TensorFlow Customer Classification Model Guide

GitHub Link: *git@github.com:johnmicky1/TensorFlow-Binary-Customer-Segmentation.git*

Overview

This professional guide walks you through the setup, development, and execution of a TensorFlow-based Customer Classification Model. The model uses simulated behavioral data to predict whether a customer is *new* (sign-up) or *existing* (login) based on their activity patterns. It is ideal for e-commerce, marketing analytics, or any customer segmentation use case.

Prerequisites

Before you begin, make sure you have the following tools installed:

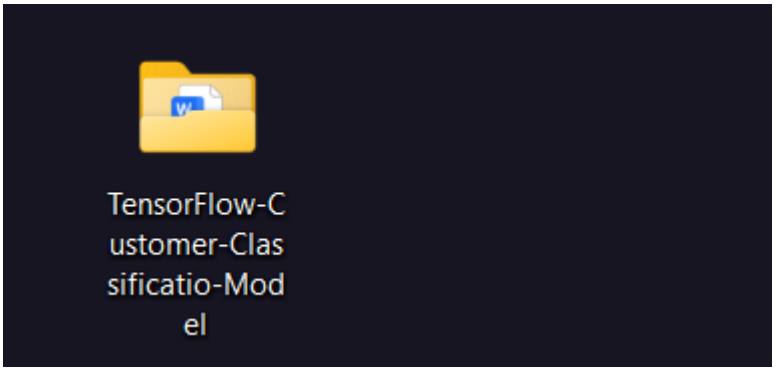
- Python 3.10+ (latest version recommended)
- TensorFlow library
- NumPy and Pandas for data handling
- Matplotlib for visualization
- Code editor or IDE (VS Code, PyCharm, Jupyter Notebook, etc.)

Install TensorFlow and NumPy using the commands below:

