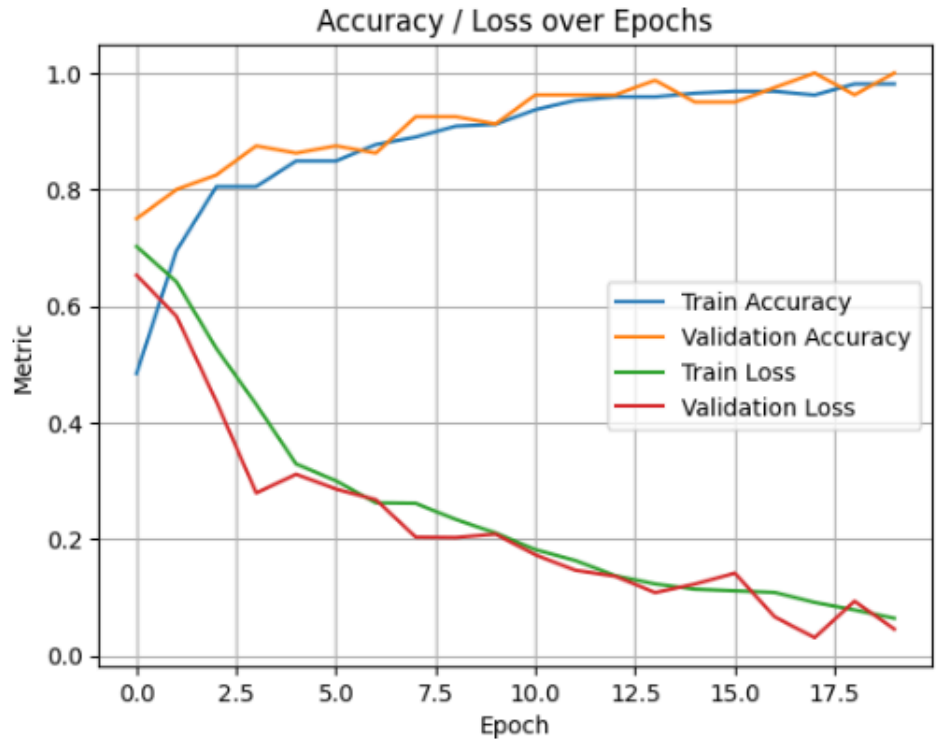
250725

```
import os
import random
import glob
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# 데이터 전처리
O_DIR = '/content/DATA/O'            # O 이미지
X_DIR = '/content/DATA/X'            # X 이미지
IMAGE_SIZE = (28, 28)               # 28×28 크기로 통일
EPOCHS = 20                         # 에폭(학습 반복 수)
BATCH_SIZE = 32                     # 미니배치 크기
CLASS_NAMES = ['O', 'X']            # 클래스 이름 (0 →'O', 1 →'X')

# 이미지 수집 및 라벨링
# 각 폴더에서 이미지 수집
o_paths = glob.glob(os.path.join(O_DIR, '*.png'))
x_paths = glob.glob(os.path.join(X_DIR, '*.png'))

# 이미지와 라벨을 튜플로 구성 (O: 0, X: 1)
image_label_pairs = [(path, 0) for path in o_paths] + [(path, 1) for path in x_pat

# 데이터 셔플 (순서 랜덤화)
random.shuffle(image_label_pairs)

# 이미지 경로와 라벨 분리
image_paths, labels = zip(*image_label_pairs)
image_paths = list(image_paths)
labels = list(labels)

# 이미지 전처리 함수
def load_and_preprocess_image(path):
```

```
    image = tf.io.read_file(path)                          # 이미지 파일 읽기
    image = tf.image.decode_png(image, channels=1)         # 디코딩 및 흑백 채널
    image = tf.image.resize(image, IMAGE_SIZE)             # 크기 통일
    image = tf.cast(image, tf.float32) / 255.0             # 0~1로 정규화
    return image

# Dataset 구성
# 이미지 경로들을 Dataset으로 변환
path_ds = tf.data.Dataset.from_tensor_slices(image_paths)
image_ds = path_ds.map(load_and_preprocess_image, num_parallel_calls=tf.d

# 라벨을 Dataset으로 변환
label_ds = tf.data.Dataset.from_tensor_slices(labels)

# 이미지와 라벨을 하나로 묶은 Dataset
full_ds = tf.data.Dataset.zip((image_ds, label_ds)).shuffle(1000)

# 훈련 및 검증 분할
train_size = int(0.8 * len(image_paths)) # 전체의 80%를 훈련에 사용
train_ds = full_ds.take(train_size).batch(BATCH_SIZE).prefetch(tf.data.AUTOTU
test_ds = full_ds.skip(train_size).batch(BATCH_SIZE).prefetch(tf.data.AUTOTU

# 모델 구성
model = tf.keras.Sequential([
    tf.keras.layers.Reshape((28, 28, 1), input_shape=(28, 28)), # 채널 차원 추가
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),     # 1번째 Conv 레이어
    tf.keras.layers.MaxPooling2D(2, 2),                        # 풀링
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),     # 2번째 Conv 레이어
    tf.keras.layers.MaxPooling2D(2, 2),                        # 풀링
    tf.keras.layers.Flatten(),                          # 1D로 펼치기
    tf.keras.layers.Dense(64, activation='relu'),       # 은닉층
    tf.keras.layers.Dense(2)                            # 출력층 (클래스 2개, 이진 분류)
])

# 컴파일
model.compile(optimizer='adam', # Adam 옵티마이저
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True
              metrics=['accuracy'])  # 정확도 지표 사용
```

```
# 학습 및 그래프 시각화 (accuracy, loss, val_accuracy, val_loss)
# 모델 학습 수행: 훈련 데이터로 지정한 횟수(에폭)만큼 학습, 검증 데이터도 함께 확인
history = model.fit(train_ds, epochs=EPOCHS, validation_data=test_ds)

# 학습 중 기록된 accuracy(정확도)/loss(손실) 그래프 시각화
plt.plot(history.history['accuracy'], label='Train Accuracy')        # 훈련 정확도
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')# 검증 정확
