SV30803

VISI BERKOMPUTER DAN APLIKASI

ASSIGNMENT 1:

IMAGE CLASSIFICATION

LECTURER: PROFESSOR DR. ABDULLAH BIN BADE

No.	Name	Matrix
1	MOHAMAD ADLI ZILIKHRAM BIN SAHARUDIN	BS22110469

TABLE OF CONTENTS

1.0 SYSTEM INTRODUCTION

This project involves developing an interactive image classification system implemented in Python using PyQt5 for the graphical user interface and scikit-learn for machine learning functionalities. The system supports dual-mode classification: species classification of Iris flowers using structured CSV data and rose color classification from image data. The application allows users to load datasets, preprocess inputs, and train Support Vector Machine (SVM) models with linear kernels. It includes real-time model performance evaluation using metrics such as precision, recall, F1-score, support, and confusion matrix visualization. The system also supports prediction tasks via direct user input (for Iris features) or image upload (for rose classification), with probability-based confidence bar charts for interpretability. Image data is processed using OpenCV for resizing and grayscale feature extraction, and the UI dynamically updates to reflect prediction outcomes and training results, offering a comprehensive platform for practical experimentation with supervised learning workflows.

2.0 SYSTEM ARCHITECTURE

1. User Interface (GUI) Layer

Class: MainWindow(QWidget)
Purpose: Handles all GUI layout, input, and user actions.

Relevant Functions:

- __init__(self): Initializes all widgets, layouts, and connects signals.
- btn_load_csv.clicked.connect(self.load_csv_dataset): Loads Iris CSV file.
- btn_load_rose.clicked.connect(self.load_rose_dataset): Loads image folder for Rose dataset.
- btn_train_both.clicked.connect(self.train_both_models): Triggers model training.
- btn_predict_iris.clicked.connect(self.predict_iris_class): Predicts Iris flower class from form inputs.
- btn_verify_rose.clicked.connect(self.verify_rose_image): Classifies loaded rose image.

🔹 2. Data Loading and Preprocessing Module

Functions:

- load_csv_dataset(self):
    - Uses pandas.read_csv() to read the Iris dataset.
    - Selects specific columns: ["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"].
    - Uses train_test_split() for 80:20 data division.
- load_rose_dataset(self):
    - Uses cv2.imread() to read images.
    - Resizes and converts images with cv2.resize() and cv2.cvtColor().
    - Flattens grayscale images to feature vectors with .flatten().

Progress bars:

- Updated using self.iris_progress_bar.setValue() and self.rose_progress_bar.setValue().

3. Model Training Module

Functions:

- train_model(self): Trains SVM on Iris dataset.
- train_rose_model(self): Trains SVM on Rose dataset.
- train_both_models(self): Encodes labels using LabelEncoder(), trains both Iris and Rose models using SVC (kernel='linear', probability=True), and stores predictions in self.model_results.

4. Prediction and Verification Module

Functions:

- predict_iris_class(self):
    - Takes input from QLineEdit fields, forms a NumPy array, and uses self.iris_model.predict() for classification.
    - Uses self.iris_model.predict_proba() for confidence values.
    - Displays prediction in self.iris_prediction_label.
- verify_rose_image(self):
    - Loads an image from disk.
    - Processes and flattens it.
    - Calls self.rose_model.predict() and predict_proba() to get predictions.
    - Updates self.rose_prediction_label.

5. Evaluation & Metrics Module

Function:

- evaluate_model(self):
    - Uses precision_recall_fscore_support() and confusion_matrix() from sklearn.metrics to compute evaluation metrics.

