```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_iris
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
```

```
# ---------- Load and Prepare the Iris Dataset ----------
iris = load_iris()
X = iris.data  # 4 features
y = iris.target  # 3 classes

# Normalize input
# scaler = StandardScaler()
# X = scaler.fit_transform(X)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ---------- Build a Simple Neural Network ----------
model = Sequential([
    Dense(10, activation='relu', input_shape=(4,)),
    Dense(3, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=50, verbose=0)

# Evaluate
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"✅ Test Accuracy: {acc:.4f}")
```

```
✅ Test Accuracy: 0.9000
```

```
# ---------- Convert to Quantized TFLite Model ----------
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]  # dynamic range quantization
tflite_model = converter.convert()

# Save .tflite
with open("iris_model.tflite", "wb") as f:
    f.write(tflite_model)

print("✅ Saved iris_model.tflite")

with open("iris_model.tflite", "rb") as f:
        model_bytes = f.read()

print(len(model_bytes))
```

```
Saved artifact at '/tmp/tmphc45s4jl'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 4), dtype=tf.float32, name='keras_tensor')
Output Type:
  TensorSpec(shape=(None, 3), dtype=tf.float32, name=None)
Captures:
  137325154637712: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137325154638864: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137325154637520: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137325154635216: TensorSpec(shape=(), dtype=tf.resource, name=None)
✅ Saved iris_model.tflite
2060
```

```
sample = np.array([X[70]])  # this will be: [5.9, 3.2, 4.8, 1.8]
print("Original sample:", sample)
```

```
Original sample: [[5.9 3.2 4.8 1.8]]
```

```
import time

# ----------------- Load the .tflite Model -----------------
interpreter = tf.lite.Interpreter(model_path="iris_model.tflite")
```

```
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

st_time = time.time()
# Ensure correct shape and dtype
input_data = np.array(sample, dtype=np.float32)
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()

# ----------------- Get and Print Prediction -----------------
output = interpreter.get_tensor(output_details[0]['index'])
predicted_class = np.argmax(output)
en_time = time.time()

print("Predicted probabilities:", output)
print("✅ Predicted class:", predicted_class)
print("Time for inference:", (en_time-st_time)*(10**6), "micro sec.")
```

```
Predicted probabilities: [[0.02981904 0.40336978 0.5668112 ]]
✅ Predicted class: 2
Time for inference: 485.1818084716797 micro sec.
```

```
def convert_tflite_to_header(tflite_path, header_path="iris_model_data.h", var_name="iris_model"):
    with open(tflite_path, "rb") as f:
        model_bytes = f.read()

    with open(header_path, "w") as f:
        f.write(f"#ifndef {var_name.upper()}_H\n")
        f.write(f"#define {var_name.upper()}_H\n\n")
        f.write("#include <cstdint>\n\n")
        f.write(f"const unsigned char {var_name}[] = {{\n")

        for i in range(0, len(model_bytes), 12):
            line = ", ".join(f"0x{b:02x}" for b in model_bytes[i:i+12])
            f.write("  " + line + ",\n")

        f.write("};\n")
        f.write(f"const unsigned int {var_name}_len = {len(model_bytes)};\n\n")
        f.write(f"#endif // {var_name.upper()}_H\n")

    print(f"✅ Generated {header_path}")

# Run header conversion
convert_tflite_to_header("iris_model.tflite")
```

```
✅ Generated iris_model_data.h with 2060 bytes
```