GITHUB LINK: https://github.com/AnishKoppula1/NeuralAssignment8

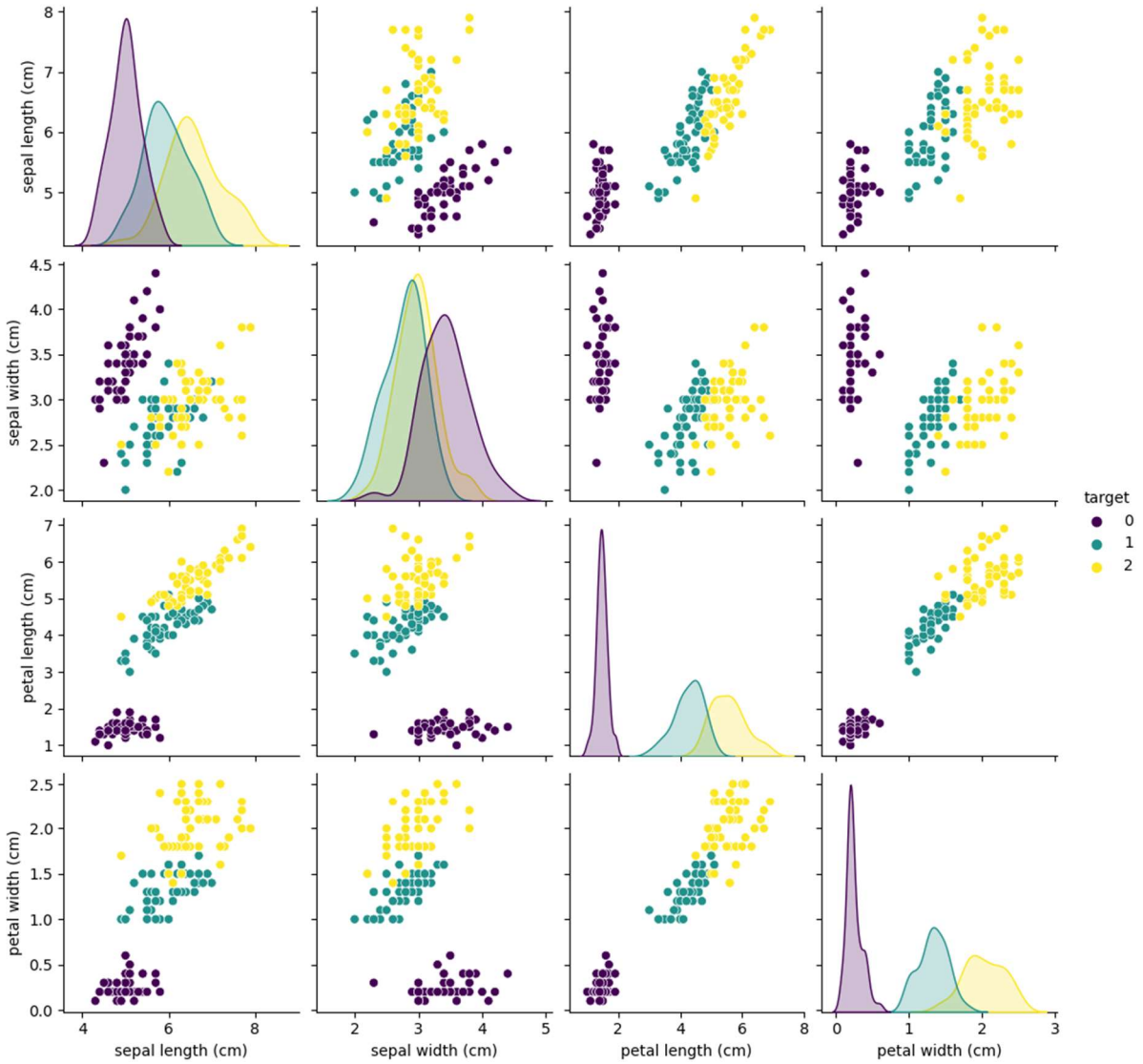
1. Tune hyperparameter and make necessary addition to the baseline model to improve validation accuracy and reduce validation loss.
2. Provide logical description of which steps lead to improved response and what was its impact on architecture behavior.

