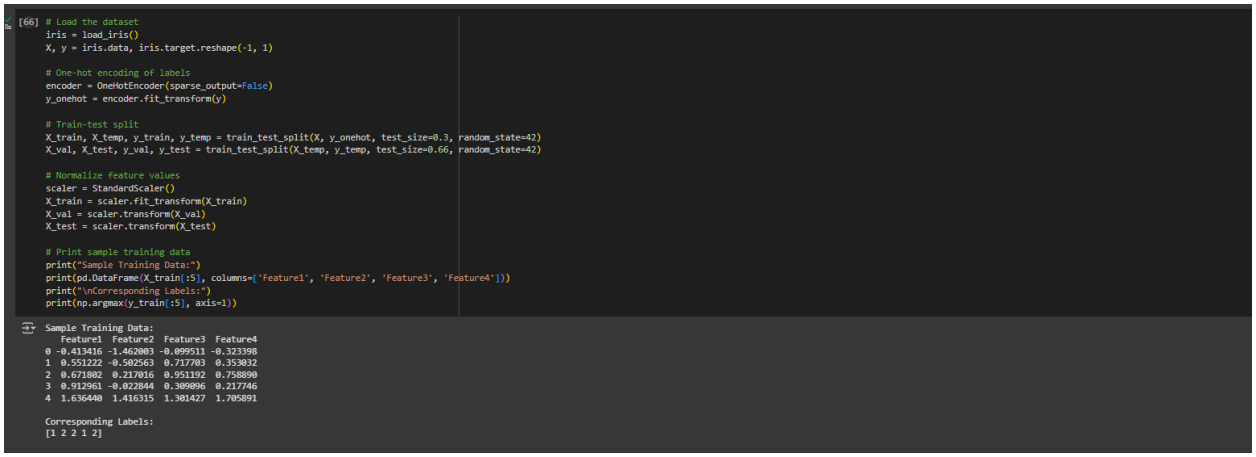
ASSIGNMENT 6

Om Gupta - 22110174 Pranav Somase - 22110200

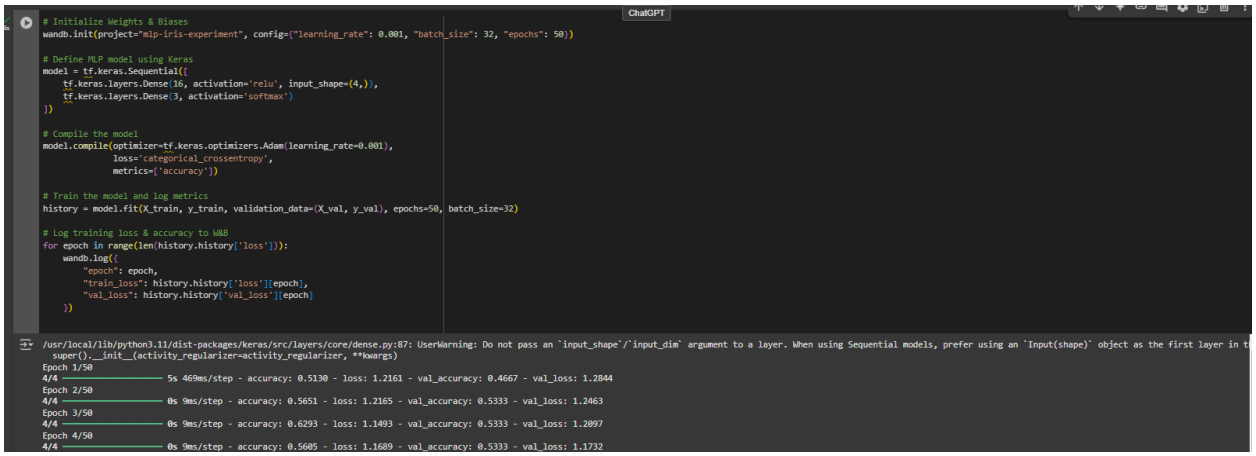
GITHUB - https://github.com/hiomgupta/STTAi_L6

