TEST1

```
import pandas as pd
from sklearn.model_selection import train_test_split

# CSV 파일에서 데이터 불러오기
data1 = pd.read_csv("C:/yelin/sp/합친파일/항치배/데이터/train_data_1.csv")

# 'DAY'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data1[numeric_feature]

# 'INGR_NM'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data1[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data1['RSLT_CONT'].map({'R': 1, 'S': 0})

# 학습 데이터와 테스트 데이터 분할
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train1.shape)
print("테스트 데이터:", X_test1.shape)
print("학습 레이블:", y_train1.shape)
print("테스트 레이블:", y_test1.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train1.to_csv('C:/yelin/sp/합친파일/항치배/데이터/1/X_train1.csv', index=False)
X_test1.to_csv('C:/yelin/sp/합친파일/항치배/데이터/1/X_test1.csv', index=False)
y_train1.to_csv('C:/yelin/sp/합친파일/항치배/데이터/1/y_train1.csv', index=False, header=['RSLT_CONT'])
y_test1.to_csv('C:/yelin/sp/합친파일/항치배/데이터/1/y_test1.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
import pandas as pd

# CSV 파일로부터 데이터 불러오기
X_train1 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/1/X_train1.csv')
X_test1 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/1/X_test1.csv')
y_train1 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/1/y_train1.csv')['RSLT_CONT']
y_test1 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/1/y_test1.csv')['RSLT_CONT']


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train1.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

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

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train1, y_train1, epochs=30, batch_size=32, validation_data=(X_test1, y_test1))

# 모델 평가
```

```
loss, accuracy = model.evaluate(X_test1, y_test1)

# 모델 예측
y_pred = model.predict(X_test1)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)

# AUROC 계산
roc_auc = roc_auc_score(y_test1, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test1, y_pred_binary)
precision = precision_score(y_test1, y_pred_binary, average='weighted')
recall = recall_score(y_test1, y_pred_binary, average='weighted')
f1 = f1_score(y_test1, y_pred_binary, average='weighted')

# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 1072

 dense_1 (Dense)             (None, 4)                 36

 dense_2 (Dense)             (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 54s 4ms/step - loss: 1.2538 - accuracy: 0.6946 - val_loss: 0.5281 - val_accuracy: 0.7368
Epoch 2/30
13104/13104 [==============================] - 58s 4ms/step - loss: 0.6317 - accuracy: 0.7178 - val_loss: 0.6094 - val_accuracy: 0.7278
Epoch 3/30
13104/13104 [==============================] - 60s 5ms/step - loss: 0.5782 - accuracy: 0.7218 - val_loss: 0.5469 - val_accuracy: 0.7283
Epoch 4/30
13104/13104 [==============================] - 60s 5ms/step - loss: 0.5622 - accuracy: 0.7159 - val_loss: 0.5508 - val_accuracy: 0.7356
Epoch 5/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.5504 - accuracy: 0.7237 - val_loss: 0.5541 - val_accuracy: 0.7286
Epoch 6/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5470 - accuracy: 0.7259 - val_loss: 0.5477 - val_accuracy: 0.7214
Epoch 7/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5434 - accuracy: 0.7281 - val_loss: 0.5355 - val_accuracy: 0.7291
Epoch 8/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5420 - accuracy: 0.7287 - val_loss: 0.5322 - val_accuracy: 0.7303
Epoch 9/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5395 - accuracy: 0.7304 - val_loss: 0.5373 - val_accuracy: 0.7285
Epoch 10/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5376 - accuracy: 0.7311 - val_loss: 0.5331 - val_accuracy: 0.7404
Epoch 11/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5387 - accuracy: 0.7308 - val_loss: 0.5334 - val_accuracy: 0.7325
Epoch 12/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5380 - accuracy: 0.7313 - val_loss: 0.5344 - val_accuracy: 0.7371
Epoch 13/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5358 - accuracy: 0.7323 - val_loss: 0.5349 - val_accuracy: 0.7329
Epoch 14/30
13104/13104 [==============================] - 61s 5ms/step - loss: 0.5353 - accuracy: 0.7324 - val_loss: 0.5408 - val_accuracy: 0.7250
Epoch 15/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.5360 - accuracy: 0.7324 - val_loss: 0.5321 - val_accuracy: 0.7365