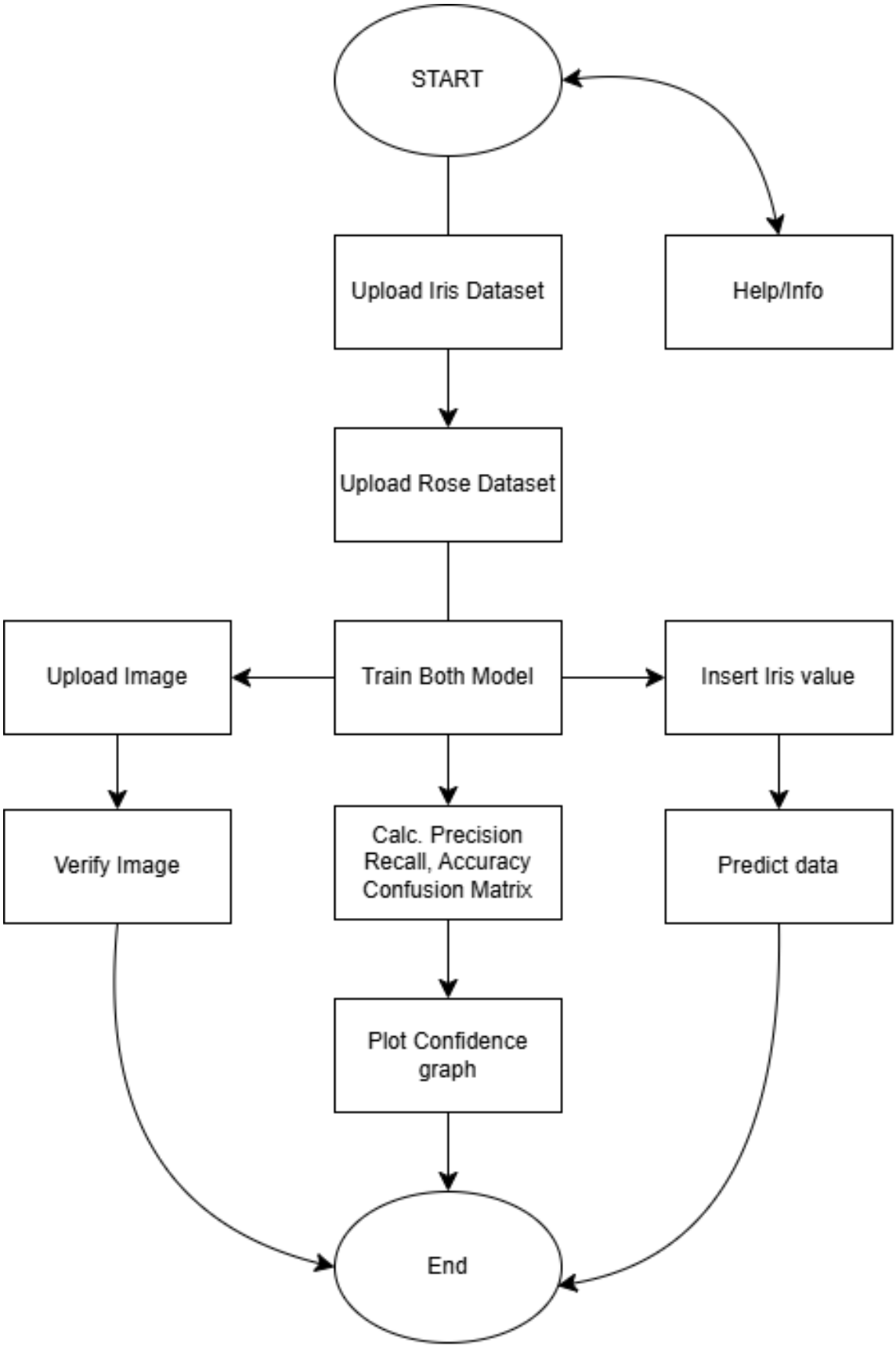
- Populates a QTableWidget (self.metrics_table) with precision, recall, F1-score, and support.
- Uses accuracy_score() for model accuracy.

6. Visualization Module

Functions:

- evaluate_model(self):
  - Uses matplotlib.pyplot.subplots() to create:
    - Confusion Matrix heatmaps.
    - Precision-Recall scatter plots.
  - Plots are rendered in PyQt using FigureCanvas.
- plot_confidence_bar(self, class_probs, class_names, canvas_label):
  - Draws horizontal bar charts using ax.barh() to show prediction confidence.
  - Displays them inside QLabel containers (self.iris_conf_canvas, self.rose_conf_canvas).

3.0 SYSTEM FLOWCHART



4.0 ALGORITHMS

Preparing Iris Dataset

Input:

```
file_path, _ = QFileDialog.getOpenFileName(self, "Open CSV", "", "CSV Files (*.csv)")
        if file_path:
            try:
                # Simulated loading steps (just visual effect)
                for i in range(1, 101, 20):
                    self.iris_progress_bar.setValue(i)
                    QApplication.processEvents()
                    QThread.msleep(50)  # Simulate delay for animation

                df = pd.read_csv(file_path)
                # Drop ID column if it exists
                if "Id" in df.columns:
                    df = df.drop("Id", axis=1)

                # Or use the actual feature names
                        X = df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]].values
                y = df["Species"].values

                self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
                    X, y, test_size=0.2, random_state=42
                )

                self.iris_progress_bar.setValue(100)
                self.iris_progress_bar.setFormat("Load complete")
            except Exception as e:
                self.iris_progress_bar.setValue(100)
                self.iris_progress_bar.setFormat("Failed to load")
        else:
            self.iris_progress_bar.setFormat("Load cancelled")
```