- pip install tensorflow
  pip install numpy

Step 1: Project Setup

1. Create a new folder or directory named `TensorFlow-Customer-Classification-Model`.



2. Open your preferred code editor (VS Code, Notepad++, etc.).
3. Create a new Python file and name it `tensorflow-customer-classification-model.py using the code below`. Then save it in the folder.

```
import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt # Import for plotting

# Set random seeds for reproducibility
tf.random.set_seed(42)
np.random.seed(42)

# --- 1. Data Simulation Function (Real-Time Scenario) ---
def simulate_customer_data(n_samples=2000):
    """
    Generates synthetic data for customer classification (New vs. Existing)
    Target (Y): 0 = New Customer (Signup), 1 = Existing Customer (Login)
    """
    # 1. Target (55% Existing, 45% New)
    y = np.random.binomial(1, 0.55, n_samples)
    df = pd.DataFrame({'is_existing': y})

    # 2. Time of Day (0-23) - New users favor peak hours (18-22)
    df['time_of_day'] = np.random.randint(0, 24, n_samples)

    # Using .loc with .values to avoid FutureWarnings and ensure consistent dtype
    new_user_indices = (df['is_existing'] == 0)
    time_of_day_new = df.loc[new_user_indices, 'time_of_day'].values + np.random.normal(0, 5,
new_user_indices.sum())
    df.loc[new_user_indices, 'time_of_day'] = np.clip(time_of_day_new, 0, 23).astype(int)

    # 3. Device Type (0=Mobile/Tablet, 1=Desktop)
    # Existing users slightly favor desktop
```

```
    df['device_type_desktop'] = np.random.binomial(1, 0.6, n_samples)
    df.loc[df['is_existing'] == 0, 'device_type_desktop'] = np.random.binomial(1, 0.45, (df['is_existing'] == 0).sum())

    # 4. Session Duration (Seconds) - New users spend slightly longer
    df['session_duration_sec'] = np.random.normal(loc=120, scale=40, size=n_samples)
    df.loc[df['is_existing'] == 0, 'session_duration_sec'] = np.random.normal(loc=150, scale=60,
size=(df['is_existing'] == 0).sum())
    df['session_duration_sec'] = np.clip(df['session_duration_sec'], 10, 600).astype(int)

    # 5. Referral Source (0=Direct/Search, 1=Social Media Campaign)
    # New users heavily influenced by social campaign
    df['referral_source_social'] = np.random.binomial(1, 0.2, n_samples)
    df.loc[df['is_existing'] == 0, 'referral_source_social'] = np.random.binomial(1, 0.7, (df['is_existing'] == 0).sum())

    # Final features and target
    X = df[['time_of_day', 'device_type_desktop', 'session_duration_sec', 'referral_source_social']]
    y = df['is_existing']

    return X, y

# Generate the data
X, y = simulate_customer_data()
print(f"Dataset generated: {X.shape[0]} samples")
print(X.head())

# --- 2. Preprocessing and Feature Engineering ---

# One-Hot Encode Time of Day (0-23 hours)
X_processed = pd.get_dummies(X, columns=['time_of_day'], prefix='hour')
```

```
# CRITICAL FIX: Ensure all columns are explicitly float32 before converting to NumPy
# This resolves the 'ValueError: Invalid dtype: object'
X_data = X_processed.astype(np.float32).values
y_data = y.values

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_data, y_data, test_size=0.2, random_state=42, stratify=y_data
)

# Scale Numerical Features (Session Duration is the only un-encoded numerical feature)
# Find the index of the 'session_duration_sec' column after one-hot encoding
duration_index = X_processed.columns.get_loc('session_duration_sec')

scaler = StandardScaler()
# Note: Reshape(-1, 1) is necessary for the scaler to work on a single column
X_train[:, duration_index] = scaler.fit_transform(X_train[:, duration_index].reshape(-1, 1)).flatten()
X_test[:, duration_index] = scaler.transform(X_test[:, duration_index].reshape(-1, 1)).flatten()

print(f"\nTraining set size: {X_train.shape[0]}")
print(f"Testing set size: {X_test.shape[0]}")
print(f"Number of features after encoding: {X_train.shape[1]}")

# --- 3. Build the TensorFlow Keras Model (Binary Classification) ---

INPUT_DIM = X_train.shape[1]

model = Sequential([
    # Input layer and first hidden layer
    Dense(64, activation='relu', input_shape=(INPUT_DIM,), name='dense_1'),
```

```
    Dropout(0.3, name='dropout_1'),

    # Second hidden layer
    Dense(32, activation='relu', name='dense_2'),
    Dropout(0.3, name='dropout_2'),

    # Output layer for Binary Classification
    # Uses 1 neuron and 'sigmoid' activation (output between 0 and 1)
    Dense(1, activation='sigmoid', name='output')
])

# Compile the model
model.compile(
    optimizer='adam',
    loss='binary_crossentropy', # Standard loss for binary classification
    metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]
)

print("\nModel Architecture Summary:")
model.summary()

# --- 4. Train the Model ---

# Use Early Stopping to prevent overfitting
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)
```

```
print("\nStarting model training...")

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_data=(X_test, y_test),
    callbacks=[early_stop],
    verbose=0 # Run silently for clean output
)

print(f"Training finished after {len(history.history['loss'])} epochs.")

# --- 5. Plot Training History Function ---
def plot_training_history(history):
    """Plots and saves the training and validation loss and accuracy."""
    print("\nGenerating training graphs...")

    # 1. Plot Loss
    plt.figure(figsize=(10, 5))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss Over Epochs')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()
    plt.grid(True)
    plt.savefig('training_loss.png')
    plt.close()
    print("-> Saved training_loss.png")
```

```
    # 2. Plot Accuracy
    plt.figure(figsize=(10, 5))
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy Over Epochs')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend()
    plt.grid(True)
    plt.savefig('training_accuracy.png')
    plt.close()
    print("-> Saved training_accuracy.png")


# --- 6. Evaluation and Real-Time Application ---

# Evaluate the model on the test set
loss, accuracy, precision, recall = model.evaluate(X_test, y_test, verbose=0)

print("\n--- Model Evaluation ---")
print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test Precision: {precision:.4f}")
print(f"Test Recall: {recall:.4f}")

# Generate and save the training history plots
plot_training_history(history)

# --- Real-Time Prediction Simulation ---
print("\n--- Real-Time Prediction Simulation ---")
```

