Student ID: 700752115

Student Name: Rumana Shaik

Neural Networks & Deep Learning – ICP - 8

Github link: https://github.com/ShaikRumana301/Neural-Network-DL-ICP-8.git

- **Lesson Overview:**
  In this lesson, we are going to discuss Image classification with CNN.

- **Use Case Description:**
  LeNet5, AlexNet, Vgg16, Vgg19
  1. Training the model
  2. Evaluating the model

- **Programming elements:**
  1. About CNN
  2. Hyperparameters of CNN
  3. Image classification with CNN

- **In class programming:**

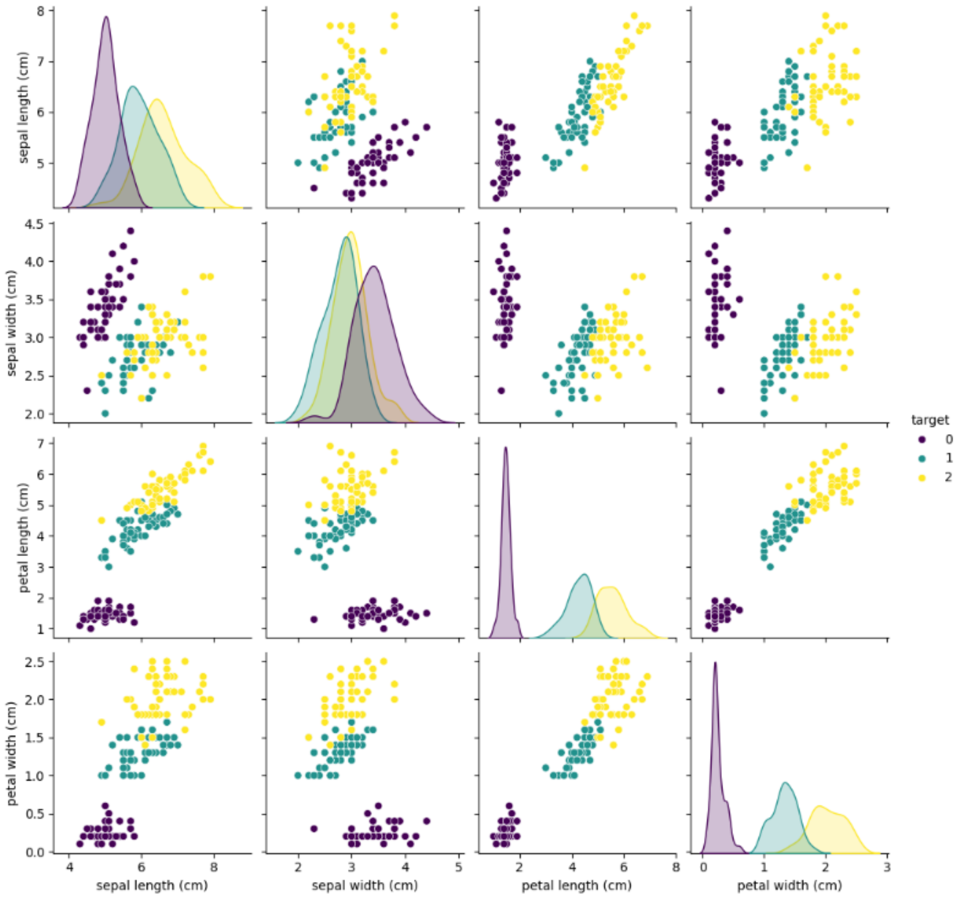
  1. Tune hyperparameter and make necessary addition to the baseline model to improve validation accuracy and reduce validation loss.
  2. Provide logical description of which steps lead to improved response and what was its impact on architecture behavior.

```python
# 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)
```