Data Preprocessing



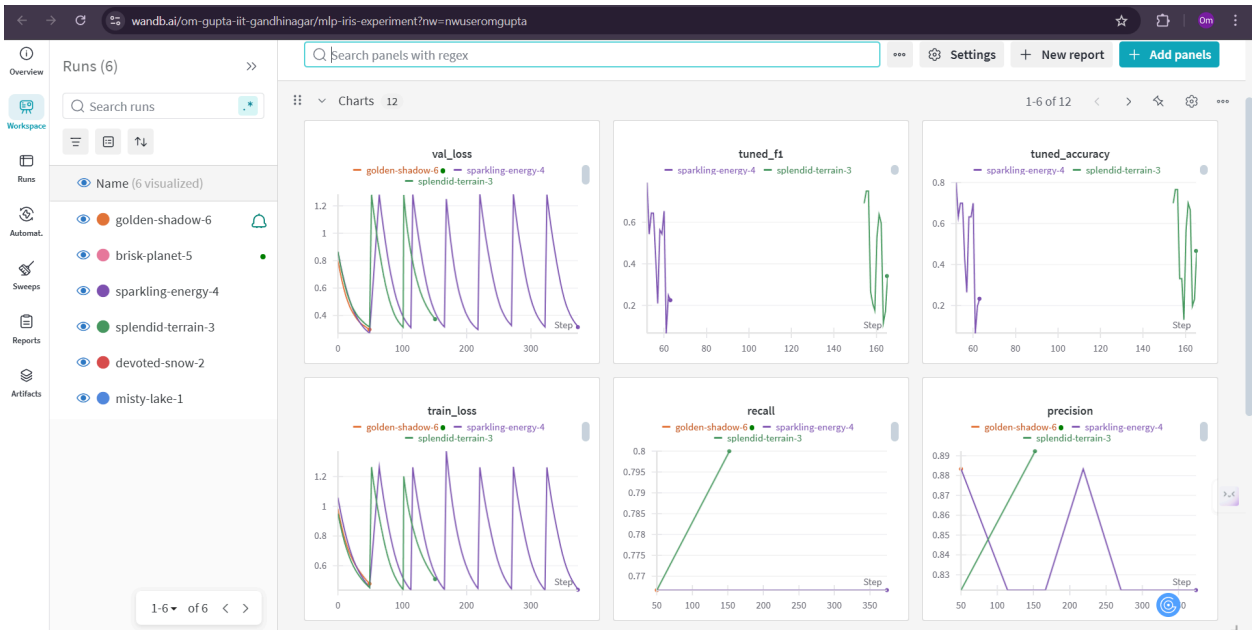
```
[66] # Load the dataset
     iris = load_iris()
     X, y = iris.data, iris.target.reshape(-1, 1)

     # One-hot encoding of labels
     encoder = OneHotEncoder(sparse_output=False)
     y_onehot = encoder.fit_transform(y)

     # Train-test split
     X_train, X_temp, y_train, y_temp = train_test_split(X, y_onehot, test_size=0.3, random_state=42)
     X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.66, random_state=42)

     # Normalize feature values
     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_val = scaler.transform(X_val)
     X_test = scaler.transform(X_test)

     # Print sample training data
     print("Sample Training Data:")
     print(pd.DataFrame(X_train[:5], columns=['Feature1', 'Feature2', 'Feature3', 'Feature4']))
     print("\nCorresponding Labels:")
     print(np.argmax(y_train[:5], axis=1))

Sample Training Data:
    Feature1  Feature2  Feature3  Feature4
0 -0.413416 -1.462003 -0.099511 -0.323398
1  0.551222 -0.592563  0.717703  0.353032
2  0.671802  0.217016  0.951192  0.758890
3  0.912961 -0.022844  0.309096  0.217746
4  1.636440  1.416315  1.301427  1.705891

Corresponding Labels:
[1 2 2 1 2]
```