Epoch 16/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.5350 - accuracy: 0.7330 - val_loss: 0.5320 - val_accuracy: 0.7303
Epoch 17/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.5333 - accuracy: 0.7339 - val_loss: 0.5299 - val_accuracy: 0.7319
Epoch 18/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5338 - accuracy: 0.7340 - val_loss: 0.5315 - val_accuracy: 0.7307
Epoch 19/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5331 - accuracy: 0.7344 - val_loss: 0.5287 - val_accuracy: 0.7400
Epoch 20/30
13104/13104 [==============================] - 61s 5ms/step - loss: 0.5313 - accuracy: 0.7346 - val_loss: 0.5281 - val_accuracy: 0.7342
Epoch 21/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.5318 - accuracy: 0.7351 - val_loss: 0.5345 - val_accuracy: 0.7401
Epoch 22/30
```

TEST2

```
import pandas as pd
from sklearn.model_selection import train_test_split

# CSV 파일에서 데이터 불러오기
data2 = pd.read_csv("C:/yelin/sp/합친파일/항치배/데이터/train_data_2.csv")

# 'DAY'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data2[numeric_feature]

# 'INGR_NM'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data2[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data2['RSLT_CONT'].map({'R': 1, 'S': 0})

# 학습 데이터와 테스트 데이터 분할
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train2.shape)
print("테스트 데이터:", X_test2.shape)
print("학습 레이블:", y_train2.shape)
print("테스트 레이블:", y_test2.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train2.to_csv('C:/yelin/sp/합친파일/항치배/데이터/2/X_train2.csv', index=False)
X_test2.to_csv('C:/yelin/sp/합친파일/항치배/데이터/2/X_test2.csv', index=False)
y_train2.to_csv('C:/yelin/sp/합친파일/항치배/데이터/2/y_train2.csv', index=False, header=['RSLT_CONT'])
y_test2.to_csv('C:/yelin/sp/합친파일/항치배/데이터/2/y_test2.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
import pandas as pd

# CSV 파일로부터 데이터 불러오기
X_train2 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/2/X_train2.csv')
X_test2 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/2/X_test2.csv')
y_train2 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/2/y_train2.csv')['RSLT_CONT']
y_test2 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/2/y_test2.csv')['RSLT_CONT']


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train2.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

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

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train2, y_train2, epochs=30, batch_size=32, validation_data=(X_test2, y_test2))

# 모델 평가
loss, accuracy = model.evaluate(X_test2, y_test2)
```

```
# 모델 예측
y_pred = model.predict(X_test2)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)

# AUROC 계산
roc_auc = roc_auc_score(y_test2, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test2, y_pred_binary)
precision = precision_score(y_test2, y_pred_binary, average='weighted')
recall = recall_score(y_test2, y_pred_binary, average='weighted')
f1 = f1_score(y_test2, y_pred_binary, average='weighted')

# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 8)                 1072

 dense_4 (Dense)             (None, 4)                 36

 dense_5 (Dense)             (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.9427 - accuracy: 0.6837 - val_loss: 0.9103 - val_accuracy: 0.6968
Epoch 2/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5742 - accuracy: 0.7168 - val_loss: 0.6322 - val_accuracy: 0.6225
Epoch 3/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5688 - accuracy: 0.7058 - val_loss: 0.5499 - val_accuracy: 0.7185
Epoch 4/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5704 - accuracy: 0.6940 - val_loss: 0.5534 - val_accuracy: 0.7267
Epoch 5/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5552 - accuracy: 0.7142 - val_loss: 0.5444 - val_accuracy: 0.7232
Epoch 6/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.5488 - accuracy: 0.7196 - val_loss: 0.5348 - val_accuracy: 0.7314
Epoch 7/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5473 - accuracy: 0.7185 - val_loss: 0.5854 - val_accuracy: 0.6601
Epoch 8/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.5431 - accuracy: 0.7246 - val_loss: 0.5328 - val_accuracy: 0.7353
Epoch 9/30
13104/13104 [==============================] - 47s 4ms/step - loss: 0.5424 - accuracy: 0.7239 - val_loss: 0.5313 - val_accuracy: 0.7347
Epoch 10/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5399 - accuracy: 0.7283 - val_loss: 0.5738 - val_accuracy: 0.6755