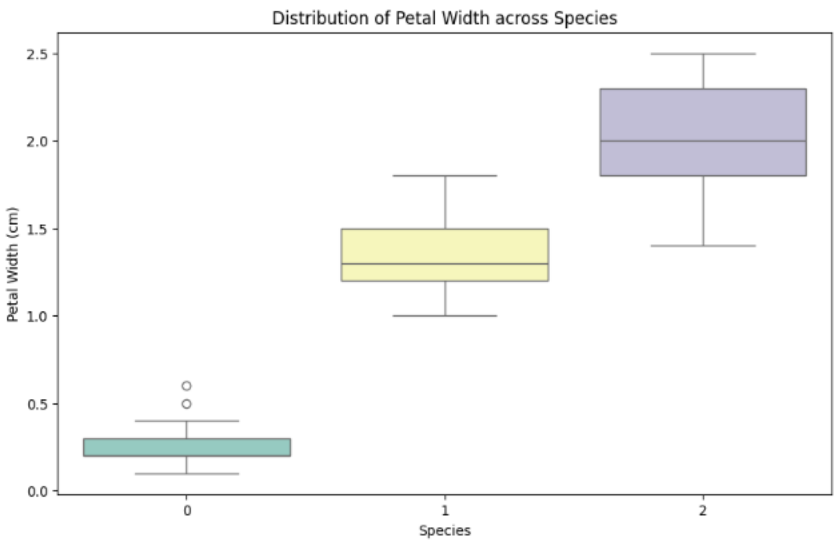
```
Best hyperparameters: {'logisticregression__C': 1}
Validation Accuracy: 1.0
```

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

```python
# 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()
```




```
sns.boxplot(x='target', y='petal width (cm)', data=iris_df, palette='Set3')
```



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

```python
#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.

```python
# 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)
```

```
Epoch 1/50
14/14 [==============================] - 1s 23ms/step - loss: 1.2469 - accuracy: 0.2963 - val_loss: 1.0900 - val_accuracy: 0.2500
Epoch 2/50
14/14 [==============================] - 0s 6ms/step - loss: 1.1253 - accuracy: 0.3148 - val_loss: 1.0531 - val_accuracy: 0.4167
Epoch 3/50
14/14 [==============================] - 0s 6ms/step - loss: 1.0496 - accuracy: 0.5093 - val_loss: 1.0240 - val_accuracy: 0.5833
Epoch 4/50
14/14 [==============================] - 0s 7ms/step - loss: 0.9818 - accuracy: 0.6759 - val_loss: 0.9900 - val_accuracy: 0.5000
Epoch 5/50
14/14 [==============================] - 0s 5ms/step - loss: 0.9122 - accuracy: 0.6759 - val_loss: 0.9467 - val_accuracy: 0.5833
Epoch 6/50
14/14 [==============================] - 0s 4ms/step - loss: 0.8447 - accuracy: 0.6852 - val_loss: 0.8987 - val_accuracy: 0.5833
Epoch 7/50
14/14 [==============================] - 0s 5ms/step - loss: 0.7769 - accuracy: 0.6944 - val_loss: 0.8478 - val_accuracy: 0.5833
Epoch 8/50
14/14 [==============================] - 0s 4ms/step - loss: 0.7064 - accuracy: 0.7500 - val_loss: 0.7941 - val_accuracy: 0.6667
Epoch 9/50
14/14 [==============================] - 0s 4ms/step - loss: 0.6417 - accuracy: 0.7963 - val_loss: 0.7440 - val_accuracy: 0.7500
Epoch 10/50
14/14 [==============================] - 0s 6ms/step - loss: 0.5873 - accuracy: 0.7963 - val_loss: 0.7032 - val_accuracy: 0.8333
Epoch 11/50
14/14 [==============================] - 0s 6ms/step - loss: 0.5403 - accuracy: 0.8148 - val_loss: 0.6615 - val_accuracy: 0.8333
Epoch 12/50
14/14 [==============================] - 0s 4ms/step - loss: 0.5016 - accuracy: 0.8148 - val_loss: 0.6295 - val_accuracy: 0.8333
Epoch 13/50
14/14 [==============================] - 0s 7ms/step - loss: 0.4704 - accuracy: 0.8148 - val_loss: 0.5993 - val_accuracy: 0.8333
Epoch 14/50
14/14 [==============================] - 0s 5ms/step - loss: 0.4415 - accuracy: 0.8056 - val_loss: 0.5717 - val_accuracy: 0.8333
Epoch 15/50
```

