**Project Report**

**Weather Classification Using Deep Learning**

**Team ID :** Team-592288

**Team Leader :** Manan Sharma

**Team member :** Kanika Rathi

**Team member :** Shreyans Jain

**Team member :** D Gunasekhar

**Index Of Contents**

1. INTRODUCTION

1.1 Project Overview

1.2 Purpose

2. LITERATURE SURVEY

2.1 Existing problem

2.2 References

2.3 Problem Statement Definition

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

3.2 Ideation & Brainstorming

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

4.2 Non-Functional requirements

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

5.2 Solution Architecture

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

6.2 Sprint Planning & Estimation

6.3 Sprint Delivery Schedule

7. CODING & SOLUTIONING

8. PERFORMANCE TESTING

8.1 Performace Metrics

9. ADVANTAGES & DISADVANTAGES

10. CONCLUSION

11. FUTURE SCOPE

12. APPENDIX

Source Code

GitHub & Project Demo Link

1. **INTRODUCTION**
   1. **Project Overview**

Categorizing weather conditions is vital for meteorologists and forecasters to anticipate and convey upcoming weather patterns. Recognizing various weather phenomena significantly impacts our day-to-day routines. Assessing these phenomena plays a pivotal role across multiple areas such as environmental monitoring, weather prediction, and evaluating environmental standards. Furthermore, distinct weather occurrences have varying impacts on agriculture, making precise identification crucial for enhancing agricultural strategies.

* 1. **Purpose**

The objectives are as follows:

* Allow user interaction with the UI to select an image.
* Process the chosen image through a VGG19 deep learning model.
* Integrate the VGG19 model into a Flask application.
* Utilize the VGG19 model to analyze the image and produce predictions.
* Display these predictions on the Flask UI for user review.
* Enable users to swiftly input an image and receive accurate predictions.

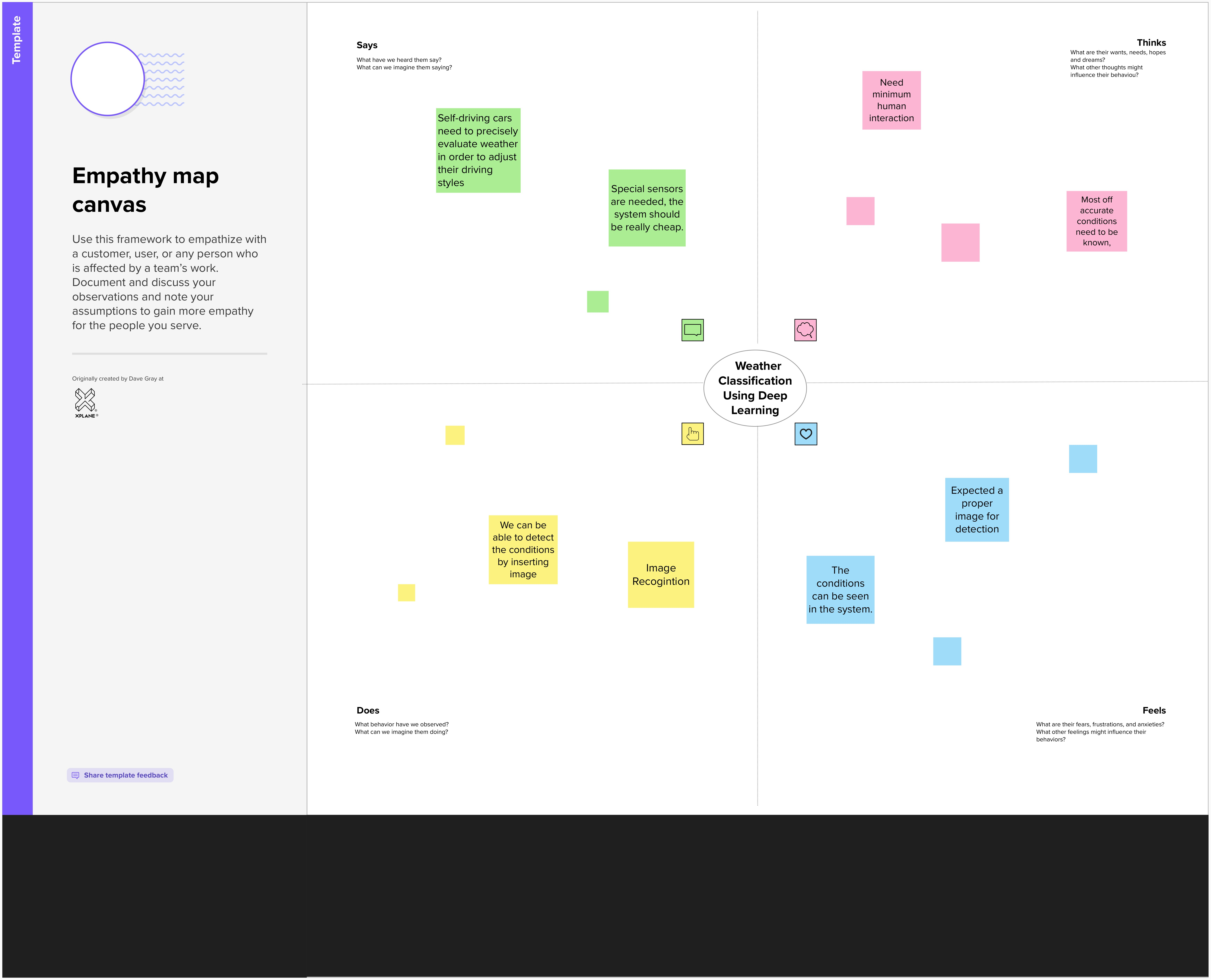
To achieve this, the following tasks need completion:

* Data Collection:
  + Download and extract the dataset.
* Image Pre-processing:
  + - Import necessary libraries.
    - Set up the ImageDataGenerator class.
    - Apply ImageDataGenerator functionality to the Trainset and Testset.
    - Model Building:
    - Use a pre-trained CNN model as a Feature Extractor.
    - Add Dense Layer.
    - Configure the Learning Process.
    - Train the model.
    - Evaluate Model Accuracy.
    - Save the Model.
    - Test the model.
    - Application Building:
    - Create HTML Pages.
    - Develop Flask Code.
    - Launch the Application.

1. **Literature Survey**
   1. **Existing Problem**
   2. **Problem Statement Definition**

Identifying weather patterns is a widespread challenge across various industries. Contemporary agriculture heavily relies on analyzing present meteorological situations. A potential solution involves a weather detection system using images, eliminating the necessity for specialized sensors, thereby ensuring a cost-effective system.

1. **Ideation & Proposed Solution**
   1. **Empathy Map Canvas**



* 1. **Ideation & Brainstorming**

Step 1: Team Gathering and Problem Statement Selection

Problem Statement:

Weather classification is vital for agriculture, transportation, and disaster management. Accurate classification aids crop yield optimization, efficient route planning, and better preparedness for extreme weather events.

Challenges:

Conventional methods lack generalization across regions and seasons, leading to inaccuracies in unique or data-scarce areas.

Proposed Solution:

Develop an automated weather classification system using transfer learning from pre-trained deep learning models, like ImageNet, to enhance accuracy and robustness.

Specific Challenges:

1. Dataset Collection: Gather diverse, labelled weather images covering various conditions.
2. Model Selection: Identify and fine-tune pre-trained models for weather classification.
3. Generalization: Ensure the model performs well across different locations and seasons.

Additional Considerations:

* + - Use data augmentation and cross-validation techniques.
    - Explore explainability methods for model understanding and bias identification.

Step 2: Brainstorming and Grouped Ideas

1. Dataset Collection:
   * + Collect diverse labeled weather images from multiple sources.
     + Preprocess images for noise removal and standardization.
2. Model Architecture and Transfer Learning:

* Evaluate pre-trained models like VGG or ResNet for weather classification.
* Adapt and fine-tune models for optimal performance.

1. Spatial and Temporal Context:

* Utilize multi-scale convolutional filters for varied weather patterns.
* Develop temporal models considering weather changes over time.