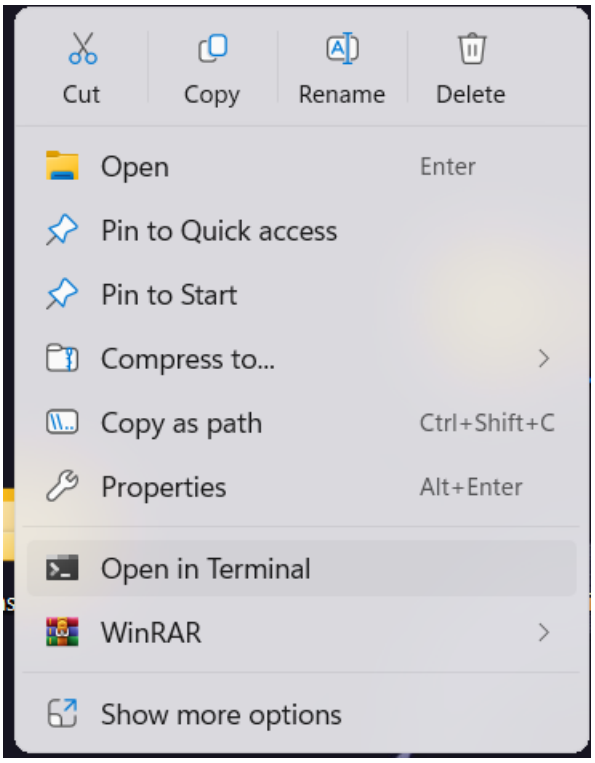
```
# Example 1: High chance of being an Existing Customer (Late time, Desktop, Short session)
sample_existing = pd.DataFrame({
    'time_of_day': [21], 'device_type_desktop': [1],
    'session_duration_sec': [60], 'referral_source_social': [0]
})

# Example 2: High chance of being a New Customer (Mid-day, Mobile, Long session, Social referral)
sample_new = pd.DataFrame({
    'time_of_day': [15], 'device_type_desktop': [0],
    'session_duration_sec': [250], 'referral_source_social': [1]
})

def predict_customer_type(sample_df, scaler, model, original_columns, duration_index):
    """Preprocesses a single sample and predicts customer type."""
    # Apply One-Hot Encoding consistent with training data
    sample_encoded = pd.get_dummies(sample_df, columns=['time_of_day'], prefix='hour')

    # Reindex to ensure all 24 hour columns exist, filling missing with 0
    missing_cols = set(original_columns) - set(sample_encoded.columns)
    for c in missing_cols:
        sample_encoded[c] = 0

    # Ensure column order matches training data
    sample_encoded = sample_encoded[original_columns]

    # Convert to float32 before passing to NumPy array
    sample_data = sample_encoded.astype(np.float32).values

    # Scale the session duration feature (at the correct index)
```

```
    sample_data[:, duration_index] = scaler.transform(sample_data[:, duration_index].reshape(-1, 1)).flatten()

    # Predict probability (0 to 1)
    prediction = model.predict(sample_data, verbose=0)[0][0]

    # Classify (0 = New, 1 = Existing)
    prediction_class = 1 if prediction > 0.5 else 0

    label = "Existing Customer (Login)" if prediction_class == 1 else "New Customer (Signup)"

    print(f"Features: TOD={sample_df['time_of_day'].iloc[0]},
Desktop={bool(sample_df['device_type_desktop'].iloc[0])},
Duration={sample_df['session_duration_sec'].iloc[0]}s,
Social={bool(sample_df['referral_source_social'].iloc[0])}")
    print(f"Predicted Probability of being Existing: {prediction:.4f}")
    print(f"Predicted Customer Type: {label}")
    print("-" * 50)

    return prediction_class

# Run predictions
print("Simulating prediction for a likely Existing Customer:")
predict_customer_type(sample_existing, scaler, model, X_processed.columns, duration_index)

print("Simulating prediction for a likely New Customer:")
predict_customer_type(sample_new, scaler, model, X_processed.columns, duration_index)

# --- 7. Model Saving (for E-commerce Integration) ---

model_save_path = 'customer_type_classifier.keras'
model.save(model_save_path)
```