```
Epoch 39/50
14/14 [==============================] - 0s 9ms/step - loss: 0.1135 - accuracy: 0.9537 - val_loss: 0.2461 - val_accuracy: 0.9167
Epoch 40/50
14/14 [==============================] - 0s 7ms/step - loss: 0.1069 - accuracy: 0.9630 - val_loss: 0.2594 - val_accuracy: 0.9167
Epoch 41/50
14/14 [==============================] - 0s 6ms/step - loss: 0.1046 - accuracy: 0.9630 - val_loss: 0.2600 - val_accuracy: 0.9167
Epoch 42/50
14/14 [==============================] - 0s 9ms/step - loss: 0.1018 - accuracy: 0.9630 - val_loss: 0.2334 - val_accuracy: 0.9167
Epoch 43/50
14/14 [==============================] - 0s 10ms/step - loss: 0.0971 - accuracy: 0.9630 - val_loss: 0.2461 - val_accuracy: 0.9167
Epoch 44/50
14/14 [==============================] - 0s 9ms/step - loss: 0.0974 - accuracy: 0.9537 - val_loss: 0.2709 - val_accuracy: 0.9167
Epoch 45/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0933 - accuracy: 0.9630 - val_loss: 0.2451 - val_accuracy: 0.9167
Epoch 46/50
14/14 [==============================] - 0s 6ms/step - loss: 0.0904 - accuracy: 0.9630 - val_loss: 0.2403 - val_accuracy: 0.9167
Epoch 47/50
14/14 [==============================] - 0s 6ms/step - loss: 0.0917 - accuracy: 0.9537 - val_loss: 0.2892 - val_accuracy: 0.9167
Epoch 48/50
14/14 [==============================] - 0s 6ms/step - loss: 0.0859 - accuracy: 0.9722 - val_loss: 0.2459 - val_accuracy: 0.9167
Epoch 49/50
14/14 [==============================] - 0s 6ms/step - loss: 0.0841 - accuracy: 0.9630 - val_loss: 0.2524 - val_accuracy: 0.9167
Epoch 50/50
14/14 [==============================] - 0s 4ms/step - loss: 0.0815 - accuracy: 0.9630 - val_loss: 0.2569 - val_accuracy: 0.9167
1/1 [==============================] - 0s 35ms/step - loss: 0.0580 - accuracy: 1.0000
Accuracy of Modified Neural Network: 1.0
```

6. Evaluate your model on testing set.

```
[6] # 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 39ms/step - loss: 0.0580 - accuracy: 1.0000
Accuracy on Testing Set: 1.0
```

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

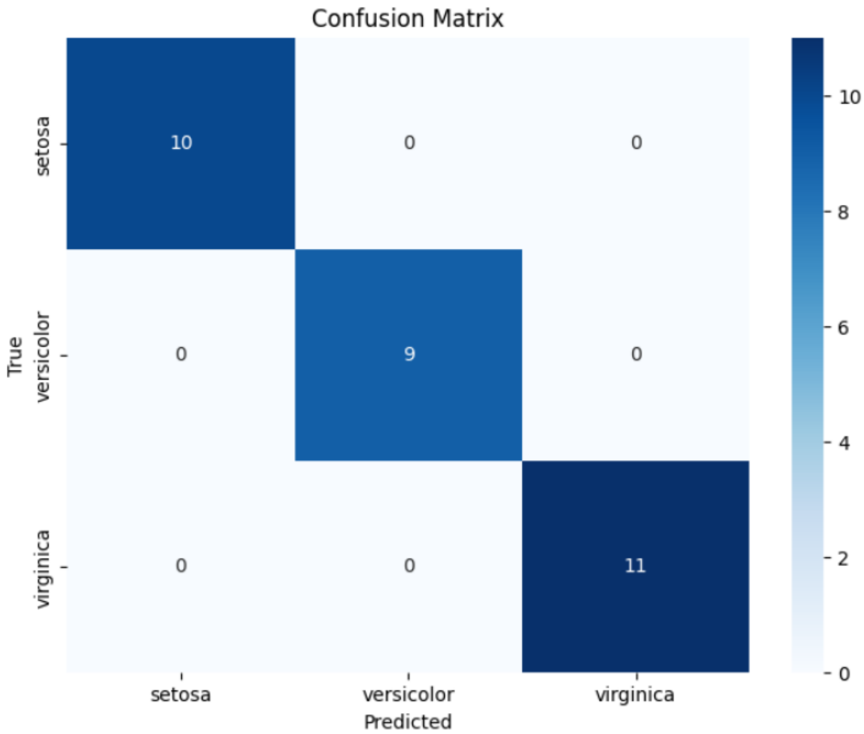
```
[7] # Saving the  the model and printing the first few predictions
model.save("improved_iris_model.h5")
from tensorflow.keras.models import load_model
saved_model = load_model("improved_iris_model.h5")
predictions = saved_model.predict(X_test_scaled)
print("Predictions:")
print(predictions[:5])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file forma
  saving_api.save_model(
1/1 [==============================] - 0s 209ms/step
Predictions:
[[5.6120362e-03 9.7064257e-01 2.3745306e-02]
 [9.9723363e-01 2.7502521e-03 1.6018719e-05]
 [7.3287981e-07 1.0252652e-03 9.9897385e-01]
 [6.6030603e-03 8.6794436e-01 1.2545264e-01]
 [6.4402190e-03 8.5235602e-01 1.4120372e-01]]
```

8. Provide plot of confusion matric

