Neural Networks and Deep Learning
Assignment-7

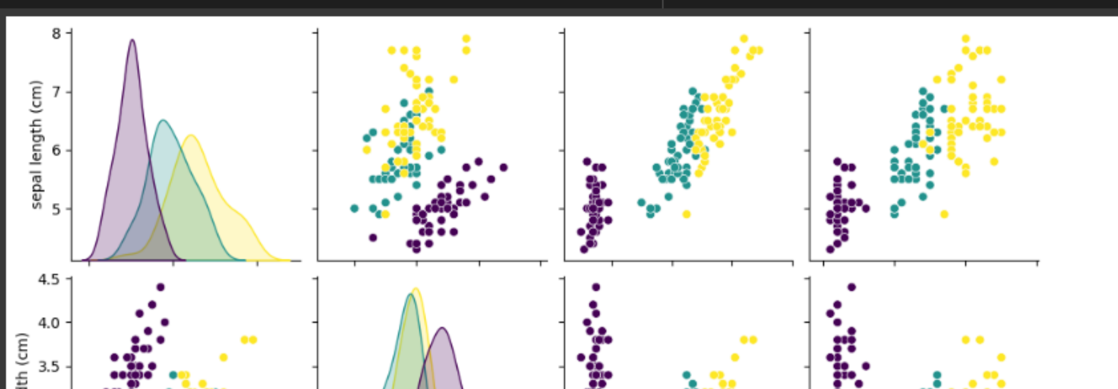
Name :Bhanu chand Garikapati
Student id :-700748274

Github Link:- https://github.com/Bhanu5423/neural_assignments

```
# 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. (module) linear_model 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
```

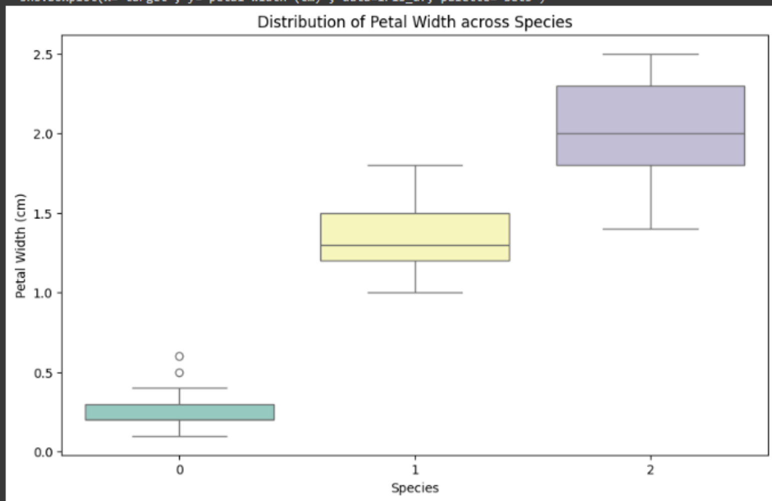
```
# Create at least two more visualizations using matplotlib (Other than provided in the source file)
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
sns.pairplot(iris_df, hue='target', 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()
```



```
<ipython-input-5-bb3dcd4abe5b>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

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