Preparing rose dataset

Input:

```
data_path = QFileDialog.getExistingDirectory(self, "Select Rose Image Folder")
        if not data_path:
            self.rose_progress_bar.setFormat("Load cancelled")
            return
```

```
        X, y = [], []
        files = [f for f in os.listdir(data_path) if f.lower().endswith((".png", ".jpg", ".jpeg"))]

        if not files:
            self.rose_progress_bar.setFormat("No valid images found")
            return

        total = len(files)
        for idx, file_name in enumerate(files):
            img_path = os.path.join(data_path, file_name)
            img = cv2.imread(img_path)
            if img is not None:
                img_resized = cv2.resize(img, (64, 64))
                gray = cv2.cvtColor(img_resized, cv2.COLOR_BGR2GRAY)
                features = gray.flatten()
                X.append(features)

                # Extract color label from filename
                label = file_name.lower().split()[0]
                y.append(label)

            # Update progress bar
            progress = int((idx + 1) / total * 80)
            self.rose_progress_bar.setValue(progress)
            QApplication.processEvents()

        if not X:
            self.rose_progress_bar.setFormat("No valid images found")
            return

        X = np.array(X)
        y = np.array(y)

        dummy_negatives = np.random.rand(len(X), X.shape[1]) * 255
        X = np.vstack((X, dummy_negatives))
        y = np.concatenate((y, np.zeros(len(dummy_negatives), dtype=int)))

        # Final loading step
            self.rose_X_train, self.rose_X_test, self.rose_y_train, self.rose_y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )
```

Training Model

For Iris:

- Encode labels (Species) using LabelEncoder
- Train SVM with linear kernel
- Store model and label encoder

For Rose:

- Encode rose color labels
- Train SVM with linear kernel
- Store model and label encoder

Store both predictions for evaluation

Input:

```
# Train Iris
        if hasattr(self, 'X_train') and hasattr(self, 'y_train'):
            # Encode target labels
            self.iris_label_encoder = LabelEncoder()
            y_train_encoded = self.iris_label_encoder.fit_transform(self.y_train)
            y_test_encoded = self.iris_label_encoder.transform(self.y_test)

            self.iris_model = SVC(kernel='linear', probability=True)
            self.iris_model.fit(self.X_train, y_train_encoded)

            y_pred = self.iris_model.predict(self.X_test)

            self.model_results.append({
                'name': 'Iris',
                'y_true': y_test_encoded,
                'y_pred': y_pred
            })

        # Train Rose
        if hasattr(self, 'rose_X_train') and hasattr(self, 'rose_y_train'):
            self.rose_label_encoder = LabelEncoder()
            y_train_encoded = self.rose_label_encoder.fit_transform(self.rose_y_train)
            y_test_encoded = self.rose_label_encoder.transform(self.rose_y_test)

            self.rose_model = SVC(kernel='linear', probability=True)
            self.rose_model.fit(self.rose_X_train, y_train_encoded)

            y_pred = self.rose_model.predict(self.rose_X_test)

            self.model_results.append({
                'name': 'Rose',
```

```
            'y_true': y_test_encoded,
            'y_pred': y_pred
        })
```

Evaluate Model

For the latest trained model:

- Calculate accuracy, precision, recall, F1
- Display results in a table
- Generate confusion matrix and precision-recall plots
- Display metrics visually with matplotlib

