FLOWER IDENTIFICATION USING DEEP LEARNING

Abstract

The classification of flowers based on their images is a practical and visually appealing application of deep learning in the domain of computer vision. This project presents the development of a flower identification system using a pre-trained ResNet50 convolutional neural network model, fine-tuned on a flower dataset. To make the model accessible to end-users, a web application is developed using Flask, allowing users to upload flower images and obtain predictions through a simple and intuitive interface. The system demonstrates the capability of transfer learning and web technologies to deliver real-time image classification.

Tech Stack Summary:

tensorflow Keras Flask docker Google Cloud Platform

Debojyoti

Project Overview

Advancing Flower Recognition with Deep Learning

Automated flower recognition is crucial for botanical research and biodiversity. This project addresses the limitations of manual identification by developing a robust deep learning model and an intuitive web application.

Develop Classification Model

Build an accurate image classification model for various flower species.

Utilize Transfer Learning

Leverage ResNet50 for efficient and effective model training.

Create Web Interface

Design a user-friendly Flask application for image uploads and predictions.

Ensure Real-Time Efficiency

Optimize the system for efficient and accurate real-time performance.

System Architecture

Model & Web Application Pipeline

Model Pipeline

1. **Dataset Preparation**

   Oxford 102 Flower Dataset resized and augmented.

2. **Model Selection**

   ResNet50 (ImageNet pre-trained) for transfer learning.

3. **Model Fine-Tuning**

   Final layers replaced; trained with custom hyperparameters.

4. **Prediction Logic**

   Model outputs probabilities; highest selected as prediction.

5. **Confidence Score**

   Extracted using np.max(predictions) from softmax output.

Web Application Pipeline

1. **Frontend**

   HTML/CSS with Bootstrap for UI and image upload.

2. **Backend (Flask)**

   Handles routing, image upload, and model prediction.

3. **Integration**

   Image sent, preprocessed, passed to model, prediction displayed.

Tech Stack Overview

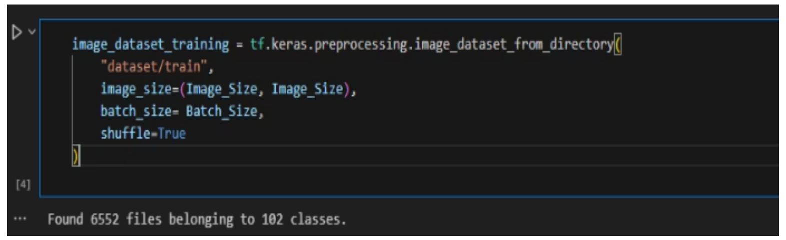
Layer	Tools Used
Deep Learning Framework	TensorFlow, Keras
Web Framework	Flask
Frontend	HTML, CSS, Bootstrap
Libraries	NumPy, Pillow
Containerization	Docker
Cloud & Deployment	Google Cloud Build, Google Cloud Run

Technical Implementation

Model Training & Data Handling

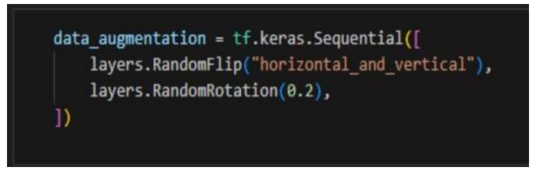
Dataset & Preparation

The Oxford 102 Flower Dataset, comprising **102 distinct classes**, was utilized. Images were uniformly resized to **256x256 pixels** and augmented to enhance model generalization. TensorFlow Keras's `image_dataset_from_directory` utility streamlined data loading and automatic label assignment based on subfolder names.

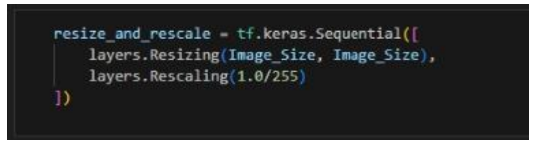


```
image_dataset_training = tf.keras.preprocessing.image_dataset_from_directory(
    "dataset/train",
    image_size=(Image_Size, Image_Size),
    batch_size= Batch_Size,
    shuffle=True
)
```

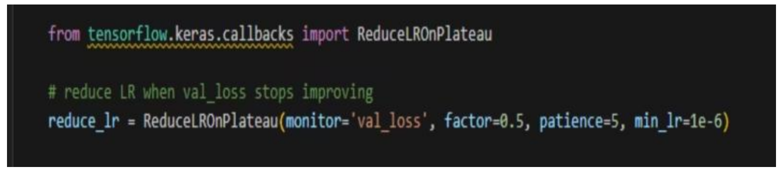
```
Found 6552 files belonging to 102 classes.
```



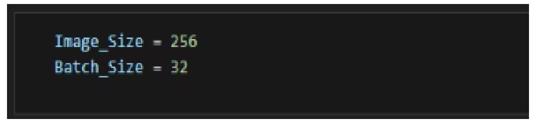
```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
])
```



```
resize_and_rescale = tf.keras.Sequential([
    layers.Resizing(Image_Size, Image_Size),
    layers.Rescaling(1.0/255)
])
```