1. Domain Adaptation and Generalization:

* Transfer knowledge across regions for improved adaptability.
* Incorporate external data sources to enhance generalization.

1. Evaluation and Performance Metrics:

* Define suitable metrics and address class imbalance issues.
* Estimate uncertainty for prediction reliability.

1. Real-time Deployment and Optimization:

* Design efficient pipelines for real-time classification.
* Optimize models for resource-constrained devices.

1. Visualization and Interpretability:
   * + Develop visualizations for model feature understanding.
     + Create user-friendly interfaces for presenting results.

These strategies outline the development path for an effective weather classification system using transfer learning.

1. **Requirement Analysis**
   1. **Functional Requirement**

Following are the functional requirements of the proposed solution.

|  |  |  |
| --- | --- | --- |
| **FR No.** | **Functional Requirement** | **Sub Requirement** |
| FR – 1 | User Registration | Registration through Form Registration, through  Gmail Registration, through LinkedIN |
| FR – 2 | User Confirmation | Confirmation via Email  Confirmation via OTP |
| FR – 3 | User Interface | It allows users to capture images of garbage and see the results of the classification in real-time. |
| FR – 4 | AI Model | The project should use an AI algorithm that can learn from data and improve over time. |
| FR – 5 | Real Time Classification | It should be able to classify images quickly and accurately as soon as they are captured by a camera. |

* 1. **Non-Functional Requirement**

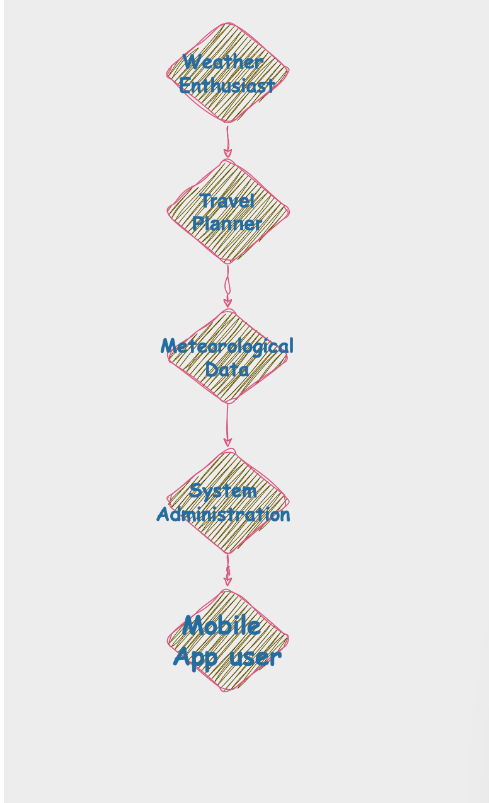
Following are the non-functional requirements of the proposed solution.

|  |  |  |
| --- | --- | --- |
| **NFR No.** | **Non- Functional Requirement** | **Description** |
| NFR – 1 | Performance | Performance forecasting is an essential service to support decision-taking in the concept, design and operational phases of an asset, meeting the production efficiency challenge by enhancing operational performance |
| NFR – 2 | Reliability | A seven-day forecast can accurately predict the weather about 80 percent of the time |
| NFR – 3 | Security | Stay indoors and move to a shelter. |
| NFR – 4 | Scalability | Global efforts to bring about crucial  improvements in supercomputing efficiency and energy usage were placed center stage this week as the European Centre for Medium-Range Weather Forecasts (ECMWF) welcomed users and vendors from around the world to London for the Cray |
| NFR – 5 | Usability | Weather Forecasting is crucial since it helps  to determine future climate changes. |