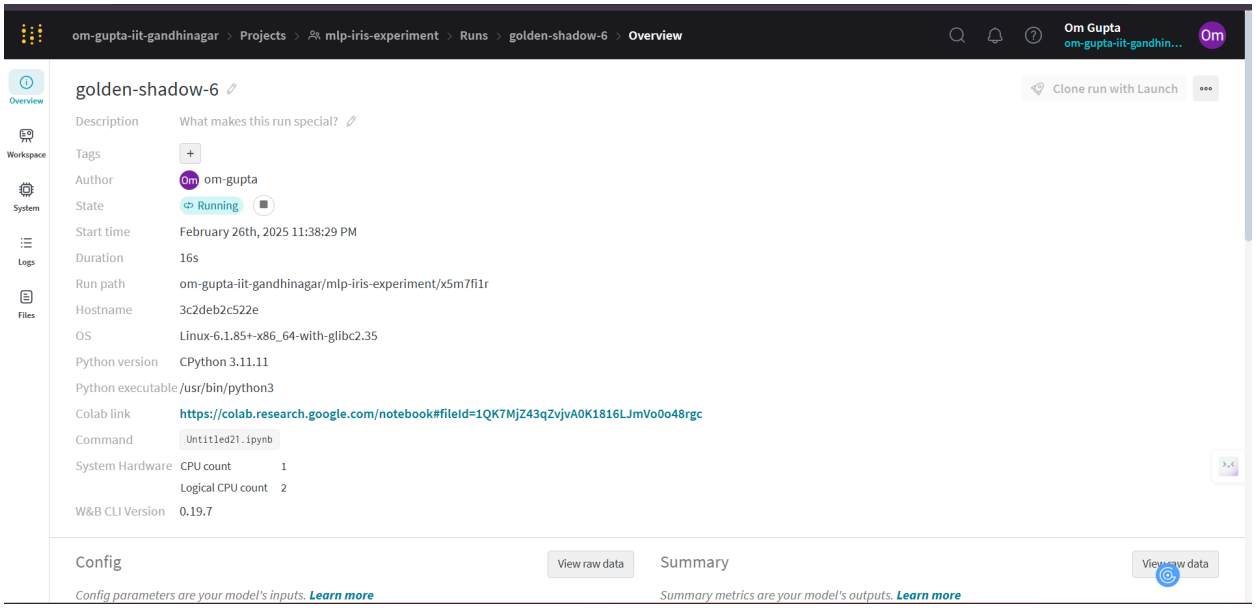
Model 1: Multi-Layer Perceptron (MLP)