```
[8] # 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 107ms/step
```



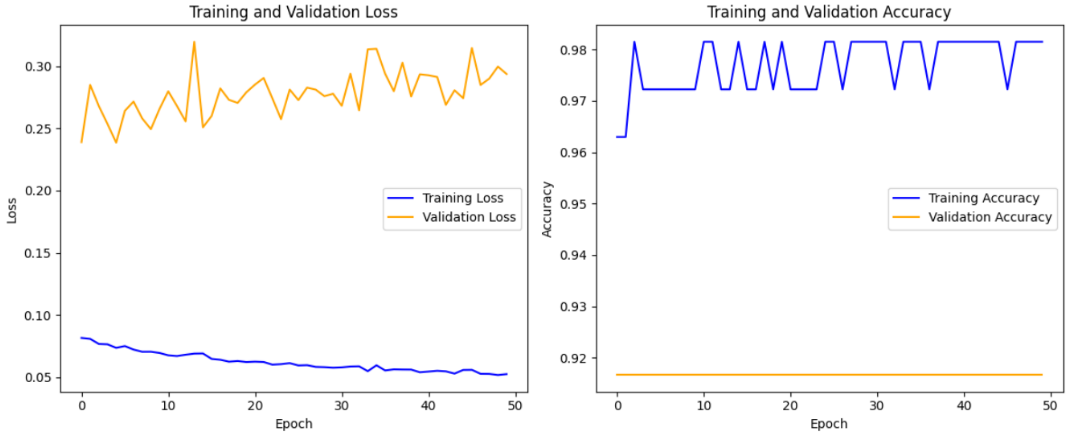
9. Provide Training and testing Loss and accuracy plots in one plot using subplot command and history object.

