```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import urllib.request
import os
def download_dataset():
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data"
    filename = "magic04.data"

    if not os.path.exists(filename):
        print("Downloading dataset...")
        urllib.request.urlretrieve(url, filename)
        print("Dataset downloaded successfully!")
    else:
        print("Dataset already exists!")

def load_data():
    # First, download the dataset if it doesn't exist
    download_dataset()

    # Load the dataset
    column_names = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1',
                    'fAsym', 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']
    data = pd.read_csv('magic04.data', names=column_names)
```

```
    # Convert 'g' (gamma) to 1 and 'h' (hadron) to 0
    data['class'] = (data['class'] == 'g').astype(int)

    # Split features and target
    X = data.drop('class', axis=1).values
    y = data['class'].values

    return X, y, data  # Return the DataFrame as well for printing

# Load the data and print the dataset
X, y, dataset = load_data()
print(dataset)
def create_model(activation_function):
    model = Sequential([
        Dense(1, input_dim=10),  # Input layer
        Dense(64, activation=activation_function),  # First hidden layer
        Dense(32, activation=activation_function),  # Second hidden layer
        Dense(1, activation=activation_function)  # Output layer
    ])

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

def train_and_evaluate(X, y, activation_function):
    # Split the data
    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_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Create and train the model
    model = create_model(activation_function)
    history = model.fit(X_train_scaled, y_train,
                        epochs=50,
                        batch_size=32,
                        validation_split=0.2,
                        verbose=1)

    # Evaluate the model
    y_pred = (model.predict(X_test_scaled) > 0.5).astype(int)
    accuracy = accuracy_score(y_test, y_pred)

    return history, accuracy, model

def plot_training_history(histories, activation_functions):
    plt.figure(figsize=(12, 5))

    # Plot training accuracy
    plt.subplot(1, 2, 1)
    for hist, act_func in zip(histories, activation_functions):
        plt.plot(hist.history['accuracy'], label=f'{act_func}')
    plt.title('Training Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot validation accuracy
    plt.subplot(1, 2, 2)
```

```
    for hist, act_func in zip(histories, activation_functions):
        plt.plot(hist.history['val_accuracy'], label=f'{act_func}')
    plt.title('Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()
# Main execution
def main():
    # Load data
    print("Loading data...")
    X, y = load_data()
    print("Data loaded successfully!")

    # Define activation functions to test
    activation_functions = ['sigmoid', 'tanh', 'relu', 'LeakyReLU']
    histories = []
    results = {}

    # Train and evaluate models with different activation functions
    for activation in activation_functions:
        print(f"\nTraining with {activation} activation function:")

        if activation == 'LeakyReLU':
            # For LeakyReLU, we need to use it differently due to its parameters
            history, accuracy, model = train_and_evaluate(X, y, tf.keras.layers.LeakyReLU(alpha=0.01))
        else:
            history, accuracy, model = train_and_evaluate(X, y, activation)
```

```
        histories.append(history)
        results[activation] = accuracy
        print(f"Test Accuracy: {accuracy:.4f}")

    # Plot training histories
    plot_training_history(histories, activation_functions)

    # Print final comparison
    print("\nFinal Results:")
    for activation, accuracy in results.items():
        print(f"{activation}: {accuracy:.4f}")

if __name__ == "__main__":
    main()
```