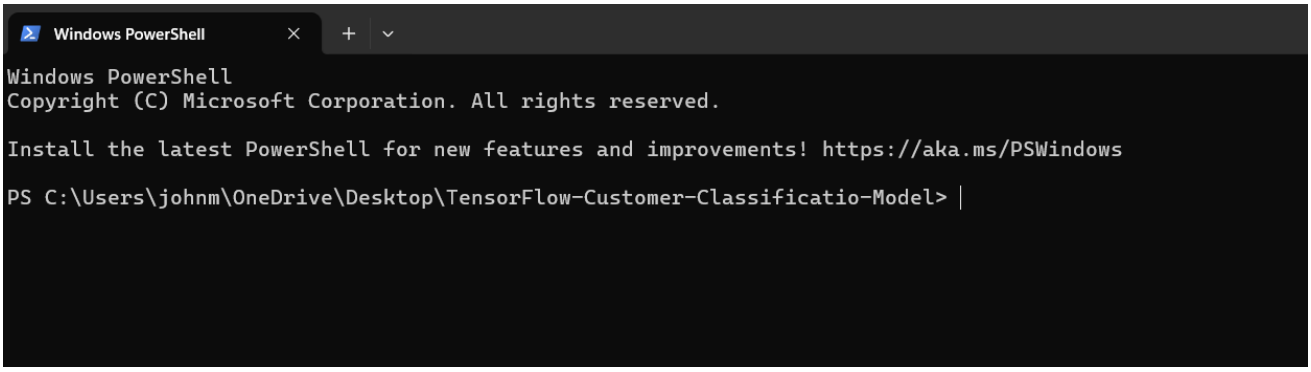
```
print(f"\nModel saved successfully for deployment: {model_save_path}")


print("\nTo integrate this model in a real-time environment, you would load the model, load the scaler, and pass the four input features to the 'predict_customer_type' function.")
```

▶ Step 2: Run the Code

1. **Right Click on folder/Directory (TensorFlow-Customer-Classificatio-Model)- Select Open in Terminal**



2. **PowerShell will Open**



⚙ Step 3: Run the Model

Use PowerShell or your terminal to run and verify the model setup.

```
python -c "import tensorflow as tf; print('TensorFlow version:', tf.__version__); print('GPUs:', tf.config.list_physical_devices('GPU'))"
```

```
PS C:\Users\johnm\OneDrive\Desktop\TensorFlow-Customer-Classificatio-Model> python -c "import tensorflow as tf; print('TensorFlow version:', tf.__version__)
; print('GPUs:', tf.config.list_physical_devices('GPU'))"
2025-10-30 00:15:43.372394: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
ating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-10-30 00:15:47.314827: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
ating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
TensorFlow version: 2.20.0
GPUs: []
```

ls

```
PS C:\Users\johnm\OneDrive\Desktop\TensorFlow-Customer-Classificatio-Model> ls


    Directory: C:\Users\johnm\OneDrive\Desktop\TensorFlow-Customer-Classificatio-Model


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         10/30/2025  12:07 AM           9785 tensorflow-customer-classification-model.py
```

python tensorflow-customer-classification-model.py

```
PS C:\Users\johnm\OneDrive\Desktop\TensorFlow-Customer-Classificatio-Model> python tensorflow-customer-classification-model.py
2025-10-30 00:16:02.427966: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
ating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-10-30 00:16:04.446489: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
ating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
C:\Users\johnm\OneDrive\Desktop\TensorFlow-Customer-Classificatio-Model\tensorflow-customer-classification-model.py:32: FutureWarning: Setting an item of in
compatible dtype is deprecated and will raise an error in a future version of pandas. Value '[ 0 23  7 13 23  1 12  9 10 11 13 15 22 11 12  2 18  0 22 22 23
 18  7  0
 23  9 13 10 11  7  8  8 23  4  0  0 20  8 20 17 23 20 16 16 11 12 23 17
 15 12  2 23  2 17 18 18 23  7  5  6 15 21 12 10  0  5  3 10  5  6 14  3
 19 15 12 12  9  9 20 20  0  6  0 14  4  0 23  7 18  0 16 14  3  4 15 23
  0 13 23  0  5  7 20 22 12 23 12 17  2  0  9  8 21 20  0 19 18 15 23  0
  4 17 23  3 16 20 10  4 19 19 13  8 23  0  7 23 13 20 18 14  6 17 15  8
 15 15 14  3 11 17 17  6 10  9 15  6 11 14 17  0 10 17  8 21  0  0  5 14
 11 15 10 12 11 12 15 17 23  3 13  0 15 16  4 12  5  3 23  2  0 23  8 14
 12 18 17 11 16  8 18  2 13 22  7  8  4 23  0 20  9 14  5 20  7 14 23 22
 19  0  5 21 13 15 23 22  7 15 18 22 21 16 13  0  0  1  0 13 21 14  2 17
 18 13  1 16 13  9 20 12 23 19  1  9 18 11  0 16  8 15  6  3 19  1 16 22
 23 23 21 23 12  0  3 12  6 23  9  0  0  4  3  0 14 16  6 11 17 12 17 13
 22 17  6 15 11  8  5 13  3  6  5 20  8 15 11  4 11  0 20  8  0 11  0  0
 22 21 16  9 10  9 23 20  8 11 21 12 11 23  7 17 18 19 18 20 19 21 17  7
  4  2 11 23  2 22 13  6  0 14 13 23 11 10 23 20 10  6  0 23 12  0 18 12
 17  5  9  6  4  1 16 10 10  7  0  2  9 10  9 15  7  7  9  5  3 19  8 15
 17 18  8 16 22 14 12 19 12 14 20 14 17 15  8 13 22  7 17 17 10 17 22 15
```

```
 10 10  1  5  8 15 11  8 12 13 23  8 12  8  0  8  0  0 10  8 11  0  2]' has dtype incompatible with int32, please explicitly cast to a compatible dtype firs
t.
  df.loc[new_user_indices, 'time_of_day'] = np.clip(time_of_day_new, 0, 23).astype(int)
Dataset generated: 2000 samples
   time_of_day  device_type_desktop  session_duration_sec  referral_source_social
0            6                    1                   132                       0
1            0                    1                   103                       1
2           23                    0                   148                       1
3            7                    0                   189                       0
4           23                    1                   203                       0

Training set size: 1600
Testing set size: 400
Number of features after encoding: 27
C:\Users\johnm\AppData\Local\Programs\Python\Python313\Lib\site-packages\keras\src\layers\core\dense.py:92: UserWarning: Do not pass an `input_shape`/`input
_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2025-10-30 00:16:06.218525: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in pe
rformance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

Model Architecture Summary:
Model: "sequential"
```

