```python
import numpy as np
import pandas as pd

# Generate synthetic data for demonstration
num_samples = 10000
num_features = 20

# Generate random features (assume each feature represents a pixel
intensity in an image)
features = np.random.rand(num_samples, num_features)

# Generate synthetic labels (0 or 1 representing not clicked and
clicked)
labels = np.random.randint(2, size=num_samples)

# Create a DataFrame to store the dataset
data = pd.DataFrame(features, columns=[f'feature_{i}' for i in
range(num_features)])
data['clicked'] = labels

# Save the dataset to a CSV file
data.to_csv('your_dataset.csv', index=False)

print("Dataset saved successfully.")
```

```
Dataset saved successfully.
```

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten,
Dense

# Load your dataset into a DataFrame (replace 'your_dataset.csv' with
your actual dataset)
data = pd.read_csv('your_dataset.csv')

# Preprocess the data
X = data.drop('clicked', axis=1).values  # Features
y = data['clicked'].values  # Labels

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```python
# Reshape the data for CNN input (assuming your features represent
sequential data)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Define the CNN model
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',
input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
250/250 [==============================] - 6s 7ms/step - loss: 0.6998
- accuracy: 0.4975 - val_loss: 0.7001 - val_accuracy: 0.4870
Epoch 2/10
250/250 [==============================] - 1s 6ms/step - loss: 0.6927
- accuracy: 0.5225 - val_loss: 0.6965 - val_accuracy: 0.5030
Epoch 3/10
250/250 [==============================] - 2s 8ms/step - loss: 0.6918
- accuracy: 0.5240 - val_loss: 0.6975 - val_accuracy: 0.5025
Epoch 4/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6894
- accuracy: 0.5341 - val_loss: 0.6975 - val_accuracy: 0.4760
Epoch 5/10
250/250 [==============================] - 1s 4ms/step - loss: 0.6867
- accuracy: 0.5512 - val_loss: 0.6972 - val_accuracy: 0.4915
Epoch 6/10
250/250 [==============================] - 1s 3ms/step - loss: 0.6846
- accuracy: 0.5556 - val_loss: 0.6977 - val_accuracy: 0.4825
Epoch 7/10
250/250 [==============================] - 1s 4ms/step - loss: 0.6815
- accuracy: 0.5589 - val_loss: 0.7000 - val_accuracy: 0.4885
Epoch 8/10
250/250 [==============================] - 1s 4ms/step - loss: 0.6778
```

```
- accuracy: 0.5739 - val_loss: 0.7094 - val_accuracy: 0.4850
Epoch 9/10
250/250 [==============================] - 1s 4ms/step - loss: 0.6744
- accuracy: 0.5739 - val_loss: 0.7068 - val_accuracy: 0.4890
Epoch 10/10
250/250 [==============================] - 1s 3ms/step - loss: 0.6692
- accuracy: 0.5915 - val_loss: 0.7118 - val_accuracy: 0.4905
63/63 [==============================] - 0s 2ms/step - loss: 0.7118 -
accuracy: 0.4905
Test Accuracy: 0.49050000309944153
```