plt.plot(history.history['loss'], label='Train Loss')             # 훈련 손실
plt.plot(history.history['val_loss'], label='Validation Loss')    # 검증 손실

# 그래프 레이아웃 설정
plt.xlabel('Epoch')              # x축: 에폭 수
plt.ylabel('Metric')             # y축: 정확도 또는 손실
plt.legend()                     # 범례 표시
plt.title('Accuracy / Loss over Epochs')  # 그래프 제목
plt.grid(True)                   # 그리드 표시
plt.show()                       # 그래프 출력

# 평가 (테스트)
test_loss, test_acc = model.evaluate(test_ds)
print(f"\nTest accuracy: {test_acc:.4f}")
```

```
Epoch 1/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 87ms/step - accuracy: 0.4989 - loss: 0.7024 - val_accuracy: 0.7500 - val_loss: 0.6531
Epoch 2/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 320ms/step - accuracy: 0.6875 - loss: 0.6506 - val_accuracy: 0.8000 - val_loss: 0.5823
Epoch 3/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 104ms/step - accuracy: 0.7885 - loss: 0.5607 - val_accuracy: 0.8250 - val_loss: 0.4361
Epoch 4/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.7994 - loss: 0.4453 - val_accuracy: 0.8750 - val_loss: 0.2792
Epoch 5/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 180ms/step - accuracy: 0.8493 - loss: 0.3280 - val_accuracy: 0.8625 - val_loss: 0.3114
Epoch 6/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.8481 - loss: 0.3005 - val_accuracy: 0.8750 - val_loss: 0.2859
Epoch 7/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 94ms/step - accuracy: 0.8821 - loss: 0.2516 - val_accuracy: 0.8625 - val_loss: 0.2677
Epoch 8/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.8709 - loss: 0.2992 - val_accuracy: 0.9250 - val_loss: 0.2033
Epoch 9/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.9194 - loss: 0.2092 - val_accuracy: 0.9250 - val_loss: 0.2028
Epoch 10/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9033 - loss: 0.2114 - val_accuracy: 0.9125 - val_loss: 0.2085
Epoch 11/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 182ms/step - accuracy: 0.9360 - loss: 0.1808 - val_accuracy: 0.9625 - val_loss: 0.1730
Epoch 12/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9489 - loss: 0.1611 - val_accuracy: 0.9625 - val_loss: 0.1465
Epoch 13/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.9578 - loss: 0.1519 - val_accuracy: 0.9625 - val_loss: 0.1364
Epoch 14/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9580 - loss: 0.1123 - val_accuracy: 0.9875 - val_loss: 0.1083
Epoch 15/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9604 - loss: 0.1240 - val_accuracy: 0.9500 - val_loss: 0.1229
Epoch 16/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9813 - loss: 0.1071 - val_accuracy: 0.9500 - val_loss: 0.1416
Epoch 17/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.9752 - loss: 0.0950 - val_accuracy: 0.9750 - val_loss: 0.0665
Epoch 18/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9650 - loss: 0.0911 - val_accuracy: 1.0000 - val_loss: 0.0310
Epoch 19/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9796 - loss: 0.0781 - val_accuracy: 0.9625 - val_loss: 0.0934