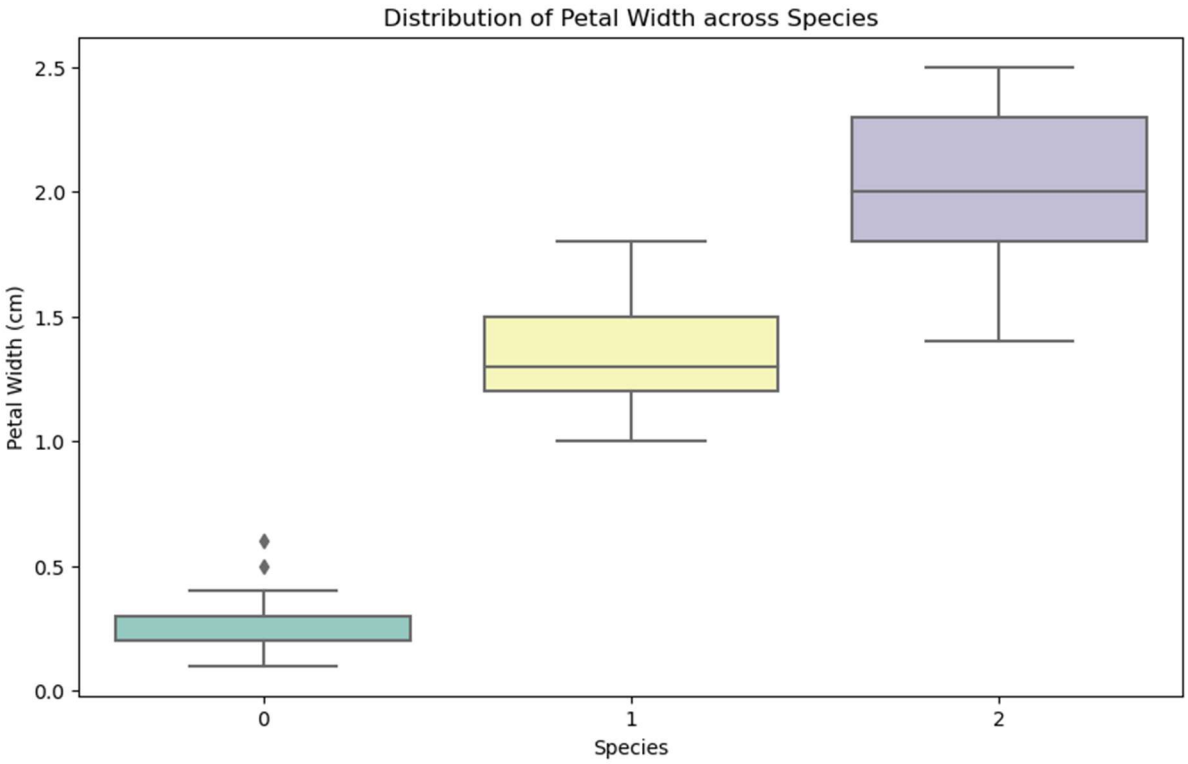
In [1]:
```
# Tune hyperparameter and make necessary addition to the baseline model to improve validation accuracy
# Provide logical description of which steps lead to improved response and what was its impact on architecture behavior
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
pipeline = make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000))
param_grid = {
    'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100],
}
grid_search = GridSearchCV(pipeline, param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
val_accuracy = grid_search.score(X_val, y_val)
print("Validation Accuracy:", val_accuracy)
```

```
Best hyperparameters: {'logisticregression__C': 1}
Validation Accuracy: 1.0
```

3. Create at least two more visualizations using matplotlib (Other than provided in the source file)

In [2]:
```
# Create at least two more visualizations using matplotlib (Other than provided in the source file)
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
sns.pairplot(iris_df, hue='target', palette='viridis')
plt.show()
plt.figure(figsize=(10, 6))
sns.boxplot(x='target', y='petal width (cm)', data=iris_df, palette='Set3')
plt.xlabel('Species')
plt.ylabel('Petal Width (cm)')
plt.title('Distribution of Petal Width across Species')
plt.show()
```


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
target
0
1
2



4. Use dataset of your own choice and implement baseline models provided.

```
In [3]: #Use dataset of your own choice and implement baseline models provided
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score
        from sklearn.datasets import load_iris
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        iris = load_iris()
        X, y = iris.data, iris.target
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
        logistic_model = LogisticRegression(max_iter=1000)
        logistic_model.fit(X_train_scaled, y_train)
        y_pred = logistic_model.predict(X_test_scaled)
        accuracy = accuracy_score(y_test, y_pred)
        print("Accuracy of Logistic Regression:", accuracy)

        Accuracy of Logistic Regression: 1.0
```