```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
```

```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  ndarray: iris.target  split
from sklearn.prepr                        r
iris = load_iris()  ndarray with shape (150,)
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 23/50
14/14 [==============================] - 0s 5ms/step - loss: 0.3139 - accuracy: 0.8889 - val_loss: 0.4355 - val_accuracy: 0.9167
Epoch 24/50
14/14 [==============================] - 0s 5ms/step - loss: 0.3022 - accuracy: 0.8889 - val_loss: 0.4269 - val_accuracy: 0.9167
Epoch 25/50
14/14 [==============================] - 0s 5ms/step - loss: 0.2902 - accuracy: 0.8889 - val_loss: 0.4029 - val_accuracy: 0.9167
Epoch 26/50
14/14 [==============================] - 0s 4ms/step - loss: 0.2752 - accuracy: 0.8981 - val_loss: 0.4038 - val_accuracy: 0.9167
Epoch 27/50
```

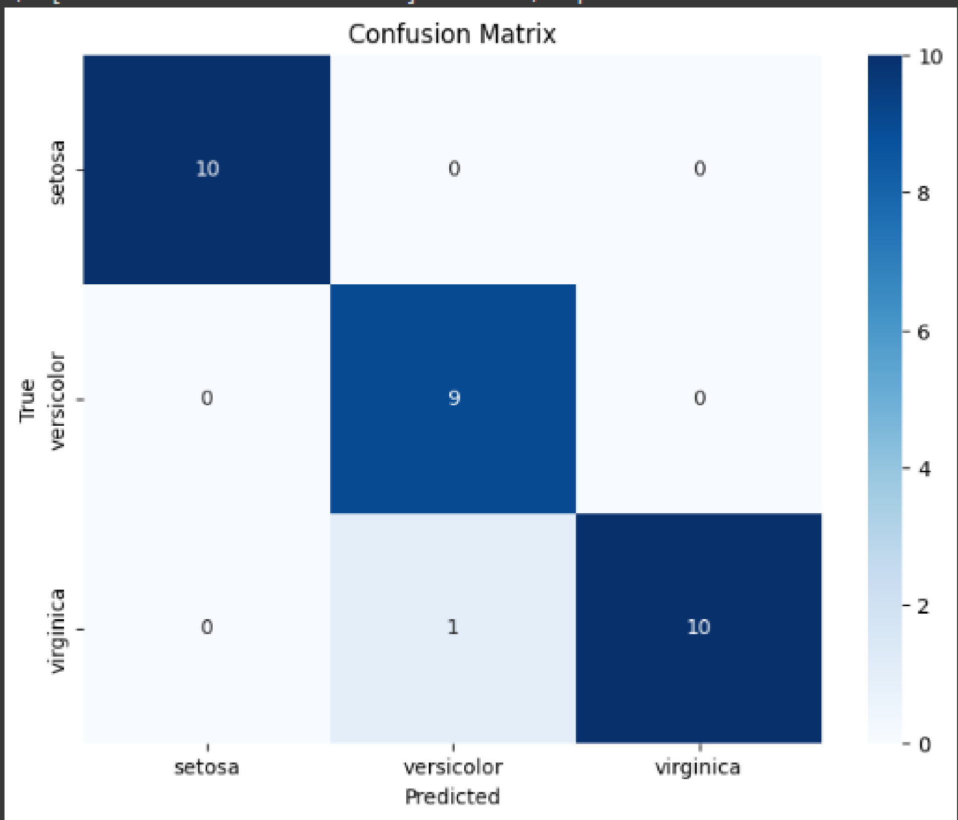
```
Epoch 39/50
14/14 [==============================] - 0s 5ms/step - loss: 0.1541 - accuracy: 0.9444 - val_loss: 0.2417 - val_accuracy: 0.9167
Epoch 40/50
14/14 [==============================] - 0s 5ms/step - loss: 0.1471 - accuracy: 0.9444 - val_loss: 0.2504 - val_accuracy: 0.9167
Epoch 41/50
14/14 [==============================] - 0s 4ms/step - loss: 0.1410 - accuracy: 0.9630 - val_loss: 0.2483 - val_accuracy: 0.9167
Epoch 42/50
14/14 [==============================] - 0s 4ms/step - loss: 0.1346 - accuracy: 0.9537 - val_loss: 0.2231 - val_accuracy: 0.9167
Epoch 43/50
14/14 [==============================] - 0s 4ms/step - loss: 0.1300 - accuracy: 0.9537 - val_loss: 0.2155 - val_accuracy: 0.9167
Epoch 44/50
14/14 [==============================] - 0s 4ms/step - loss: 0.1241 - accuracy: 0.9537 - val_loss: 0.2142 - val_accuracy: 0.9167
Epoch 45/50
14/14 [==============================] - 0s 6ms/step - loss: 0.1196 - accuracy: 0.9630 - val_loss: 0.2180 - val_accuracy: 0.9167
Epoch 46/50
14/14 [==============================] - 0s 4ms/step - loss: 0.1154 - accuracy: 0.9537 - val_loss: 0.1986 - val_accuracy: 0.9167
Epoch 47/50
14/14 [==============================] - 0s 5ms/step - loss: 0.1096 - accuracy: 0.9630 - val_loss: 0.2008 - val_accuracy: 0.9167
Epoch 48/50
14/14 [==============================] - 0s 4ms/step - loss: 0.1069 - accuracy: 0.9630 - val_loss: 0.1852 - val_accuracy: 0.9167
Epoch 49/50
14/14 [==============================] - 0s 4ms/step - loss: 0.1032 - accuracy: 0.9722 - val_loss: 0.1933 - val_accuracy: 0.9167
Epoch 50/50
14/14 [==============================] - 0s 5ms/step - loss: 0.0982 - accuracy: 0.9722 - val_loss: 0.1808 - val_accuracy: 0.9167
1/1 [==============================] - 0s 27ms/step - loss: 0.1255 - accuracy: 0.9667
Accuracy of Modified Neural Network: 0.9666666388511658
```