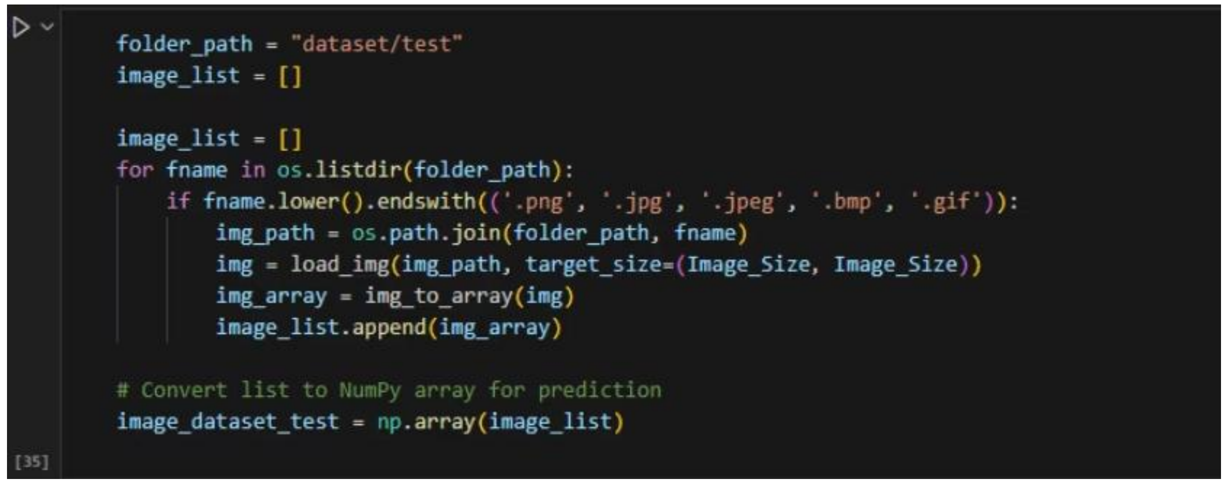
```
from tensorflow.keras.callbacks import ReduceLROnPlateau

# reduce LR when val_loss stops improving
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)
```



```
Image_Size = 256
Batch_Size = 32
```

Test Image Handling

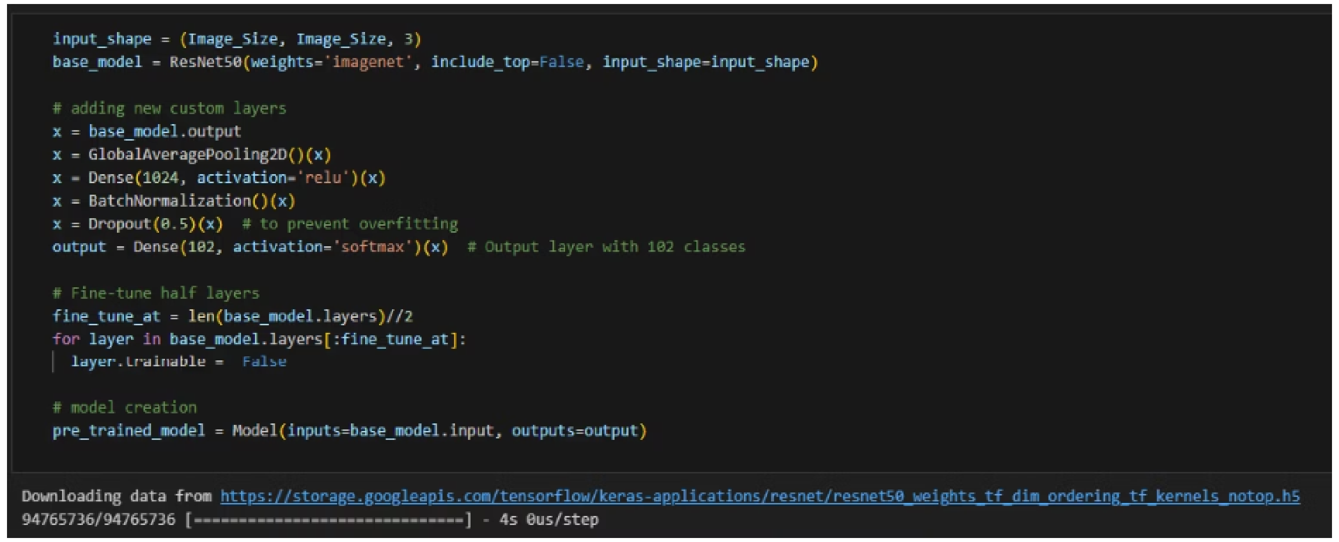
For test images, which lacked predefined class labels due to their individual file structure, a custom loading approach was implemented. Each image was manually loaded, consistently preprocessed with resizing, and compiled into a NumPy array for direct prediction. This ensured uniformity for inference.



```
folder_path = "dataset/test"
image_list = []

image_list = []
for fname in os.listdir(folder_path):
    if fname.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):
        img_path = os.path.join(folder_path, fname)
        img = load_img(img_path, target_size=(Image_Size, Image_Size))
        img_array = img_to_array(img)
        image_list.append(img_array)

# Convert list to NumPy array for prediction
image_dataset_test = np.array(image_list)
```