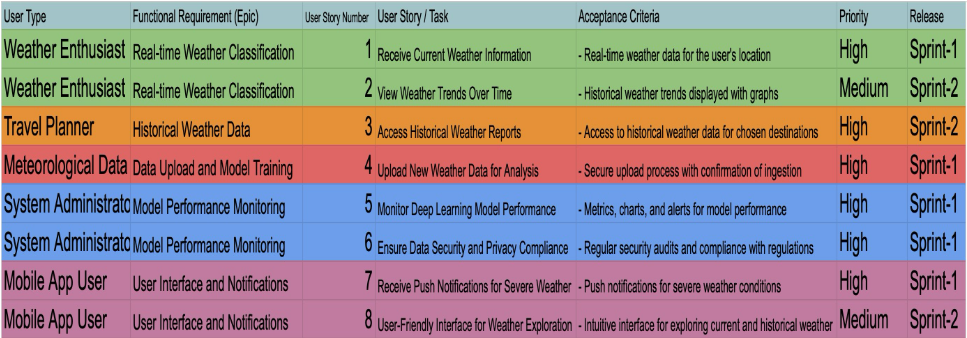
1. **Project Design**
   1. **Data Flow Diagrams & User Stories**

**Data Flow Diagrams:**

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



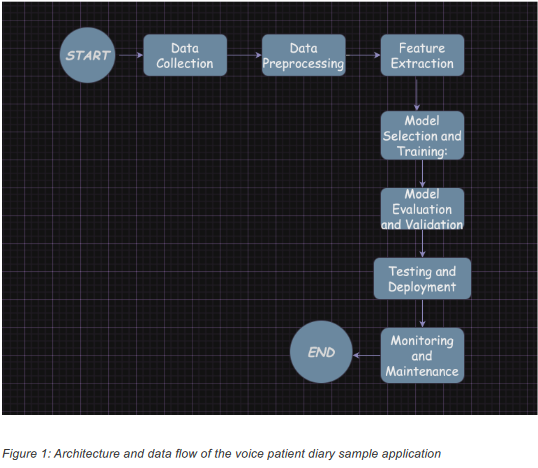
**User Stories:**



* 1. **Solution Architecture**

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

* Find the best tech solution to solve existing business problems.
* Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.
* Define features, development phases, and solution requirements.
* Provide specifications according to which the solution is defined, managed and delivered



1. **Project Planning & Scheduling**
   1. **Technical Architecture**

Table-1: Tech Stack

|  |  |  |
| --- | --- | --- |
| **S No.** | **Component** | **Technology/Service Used** |
| 1. | User Interface | Tkinter (Python GUI toolkit) |
| 2. | Application Logic-1 | TensorFlow (Python) |
| 3. | Application Logic-2 | IBM Watson Speech to Text (STT) service |
| 4. | Application Logic-3 | IBM Watson Assistant |
| 5. | Cloud Database | IBM Cloudant (Assumed based on Watson services) |
| 6. | Machine Learning Model | VGG19 model (pre-trained) |
| 7. | Infrastructure (Server / Cloud) | Local Server Configuration: N/A (Code not designed for local server) Cloud Server Configuration: IBM Cloud Foundry or Kubernetes (assumed based on IBM Watson services) |

Table-2: Application Characteristics

|  |  |  |  |
| --- | --- | --- | --- |
| **S No.** | **Characteristics** | **Description** | **Technology** |
| 1. | Open-Source Frameworks | TensorFlow (open-source machine learning framework) | TensorFlow |
| 2. | Scalable Architecture | Limited scalability due to the frozen VGG19 architecture | VGG19 (pre-trained and frozen layers) |
| 3. | Availability | Dataset availability is crucial for training and evaluation | N/A (Dependent on dataset availability) |
| 4. | Performance | Data augmentation, GPU utilization, and batch size considerations | TensorFlow (Data augmentation, GPU support, batch processing) |

