val_accuracy: 0.7389 - val_loss: 0.5211
Epoch 18/20
641/641 ————— 20s 32ms/step - accuracy: 0.7411 - loss: 0.5221 - val_accuracy: 0.7458 - val_loss: 0.5187
Epoch 19/20
641/641 ————— 22s 34ms/step - accuracy: 0.7439 - loss: 0.5120 - val_accuracy: 0.7477 - val_loss: 0.5125
Epoch 20/20
641/641 ————— 21s 33ms/step - accuracy: 0.7425 - loss: 0.5163 - val_accuracy: 0.7514 - val_loss: 0.5087

```

1. batch_size=10 → 자주 업데이트

- 작은 배치는 gradient update를 더 자주 하므로 초기 학습 속도가 빠름

2. kernel_regularizer=l2(0.01) → 과적합 억제

- Dense 층의 가중치가 너무 커지는 걸 방지
- validation 데이터에서도 좋은 성능 유지 가능

3. epochs=20 → 충분한 학습 기회

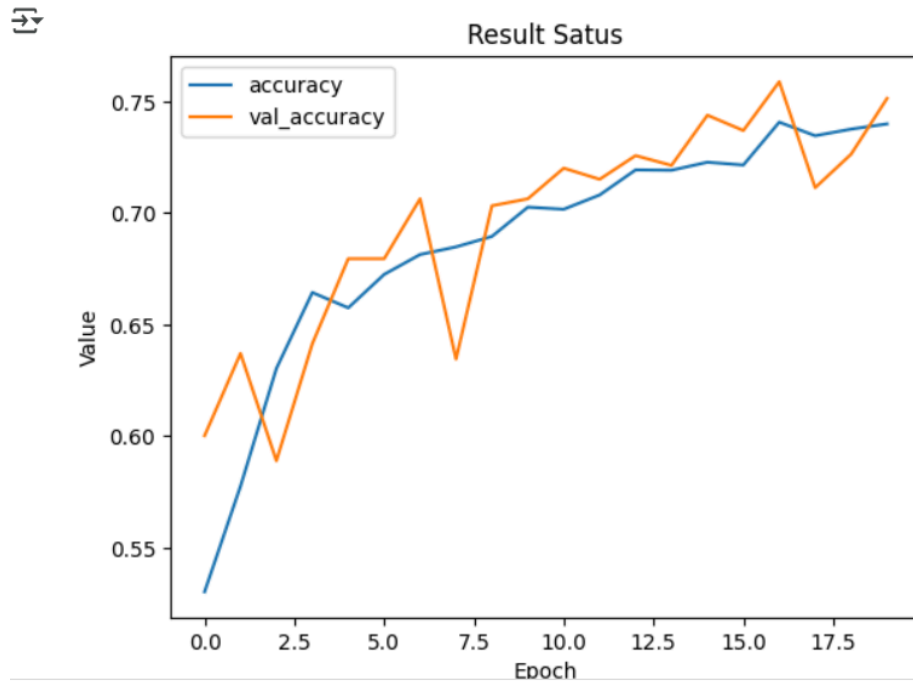
- 에폭을 늘리면 모델이 더 많은 패턴을 배울 수 있음
- 조기 과적합이 없다면 아주 효과적

```

.. Epoch 8/20
641/641 ————— 19s 30ms/step - accuracy: 0.6791 - loss
Epoch 9/20
641/641 ————— 22s 33ms/step - accuracy: 0.6830 - loss
Epoch 10/20
641/641 ————— 20s 32ms/step - accuracy: 0.7175 - loss
Epoch 11/20
639/641 ————— 0s 25ms/step - accuracy: 0.6951 - loss:
641/641 ————— 20s 30ms/step - accuracy: 0.6951 - loss
Epoch 12/20
641/641 ————— 20s 32ms/step - accuracy: 0.7079 - loss
Epoch 13/20
640/641 ————— 0s 25ms/step - accuracy: 0.7178 - loss:
641/641 ————— 20s 32ms/step - accuracy: 0.7178 - loss
Epoch 14/20
641/641 ————— 19s 30ms/step - accuracy: 0.7142 - loss
Epoch 15/20
640/641 ————— 0s 26ms/step - accuracy: 0.7123 - loss:
641/641 ————— 20s 32ms/step - accuracy: 0.7124 - loss
Epoch 16/20
641/641 ————— 20s 31ms/step - accuracy: 0.7290 - loss
Epoch 17/20
639/641 ————— 0s 25ms/step - accuracy: 0.7423 - loss:
641/641 ————— 20s 31ms/step - accuracy: 0.7423 - loss
Epoch 18/20
641/641 ————— 20s 32ms/step - accuracy: 0.7321 - loss
Epoch 19/20
641/641 ————— 21s 32ms/step - accuracy: 0.7284 - loss
Epoch 20/20
13/641 ————— 14s 23ms/step - accuracy: 0.7375 - loss

```

시각화 그래프



3차 시도(최종 정확도 약76%)

목표했던 73%를 넘었지만 75% 넘겨보고 싶어 마지막 시도를 해보았다

3차시도 dropout 삭제(과적합 방지 2번 중 1개 삭제)
batch_size 10 → 16으로 늘림