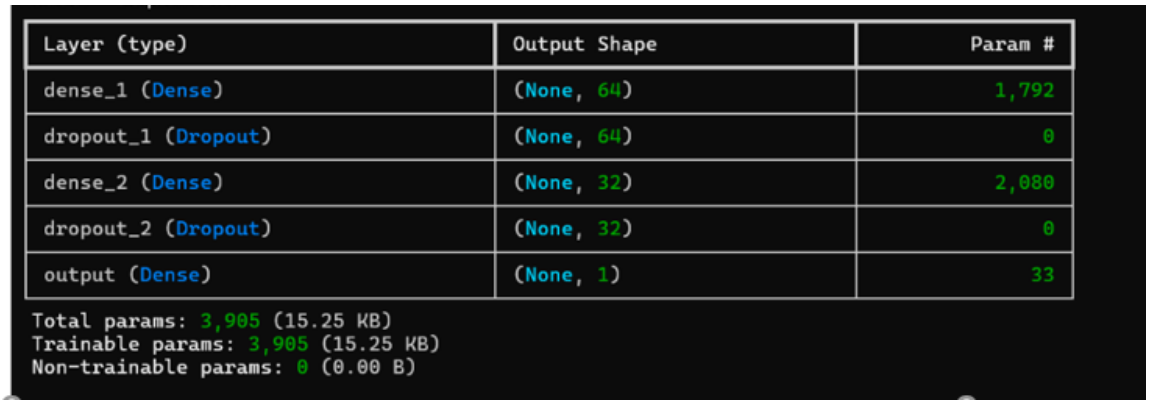
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 64)	1,792
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dropout_2 (Dropout)	(None, 32)	0
output (Dense)	(None, 1)	33

```
 Total params: 3,905 (15.25 KB)
 Trainable params: 3,905 (15.25 KB)
 Non-trainable params: 0 (0.00 B)
```

```
 Total params: 3,905 (15.25 KB)
 Trainable params: 3,905 (15.25 KB)
 Non-trainable params: 0 (0.00 B)

Starting model training...
Training finished after 16 epochs.

--- Model Evaluation ---
Test Accuracy: 0.7525
Test Precision: 0.7644
Test Recall: 0.7890

Generating training graphs...
-> Saved training_loss.png
-> Saved training_accuracy.png

--- Real-Time Prediction Simulation ---
Simulating prediction for a likely Existing Customer:
Features: TOD=21, Desktop=True, Duration=60s, Social=False
Predicted Probability of being Existing: 0.9275
Predicted Customer Type: Existing Customer (Login)
----------------------------------------------------
Simulating prediction for a likely New Customer:
Features: TOD=15, Desktop=False, Duration=250s, Social=True
Predicted Probability of being Existing: 0.0189
Predicted Customer Type: New Customer (Signup)
----------------------------------------------------

Model saved successfully for deployment: customer_type_classifier.keras

To integrate this model in a real-time environment, you would load the model, load the scaler, and pass the four input features to the 'predict_customer_typ
e' function.
```