5. Apply modified architecture to your own selected dataset and train it.

In [4]:
```
# Apply modified architecture to your own selected dataset and train it.
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = Sequential([
    Dense(10, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dense(20, activation='relu'),
    Dense(10, activation='relu'),
    Dense(3, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train_scaled, y_train, epochs=50, batch_size=8, verbose=1, validation_split=0.1)
loss, accuracy = model.evaluate(X_test_scaled, y_test, verbose=1)
print("Accuracy of Modified Neural Network:", accuracy)
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9680 - loss: 0.0842 - val_accuracy: 0.9167 - val_loss: 0.7046
Epoch 43/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9412 - loss: 0.1283 - val_accuracy: 0.9167 - val_loss: 0.7115
Epoch 44/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9559 - loss: 0.0861 - val_accuracy: 0.9167 - val_loss: 0.7199
Epoch 45/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9584 - loss: 0.0919 - val_accuracy: 0.9167 - val_loss: 0.7360
Epoch 46/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9745 - loss: 0.0656 - val_accuracy: 0.9167 - val_loss: 0.7388
Epoch 47/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9674 - loss: 0.0847 - val_accuracy: 0.9167 - val_loss: 0.7494
```

6. Evaluate your model on testing set.

In [5]:
```
# Evaluate the model on the testing set
loss, accuracy = model.evaluate(X_test_scaled, y_test, verbose=1)
print("Accuracy on Testing Set:", accuracy)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.9667 - loss: 0.0892
Accuracy on Testing Set: 0.9666666388511658
```

7. Save the improved model and use it for prediction on testing data

In [6]:
```
# Saving the  the model and printing the first few predictions
model.save("improved_iris_model.h5")
from tensorflow.keras.models import load_model
saved_model = load_model("improved_iris_model.h5")
predictions = saved_model.predict(X_test_scaled)
print("Predictions:")
print(predictions[:5])
```