```python
# 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 format is considered legacy. We recommend usin
  saving_api.save_model(
1/1 [==============================] - 0s 105ms/step
Predictions:
[[6.5732480e-04 8.4889555e-01 1.5044701e-01]
 [9.9981046e-01 1.8951908e-04 2.7959034e-08]
 [1.6832097e-16 2.4725811e-04 9.9975276e-01]
 [2.8943064e-04 7.1446979e-01 2.8524089e-01]
 [3.2435542e-05 6.4441466e-01 3.5555279e-01]]
```

```python
# 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 88ms/step
```

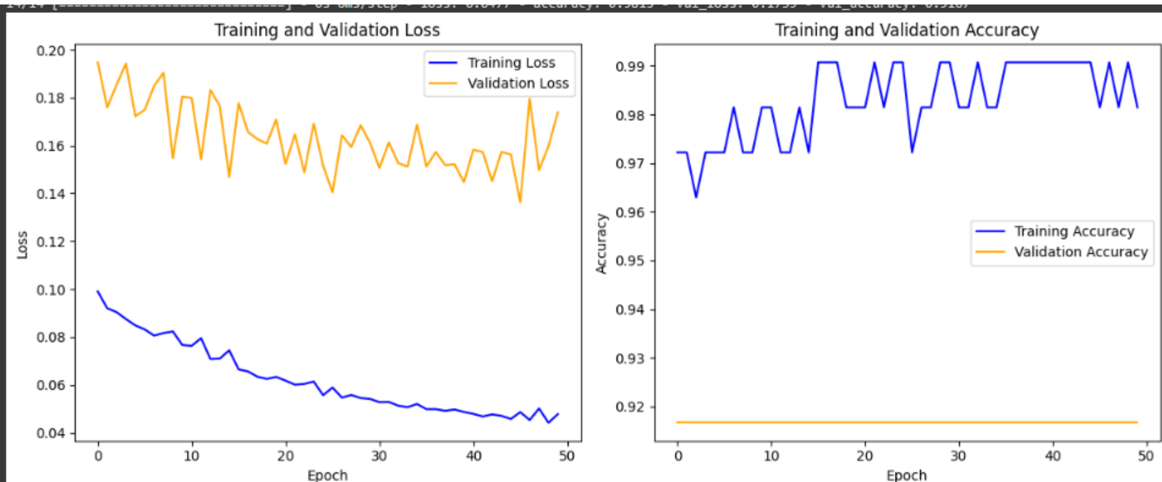