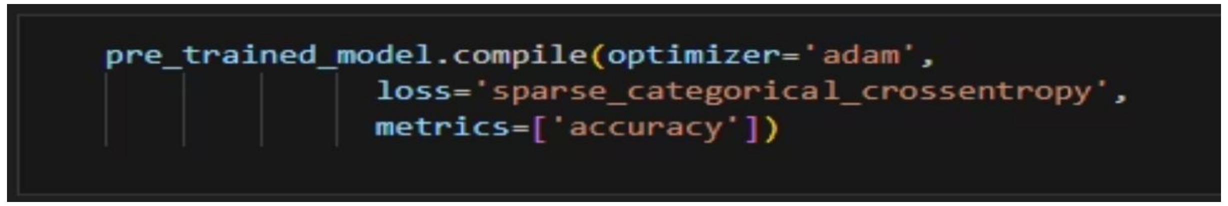
Model Fine-Tuning

The pre-trained ResNet50 model's final layers were reconfigured to match the 102 flower classes. Custom hyperparameters were applied during training to optimize performance for the specific dataset.

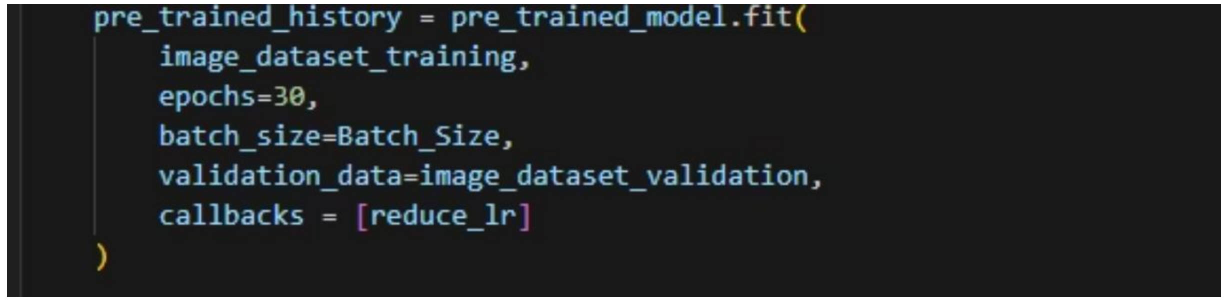


```
input_shape = (Image_Size, Image_Size, 3)
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)

# adding new custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)  # to prevent overfitting
output = Dense(102, activation='softmax')(x)  # Output layer with 102 classes

# Fine-tune half layers
fine_tune_at = len(base_model.layers)//2
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable =  False

# model creation
pre_trained_model = Model(inputs=base_model.input, outputs=output)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 4s 0us/step
```



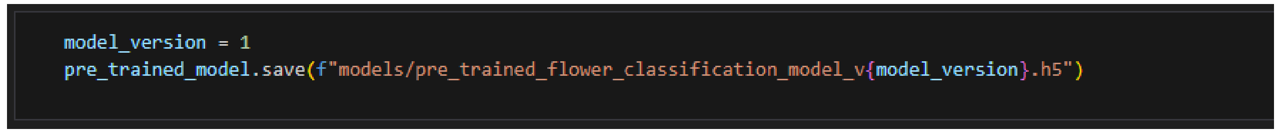
```
pre_trained_model.compile(optimizer='adam',
                          loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'])
```



```
pre_trained_history = pre_trained_model.fit(
    image_dataset_training,
    epochs=30,
    batch_size=Batch_Size,
    validation_data=image_dataset_validation,
    callbacks = [reduce_lr]
)
```

Exporting the Model

Finally saved the model as h5 file for future use for prediction.



```
model_version = 1
pre_trained_model.save(f"models/pre_trained_flower_classification_model_v{model_version}.h5")
```

Model Integration

To enhance flexibility and performance comparison, two separate deep learning models were integrated into the web application:

- A **ResNet50-based model**, pretrained on ImageNet and fine-tuned on flower datasets.
- A **custom-trained CNN model**, optimized for speed and smaller memory footprint.

The application allows the user to select the model from a dropdown menu before uploading an image for classification. This enables interactive evaluation of both models and supports use cases with different computational constraints.

Model Specification

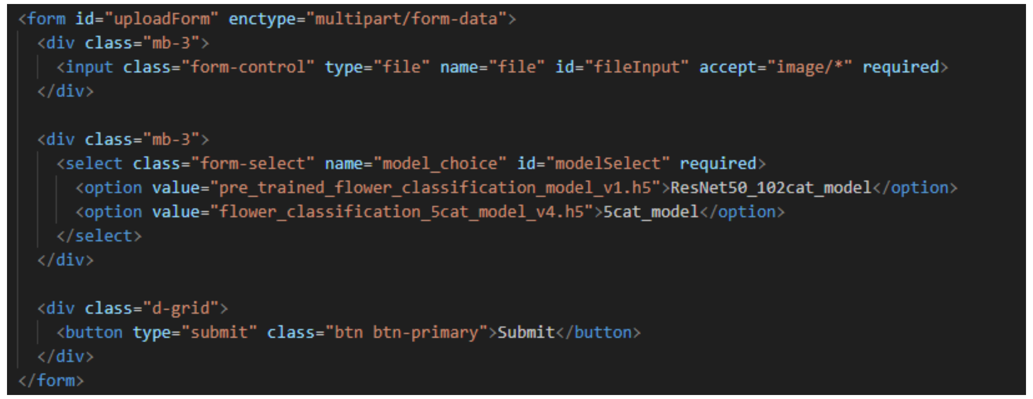
ResNet50-based model is trained on following flower set:

"pink primrose", "hard-leaved pocket orchid", "canterbury bells", "sweet pea", "english marigold", "tiger lily", "moon orchid", "bird of paradise", "monkshood", "globe thistle", "snapdragon", "colt's foot", "king protea", "spear thistle", "yellow iris", "globe-flower", "purple coneflower", "peruvian lily", "balloon flower", "giant white arum lily", "fire lily", "pincushion flower", "fritillary", "red ginger", "grape hyacinth", "corn poppy", "prince of wales feathers", "stemless gentian", "artichoke", "sweet william", "carnation", "garden phlox", "love in the mist", "mexican aster", "alpine sea holly", "ruby-lipped cattleya", "cape flower", "great masterwort", "siam tulip", "lenten rose", "barbeton daisy", "daffodil", "sword lily", "poinsettia", "bolero deep blue", "wallflower", "marigold", "buttercup", "oxeye daisy", "common dandelion", "petunia", "wild pansy", "primula", "sunflower", "pelargonium", "bishop of llandaff", "gaura", "geranium", "orange dahlia", "pink-yellow dahlia", "cautleya spicata", "japanese anemone", "black-eyed susan", "silverbush", "californian poppy", "osteospermum", "spring crocus", "bearded iris", "windflower", "tree poppy", "gazania", "azalea", "water lily", "rose", "thorn apple", "morning glory", "passion flower", "lotus lotus", "toad lily", "anthurium", "frangipani", "clematis", "hibiscus", "columbine", "desert-rose", "tree mallow", "magnolia", "cyclamen", "watercress", "canna lily", "hippeastrum", "bee balm", "ball moss", "foxglove", "bougainvillea", "camellia", "mallow", "mexican petunia", "bromelia", "blanket flower", "trumpet creeper", "blackberry lily"

Custom-trained CNN model is trained on following flower set:

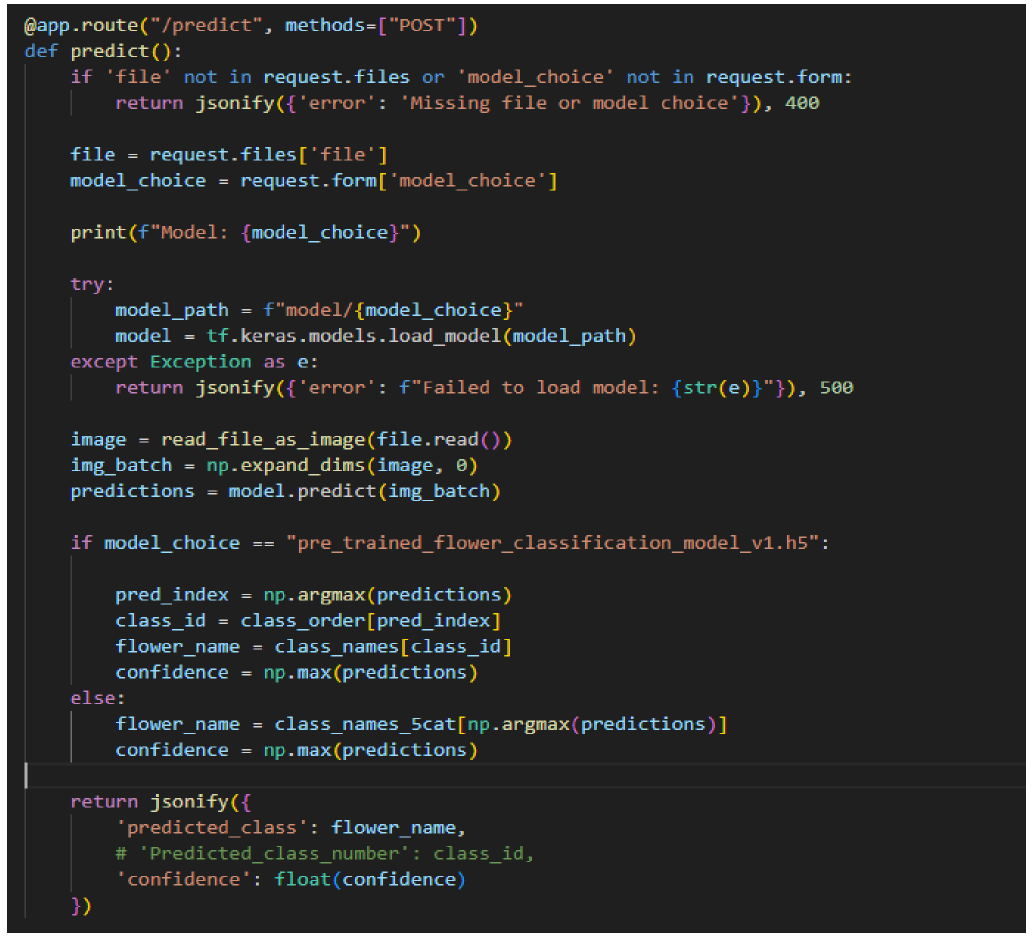
"daisy", "dandelion", "rose", "sunflower", "tulip"