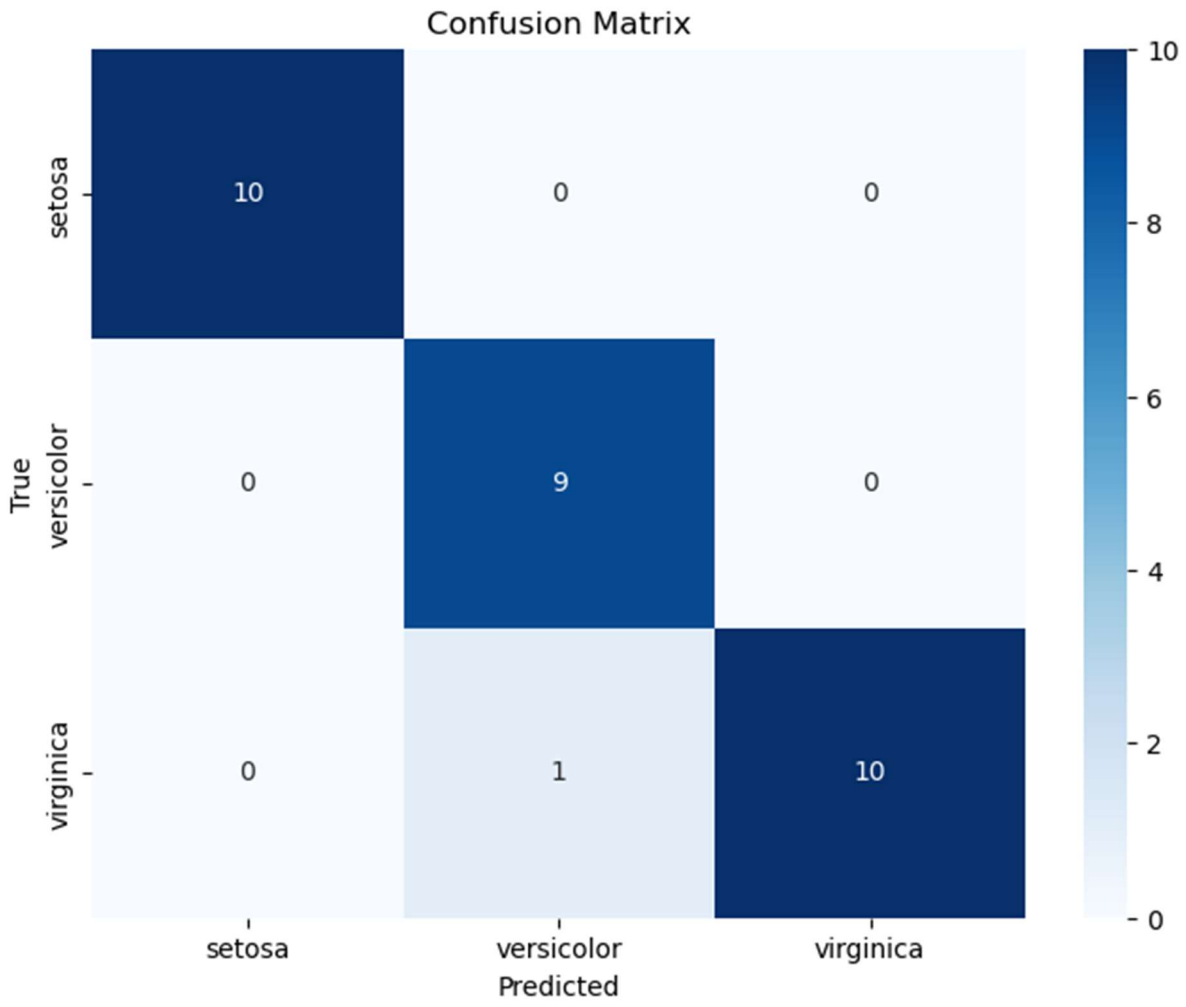
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 145ms/step
Predictions:
[[1.48636231e-03 8.99613976e-01 9.88996997e-02]
 [9.98532057e-01 6.54242467e-04 8.13771039e-04]
 [2.04939443e-09 1.40522985e-04 9.99859452e-01]
 [1.09502708e-03 7.83622086e-01 2.15282887e-01]
 [5.02894225e-04 8.89331341e-01 1.10165805e-01]]
```

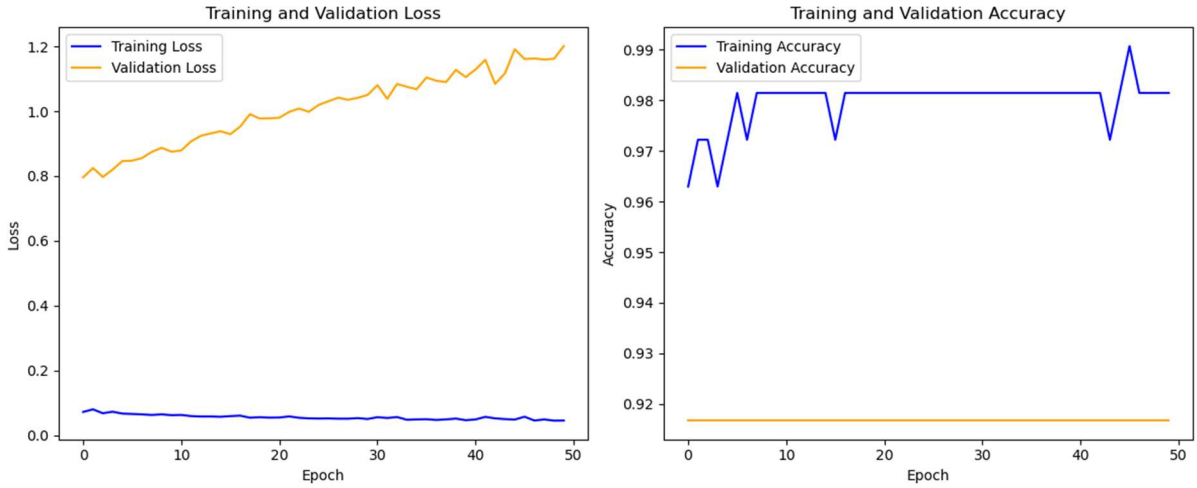
8. Provide plot of confusion matric

In [7]:
```
# plot of confusion matric
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