Epoch 20/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9767 - loss: 0.0674 - val_accuracy: 1.0000 - val_loss: 0.0456
```




```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9898 - loss: 0.0816

Test accuracy: 0.9875
```

```
# 예측 테스트 (O 이미지)
# O 클래스 하나의 이미지 가져오기
sample_path = glob.glob('/content/DATA/O/*.png')[0]

# 해당 이미지 로드 및 전처리
sample_image = load_and_preprocess_image(sample_path)

# 배치 차원 추가 (모델 입력을 위해 4D로 만듦)
sample_batch = tf.expand_dims(sample_image, 0)

# 예측 수행 → 모델 출력은 logits 형태
logits = model.predict(sample_batch)

# logits 값을 확률로 변환 (softmax)
probs = tf.nn.softmax(logits)
```

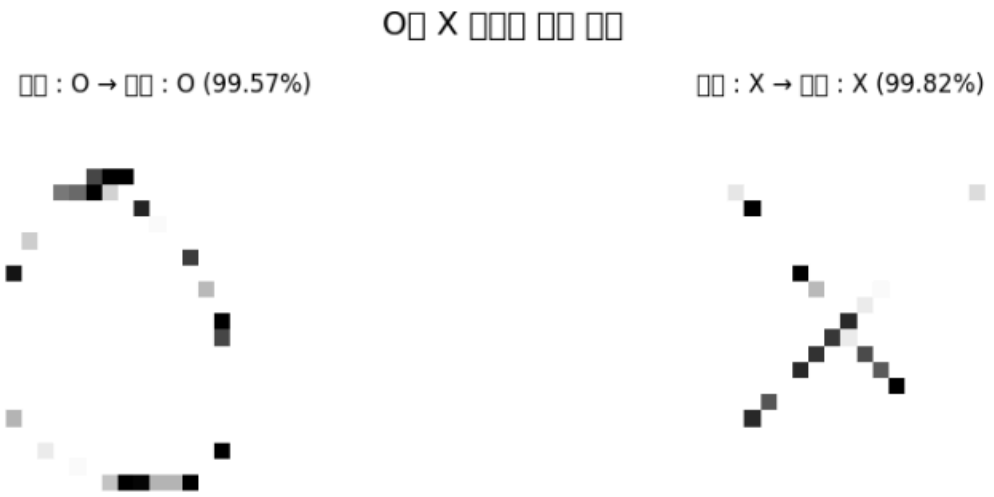
```
# 예측 결과 출력
print("softmax 확률 :", probs.numpy()[0])                    # 각 클래스에 대한 확률
print("예측되는 클래스 :", CLASS_NAMES[np.argmax(probs.numpy()[0])])# 클래스
```

```
# 예측 테스트 (X 이미지)
# X 클래스 하나의 이미지 가져오기
sample_path = glob.glob('/content/DATA/X/*.png')[0]

# 해당 이미지 로드 및 전처리
sample_image = load_and_preprocess_image(sample_path)

# 배치 차원 추가 (4D)
sample_batch = tf.expand_dims(sample_image, 0)

# 예측 수행
logits = model.predict(sample_batch)

# 확률 변환
probs = tf.nn.softmax(logits)

# 예측 결과 출력
print("softmax 확률 :", probs.numpy()[0])                    # 각 클래스에 대한 확률
print("예측되는 클래스 :", CLASS_NAMES[np.argmax(probs.numpy()[0])])# 클래스
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
softmax 확률 : [0.99569595 0.00430399]
예측되는 클래스 : O
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
softmax 확률 : [0.00178714 0.99821293]
예측되는 클래스 : X
```

```
# O와 X 이미지 하나씩 예측 및 시각화
sample_paths = {
    "O": glob.glob(os.path.join(O_DIR, '*.png'))[0],  # O 클래스 중 하나
    "X": glob.glob(os.path.join(X_DIR, '*.png'))[0]   # X 클래스 중 하나
}
```

```
plt.figure(figsize=(10, 4)) # 출력 크기 설정 (가로 10인치, 세로 4인치)

# O와 X 각각에 대해 예측 및 시각화 반복
for i, (label_name, path) in enumerate(sample_paths.items()):
    # 이미지 불러오기 및 전처리
    image = load_and_preprocess_image(path)

    # 모델 입력을 위해 4D 텐서로 변환 (batch 차원 추가)
    image_batch = tf.expand_dims(image, 0)

    # 예측 (logits 출력 → softmax로 변환)
    logits = model.predict(image_batch, verbose=0)
    probs = tf.nn.softmax(logits)

    # 예측 결과 : 가장 높은 확률 클래스 이름
    pred_index = np.argmax(probs.numpy()[0])
    pred_label = CLASS_NAMES[pred_index]

    # 해당 클래스의 확률값 (confidence)
    confidence = probs.numpy()[0][pred_index]

    # 이미지 시각화
    plt.subplot(1, 2, i + 1)  # 1행 2열 중 i+1번째 위치에 출력
    plt.imshow(tf.squeeze(image), cmap='gray')  # 흑백 이미지 출력
    plt.title(f"실제 : {label_name} → 예측 : {pred_label} ({confidence:.2%})")  # 여
    plt.axis('off')  # 축 제거

plt.suptitle("O와 X 이미지 예측 결과", fontsize=16) # 전체 그래프 제목 및 여백 설정
plt.tight_layout()  # 그래프 간격 자동 조정
plt.show()  # 전체 출력
```



LOOCV (Leave-One-Out Cross-Validation)

- 교차 검증 방식 → 모델이 각각의 테스트 샘플을 얼마나 잘 맞추는지 평균 측정
- 전체 데이터 중 1개만 테스트로 쓰고 나머지는 모두 모델 훈련에 사용
- 데이터 개수만큼 반복 (N개의 데이터면 N번 학습 및 평가)
- 작은 데이터셋에서도 일반화 성능을 정확히 평가하려고 사용
- N개의 데이터에 대해 학습을 N번 하므로 시간이 오래 걸림 (비효율적일 수 있음)