A dropdown is added to let the user select a model:



```
<form id="uploadForm" enctype="multipart/form-data">
  <div class="mb-3">
    <input class="form-control" type="file" name="file" id="fileInput" accept="image/*" required>
  </div>

  <div class="mb-3">
    <select class="form-select" name="model_choice" id="modelSelect" required>
      <option value="pre_trained_flower_classification_model_v1.h5">ResNet50_102cat_model</option>
      <option value="flower_classification_5cat_model_v4.h5">5cat_model</option>
    </select>
  </div>

  <div class="d-grid">
    <button type="submit" class="btn btn-primary">Submit</button>
  </div>
</form>
```

The /predict route processes the form input:



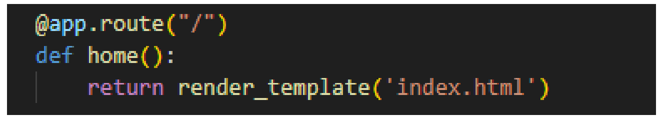
```
@app.route("/predict", methods=["POST"])
def predict():
    if 'file' not in request.files or 'model_choice' not in request.form:
        return jsonify({'error': 'Missing file or model choice'}), 400

    file = request.files['file']
    model_choice = request.form['model_choice']

    print(f"Model: {model_choice}")

    try:
        model_path = f"model/{model_choice}"
        model = tf.keras.models.load_model(model_path)
    except Exception as e:
        return jsonify({'error': f"Failed to load model: {str(e)}"}), 500

    image = read_file_as_image(file.read())
    img_batch = np.expand_dims(image, 0)
    predictions = model.predict(img_batch)

    if model_choice == "pre_trained_flower_classification_model_v1.h5":

        pred_index = np.argmax(predictions)
        class_id = class_order[pred_index]
        flower_name = class_names[class_id]
        confidence = np.max(predictions)
    else:
        flower_name = class_names_5cat[np.argmax(predictions)]
        confidence = np.max(predictions)

    return jsonify({
        'predicted_class': flower_name,
        # 'Predicted_class_number': class_id,
        'confidence': float(confidence)
    })
```

Flask Application

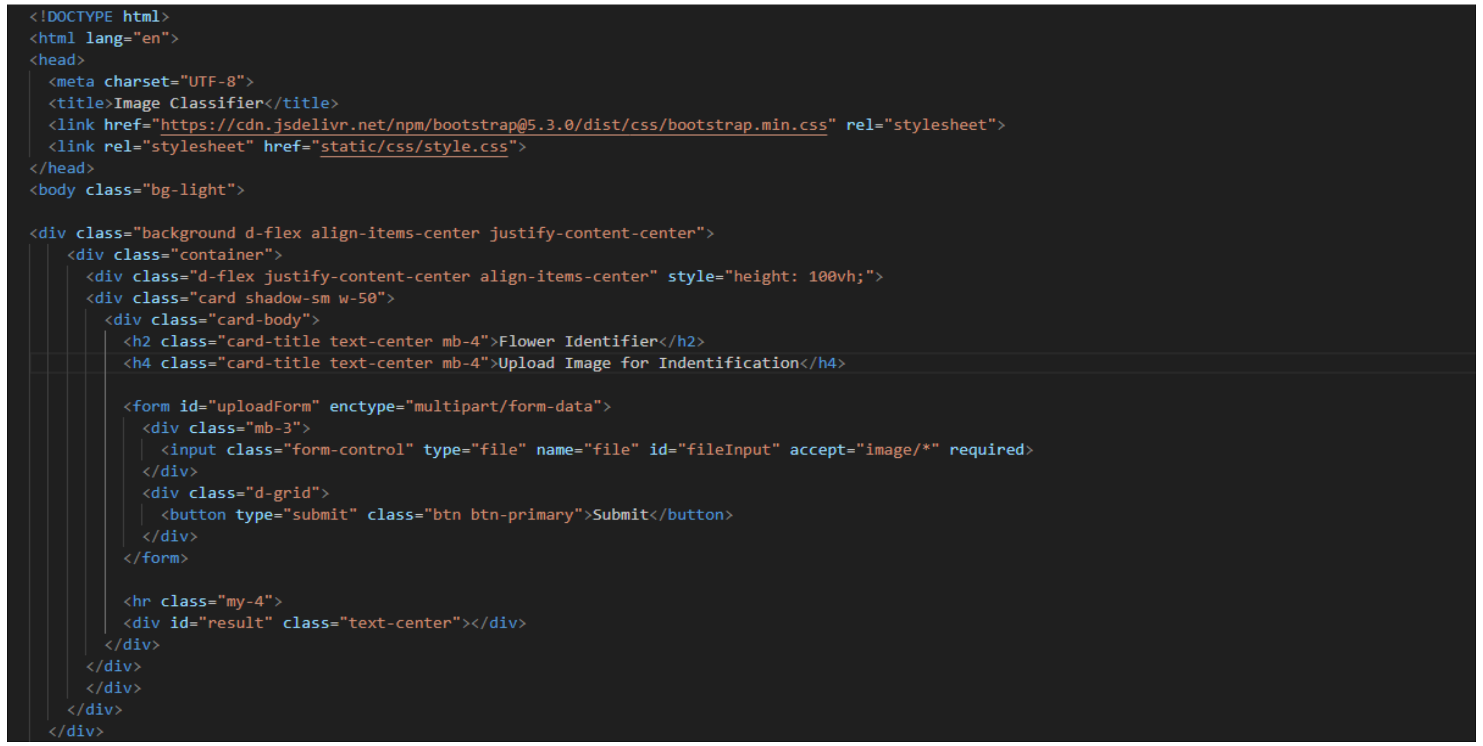
The Flask web application provides a user interface for flower image upload and displays prediction results. It has two main routes:

/ - Home Route

This route renders the **main upload interface** where users can select and upload an image for classification.



```python
@app.route("/")
def home():
    return render_template('index.html')
```

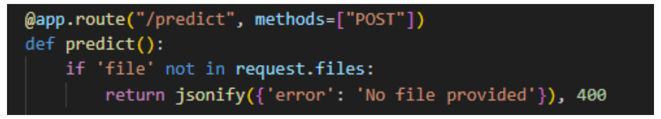


```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Image Classifier</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
  <link rel="stylesheet" href="static/css/style.css">
</head>
<body class="bg-light">

<div class="background d-flex align-items-center justify-content-center">
    <div class="container">
      <div class="d-flex justify-content-center align-items-center" style="height: 100vh;">
      <div class="card shadow-sm w-50">
        <div class="card-body">
          <h2 class="card-title text-center mb-4">Flower Identifier</h2>
          <h4 class="card-title text-center mb-4">Upload Image for Indentification</h4>

          <form id="uploadForm" enctype="multipart/form-data">
            <div class="mb-3">
              <input class="form-control" type="file" name="file" id="fileInput" accept="image/*" required>
            </div>
            <div class="d-grid">
              <button type="submit" class="btn btn-primary">Submit</button>
            </div>
          </form>

          <hr class="my-4">
          <div id="result" class="text-center"></div>
        </div>
      </div>
      </div>
    </div>
  </div>
```

/predict - Prediction Route

This route handles: Receiving the uploaded file, Reading and preprocessing the image, Model inference, Returning prediction and confidence.

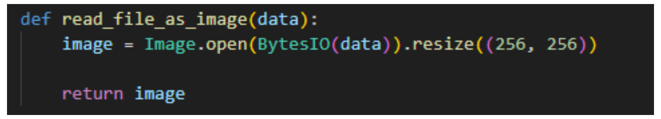
Upload Handling:



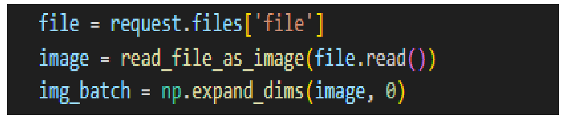
```python
@app.route("/predict", methods=["POST"])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No file provided'}), 400
```

Image Preprocessing:

The image is resized to 256x256 and converted to a format compatible with the model.



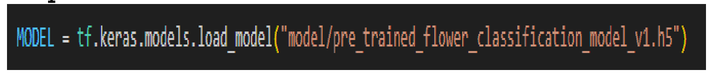
```python
def read_file_as_image(data):
    image = Image.open(BytesIO(data)).resize((256, 256))

    return image
```



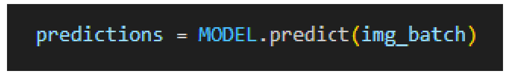
```python
file = request.files['file']
image = read_file_as_image(file.read())
img_batch = np.expand_dims(image, 0)
```

Prediction:

The model is loaded in the main.py file. The processed image is passed to the loaded ResNet50 model for prediction.

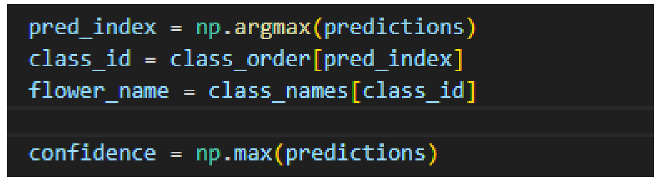


```python
MODEL = tf.keras.models.load_model("model/pre_trained_flower_classification_model_v1.h5")
```



```python
predictions = MODEL.predict(img_batch)
```