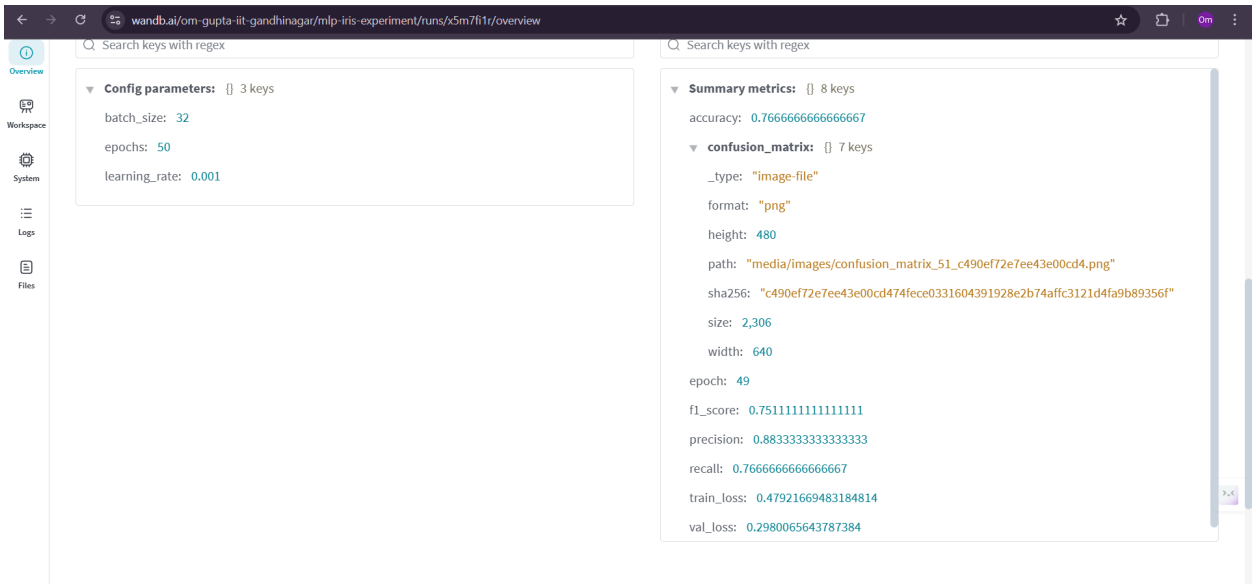


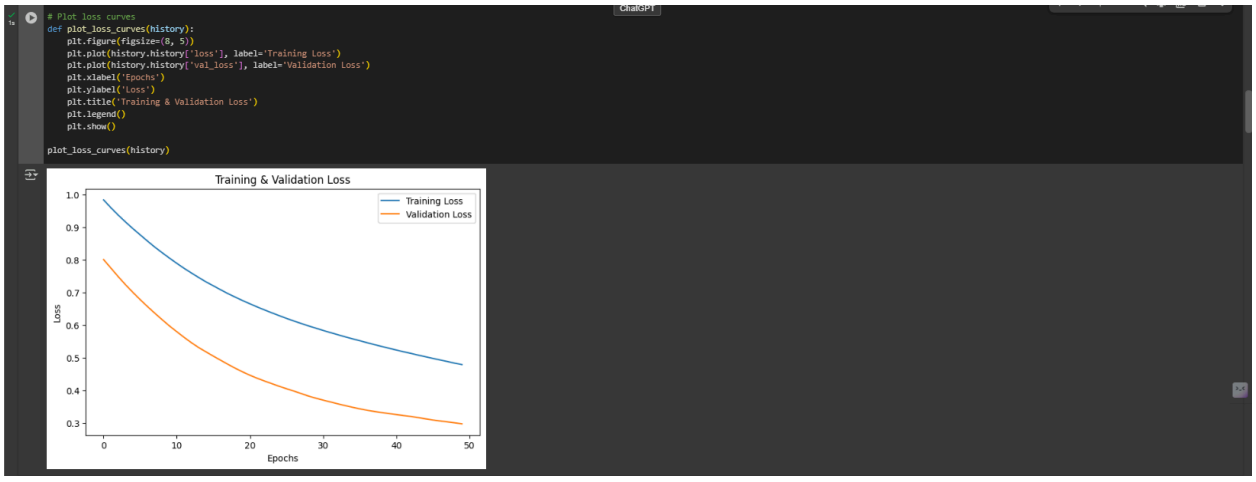
```
# Initialize Weights & Biases
wandb.init(project="mlp-iris-experiment", config={"learning_rate": 0.001, "batch_size": 32, "epochs": 50})

# Define MLP model using Keras
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu', input_shape=(4,)),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model and log metrics
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50, batch_size=32)

# Log training loss & accuracy to W&B
for epoch in range(len(history.history['loss'])):
    wandb.log({
        "epoch": epoch,
        "train_loss": history.history['loss'][epoch],
        "val_loss": history.history['val_loss'][epoch]
    })

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in t
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 469ms/step - accuracy: 0.5130 - loss: 1.2161 - val_accuracy: 0.4667 - val_loss: 1.2844
Epoch 2/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5651 - loss: 1.2165 - val_accuracy: 0.5333 - val_loss: 1.2463
Epoch 3/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6293 - loss: 1.1493 - val_accuracy: 0.5333 - val_loss: 1.2097
Epoch 4/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5605 - loss: 1.1689 - val_accuracy: 0.5333 - val_loss: 1.1732
```