Epoch 11/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.5377 - accuracy: 0.7302 - val_loss: 0.5529 - val_accuracy: 0.7316
Epoch 12/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.5375 - accuracy: 0.7299 - val_loss: 0.5415 - val_accuracy: 0.7206
Epoch 13/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.5353 - accuracy: 0.7321 - val_loss: 0.5260 - val_accuracy: 0.7387
Epoch 14/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5343 - accuracy: 0.7321 - val_loss: 0.5292 - val_accuracy: 0.7374
Epoch 15/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5328 - accuracy: 0.7338 - val_loss: 0.5288 - val_accuracy: 0.7388
Epoch 16/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5341 - accuracy: 0.7329 - val_loss: 0.5775 - val_accuracy: 0.6723
Epoch 17/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5319 - accuracy: 0.7349 - val_loss: 0.5282 - val_accuracy: 0.7375
Epoch 18/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.5319 - accuracy: 0.7345 - val_loss: 0.5489 - val_accuracy: 0.7066
Epoch 19/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5305 - accuracy: 0.7360 - val_loss: 0.5427 - val_accuracy: 0.7126
Epoch 20/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.5306 - accuracy: 0.7354 - val_loss: 0.5240 - val_accuracy: 0.7391
Epoch 21/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5303 - accuracy: 0.7358 - val_loss: 0.5291 - val_accuracy: 0.7362
Epoch 22/30
```

TEST3

```
# CSV 파일에서 데이터 불러오기
data3 = pd.read_csv("C:/yelin/sp/합친파일/항치배/데이터/train_data_3.csv")

# 'DAY'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data3[numeric_feature]

# 'INGR_NM'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data3[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data3['RSLT_CONT'].map({'R': 1, 'S': 0})

# 학습 데이터와 테스트 데이터 분할
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train3.shape)
print("테스트 데이터:", X_test3.shape)
print("학습 레이블:", y_train3.shape)
print("테스트 레이블:", y_test3.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train3.to_csv('C:/yelin/sp/합친파일/항치배/데이터/3/X_train3.csv', index=False)
X_test3.to_csv('C:/yelin/sp/합친파일/항치배/데이터/3/X_test3.csv', index=False)
y_train3.to_csv('C:/yelin/sp/합친파일/항치배/데이터/3/y_train3.csv', index=False, header=['RSLT_CONT'])
y_test3.to_csv('C:/yelin/sp/합친파일/항치배/데이터/3/y_test3.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
# CSV 파일로부터 데이터 불러오기
X_train3 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/3/X_train3.csv')
X_test3 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/3/X_test3.csv')
y_train3 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/3/y_train3.csv')['RSLT_CONT']
y_test3 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/3/y_test3.csv')['RSLT_CONT']



import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train3.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

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

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train3, y_train3, epochs=30, batch_size=32, validation_data=(X_test3, y_test3))

# 모델 평가
loss, accuracy = model.evaluate(X_test3, y_test3)

# 모델 예측
y_pred = model.predict(X_test3)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)
```

```
# AUROC 계산
roc_auc = roc_auc_score(y_test3, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test3, y_pred_binary)
precision = precision_score(y_test3, y_pred_binary, average='weighted')
recall = recall_score(y_test3, y_pred_binary, average='weighted')
f1 = f1_score(y_test3, y_pred_binary, average='weighted')

# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 8)                 1072

 dense_7 (Dense)             (None, 4)                 36

 dense_8 (Dense)             (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 52s 4ms/step - loss: 1.4673 - accuracy: 0.6736 - val_loss: 0.6287 - val_accuracy: 0.6774
Epoch 2/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6291 - accuracy: 0.6774 - val_loss: 0.6287 - val_accuracy: 0.6773
Epoch 3/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.6287 - accuracy: 0.6774 - val_loss: 0.6284 - val_accuracy: 0.6775
Epoch 4/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.6260 - accuracy: 0.6795 - val_loss: 0.6093 - val_accuracy: 0.6889
Epoch 5/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.6096 - accuracy: 0.6954 - val_loss: 0.6236 - val_accuracy: 0.6818