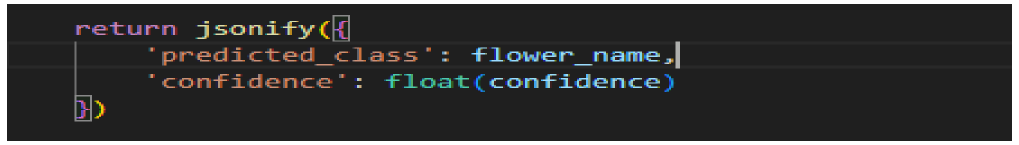
Interpreting Results:



```python
pred_index = np.argmax(predictions)
class_id = class_order[pred_index]
flower_name = class_names[class_id]

confidence = np.max(predictions)
```

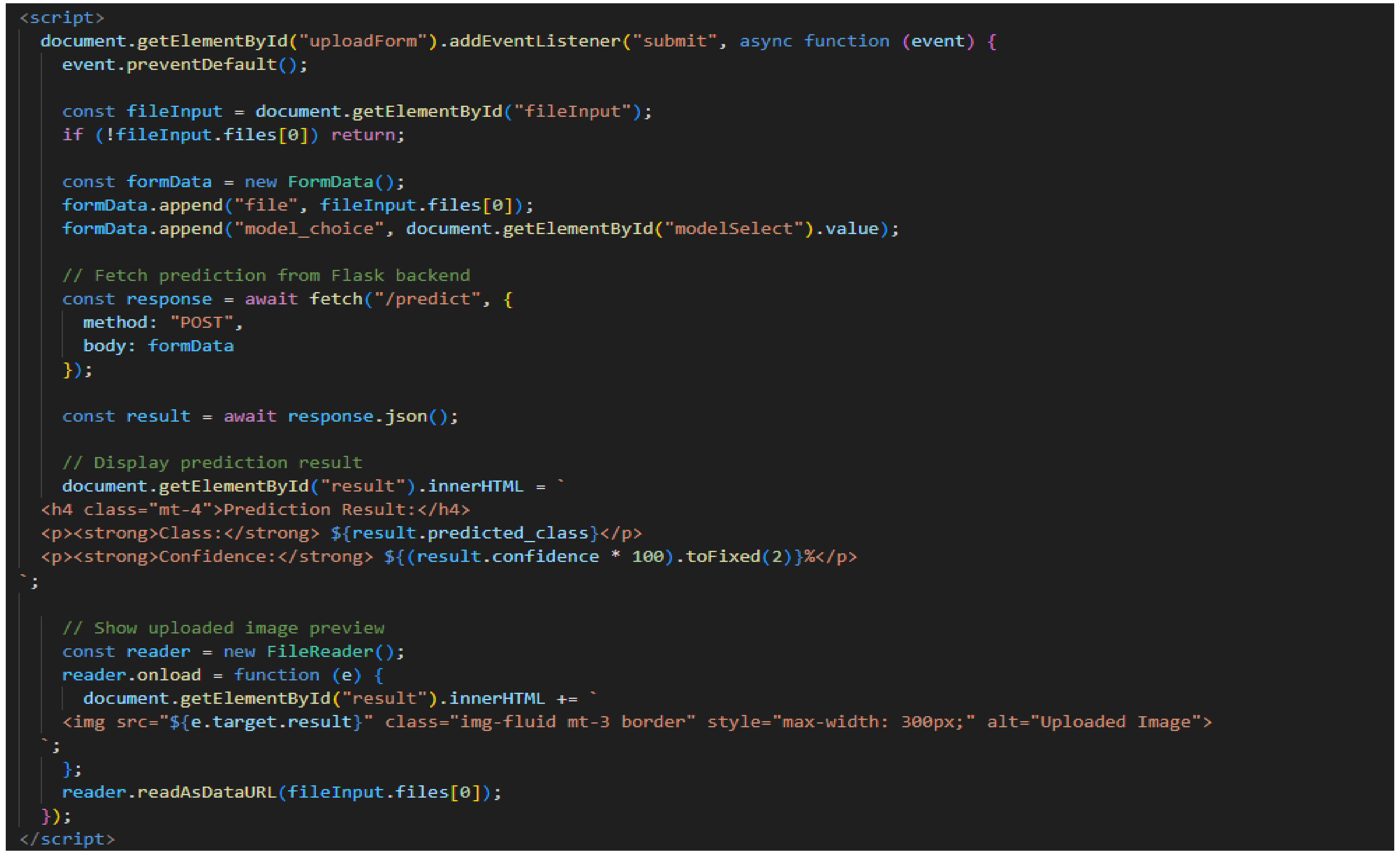
Returning Response

The result is returned as a JSON object, including:
- Predicted flower name
- Confidence score



```
return jsonify({
    'predicted_class': flower_name,
    'confidence': float(confidence)
})
```