```
[9] # 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()
```

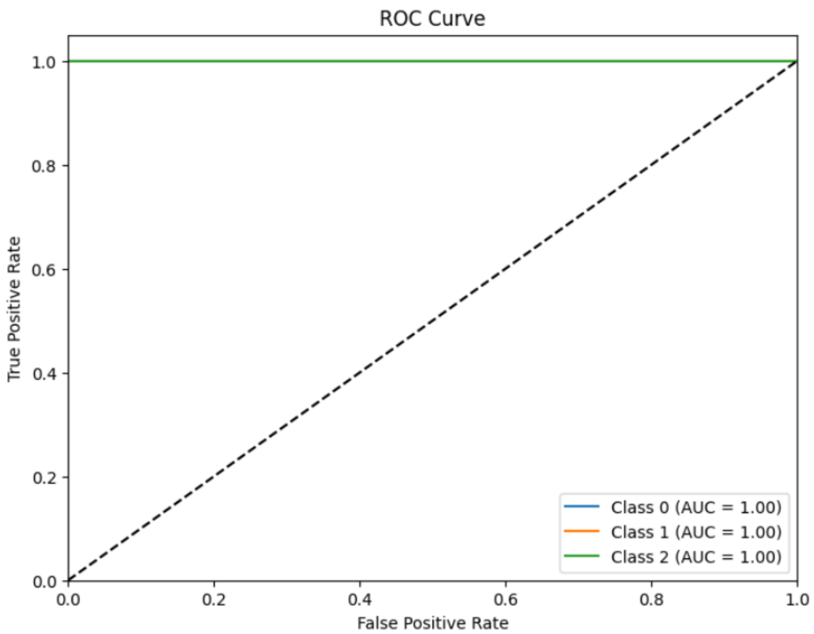
```
Epoch 35/50
14/14 [==============================] - 0s 9ms/step - loss: 0.0596 - accuracy: 0.9815 - val_loss: 0.3140 - val_accuracy: 0.9167
Epoch 36/50
14/14 [==============================] - 0s 17ms/step - loss: 0.0554 - accuracy: 0.9815 - val_loss: 0.2940 - val_accuracy: 0.9167
Epoch 37/50
14/14 [==============================] - 0s 15ms/step - loss: 0.0563 - accuracy: 0.9722 - val_loss: 0.2799 - val_accuracy: 0.9167
Epoch 38/50
14/14 [==============================] - 0s 10ms/step - loss: 0.0562 - accuracy: 0.9815 - val_loss: 0.3029 - val_accuracy: 0.9167
Epoch 39/50
14/14 [==============================] - 0s 7ms/step - loss: 0.0561 - accuracy: 0.9815 - val_loss: 0.2757 - val_accuracy: 0.9167
Epoch 40/50
14/14 [==============================] - 0s 9ms/step - loss: 0.0540 - accuracy: 0.9815 - val_loss: 0.2935 - val_accuracy: 0.9167
Epoch 41/50
14/14 [==============================] - 0s 8ms/step - loss: 0.0546 - accuracy: 0.9815 - val_loss: 0.2927 - val_accuracy: 0.9167
Epoch 42/50
14/14 [==============================] - 0s 8ms/step - loss: 0.0552 - accuracy: 0.9815 - val_loss: 0.2914 - val_accuracy: 0.9167
Epoch 43/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0547 - accuracy: 0.9815 - val_loss: 0.2689 - val_accuracy: 0.9167
Epoch 44/50
14/14 [==============================] - 0s 8ms/step - loss: 0.0529 - accuracy: 0.9815 - val_loss: 0.2807 - val_accuracy: 0.9167
Epoch 45/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0559 - accuracy: 0.9815 - val_loss: 0.2743 - val_accuracy: 0.9167
Epoch 46/50
14/14 [==============================] - 0s 15ms/step - loss: 0.0559 - accuracy: 0.9722 - val_loss: 0.3146 - val_accuracy: 0.9167
Epoch 47/50
14/14 [==============================] - 0s 14ms/step - loss: 0.0527 - accuracy: 0.9815 - val_loss: 0.2849 - val_accuracy: 0.9167
Epoch 48/50
14/14 [==============================] - 0s 15ms/step - loss: 0.0527 - accuracy: 0.9815 - val_loss: 0.2900 - val_accuracy: 0.9167
Epoch 49/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0518 - accuracy: 0.9815 - val_loss: 0.2997 - val_accuracy: 0.9167
Epoch 50/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0525 - accuracy: 0.9815 - val_loss: 0.2937 - val_accuracy: 0.9167
```

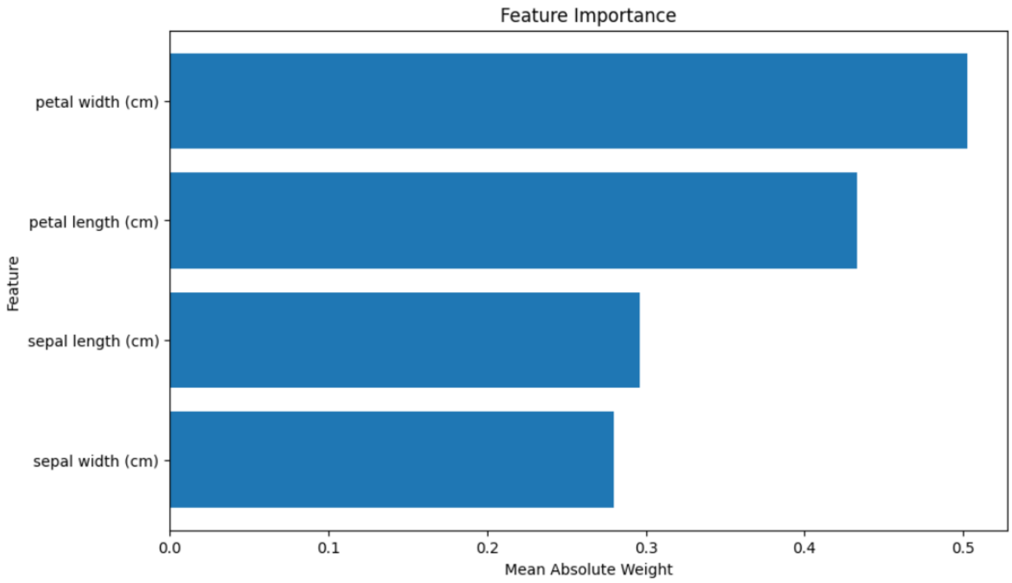


```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score
y_test_one_hot = label_binarize(y_test, classes=[0, 1, 2])
y_probs = model.predict(X_test_scaled)
fpr = dict()
tpr = dict()
roc_auc = 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()
```

```
1/1 [==============================] - 0s 31ms/step
```




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