Epoch 6/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5899 - accuracy: 0.7113 - val_loss: 0.5720 - val_accuracy: 0.7234
Epoch 7/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5776 - accuracy: 0.7190 - val_loss: 0.5772 - val_accuracy: 0.7179
Epoch 8/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5673 - accuracy: 0.7235 - val_loss: 0.5507 - val_accuracy: 0.7317
Epoch 9/30
13104/13104 [==============================] - 63s 5ms/step - loss: 0.5594 - accuracy: 0.7271 - val_loss: 0.5559 - val_accuracy: 0.7323
Epoch 10/30
13104/13104 [==============================] - 58s 4ms/step - loss: 0.5532 - accuracy: 0.7276 - val_loss: 0.5560 - val_accuracy: 0.7202
Epoch 11/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.5498 - accuracy: 0.7276 - val_loss: 0.5447 - val_accuracy: 0.7261
Epoch 12/30
13104/13104 [==============================] - 62s 5ms/step - loss: 0.5443 - accuracy: 0.7308 - val_loss: 0.5399 - val_accuracy: 0.7356
Epoch 13/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5442 - accuracy: 0.7300 - val_loss: 0.5364 - val_accuracy: 0.7333
Epoch 14/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5413 - accuracy: 0.7305 - val_loss: 0.5348 - val_accuracy: 0.7406
Epoch 15/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5395 - accuracy: 0.7318 - val_loss: 0.5413 - val_accuracy: 0.7299
Epoch 16/30
13104/13104 [==============================] - 60s 5ms/step - loss: 0.5393 - accuracy: 0.7316 - val_loss: 0.5293 - val_accuracy: 0.7383
Epoch 17/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.5377 - accuracy: 0.7329 - val_loss: 0.5390 - val_accuracy: 0.7279
Epoch 18/30
13104/13104 [==============================] - 55s 4ms/step - loss: 0.5380 - accuracy: 0.7323 - val_loss: 0.5387 - val_accuracy: 0.7288
Epoch 19/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5361 - accuracy: 0.7326 - val_loss: 0.5269 - val_accuracy: 0.7380
Epoch 20/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.5351 - accuracy: 0.7339 - val_loss: 0.5312 - val_accuracy: 0.7345
Epoch 21/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5355 - accuracy: 0.7341 - val_loss: 0.5405 - val_accuracy: 0.7238
Epoch 22/30
```

TEST4

```
# CSV 파일에서 데이터 불러오기
data4 = pd.read_csv("C:/yelin/sp/합친파일/항치배/데이터/train_data_4.csv")

# 'DAY'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data4[numeric_feature]
```

```
# 'INGR_NM'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data4[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data4['RSLT_CONT'].map({'R': 1, 'S': 0})

# 학습 데이터와 테스트 데이터 분할
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train4.shape)
print("테스트 데이터:", X_test4.shape)
print("학습 레이블:", y_train4.shape)
print("테스트 레이블:", y_test4.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train4.to_csv('C:/yelin/sp/합친파일/항치배/데이터/4/X_train4.csv', index=False)
X_test4.to_csv('C:/yelin/sp/합친파일/항치배/데이터/4/X_test4.csv', index=False)
y_train4.to_csv('C:/yelin/sp/합친파일/항치배/데이터/4/y_train4.csv', index=False, header=['RSLT_CONT'])
y_test4.to_csv('C:/yelin/sp/합친파일/항치배/데이터/4/y_test4.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
# CSV 파일로부터 데이터 불러오기
X_train4 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/4/X_train4.csv')
X_test4 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/4/X_test4.csv')
y_train4 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/4/y_train4.csv')['RSLT_CONT']
y_test4 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/4/y_test4.csv')['RSLT_CONT']


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train4.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

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

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train4, y_train4, epochs=30, batch_size=32, validation_data=(X_test4, y_test4))

# 모델 평가
loss, accuracy = model.evaluate(X_test4, y_test4)

# 모델 예측
y_pred = model.predict(X_test4)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)

# AUROC 계산
roc_auc = roc_auc_score(y_test4, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test4, y_pred_binary)
precision = precision_score(y_test4, y_pred_binary, average='weighted')
recall = recall_score(y_test4, y_pred_binary, average='weighted')
f1 = f1_score(y_test4, y_pred_binary, average='weighted')
```

```
# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 8)                 1072

 dense_10 (Dense)            (None, 4)                 36