Input:

```
(1)
latest_result = self.model_results[-1]
        y_true = latest_result['y_true']
        y_pred = latest_result['y_pred']
        label_names = list(np.unique(y_true))

        # Fill metrics table (top of layout)
        precision, recall, f1, support = precision_recall_fscore_support(
            y_true, y_pred, average=None, zero_division=0
        )

(2)
for result in self.model_results:
            y_true = result['y_true']
            y_pred = result['y_pred']
            name = result['name']

            # Encode string labels if necessary
            if isinstance(y_true[0], str):
                encoder = LabelEncoder()
                y_true = encoder.fit_transform(y_true)
                y_pred = encoder.transform(y_pred)

            labels = sorted(np.unique(np.concatenate((y_true, y_pred))))
            cmx = confusion_matrix(y_true, y_pred, labels=labels)
            accuracy = accuracy_score(y_true, y_pred)

            # Metrics
            precision, recall, f1, support = precision_recall_fscore_support(
                y_true, y_pred, average=None, zero_division=0
            )
```

```
(3)
 # Confusion Matrix
          cax = axes[0].matshow(cmx, cmap='Blues')
          fig.colorbar(cax, ax=axes[0])
          axes[0].set_xticks(range(len(labels)))
          axes[0].set_yticks(range(len(labels)))
          axes[0].set_xticklabels(labels, rotation=45, ha='left')
          axes[0].set_yticklabels(labels)
          axes[0].set_xlabel('Prediction')
          axes[0].set_ylabel('Label')
          axes[0].set_title(f"{name} Accuracy: {accuracy:.2%}")

          # Precision-Recall Points
          for i, label in enumerate(labels):
             axes[1].plot(recall[i], precision[i], marker='o', linestyle='-', label=f"Class
{label}")
          axes[1].set_xlim(0, 1.05)
          axes[1].set_ylim(0, 1.05)
          axes[1].set_xlabel("Recall")
          axes[1].set_ylabel("Precision")
          axes[1].set_title("Precision-Recall Points")
          axes[1].legend()
          axes[1].grid(True)
          fig.tight_layout(pad=5.0)
```

Predict Iris

- Read float values from 4 QLineEdit inputs
- Form a NumPy feature vector
- Use trained iris_model to predict class
- Inverse transform predicted label
- Show result in GUI label
- Update confidence bar using predict_proba

Input:

```
# Predict and decode class name
          predicted_class = self.iris_model.predict(features)[0]
                                                                predicted_label          =
self.iris_label_encoder.inverse_transform([predicted_class])[0]

          self.iris_prediction_label.setText(f"Iris prediction: {predicted_label}")
```

Predict Rose Color

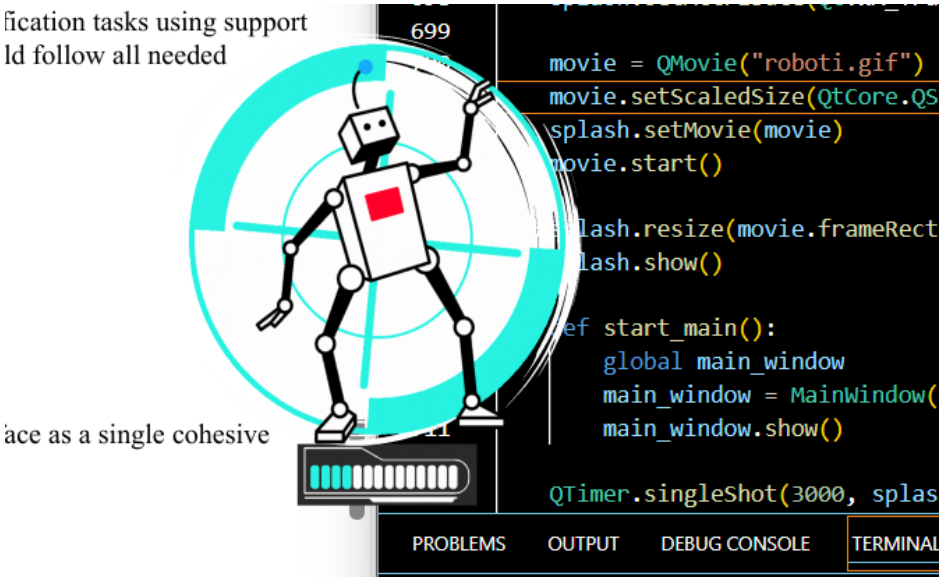
- Read loaded image from disk
- Resize to 64x64
- Convert to grayscale
- Flatten and reshape to vector
- Predict rose color using rose_model
- Inverse transform label
- Show result in GUI
- Plot confidence probabilities

Input:

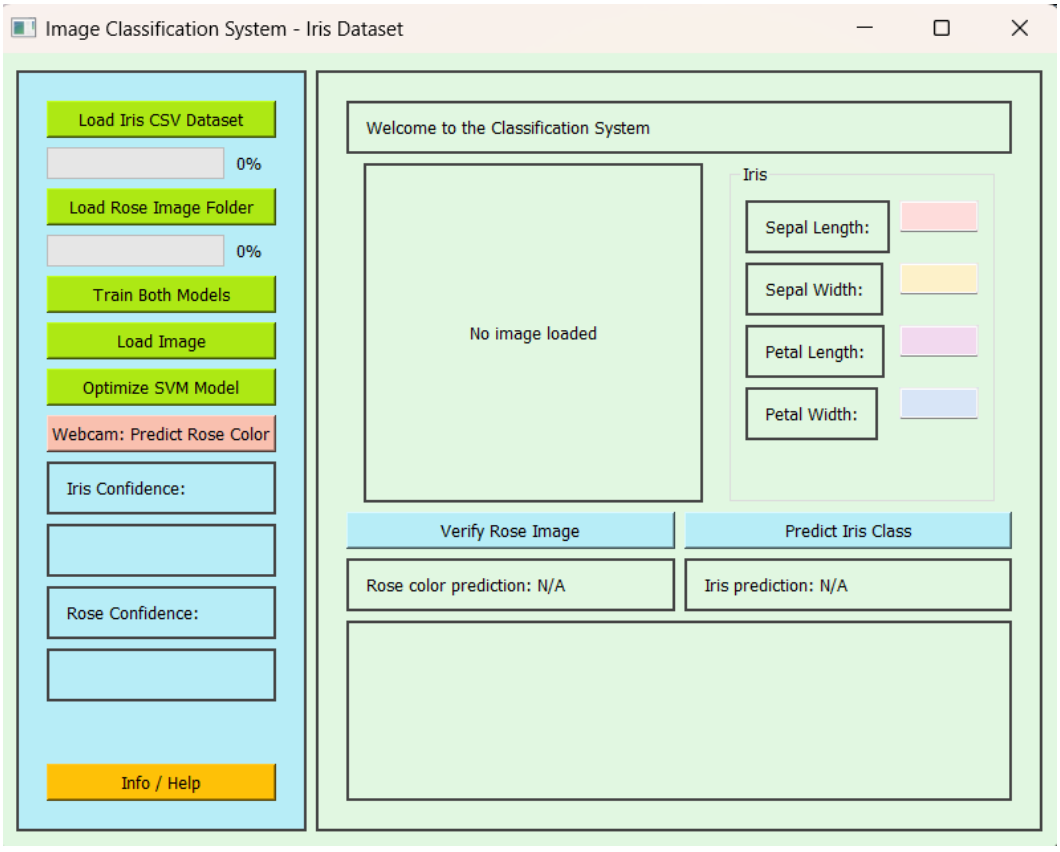
```
if not hasattr(self, 'rose_model') or not hasattr(self, 'rose_label_encoder'):
            self.rose_prediction_label.setText("Error: Train the Rose model first.")
            return

                        if not hasattr(self, 'loaded_image_path') or not
os.path.exists(self.loaded_image_path):
            QMessageBox.warning(self, "Error", "No image has been loaded.")
            return

        img = cv2.imread(self.loaded_image_path)
        if img is None:
            QMessageBox.warning(self, "Error", "Loaded image is invalid.")
            return

        img_resized = cv2.resize(img, (64, 64))
        gray = cv2.cvtColor(img_resized, cv2.COLOR_BGR2GRAY)
        features = gray.flatten().reshape(1, -1)

        predicted_class = self.rose_model .predict(features)[0]
                                                    predicted_label                =
self.rose_label_encoder.inverse_transform([predicted_class])[0]

                        self.rose_prediction_label.setText(f"Rose color prediction:
{predicted_label.capitalize()}")
```