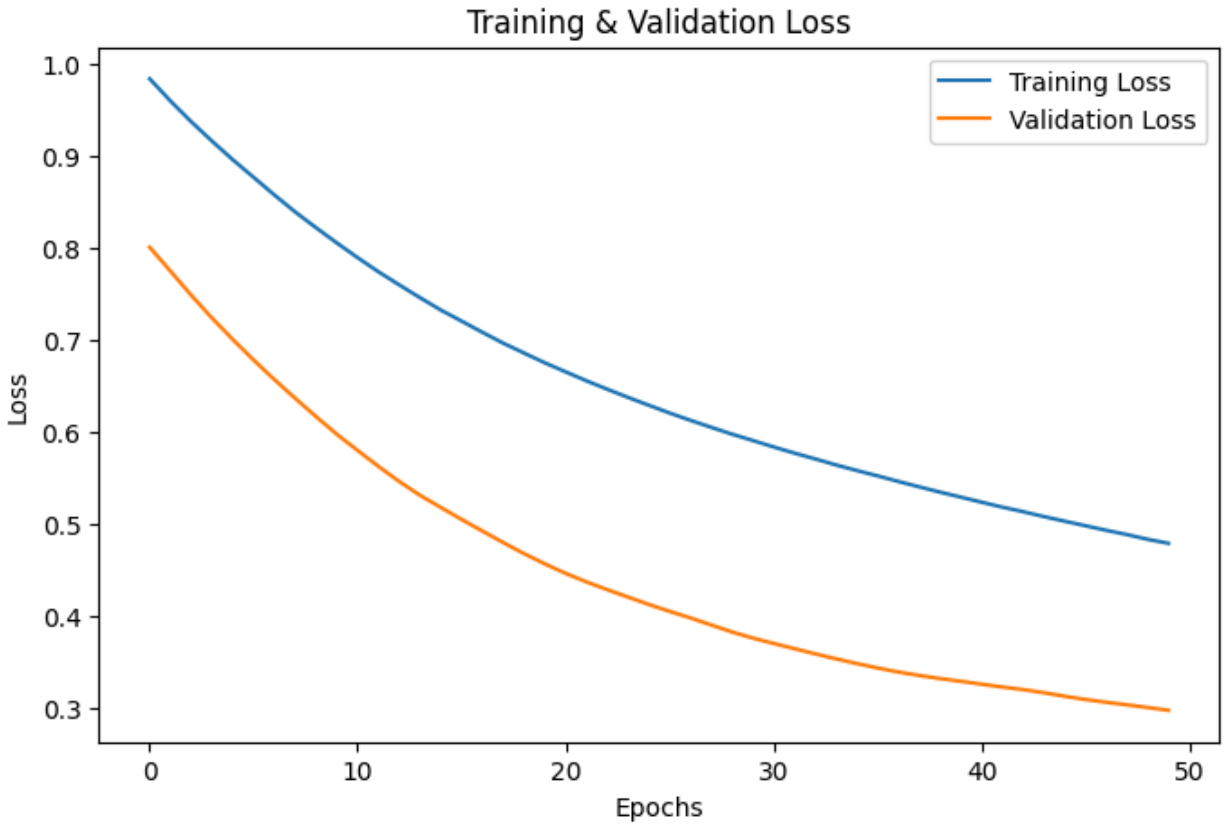
```
Epoch 24/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7756 - loss: 0.7006 - val_accuracy: 0.8000 - val_loss: 0.6151
Epoch 25/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8051 - loss: 0.6773 - val_accuracy: 0.8000 - val_loss: 0.5943
Epoch 26/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8138 - loss: 0.6694 - val_accuracy: 0.8000 - val_loss: 0.5741
Epoch 27/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8179 - loss: 0.6334 - val_accuracy: 0.8667 - val_loss: 0.5551
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8144 - loss: 0.6551 - val_accuracy: 0.8667 - val_loss: 0.5374
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8089 - loss: 0.6268 - val_accuracy: 0.8667 - val_loss: 0.5212
Epoch 30/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8457 - loss: 0.6243 - val_accuracy: 0.8667 - val_loss: 0.5048
Epoch 31/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8255 - loss: 0.6138 - val_accuracy: 0.8667 - val_loss: 0.4897
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8238 - loss: 0.5811 - val_accuracy: 0.8667 - val_loss: 0.4749
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8380 - loss: 0.5757 - val_accuracy: 0.9333 - val_loss: 0.4611
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8418 - loss: 0.5766 - val_accuracy: 0.9333 - val_loss: 0.4492
Epoch 35/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8481 - loss: 0.5632 - val_accuracy: 0.9333 - val_loss: 0.4383
Epoch 36/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8148 - loss: 0.5621 - val_accuracy: 0.9333 - val_loss: 0.4280
Epoch 37/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8252 - loss: 0.5517 - val_accuracy: 0.9333 - val_loss: 0.4179
Epoch 38/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8273 - loss: 0.5237 - val_accuracy: 0.9333 - val_loss: 0.4087
Epoch 39/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8477 - loss: 0.4947 - val_accuracy: 0.9333 - val_loss: 0.3994
Epoch 40/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8342 - loss: 0.5384 - val_accuracy: 0.9333 - val_loss: 0.3907
Epoch 41/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8332 - loss: 0.5234 - val_accuracy: 0.9333 - val_loss: 0.3825
Epoch 42/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8557 - loss: 0.4735 - val_accuracy: 0.9333 - val_loss: 0.3747
Epoch 43/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8276 - loss: 0.5041 - val_accuracy: 0.9333 - val_loss: 0.3675
Epoch 44/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8359 - loss: 0.4969 - val_accuracy: 0.9333 - val_loss: 0.3609
Epoch 45/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8568 - loss: 0.4664 - val_accuracy: 0.9333 - val_loss: 0.3547
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8401 - loss: 0.4790 - val_accuracy: 0.9333 - val_loss: 0.3491
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8599 - loss: 0.4610 - val_accuracy: 0.9333 - val_loss: 0.3436
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8411 - loss: 0.4631 - val_accuracy: 0.9333 - val_loss: 0.3380
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8297 - loss: 0.4414 - val_accuracy: 0.9333 - val_loss: 0.3327
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8401 - loss: 0.4482 - val_accuracy: 0.9333 - val_loss: 0.3269
```



