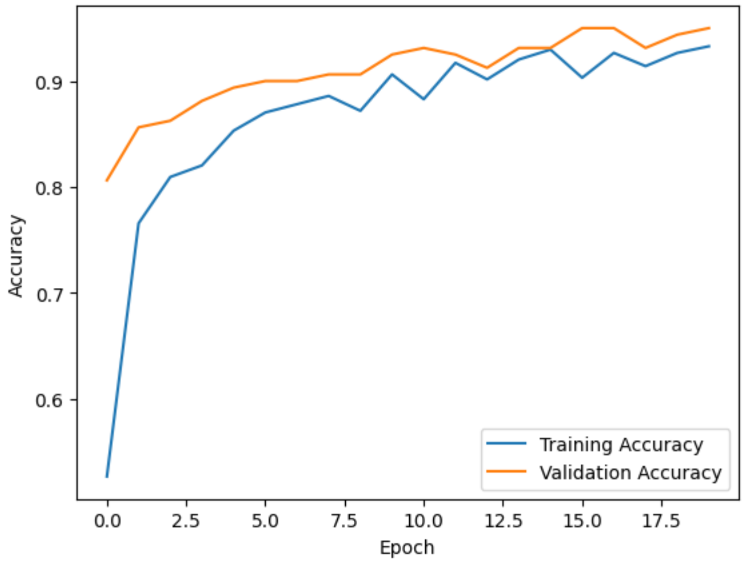
develop a nn tailored for pattern rexogniton by leveraging a dataset characterised by inticate patterns, and thoroughly assess its performance and create a sophisticated neural network design that is adept at recognizing complex patterns followed by a comprehensive evaluation of its efficacy in handling the intricacies present in the dataset

```
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=20, n_informative=10, n_redundant=10,
                           n_classes=2, random_state=42)


# Example: X, y = load_dataset()

# Split the dataset 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)

# Define the neural network architecture
model = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dropout(0.5),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # Assuming 10 classes for classification
])

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

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)

# Evaluate the model on the test set
y_pred = model.predict(X_test)
y_pred_classes = tf.argmax(y_pred, axis=1).numpy()

# Print accuracy and classification report
accuracy = accuracy_score(y_test, y_pred_classes)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("Classification Report:")
print(classification_report(y_test, y_pred_classes))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_classes)
print("Confusion Matrix:")
print(conf_matrix)

# Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

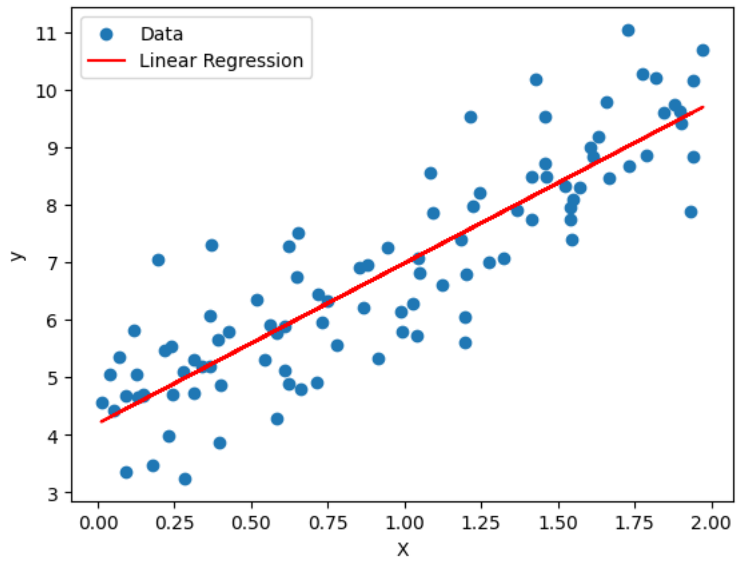
```
Epoch 1/20
20/20 [==============================] - 3s 41ms/step - loss: 1.4634 - accuracy: 0.5266 - val_loss: 0.5126 - val_accuracy: 0.8062
Epoch 2/20
20/20 [==============================] - 0s 16ms/step - loss: 0.5474 - accuracy: 0.7656 - val_loss: 0.3962 - val_accuracy: 0.8562
Epoch 3/20
20/20 [==============================] - 0s 17ms/step - loss: 0.4603 - accuracy: 0.8094 - val_loss: 0.3355 - val_accuracy: 0.8625
Epoch 4/20
20/20 [==============================] - 0s 17ms/step - loss: 0.4014 - accuracy: 0.8203 - val_loss: 0.3018 - val_accuracy: 0.8813
Epoch 5/20
20/20 [==============================] - 0s 16ms/step - loss: 0.3567 - accuracy: 0.8531 - val_loss: 0.2733 - val_accuracy: 0.8938
Epoch 6/20
20/20 [==============================] - 0s 18ms/step - loss: 0.3165 - accuracy: 0.8703 - val_loss: 0.2541 - val_accuracy: 0.9000
Epoch 7/20
20/20 [==============================] - 0s 14ms/step - loss: 0.3117 - accuracy: 0.8781 - val_loss: 0.2346 - val_accuracy: 0.9000
Epoch 8/20
20/20 [==============================] - 0s 17ms/step - loss: 0.2931 - accuracy: 0.8859 - val_loss: 0.2245 - val_accuracy: 0.9062
Epoch 9/20
20/20 [==============================] - 0s 15ms/step - loss: 0.3004 - accuracy: 0.8719 - val_loss: 0.2064 - val_accuracy: 0.9062
Epoch 10/20
20/20 [==============================] - 0s 17ms/step - loss: 0.2530 - accuracy: 0.9062 - val_loss: 0.2042 - val_accuracy: 0.9250
Epoch 11/20
20/20 [==============================] - 0s 14ms/step - loss: 0.2555 - accuracy: 0.8828 - val_loss: 0.1965 - val_accuracy: 0.9312
Epoch 12/20
20/20 [==============================] - 0s 24ms/step - loss: 0.2173 - accuracy: 0.9172 - val_loss: 0.1843 - val_accuracy: 0.9250
Epoch 13/20
20/20 [==============================] - 0s 13ms/step - loss: 0.2409 - accuracy: 0.9016 - val_loss: 0.1832 - val_accuracy: 0.9125
Epoch 14/20
20/20 [==============================] - 0s 16ms/step - loss: 0.2227 - accuracy: 0.9203 - val_loss: 0.1777 - val_accuracy: 0.9312
Epoch 15/20
20/20 [==============================] - 0s 21ms/step - loss: 0.2046 - accuracy: 0.9297 - val_loss: 0.1736 - val_accuracy: 0.9312
Epoch 16/20
20/20 [==============================] - 0s 17ms/step - loss: 0.2309 - accuracy: 0.9031 - val_loss: 0.1610 - val_accuracy: 0.9500
Epoch 17/20
20/20 [==============================] - 0s 10ms/step - loss: 0.2047 - accuracy: 0.9266 - val_loss: 0.1593 - val_accuracy: 0.9500
Epoch 18/20
20/20 [==============================] - 0s 19ms/step - loss: 0.2035 - accuracy: 0.9141 - val_loss: 0.1567 - val_accuracy: 0.9312
Epoch 19/20
20/20 [==============================] - 0s 13ms/step - loss: 0.1851 - accuracy: 0.9266 - val_loss: 0.1504 - val_accuracy: 0.9438
Epoch 20/20
20/20 [==============================] - 0s 15ms/step - loss: 0.1713 - accuracy: 0.9328 - val_loss: 0.1502 - val_accuracy: 0.9500
7/7 [==============================] - 0s 7ms/step
Accuracy: 94.00%
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.91      0.94       101
           1       0.91      0.97      0.94        99