 dense_11 (Dense)            (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6307 - accuracy: 0.6772 - val_loss: 0.6290 - val_accuracy: 0.6772
Epoch 2/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6289 - accuracy: 0.6772 - val_loss: 0.6289 - val_accuracy: 0.6772
Epoch 3/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.6289 - accuracy: 0.6772 - val_loss: 0.6289 - val_accuracy: 0.6772
Epoch 4/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6289 - accuracy: 0.6772 - val_loss: 0.6289 - val_accuracy: 0.6772
Epoch 5/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6289 - accuracy: 0.6772 - val_loss: 0.6289 - val_accuracy: 0.6772
Epoch 6/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6288 - accuracy: 0.6772 - val_loss: 0.6289 - val_accuracy: 0.6772
Epoch 7/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.6294 - accuracy: 0.6772 - val_loss: 0.6290 - val_accuracy: 0.6772
Epoch 8/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.6288 - accuracy: 0.6772 - val_loss: 0.6289 - val_accuracy: 0.6772
Epoch 9/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.6289 - accuracy: 0.6772 - val_loss: 0.6288 - val_accuracy: 0.6772
Epoch 10/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6298 - accuracy: 0.6772 - val_loss: 0.6288 - val_accuracy: 0.6772
Epoch 11/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6294 - accuracy: 0.6772 - val_loss: 0.6289 - val_accuracy: 0.6772
Epoch 12/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6289 - accuracy: 0.6772 - val_loss: 0.6290 - val_accuracy: 0.6772
Epoch 13/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.6288 - accuracy: 0.6772 - val_loss: 0.6289 - val_accuracy: 0.6772
Epoch 14/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6287 - accuracy: 0.6772 - val_loss: 0.6288 - val_accuracy: 0.6772
Epoch 15/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6289 - accuracy: 0.6772 - val_loss: 0.6288 - val_accuracy: 0.6772
Epoch 16/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6287 - accuracy: 0.6772 - val_loss: 0.6287 - val_accuracy: 0.6772
Epoch 17/30
13104/13104 [==============================] - 60s 5ms/step - loss: 0.6295 - accuracy: 0.6772 - val_loss: 0.6287 - val_accuracy: 0.6772
Epoch 18/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.6282 - accuracy: 0.6772 - val_loss: 0.6286 - val_accuracy: 0.6772
Epoch 19/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6245 - accuracy: 0.6772 - val_loss: 0.6260 - val_accuracy: 0.6772
Epoch 20/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.6140 - accuracy: 0.6772 - val_loss: 0.6023 - val_accuracy: 0.6772
Epoch 21/30
13104/13104 [==============================] - 51s 4ms/step - loss: 0.5970 - accuracy: 0.6772 - val_loss: 0.5871 - val_accuracy: 0.6772
Epoch 22/30
```

TEST5

```
# CSV 파일에서 데이터 불러오기
data5 = pd.read_csv("C:/yelin/sp/합친파일/항치배/데이터/train_data_5.csv")

# 'DAY'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data5[numeric_feature]

# 'INGR_NM'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data5[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data5['RSLT_CONT'].map({'R': 1, 'S': 0})
```

```
# 학습 데이터와 테스트 데이터 분할
X_train5, X_test5, y_train5, y_test5 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train5.shape)
print("테스트 데이터:", X_test5.shape)
print("학습 레이블:", y_train5.shape)
print("테스트 레이블:", y_test5.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train5.to_csv('C:/yelin/sp/합친파일/항치배/데이터/5/X_train5.csv', index=False)
X_test5.to_csv('C:/yelin/sp/합친파일/항치배/데이터/5/X_test5.csv', index=False)
y_train5.to_csv('C:/yelin/sp/합친파일/항치배/데이터/5/y_train5.csv', index=False, header=['RSLT_CONT'])
y_test5.to_csv('C:/yelin/sp/합친파일/항치배/데이터/5/y_test5.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
# CSV 파일로부터 데이터 불러오기
X_train5 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/5/X_train5.csv')
X_test5 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/5/X_test5.csv')
y_train5 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/5/y_train5.csv')['RSLT_CONT']
y_test5 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/5/y_test5.csv')['RSLT_CONT']


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train5.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

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

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train5, y_train5, epochs=30, batch_size=32, validation_data=(X_test5, y_test5))