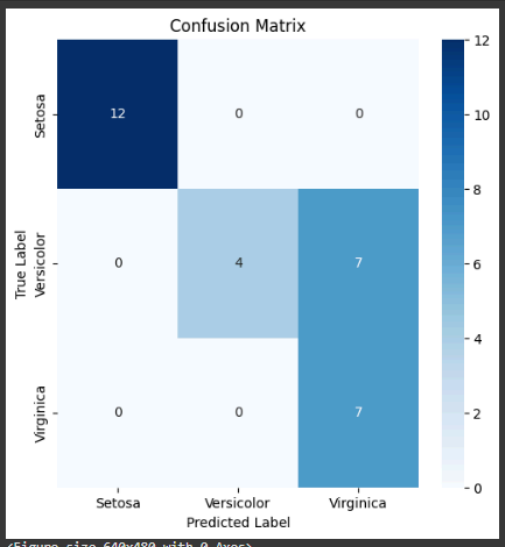

om-gupta-iit-gandhinagar > Projects > mlp-iris-experiment > Runs > golden-shadow-6 > Overview
Om Gupta
om-gupta-iit-gandhin...
Overview
Workspace
System
Logs
Files
golden-shadow-6
Clone run with Launch
Description
What makes this run special?
Tags
Author
om-gupta
State
Running
Start time
February 26th, 2025 11:38:29 PM
Duration
16s
Run path
om-gupta-iit-gandhinagar/mlp-iris-experiment/x5m7fi1r
Hostname
3c2deb2c522e
OS
Linux-6.1.85+-x86_64-with-glibc2.35
Python version
CPython 3.11.11
Python executable
/usr/bin/python3
Colab link
https://colab.research.google.com/notebook#fileId=1QK7MjZ43qZvjvA0K1816LJmVo0o48rgc
Command
Untitled21.ipynb
System Hardware
CPU count 1
Logical CPU count 2
W&B CLI Version
0.19.7
Config
View raw data
Config parameters are your model's inputs. Learn more
Summary
View raw data
Summary metrics are your model's outputs. Learn more


wandb.ai/om-gupta-iit-gandhinagar/mlp-iris-experiment/runs/x5m7fi1r/overview
Search keys with regex
Config parameters: {} 3 keys
batch_size: 32
epochs: 50
learning_rate: 0.001
Search keys with regex
Summary metrics: {} 8 keys
accuracy: 0.7666666666666667
confusion_matrix: {} 7 keys
_type: "image-file"
format: "png"
height: 480
path: "media/images/confusion_matrix_51_c490ef72e7ee43e00cd4.png"
sha256: "c490ef72e7ee43e00cd474fece0331604391928e2b74affc3121d4fa9b89356f"
size: 2,306
width: 640
epoch: 49
f1_score: 0.7511111111111111
precision: 0.8833333333333333
recall: 0.7666666666666667
train_loss: 0.47921669483184814
val_loss: 0.2980065643787384