5.0 SEVERAL SIGNIFICANT OUTPUT

Splashscreen

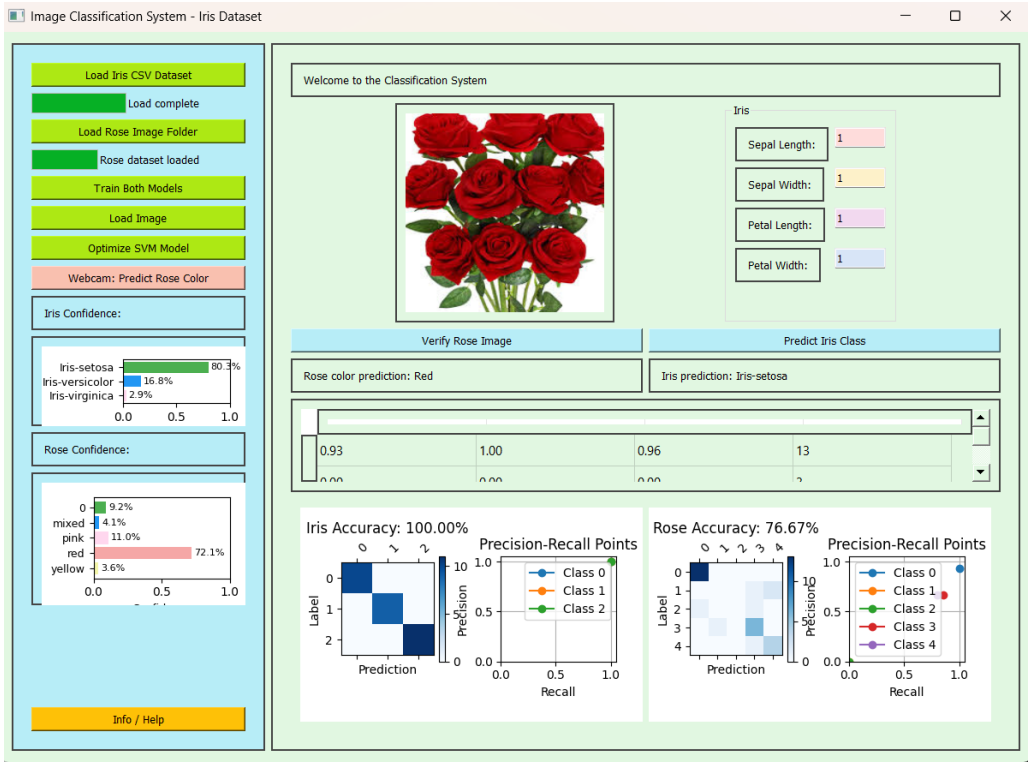


Initial UI

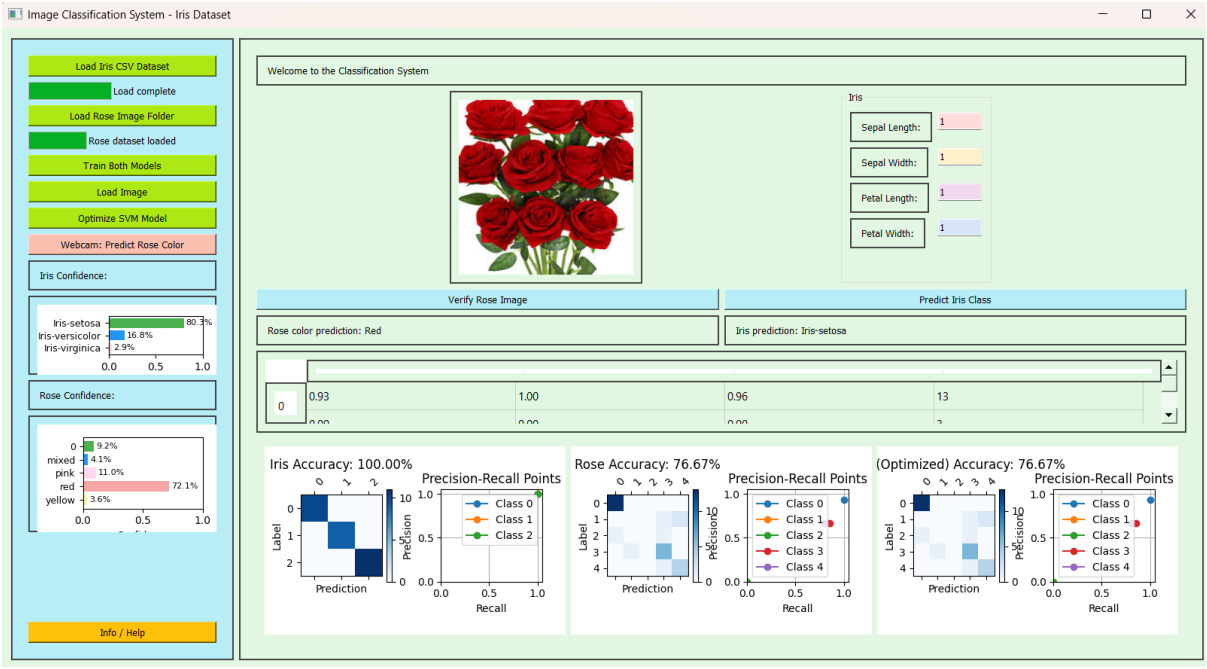


After Load and Train Data

After load rose image and enter iris input



After optimizing the model



Webcam