Step 4: Model Training and Evaluation

During execution, the script will automatically:
• Generate and preprocess customer data
• Train the TensorFlow Keras model using 64 and 32 neuron dense layers



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 64)	1,792
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dropout_2 (Dropout)	(None, 32)	0
output (Dense)	(None, 1)	33

Total params: 3,905 (15.25 KB)
Trainable params: 3,905 (15.25 KB)
Non-trainable params: 0 (0.00 B)

• Apply dropout regularization to reduce overfitting
• Evaluate performance using Accuracy, Precision, and Recall metrics

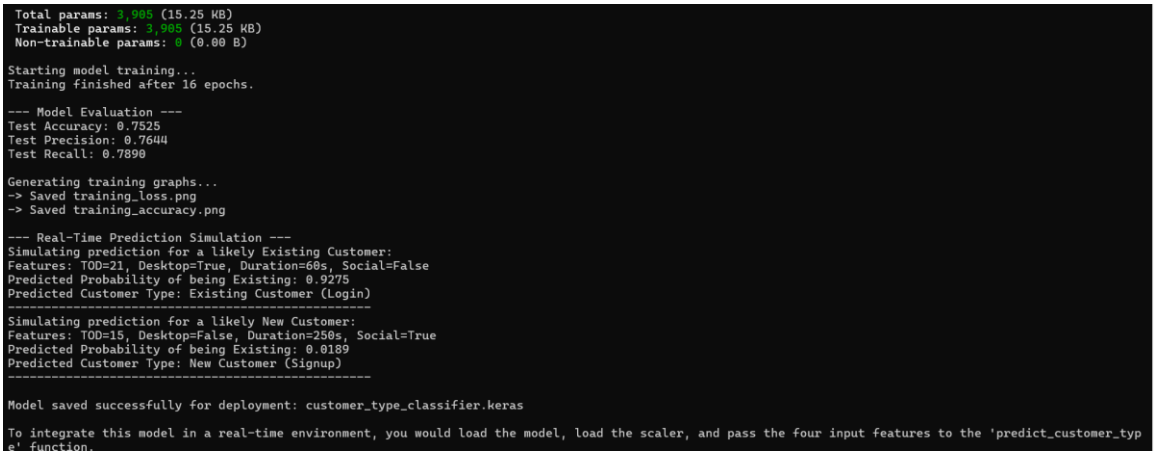
Training history graphs (loss and accuracy) will be saved as:
• training_loss.png
• training_accuracy.png

Step 5: Prediction Examples

Two prediction simulations are included in the script:
1. Existing Customer — uses late time (21:00), Desktop device, and short session.
2. New Customer — uses midday time (15:00), Mobile device, long session, and social referral.

Step 6: Model Saving and Deployment

The trained model is automatically saved as `customer_type_classifier.keras`. This file can be loaded into any TensorFlow environment for real-time deployment.