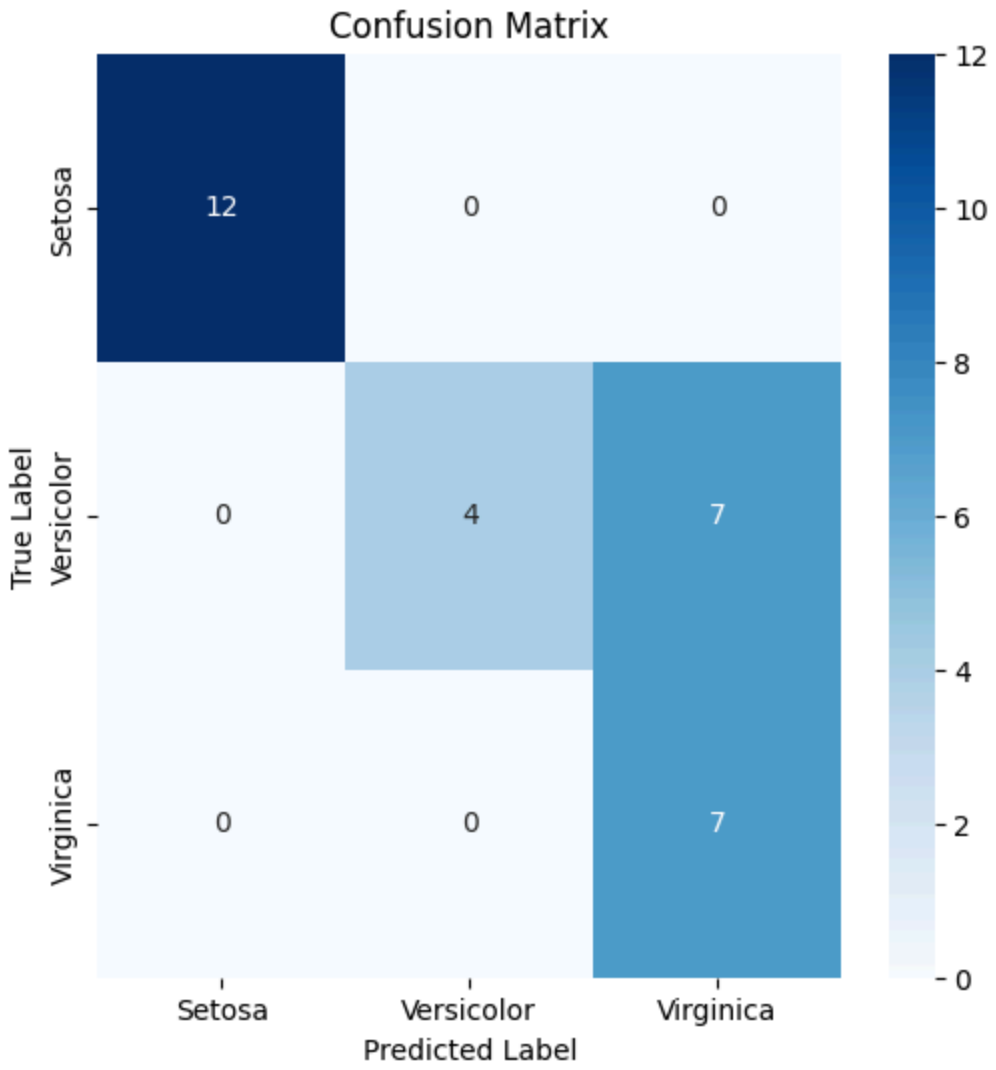

ChatGPT
# Plot loss curves
def plot_loss_curves(history):
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training & Validation Loss')
plt.legend()
plt.show()
plot_loss_curves(history)
Training & Validation Loss
Training Loss
Validation Loss
Loss
Epochs


Training & Validation Loss
Training Loss
Validation Loss
Loss
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50

```python
# Improved Confusion Matrix Visualization
def plot_confusion_matrix(y_true, y_pred, class_names):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.show()
    wandb.log({"confusion_matrix": wandb.Image(plt)})

plot_confusion_matrix(y_true, y_pred, class_names=['Setosa', 'Versicolor', 'Virginica'])
```




```
<Figure size 640x480 with 0 Axes>
```