# 모델 평가
loss, accuracy = model.evaluate(X_test5, y_test5)

# 모델 예측
y_pred = model.predict(X_test5)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)

# AUROC 계산
roc_auc = roc_auc_score(y_test5, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test5, y_pred_binary)
precision = precision_score(y_test5, y_pred_binary, average='weighted')
recall = recall_score(y_test5, y_pred_binary, average='weighted')
f1 = f1_score(y_test5, y_pred_binary, average='weighted')

# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# 모델 저장
```

```
model.save('saved_model.h5')
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 8)                 1072

 dense_19 (Dense)            (None, 4)                 36

 dense_20 (Dense)            (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 47s 4ms/step - loss: 0.7945 - accuracy: 0.6958 - val_loss: 0.6284 - val_accuracy: 0.7294
Epoch 2/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.6800 - accuracy: 0.7132 - val_loss: 0.7632 - val_accuracy: 0.7243
Epoch 3/30
13104/13104 [==============================] - 38s 3ms/step - loss: 0.6273 - accuracy: 0.7178 - val_loss: 0.5209 - val_accuracy: 0.7428
Epoch 4/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5961 - accuracy: 0.7221 - val_loss: 0.5394 - val_accuracy: 0.7330
Epoch 5/30
13104/13104 [==============================] - 43s 3ms/step - loss: 0.5623 - accuracy: 0.7227 - val_loss: 0.6322 - val_accuracy: 0.7208
Epoch 6/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5540 - accuracy: 0.7220 - val_loss: 0.5341 - val_accuracy: 0.7286
Epoch 7/30
13104/13104 [==============================] - 31s 2ms/step - loss: 0.5474 - accuracy: 0.7261 - val_loss: 0.5592 - val_accuracy: 0.7270
Epoch 8/30
13104/13104 [==============================] - 46s 3ms/step - loss: 0.5439 - accuracy: 0.7280 - val_loss: 0.5380 - val_accuracy: 0.7304
Epoch 9/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5402 - accuracy: 0.7309 - val_loss: 0.5357 - val_accuracy: 0.7387
Epoch 10/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5386 - accuracy: 0.7309 - val_loss: 0.5427 - val_accuracy: 0.7293
Epoch 11/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5368 - accuracy: 0.7315 - val_loss: 0.5451 - val_accuracy: 0.7235
Epoch 12/30
13104/13104 [==============================] - 50s 4ms/step - loss: 0.5351 - accuracy: 0.7328 - val_loss: 0.5300 - val_accuracy: 0.7300
Epoch 13/30
13104/13104 [==============================] - 49s 4ms/step - loss: 0.5344 - accuracy: 0.7333 - val_loss: 0.5243 - val_accuracy: 0.7415
Epoch 14/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5323 - accuracy: 0.7348 - val_loss: 0.5234 - val_accuracy: 0.7397
Epoch 15/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5321 - accuracy: 0.7346 - val_loss: 0.5387 - val_accuracy: 0.7359
Epoch 16/30
13104/13104 [==============================] - 43s 3ms/step - loss: 0.5319 - accuracy: 0.7348 - val_loss: 0.5266 - val_accuracy: 0.7368
Epoch 17/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5306 - accuracy: 0.7353 - val_loss: 0.5226 - val_accuracy: 0.7422
Epoch 18/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5295 - accuracy: 0.7357 - val_loss: 0.5290 - val_accuracy: 0.7337
Epoch 19/30
13104/13104 [==============================] - 48s 4ms/step - loss: 0.5290 - accuracy: 0.7365 - val_loss: 0.5317 - val_accuracy: 0.7338
Epoch 20/30
13104/13104 [==============================] - 47s 4ms/step - loss: 0.5289 - accuracy: 0.7364 - val_loss: 0.5242 - val_accuracy: 0.7410
Epoch 21/30
```

TEST6

```
# CSV 파일에서 데이터 불러오기
data6 = pd.read_csv("C:/yelin/sp/합친파일/항치배/데이터/train_data_6.csv")

# 'DAY'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data6[numeric_feature]

# 'INGR_NM'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data6[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data6['RSLT_CONT'].map({'R': 1, 'S': 0})

# 학습 데이터와 테스트 데이터 분할
X_train6, X_test6, y_train6, y_test6 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)
```

```
# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train6.shape)
print("테스트 데이터:", X_test6.shape)
print("학습 레이블:", y_train6.shape)
print("테스트 레이블:", y_test6.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train6.to_csv('C:/yelin/sp/합친파일/항치배/데이터/6/X_train6.csv', index=False)
X_test6.to_csv('C:/yelin/sp/합친파일/항치배/데이터/6/X_test6.csv', index=False)