    accuracy                           0.94       200
   macro avg       0.94      0.94      0.94       200
weighted avg       0.94      0.94      0.94       200

Confusion Matrix:
[[92  9]
 [ 3 96]]
```



develop and implement the back propagation algorithm to train a nn showcasing it convergence on representative dataset. involve the creation of the algorithm that effectively employs backporpagation demonstrating its ability to optimise the nn parameter and achieve convergence when applied to a sample dataset

```
import numpy as np
import matplotlib.pyplot as plt

# Generate a synthetic dataset
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Add a bias term to input features
X_b = np.c_[np.ones((100, 1)), X]

# Initialize random weights
np.random.seed(42)
theta = np.random.randn(2, 1)

# Set hyperparameters
learning_rate = 0.01
n_iterations = 1000

# Backpropagation algorithm
for iteration in range(n_iterations):
    # Forward propagation
    y_pred = X_b.dot(theta)
    error = y_pred - y

    # Calculate gradients
    gradients = 2/100 * X_b.T.dot(error)

    # Update weights using gradient descent
    theta = theta - learning_rate * gradients

# Plot the data and the linear regression line
plt.scatter(X, y, label='Data')
plt.plot(X, X_b.dot(theta), color='red', label='Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x78b2d1a83fa0>
```


Data
Linear Regression
y
X