Finally, the data from the form collected and transferred to the model at the backend and result from this JSON file is displayed in the UI with the help of JavaScript.



```
<script>
  document.getElementById("uploadForm").addEventListener("submit", async function (event) {
    event.preventDefault();

    const fileInput = document.getElementById("fileInput");
    if (!fileInput.files[0]) return;

    const formData = new FormData();
    formData.append("file", fileInput.files[0]);
    formData.append("model_choice", document.getElementById("modelSelect").value);

    // Fetch prediction from Flask backend
    const response = await fetch("/predict", {
      method: "POST",
      body: formData
    });

    const result = await response.json();

    // Display prediction result
    document.getElementById("result").innerHTML = `
  <h4 class="mt-4">Prediction Result:</h4>
  <p><strong>Class:</strong> ${result.predicted_class}</p>
  <p><strong>Confidence:</strong> ${(result.confidence * 100).toFixed(2)}%</p>
`;

    // Show uploaded image preview
    const reader = new FileReader();
    reader.onload = function (e) {
      document.getElementById("result").innerHTML += `
    <img src="${e.target.result}" class="img-fluid mt-3 border" style="max-width: 300px;" alt="Uploaded Image">
  `;
    };
    reader.readAsDataURL(fileInput.files[0]);
  });
</script>
```

3. Cloud Build and Deployment Steps

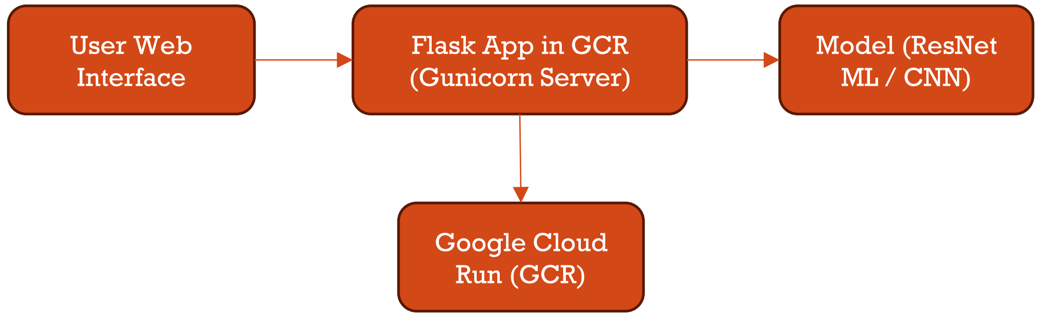
After preparing the Dockerfile and .dockerignore file the deployment steps were:
- Build container image using Google Cloud Build.
- Deploy container to Cloud Run.

Deployment

The trained deep learning models and Flask-based web application were deployed to **Google Cloud Run** to ensure:
- High availability and global access
- Scalability without manual infrastructure setup
- Isolation of compute resources for security and performance

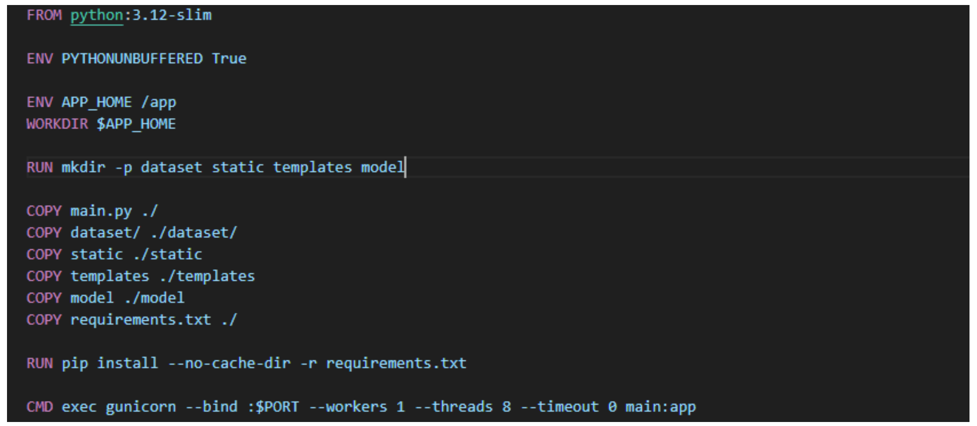
This allowed the application to be used in real-world scenarios, enabling users to upload flower images, choose a classification model, and receive predictions in real time.

1. Deployment Architecture



2. Dockerization

The entire project was containerized using Docker. This allows the same environment to be replicated during deployment.



```
FROM python:3.12-slim

ENV PYTHONUNBUFFERED True

ENV APP_HOME /app
WORKDIR $APP_HOME

RUN mkdir -p dataset static templates model

COPY main.py ./
COPY dataset/ ./dataset/
COPY static ./static
COPY templates ./templates
COPY model ./model
COPY requirements.txt ./

RUN pip install --no-cache-dir -r requirements.txt

CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 --timeout 0 main:app
```

4. Result

A live URL was generated to access the app from any browser.

Conclusion

This project successfully demonstrates how a deep learning model like ResNet50 can be integrated into a web-based application to identify flower species accurately. The combination of transfer learning and Flask web deployment enables a scalable and user-friendly solution. This work can be extended further for mobile deployment or real-time camera-based recognition systems.

Future Scope

- Expand dataset to include more flower types and variants.
- Add a feedback mechanism to improve model performance via user corrections.
- Build a mobile app version using Flutter or React Native with API integration.

Reference

- TensorFlow/Keras Documentation
- Flask Documentation
- Oxford 102 Category Flower Dataset
- Docker Documentation

- GitHub: https://github.com/CwDebojyoti/flower_identification.git

- Try the Application from here: https://flowerclassificationv2-17376196659.asia-east1.run.app

--End of Document--