```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from keras.layers import Input
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, RocCurveDisplay
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
import pickle
```

```
# Load your dataset (ensure the path is correct)
data = pd.read_csv('Dataset-ATS.csv')

# Display the first few rows to understand the structure of the dataset
print(data.head())

# Encode categorical variables using LabelEncoder
label_encoders = {}
categorical_columns = ['gender', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'Contract', 'Churn']

for col in categorical_columns:
    label_encoders[col] = LabelEncoder()
    data[col] = label_encoders[col].fit_transform(data[col])

# Define features (X) and target (y)
X = data.drop('Churn', axis=1)
y = data['Churn']

# Handle class imbalance using SMOTE (Synthetic Minority Oversampling Technique)
smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X, y)

# Split dataset into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize numerical features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
   gender  SeniorCitizen Dependents  tenure PhoneService MultipleLines  \
0  Female              0         No       1           No            No
1    Male              0         No      34          Yes            No
2    Male              0         No       2          Yes            No
3    Male              0         No      45           No            No
4  Female              0         No       2          Yes            No

  InternetService        Contract  MonthlyCharges Churn
0             DSL  Month-to-month           29.85    No
1             DSL        One year           56.95    No
2             DSL  Month-to-month           53.85   Yes
3             DSL        One year           42.30    No
4     Fiber optic  Month-to-month           70.70   Yes
```

```
# Build the ANN model
model = Sequential()

# Input Layer (using Input layer instead of input_dim)
model.add(Input(shape=(X_train.shape[1],)))  # This defines the input shape

# Hidden Layers
model.add(Dense(units=32, activation='relu'))
```

```
model.add(Dense(units=16, activation='relu'))

# Output Layer (binary classification: churn or not)
model.add(Dense(units=1, activation='sigmoid'))

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

# Display the model architecture
model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_22 (Dense)	(None, 32)	320
dense_23 (Dense)	(None, 16)	528
dense_24 (Dense)	(None, 1)	17

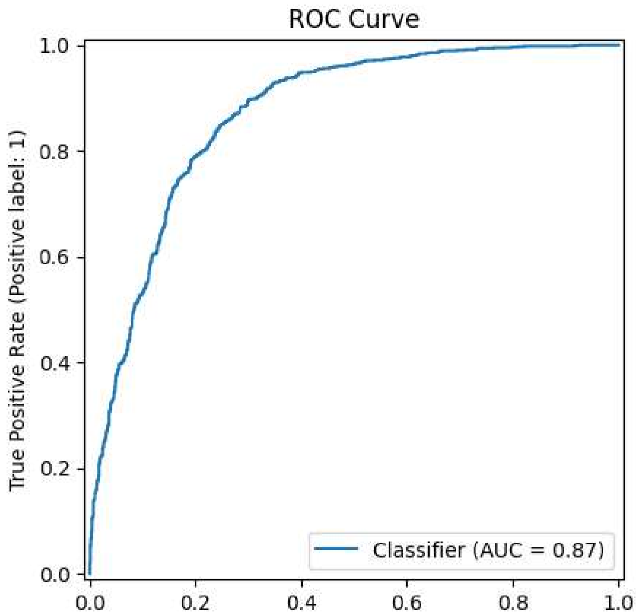
Total params: 865 (3.38 KB)
Trainable params: 865 (3.38 KB)
Non-trainable params: 0 (0.00 B)

```
# Train the model on the training data
history = model.fit(X_train, y_train, batch_size=32, epochs=50, validation_split=0.2)