* 1. **Sprint Delivery Schedule**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sprint** | **Phase** | **Tasks** | **Start Date** | **End Date** |
| 1. | Project Initiation Phase | Define Project Goals and Objectives | August 1,  2023 | August 7,  2023 |
|  |  | Assessment of Scope and Requirements |  |  |
|  |  | Identify Key Stakeholders and Expectations |  |  |
| 2. | Data Collection and Preparation | Data Collection | August 8,  2023 | August 29,  2023 |
|  |  | Model-specific Data Requirements |  |  |
|  |  | Ethical Considerations |  |  |
| 3. | Team Formation and Resource Allocation | Assemble the Project Team and Assign Responsibilities | August 30,  2023 | September 6,  2023 |
|  |  | Identify Necessary Resources,  Including Hardware, Software, and External Tools/APIs |  |  |
| 4. | Task Breakdown and Timeline Planning | Task Breakdown | September 7,  2023 | September 13, 2023 |
|  |  | Detailed Timeline |  |  |
|  |  | Set Milestones |  |  |
| 5. | Model Selection and Architecture Design | Research and Model Selection | September 14, 2023 | September 28, 2023 |
|  |  | Architecture Design |  |  |
| 6. | Data Pre-processing Plan | Data Pre-processing Steps | September 29, 2023 | October 6,  2023 |
|  |  | Address Potential Challenges |  |  |
| 7. | Training and Evaluation Strategy | Dataset Splitting and Training Strategy | October 7,  2023 | October 21,  2023 |
|  |  | Hyperparameter Tuning and Model Training |  |  |
|  |  | Evaluation Metrics and Criteria |  |  |
| 8. | User Interface Design and Real- time Integration | UI Design | October 22,  2023 | November 5,  2023 |
|  |  | Real-time Integration |  |  |
|  |  | Features and Functionalities |  |  |
| 9. | Documentation and Reporting Structure | Code Documentation | November 6,  2023 | November 13, 2023 |
|  |  | Project Report |  |  |
|  |  | Metrics and KPIs |  |  |
| 10. | Testing and Deployment Plan | Testing Strategy | November 14, 2023 | November 28, 2023 |
|  |  | Deployment Plan |  |  |
|  |  | Configuration Considerations |  |  |
| 11. | Continuous Improvement and Iteration | Regular Reviews and Iterations | November 29, 2023 | Ongoing |
|  |  |  |  |  |

1. **Coding & Solutioning**

from flask import Flask, render\_template, request, url\_for

import os

from flask import flash,redirect

from werkzeug.utils import secure\_filename

import os

import shutil

from PIL import Image

from tensorflow import keras

from keras.layers import Dense

from keras.models import Sequential, load\_model

from tensorflow.keras.preprocessing import image

import numpy as np

def predict\_model(img):

model = load\_model('weathermodel.h5')

x = image.img\_to\_array(img)

x = np.expand\_dims(x,axis = 0)

preds = model.predict(x)

pred = np.argmax(preds,axis = 1)

index = ['cloudy','foggy','rainy','sunshine','sunrise']

result = str(index[pred[0]])

return(result)

def resize\_image(image\_path, width, height):

with Image.open(image\_path) as img:

resized\_img = img.resize((width, height))

resized\_img.save(image\_path)

app = Flask(\_\_name\_\_)

app.secret\_key = 'supersecretkey'

@app.route('/')

@app.route('/index.html')

def index():

return render\_template('index.html')

@app.route('/about.html')

def about():

return render\_template('about.html')

@app.route('/pictures.html')

def pictures():

return render\_template('pictures.html')

@app.route('/upload.html')

def up():

return render\_template('upload.html')

@app.route('/upload', methods=['POST'])

def upload():

file = request.files['file']

filename = secure\_filename(file.filename)

print(filename)

file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

path = 'static/{}'.format(filename)

resize\_image(path,300,300)

img = image.load\_img(path,target\_size = (180,180))

result = predict\_model(img)

return render\_template('res.html', image\_url=url\_for('static', filename=filename),result = result)

if \_\_name\_\_ == '\_\_main\_\_':

app.config['UPLOAD\_FOLDER'] = 'static'

app.run(debug=True)

1. **Performance Testing**
   1. **Performance Metrics**
2. Precision, Recall, Accuracy, F1

These metrics are widely used in binary classification where only two categories are taken into consideration. In multi classification solutions they might be calculated in multiple ways, but the most popular is to calculate them as the average of every single metric across all classes. Precision represents proportion of predicted positives that are truly positive. Values closer to 1 means high precision and shows that there is a small number of false positives.

Precision = True Positives

True Positives + False Positives

Recall is calculated as a proportion of actual positives that have been classified

correctly. Values closer to 1 means high recall and shows that there is a small

number of false negatives.