Model 2: AutoGluon (AutoML Approach)

```
# Hyperparameter tuning (manual search)
batch_sizes = [2, 4]
learning_rates = [1e-3, 1e-5]
epochs = [1, 3, 5]
results = []

for batch in batch_sizes:
    for lr in learning_rates:
        for epoch in epochs:
            model = tf.keras.Sequential([
                tf.keras.layers.Dense(16, activation='relu', input_shape=(4,)),
                tf.keras.layers.Dense(3, activation='softmax')
            ])
            model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
                          loss='categorical_crossentropy',
                          metrics=['accuracy'])
            model.fit(X_train, y_train, epochs=epoch, batch_size=batch, verbose=0)
            y_pred = np.argmax(model.predict(X_test), axis=1)
            accuracy = accuracy_score(y_true, y_pred)
            f1 = f1_score(y_true, y_pred, average='weighted')

            # Append results
            model_name = f"Batch {batch}, LR {lr}, Epoch {epoch}"
            results.append([model_name, batch, lr, epoch, accuracy, f1])

# Convert results to DataFrame
df_results = pd.DataFrame(results, columns=['Model', 'Batch Size', 'Learning Rate', 'Epochs', 'Accuracy', 'F1 Score'])
print(df_results)
```

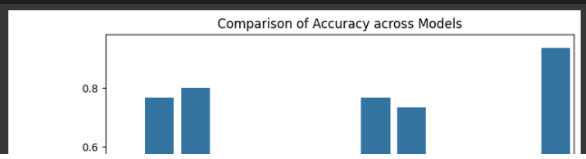
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
```

```
# AutoGluon model
train_data = pd.DataFrame(X_train, columns=['Feature1', 'Feature2', 'Feature3', 'Feature4'])
train_data['Label'] = np.argmax(y_train, axis=1)

test_data = pd.DataFrame(X_test, columns=['Feature1', 'Feature2', 'Feature3', 'Feature4'])
test_data['Label'] = np.argmax(y_test, axis=1)

# Run AutoGluon hyperparameter search
predictor = TabularPredictor(label="Label").fit(train_data, presets="optimize_for_deployment")
```

```
        0.0s     = Validation runtime
Fitting model: LightGBM ...
        0.8571   = Validation score   (accuracy)
        0.66s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: RandomForestGini ...
        0.8571   = Validation score   (accuracy)
        0.57s    = Training   runtime
        0.06s    = Validation runtime
```

```
# Evaluate AutoGluon model
predictions = predictor.predict(test_data)
accuracy = accuracy_score(test_data["Label"], predictions)
f1 = f1_score(test_data["Label"], predictions, average='weighted')

# Store AutoGluon results
auto_results = pd.DataFrame({"Model": ["AutoGluon"], "Accuracy": [accuracy], "F1 Score": [f1]})
df_results = pd.concat([df_results, auto_results])
```

```
# Visualize accuracy across models
plt.figure(figsize=(8, 5))
sns.barplot(x="Model", y="Accuracy", data=df_results)
plt.title("Comparison of Accuracy across Models")
plt.xticks(rotation=45, ha='right')
plt.show()

print(f"AutoGluon Model Accuracy: {accuracy}")
print(f"AutoGluon Model F1 Score: {f1}")

# Print final model performance
print("Model Performance Results:")
print(df_results)
```



AutoGluon Model: The AutoGluon model performed exceptionally well, achieving an accuracy of 93.33% and an F1 Score of 93.42%, making it the best model in this experiment.
Hyperparameter Tuning:

- Learning Rate (LR): Models with a learning rate of 0.001 consistently performed better, with the highest accuracy seen at Epoch 5.
- Batch Size: The batch size of 2 produced better results compared to 4, with smaller batch sizes enabling quicker convergence during training.
- Epochs: Training for 5 epochs generally resulted in the best performance. Shorter epochs (1-3) tended to result in underfitting.

Performance Trends:

- Accuracy: The highest accuracy of 83.33% was achieved using batch size 2 and learning rate 0.001 over 5 epochs.
- F1 Score: Models that achieved the highest accuracy also exhibited the highest F1 Scores, confirming a balanced performance across both metrics.