```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import plot_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt

# Step 1: Data Preprocessing
def preprocess_data(dataset_path):
    # Load dataset
    data = pd.read_csv(dataset_path)

    # Handle missing data (remove rows with missing values)
    data = data.dropna()

    # Separate features and labels
    X = data.drop(columns=['Soil Type'])
    y = data['Soil Type']

    # Encode categorical features
    categorical_features = ['Soil Texture']
    numerical_features = X.columns.difference(categorical_features)

    preprocessor = ColumnTransformer([
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

    # Transform features
    X = preprocessor.fit_transform(X)

    # Encode labels
    label_encoder = OneHotEncoder()
    y = label_encoder.fit_transform(y.values.reshape(-1, 1)).toarray()

    return X, y, preprocessor, label_encoder

# Step 2: Train-Test Split
def split_data(X, y):
    return train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Build the Deep Learning Model
def build_model(input_dim):
    model = Sequential([
        Dense(128, activation='relu', input_dim=input_dim),
        Dense(64, activation='relu'),
        Dense(3, activation='softmax')  # Assuming 3 soil types
    ])
    return model

# Step 4: Compile the Model
def compile_model(model):
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Step 5: Evaluate the Model
def evaluate_model(model, X_test, y_test):
    return model.evaluate(X_test, y_test)

# Step 6: Visualize Training and Validation Accuracy
def plot_training_history(history):
    plt.figure(figsize=(12, 6))

    # Accuracy plot
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Accuracy over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Loss over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

# Step 7: Make Predictions
def make_predictions(model, X_new):
    return model.predict(X_new)

# Step 8: Save and Load the Model
def save_model(model, file_path):
    model.save(file_path)

def load_model(file_path):
    return tf.keras.models.load_model(file_path)

# Step 9: Model Size
def get_model_size(file_path):
    import os
    return os.path.getsize(file_path)

# Main Workflow
if __name__ == "__main__":
    dataset_path = 'semi_arid_soil_dataset.csv'

    # Preprocess data
    X, y, preprocessor, label_encoder = preprocess_data(dataset_path)

    # Split data
    X_train, X_test, y_train, y_test = split_data(X, y)

    # Build model
    model = build_model(input_dim=X_train.shape[1])

    # Compile model
    compile_model(model)
```

```
# Train model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batc

# Evaluate model
loss, accuracy = evaluate_model(model, X_test, y_test)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

# Visualize training history
plot_training_history(history)

# Save and load model
model_path = 'soil_model.h5'
save_model(model, model_path)
loaded_model = load_model(model_path)

# Get model size
model_size = get_model_size(model_path)
print(f"Model Size: {model_size / 1024:.2f} KB")

# Visualize architecture
plot_model(model, to_file='model_architecture.png', show_shapes=True)
print("Model architecture saved as 'model_architecture.png'")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserW
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.5303 - loss: 1.0233 - va
Epoch 2/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9707 - loss: 0.6373 - va
Epoch 3/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.2306 - va
Epoch 4/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0521 - va
Epoch 5/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0190 - va
Epoch 6/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0101 - va
Epoch 7/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0066 - va
Epoch 8/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0047 - va
Epoch 9/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0036 - va
Epoch 10/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0028 - va
Epoch 11/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0023 - va
Epoch 12/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0019 - va
Epoch 13/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0016 - va
Epoch 14/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0014 - va
Epoch 15/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0012 - va
Epoch 16/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.0010 - v
Epoch 17/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 8.9126e-04
Epoch 18/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 7.8354e-04
Epoch 19/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 7.2637e-04
Epoch 20/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 6.4726e-04
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 6.5210e-04
Test Loss: 0.0006505745695903897, Test Accuracy: 1.0
```