```python
# Training and testing Loss and accuracy plots in one plot using subplot command and history object
history = model.fit(X_train_scaled, y_train, epochs=50, batch_size=8, verbose=1, validation_split=0.1)
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 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()
```

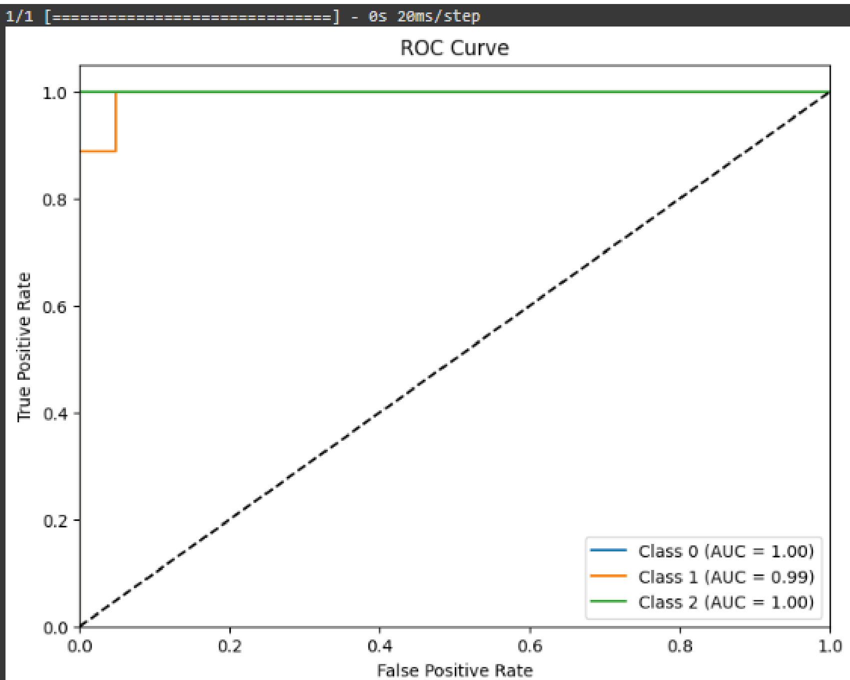
```
14/14 [==============================] - 0s 20ms/step - loss: 0.0498 - accuracy: 0.9907 - val_loss: 0.1512 - val_accuracy: 0.9167
Epoch 37/50
14/14 [==============================] - 0s 13ms/step - loss: 0.0498 - accuracy: 0.9907 - val_loss: 0.1573 - val_accuracy: 0.9167
Epoch 38/50
14/14 [==============================] - 0s 16ms/step - loss: 0.0491 - accuracy: 0.9907 - val_loss: 0.1518 - val_accuracy: 0.9167
Epoch 39/50
14/14 [==============================] - 0s 25ms/step - loss: 0.0497 - accuracy: 0.9907 - val_loss: 0.1522 - val_accuracy: 0.9167
Epoch 40/50
14/14 [==============================] - 0s 21ms/step - loss: 0.0487 - accuracy: 0.9907 - val_loss: 0.1447 - val_accuracy: 0.9167
Epoch 41/50
14/14 [==============================] - 0s 14ms/step - loss: 0.0479 - accuracy: 0.9907 - val_loss: 0.1583 - val_accuracy: 0.9167
Epoch 42/50
14/14 [==============================] - 0s 23ms/step - loss: 0.0468 - accuracy: 0.9907 - val_loss: 0.1573 - val_accuracy: 0.9167
Epoch 43/50
14/14 [==============================] - 0s 26ms/step - loss: 0.0476 - accuracy: 0.9907 - val_loss: 0.1451 - val_accuracy: 0.9167