Recall = True Positives

True Positives + False Negatives

Accuracy measures proportion of number of correct predictions to total number of samples. It helps to detect over-fitting problem (models that overfit have usually an accuracy of 1).

Accuracy = Correct Predictions

Total Predictions

F1 Score combines precision and recall metrics by calculating their harmonic

mean.

F1=2 ∗ Precision ∗ Recall

Precision + Recall

1. Log-Loss, Log-Loss Reduction

Logarithmic loss quantifies the accuracy of a classifier by penalizing incorrect

classifications. This value shows uncertainty of prediction using probability estimates for each class in the dataset. Log-loss increases as the predicted probability diverges from the actual label. Maximizing the accuracy of the classifier causes minimizing this function.

Logarithmic loss reduction (also called reduction in information gain - RIG)

gives a measure of how much improves on a model that gives random prediction.Value closer to 1 means a better model.

1. Confusion Matrix, Micro-averages, Macro-averages

Confusion matrix contains precision and recall for each class in multi-class classification problem.

A macro-average computes the metric independently for each class and then

take the average (treats all classes equally).

A micro-average aggregates the contributions of all classes to compute the

average metric.

Micro- and macro-averages may be applied for every metric.

In a multi-class classification problem, micro-average is preferred because

there might be class imbalance (significant difference between number of class

examples).

1. **Advantages and Disadvantages**
   1. **Advantages**

Advantages for weather forecasting in agriculture:

For example, weather forecasting enables you to properly plan your farm operations, such as planting, irrigation, fertilizer application, pruning/weeding, harvesting or livestock mating, since farming and agriculture as a whole chiefly depend on seasons and weather.

* 1. **Disadvantages**

Model Limitations: Forecasting models can only make predictions based on existing data and are limited by the quality and quantity of that data. Limited Time Frame: Forecasts are usually only accurate for a short time frame, making it difficult to plan ahead.

1. **Conclusion**

In summary, weather forecasts are increasingly accurate and useful, and their benefits extend widely across the economy.

1. **Future Scope**

In addition to predictions of atmospheric phenomena themselves, weather forecasting includes predictions of changes on the Earth's surface climate. These changes are caused by atmospheric conditions like snow and ice cover, storm tides, and floods.

1. **Appendix**
   1. **Source Code**

from flask import Flask, render\_template, request, url\_for

import os

from flask import flash,redirect

from werkzeug.utils import secure\_filename

import os

import shutil

from PIL import Image

from tensorflow import keras

from keras.layers import Dense

from keras.models import Sequential, load\_model

from tensorflow.keras.preprocessing import image

import numpy as np

def predict\_model(img):

model = load\_model('weathermodel.h5')

x = image.img\_to\_array(img)

x = np.expand\_dims(x,axis = 0)

preds = model.predict(x)

pred = np.argmax(preds,axis = 1)

index = ['cloudy','foggy','rainy','sunshine','sunrise']

result = str(index[pred[0]])

return(result)

def resize\_image(image\_path, width, height):

with Image.open(image\_path) as img:

resized\_img = img.resize((width, height))

resized\_img.save(image\_path)

app = Flask(\_\_name\_\_)

app.secret\_key = 'supersecretkey'

@app.route('/')

@app.route('/index.html')

def index():

return render\_template('index.html')

@app.route('/about.html')

def about():

return render\_template('about.html')

@app.route('/pictures.html')

def pictures():

return render\_template('pictures.html')

@app.route('/upload.html')

def up():

return render\_template('upload.html')

@app.route('/upload', methods=['POST'])

def upload():

file = request.files['file']

filename = secure\_filename(file.filename)

print(filename)

file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

path = 'static/{}'.format(filename)

resize\_image(path,300,300)

img = image.load\_img(path,target\_size = (180,180))

result = predict\_model(img)

return render\_template('res.html', image\_url=url\_for('static', filename=filename),result = result)

if \_\_name\_\_ == '\_\_main\_\_':

app.config['UPLOAD\_FOLDER'] = 'static'

app.run(debug=True)

* 1. **GitHub & Project Demo Link**

https://github.com/smartinternz02/SI-GuidedProject-609761-1700581963