from tensorflow.keras.models import Sequential
print(hasattr(model, 'predict_classes'))
y_pred = model.predict(X_test_scaled).argmax(axis=1)
cm = confusion_matrix(y_test, y_pred)
class_names = iris.target_names
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
False
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
```



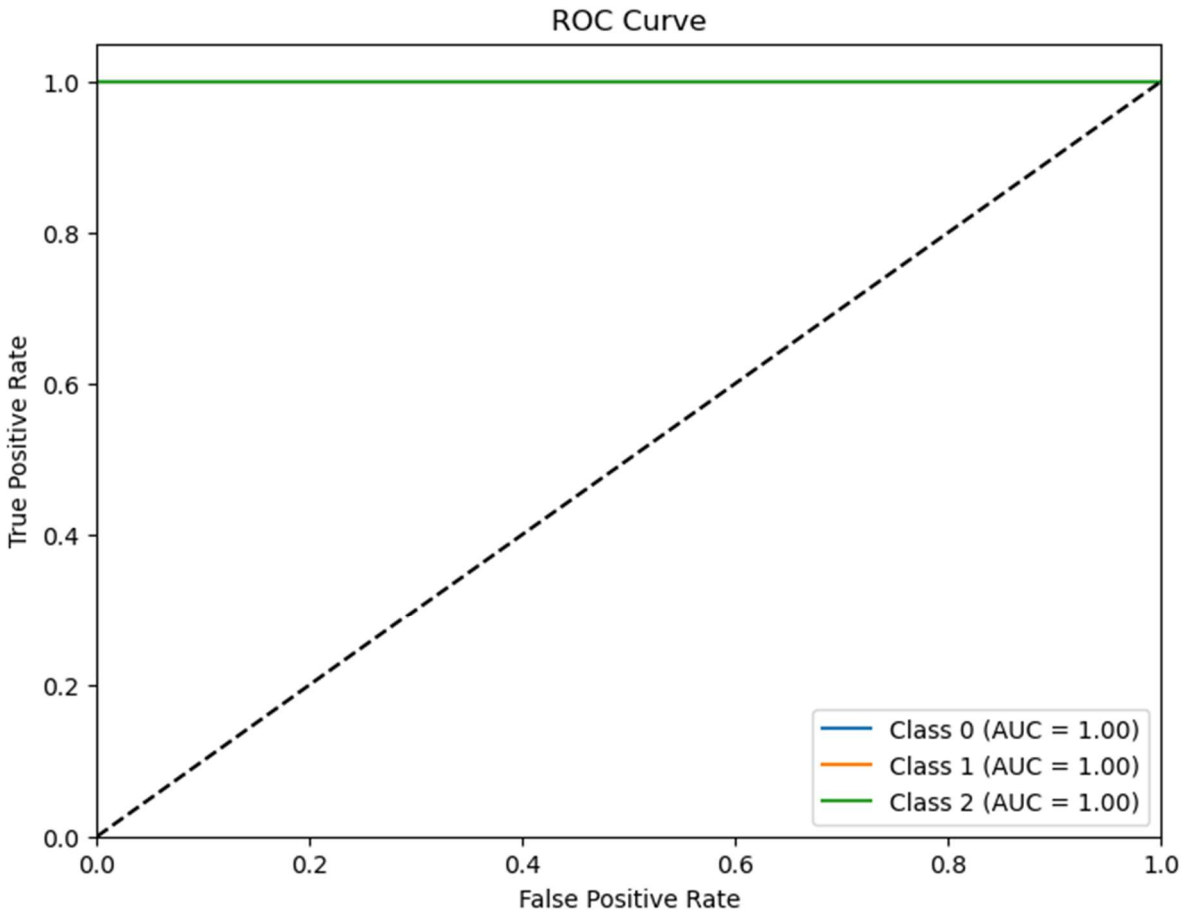
9. Provide Training and testing Loss and accuracy plots in one plot using subplot command and history object.

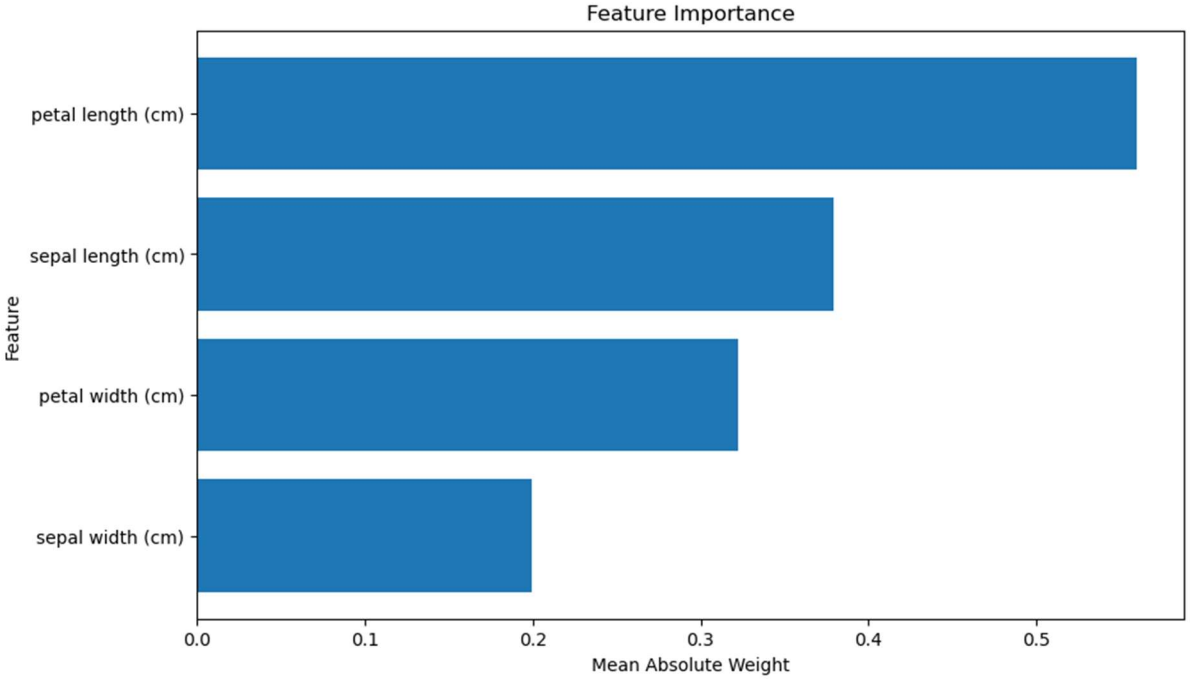