Epoch 44/50
14/14 [==============================] - 0s 10ms/step - loss: 0.0470 - accuracy: 0.9907 - val_loss: 0.1574 - val_accuracy: 0.9167
Epoch 45/50
14/14 [==============================] - 0s 8ms/step - loss: 0.0457 - accuracy: 0.9907 - val_loss: 0.1563 - val_accuracy: 0.9167
Epoch 46/50
14/14 [==============================] - 0s 7ms/step - loss: 0.0486 - accuracy: 0.9815 - val_loss: 0.1363 - val_accuracy: 0.9167
Epoch 47/50
14/14 [==============================] - 0s 7ms/step - loss: 0.0453 - accuracy: 0.9907 - val_loss: 0.1795 - val_accuracy: 0.9167
Epoch 48/50
14/14 [==============================] - 0s 7ms/step - loss: 0.0501 - accuracy: 0.9815 - val_loss: 0.1498 - val_accuracy: 0.9167
Epoch 49/50
14/14 [==============================] - 0s 4ms/step - loss: 0.0442 - accuracy: 0.9907 - val_loss: 0.1597 - val_accuracy: 0.9167
Epoch 50/50
14/14 [==============================] - 0s 6ms/step - loss: 0.0477 - accuracy: 0.9815 - val_loss: 0.1739 - val_accuracy: 0.9167
```



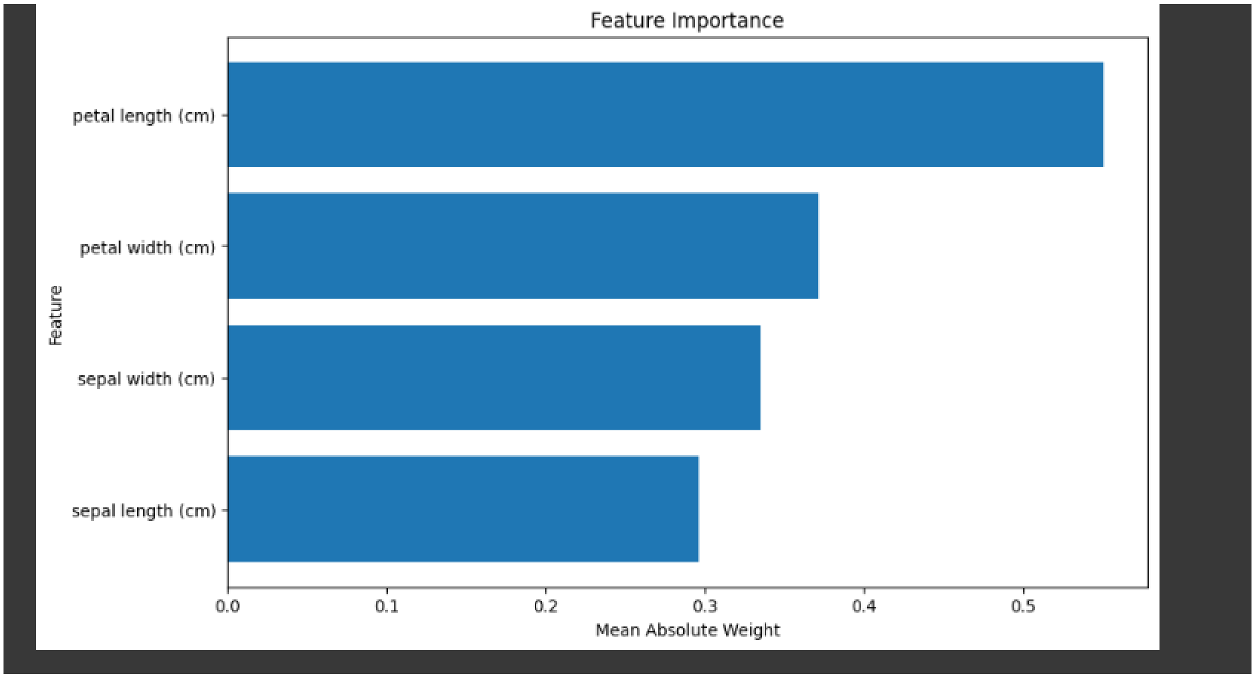
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score
y_test_one_hot = label_binarize(y_test, classes=[0, 1, 2])
y_probs = model.predict(X_test_scaled)
fpr = dict()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

```
1/1 [==============================] - 0s 20ms/step
```



Feature Importance


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