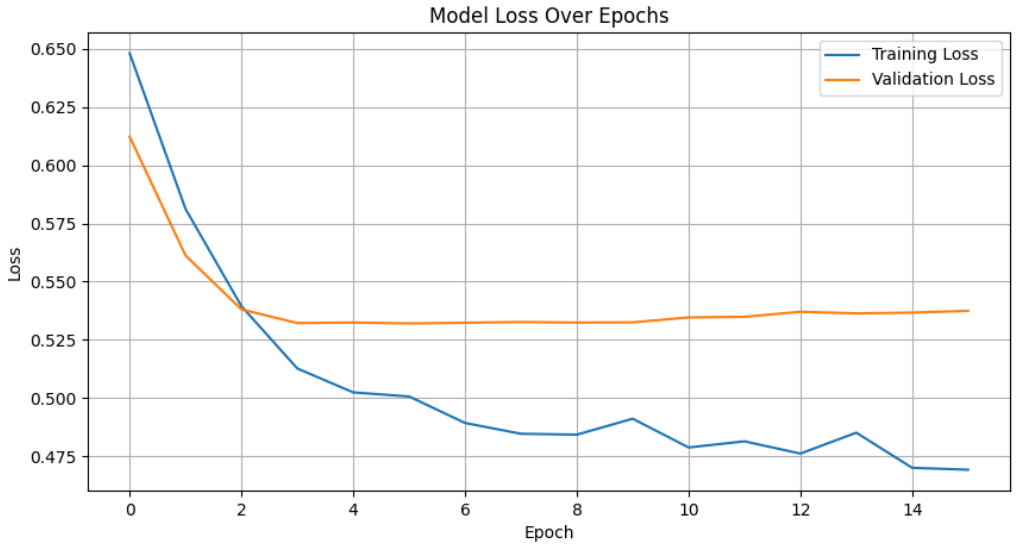


```
 Total params: 3,905 (15.25 KB)
 Trainable params: 3,905 (15.25 KB)
 Non-trainable params: 0 (0.00 B)

Starting model training...
Training finished after 16 epochs.

--- Model Evaluation ---
Test Accuracy: 0.7525
Test Precision: 0.7644
Test Recall: 0.7890

Generating training graphs...
-> Saved training_loss.png
-> Saved training_accuracy.png

--- Real-Time Prediction Simulation ---
Simulating prediction for a likely Existing Customer:
Features: TOD=21, Desktop=True, Duration=60s, Social=False
Predicted Probability of being Existing: 0.9275
Predicted Customer Type: Existing Customer (Login)
--------------------------------------------------
Simulating prediction for a likely New Customer:
Features: TOD=15, Desktop=False, Duration=250s, Social=True
Predicted Probability of being Existing: 0.0189
Predicted Customer Type: New Customer (Signup)
--------------------------------------------------

Model saved successfully for deployment: customer_type_classifier.keras

To integrate this model in a real-time environment, you would load the model, load the scaler, and pass the four input features to the 'predict_customer_typ
e' function.
```

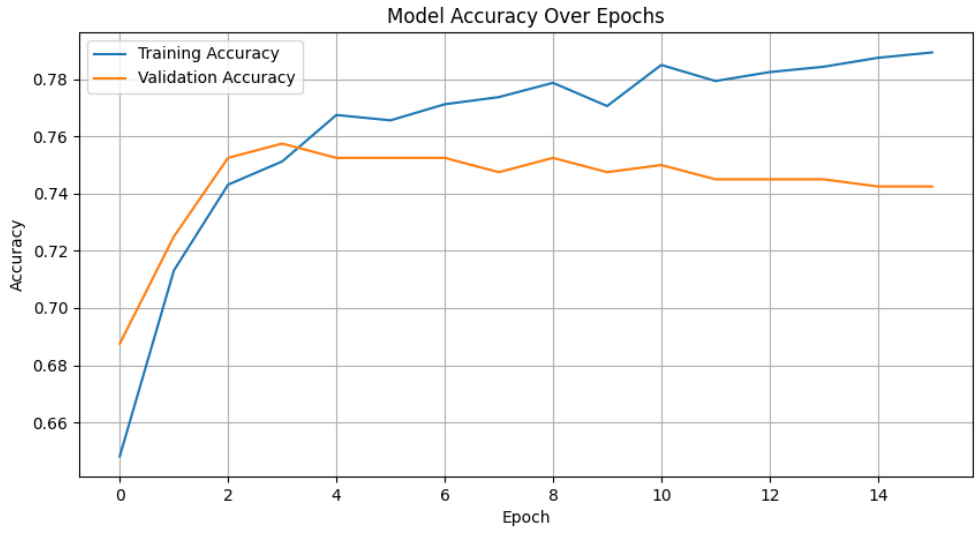
Step 7: Output Files

After successful execution, you will find the following files:

- customer_type_classifier.keras — Trained TensorFlow model
- training_loss.png — Loss curve visualization



- training_accuracy.png — Accuracy curve visualization



Step 8: Integration Hint

For real-world integration, load the saved model and scaler, then pass input features (`time_of_day`, `device_type_desktop`, `session_duration_sec`, `referral_source_social`) to the `predict_customer_type()` function to classify users dynamically.

☑ End of Guide — TensorFlow Customer Classification Model successfully documented.

GitHub Link: *git@github.com:johnmicky1/TensorFlow-Binary-Customer-Segmentation.git*