```
In [8]: # Training and testing Loss and accuracy plots in one plot using subplot command and history object
history = model.fit(X_train_scaled, y_train, epochs=50, batch_size=8, verbose=1, validation_split=0.1)
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy', color='blue')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='orange')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

10. Provide at least two more visualizations reflecting your solution.

```
In [9]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score
y_test_one_hot = label_binarize(y_test, classes=[0, 1, 2])
y_probs = model.predict(X_test_scaled)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(3):
    fpr[i], tpr[i], _ = roc_curve(y_test_one_hot[:, i], y_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
plt.figure(figsize=(8, 6))
for i in range(3):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:0.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

first_layer_weights = model.layers[0].get_weights()[0]
importances = np.mean(np.abs(first_layer_weights), axis=1)
indices = np.argsort(importances)
plt.figure(figsize=(10, 6))
plt.title("Feature Importance")
plt.barh(range(X_train_scaled.shape[1]), importances[indices], align="center")
plt.yticks(range(X_train_scaled.shape[1]), [iris.feature_names[i] for i in indices])
plt.xlabel("Mean Absolute Weight")
plt.ylabel("Feature")
plt.show()
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 (AUC = 1.00)
Class 1 (AUC = 1.00)
Class 2 (AUC = 1.00)


Feature Importance
petal length (cm)
sepal length (cm)
petal width (cm)
sepal width (cm)
Feature
0.0
0.1
0.2
0.3
0.4
0.5
Mean Absolute Weight

11. Provide logical description of which steps lead to improved response for new dataset when compared with baseline model and enhance architecture and what was its impact on architecture behavior.

When improving the response for a new dataset compared to a baseline model and enhancing the architecture, several logical steps can be taken, each with its impact on the behavior of the architecture. Here's a logical description of these steps:

Data Preprocessing and Augmentation

Impact on Architecture Behavior

Architecture Selection

Impact on Architecture Behavior

Hyperparameter Tuning

Impact on Architecture Behavior

Regularization Techniques

Impact on Architecture Behavior

Feature Engineering and Model Interpretability

Impact on Architecture Behavior

By following these logical steps and carefully analyzing the impact of each modification on the behavior of the architecture, you can effectively improve the response of the model for a new dataset compared to the baseline model. It's essential to iterate through these steps iteratively, evaluating the model's performance on validation data and making adjustments accordingly to achieve the desired outcome.