y_train6.to_csv('C:/yelin/sp/합친파일/항치배/데이터/6/y_train6.csv', index=False, header=['RSLT_CONT'])
y_test6.to_csv('C:/yelin/sp/합친파일/항치배/데이터/6/y_test6.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
# CSV 파일로부터 데이터 불러오기
X_train6 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/6/X_train6.csv')
X_test6 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/6/X_test6.csv')
y_train6 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/6/y_train6.csv')['RSLT_CONT']
y_test6 = pd.read_csv('C:/yelin/sp/합친파일/항치배/데이터/6/y_test6.csv')['RSLT_CONT']


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train6.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

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

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train6, y_train6, epochs=30, batch_size=32, validation_data=(X_test6, y_test6))

# 모델 평가
loss, accuracy = model.evaluate(X_test6, y_test6)

# 모델 예측
y_pred = model.predict(X_test6)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)

# AUROC 계산
roc_auc = roc_auc_score(y_test6, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test6, y_pred_binary)
precision = precision_score(y_test6, y_pred_binary, average='weighted')
recall = recall_score(y_test6, y_pred_binary, average='weighted')
f1 = f1_score(y_test6, y_pred_binary, average='weighted')

# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 8)                 1072

 dense_16 (Dense)            (None, 4)                 36
```

```
 dense_17 (Dense)            (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 55s 4ms/step - loss: 0.7507 - accuracy: 0.6970 - val_loss: 1.0644 - val_accuracy: 0.7120
Epoch 2/30
13104/13104 [==============================] - 54s 4ms/step - loss: 0.6223 - accuracy: 0.7159 - val_loss: 0.6040 - val_accuracy: 0.6859
Epoch 3/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5751 - accuracy: 0.7136 - val_loss: 0.5289 - val_accuracy: 0.7402
Epoch 4/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5569 - accuracy: 0.7232 - val_loss: 0.5276 - val_accuracy: 0.7389
Epoch 5/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5506 - accuracy: 0.7266 - val_loss: 0.5529 - val_accuracy: 0.7123
Epoch 6/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5445 - accuracy: 0.7288 - val_loss: 0.5245 - val_accuracy: 0.7404
Epoch 7/30
13104/13104 [==============================] - 54s 4ms/step - loss: 0.5421 - accuracy: 0.7295 - val_loss: 0.5307 - val_accuracy: 0.7371
Epoch 8/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5416 - accuracy: 0.7293 - val_loss: 0.5488 - val_accuracy: 0.7249
Epoch 9/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5402 - accuracy: 0.7299 - val_loss: 0.5359 - val_accuracy: 0.7340
Epoch 10/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5374 - accuracy: 0.7314 - val_loss: 0.5409 - val_accuracy: 0.7336
Epoch 11/30
13104/13104 [==============================] - 54s 4ms/step - loss: 0.5359 - accuracy: 0.7323 - val_loss: 0.5570 - val_accuracy: 0.7340
Epoch 12/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5344 - accuracy: 0.7328 - val_loss: 0.5336 - val_accuracy: 0.7386
Epoch 13/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5340 - accuracy: 0.7333 - val_loss: 0.5287 - val_accuracy: 0.7339
Epoch 14/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5340 - accuracy: 0.7327 - val_loss: 0.5328 - val_accuracy: 0.7355
Epoch 15/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5337 - accuracy: 0.7333 - val_loss: 0.5297 - val_accuracy: 0.7362
Epoch 16/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5325 - accuracy: 0.7332 - val_loss: 0.5302 - val_accuracy: 0.7378
Epoch 17/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5324 - accuracy: 0.7336 - val_loss: 0.5288 - val_accuracy: 0.7375
Epoch 18/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5313 - accuracy: 0.7348 - val_loss: 0.5188 - val_accuracy: 0.7432
Epoch 19/30
13104/13104 [==============================] - 53s 4ms/step - loss: 0.5307 - accuracy: 0.7354 - val_loss: 0.5266 - val_accuracy: 0.7384
Epoch 20/30
13104/13104 [==============================] - 54s 4ms/step - loss: 0.5300 - accuracy: 0.7351 - val_loss: 0.5267 - val_accuracy: 0.7397
Epoch 21/30
13104/13104 [==============================] - 52s 4ms/step - loss: 0.5298 - accuracy: 0.7356 - val_loss: 0.5252 - val_accuracy: 0.7377
Epoch 22/30
```