```
# 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=16,
    class_mode='binary',
    subset='training'
)

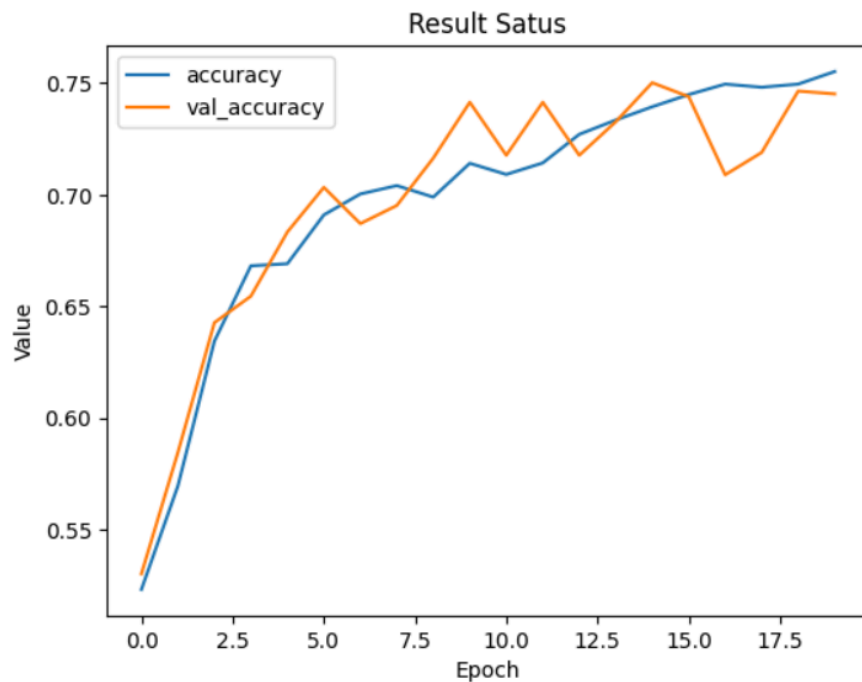
# 검증용 제너레이터
validation_generator = train_datagen.flow_from_directory(
    '/content/data/',
    target_size=(64, 64),
    batch_size=16,
    class_mode='binary',
    subset='validation'
)

# 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'), #은닉층(과적합 자주 발생)
    tf.keras.layers.Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.reg
])#출력층, 과적합 억제 추가

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

```

201/201	0s	80ms/step - accuracy: 0.7175 - loss:
201/201	20s	100ms/step - accuracy: 0.7175 - loss:
Epoch 11/20		
201/201	20s	102ms/step - accuracy: 0.7107 - loss:
Epoch 12/20		
201/201	19s	96ms/step - accuracy: 0.7160 - loss:
Epoch 13/20		
201/201	19s	96ms/step - accuracy: 0.7287 - loss:
Epoch 14/20		
201/201	19s	93ms/step - accuracy: 0.7357 - loss:
Epoch 15/20		
201/201	0s	78ms/step - accuracy: 0.7484 - loss:
201/201	20s	99ms/step - accuracy: 0.7483 - loss:
Epoch 16/20		
201/201	19s	94ms/step - accuracy: 0.7431 - loss:
Epoch 17/20		
201/201	20s	98ms/step - accuracy: 0.7419 - loss:
Epoch 18/20		
201/201	19s	97ms/step - accuracy: 0.7526 - loss:
Epoch 19/20		
201/201	20s	94ms/step - accuracy: 0.7466 - loss:
Epoch 20/20		
201/201	19s	97ms/step - accuracy: 0.7590 - loss:



과적합도 마지막엔 줄어든 모습이고 정확도도 마지막에는 75.9로 **약76%**정도 나온 모습이다.

느낀점

단순한 이미지 분류 학습이었지만 쉽지 않았다. 쉽게 사용하던 AI를 내가 직접 학습 시켜보니 실제 자주 쓰는 대화형 AI는 더욱 복잡한 과정을 거쳐 만들어졌을 것이라는 생각이 들었다. 시각화 그래프를 사용해서 눈으로 볼 수 있는 결과를 보니 실습이 재밌었다. 잘 접하지 않았던 분야지만 이런 실습을 통해 관심을 가지게 되어 좋았던 것 같다. 🙌