# Save the trained model in the native Keras format
model.save('ann_model.keras')

# Save preprocessing objects (scaler and label encoders)
with open('preprocessing.pkl', 'wb') as f:
    pickle.dump({'scaler': scaler, 'encoders': label_encoders}, f)
```

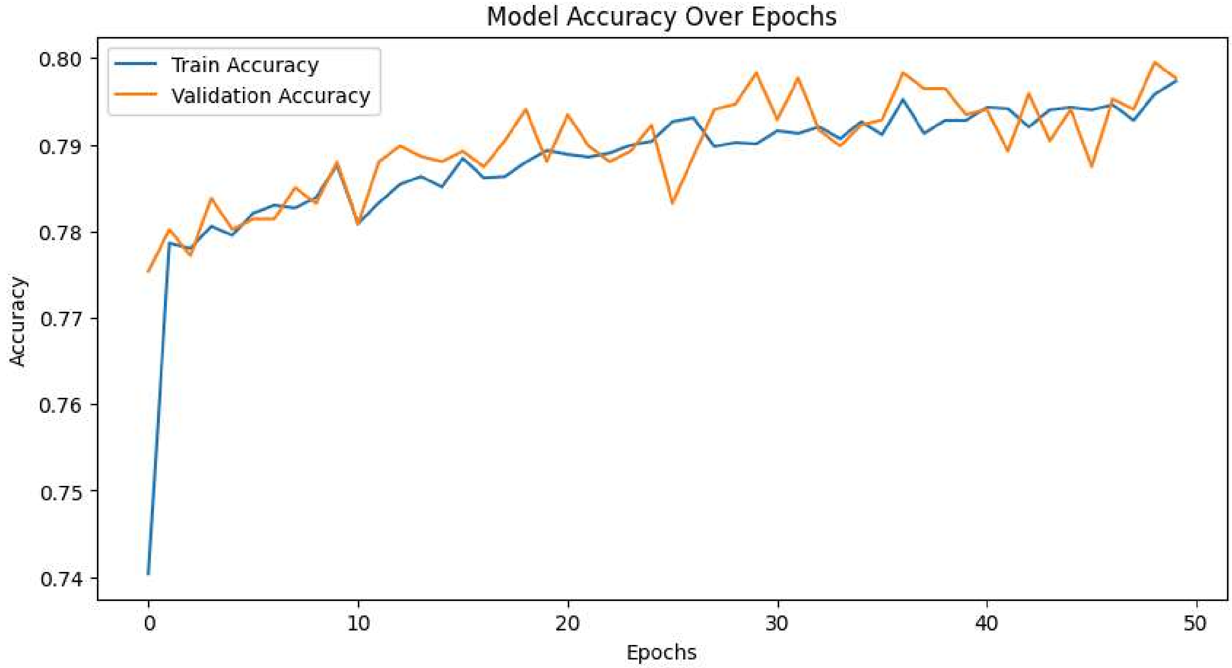
```
Epoch 40/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7920 - loss: 0.4470 - val_accuracy: 0.7935 - val_loss: 0.4397
Epoch 41/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8024 - loss: 0.4264 - val_accuracy: 0.7941 - val_loss: 0.4416
Epoch 42/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7968 - loss: 0.4451 - val_accuracy: 0.7893 - val_loss: 0.4462
Epoch 43/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7913 - loss: 0.4497 - val_accuracy: 0.7959 - val_loss: 0.4399
Epoch 44/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7920 - loss: 0.4465 - val_accuracy: 0.7905 - val_loss: 0.4374
Epoch 45/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8011 - loss: 0.4400 - val_accuracy: 0.7941 - val_loss: 0.4368
Epoch 46/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7937 - loss: 0.4442 - val_accuracy: 0.7874 - val_loss: 0.4392
Epoch 47/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7980 - loss: 0.4375 - val_accuracy: 0.7953 - val_loss: 0.4378
Epoch 48/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8000 - loss: 0.4348 - val_accuracy: 0.7941 - val_loss: 0.4383
Epoch 49/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7977 - loss: 0.4423 - val_accuracy: 0.7995 - val_loss: 0.4385
Epoch 50/50
207/207 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8051 - loss: 0.4376 - val_accuracy: 0.7977 - val_loss: 0.4417
```

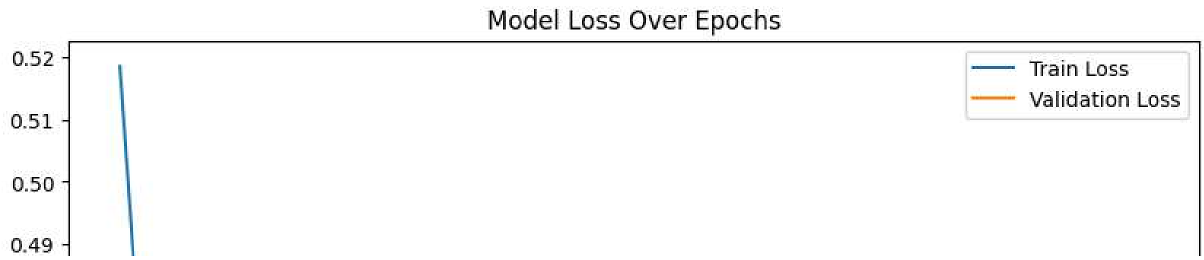
```python
# Predict churn probabilities for the test data
y_pred_prob = model.predict(X_test)

# Convert probabilities to binary predictions (0 or 1)
y_pred = (y_pred_prob > 0.5).astype("int32")

# Confusion Matrix and Classification Report
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Calculate AUC-ROC (Area Under Curve for Receiver Operating Characteristic)
auc_score = roc_auc_score(y_test, y_pred_prob)
print(f"AUC-ROC Score: {auc_score:.4f}")

# Plot the ROC Curve
RocCurveDisplay.from_predictions(y_test, y_pred_prob)
plt.title('ROC Curve')
plt.show()
```

```
65/65 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Confusion Matrix:
[[743 278]
 [139 910]]

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.73      0.78      1021
           1       0.77      0.87      0.81      1049

    accuracy                           0.80      2070
   macro avg       0.80      0.80      0.80      2070
weighted avg       0.80      0.80      0.80      2070

AUC-ROC Score: 0.8689
```




```
# Plot training and validation accuracy over epochs
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss over epochs
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```


Model Accuracy Over Epochs
Train Accuracy
Validation Accuracy
Accuracy
0.80
0.79
0.78
0.77
0.76
0.75
0.74
0
10
20
30
40
50
Epochs


Model Loss Over Epochs
Train Loss
Validation Loss
0.52
0.51
0.50
0.49