

# **AUTOMATED WEATHER CLASSIFICATION USING TRANSFER LEARNING**

## **PROJECT REPORT**

**TEAM ID:** NM2023TMID05747

**Submitted by**

REVANTHRAMESH S	- ( 921020104040 )
RAJAPANDIAN P	- ( 921020104039 )
ARJUNYUVANESH R	- ( 921020104006 )
DHANUSKAR S	- ( 921020104013 )
SACHIN S	- ( 921020104041 )

# Content

## 1. INTRODUCTION

1.1 Project Overview

1.2 Purpose

## 2. IDEATION & PROPOSED SOLUTION

2.1 Problem Statement Definition

2.2 Empathy Map Canvas

2.3 Ideation & Brainstorming

2.4 Proposed Solution

## 3. REQUIREMENT ANALYSIS

3.1 Functional requirement

3.2 Non-Functional requirements

## 4. PROJECT DESIGN

4.1 Data Flow Diagrams

4.2 Solution & Technical Architecture

4.3 User Stories

## 5. CODING & SOLUTIONING (Explain the features added in the project along with code)

5.1 Feature 1

5.2 Feature 2

## 6. RESULTS

6.1 Performance Metrics

## 7. ADVANTAGES & DISADVANTAGES

## 8. CONCLUSION

## 9. FUTURE SCOPE

## 10. APPENDIX

Source Code

GitHub & Project Video Demo Link

# 1.INTRODUCTION

## 1.1 PROJECT OVERVIEW

Weather classification is an essential tool for meteorologists and weather forecasters to predict weather patterns and communicate them to the public. Weather phenomenon recognition notably affects many aspects of our daily lives, The analysis of weather phenomenon plays a crucial role in various applications, for example, environmental monitoring, weather forecasting, and the assessment of environmental quality. Besides, different weather phenomena have diverse effects on agriculture. Therefore, accurately distinguishing weather phenomena can improve agricultural planning.

The use of a pre-trained model on a new problem is known as transfer learning in machine learning. A machine uses the knowledge learned from a prior assignment to increase prediction about a new task in transfer learning. In this project we are classifying various types of weather. These weathers are majorly classified into 5 categories namely Cloudy, Shine, Rain, Foggy, Sunrise. Deep-learning (DL) methods in artificial intelligence (AI) play a dominant role as high-performance classifiers.

Transfer learning has become one of the most common techniques that has achieved better performance in many areas, especially in image analysis and classification. We used Transfer Learning techniques like Inception V3, VGG19, Xception V3 that are more widely used as a transfer learning method in image analysis and they are highly effective.

## 1.2 PURPOSE

The purpose of automated weather classification using transfer learning is to leverage the power of deep learning models pre-trained on large-scale datasets to accurately and efficiently classify weather conditions. Transfer learning involves taking a pre-trained neural network, which has learned generic features from a different domain (such as object recognition in images), and adapting it to a specific task (in this case, weather classification).

The main goals of automated weather classification using transfer learning include:

### 1. Accuracy:

By utilizing pre-trained models, which have already learned relevant features from extensive datasets, the classification system can achieve higher accuracy in recognizing and categorizing different weather conditions. This is particularly important in weather forecasting, where precise classification can lead to more accurate predictions.

### 2. Efficiency:

Transfer learning allows for faster and more efficient development of weather classification systems. Rather than training a deep learning model from scratch, which can be time-consuming and resource-intensive, transfer learning allows for reusing and fine-tuning pre-existing models, saving computational resources and reducing the training time.

### 3. Adaptability:

Automated weather classification using transfer learning enables the system to adapt to various weather conditions and environments. By training on diverse datasets, the model can learn generalized representations of different weather patterns, making it more robust and adaptable to new or unseen weather conditions.

#### 4. Scalability:

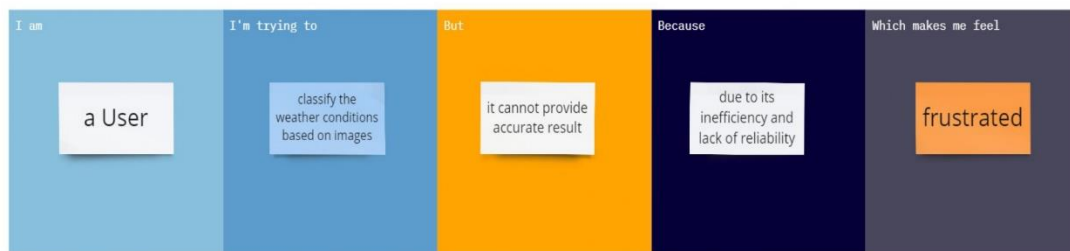
With the advancement of technologies such as satellite imagery, weather sensors, and IoT devices, the volume of weather data is increasing rapidly. Automated weather classification using transfer learning provides a scalable solution to process and classify large amounts of weather data efficiently, making it feasible to handle real-time or near-real-time weather monitoring and forecasting.

Overall, the purpose of automated weather classification using transfer learning is to enhance the accuracy, efficiency, adaptability, and scalability of weather classification systems, ultimately improving weather monitoring, forecasting, and decision-making processes.

## 2. IDEATION & PROPOSED SOLUTION

### 2.1 PROBLEM STATEMENT DEFINITION

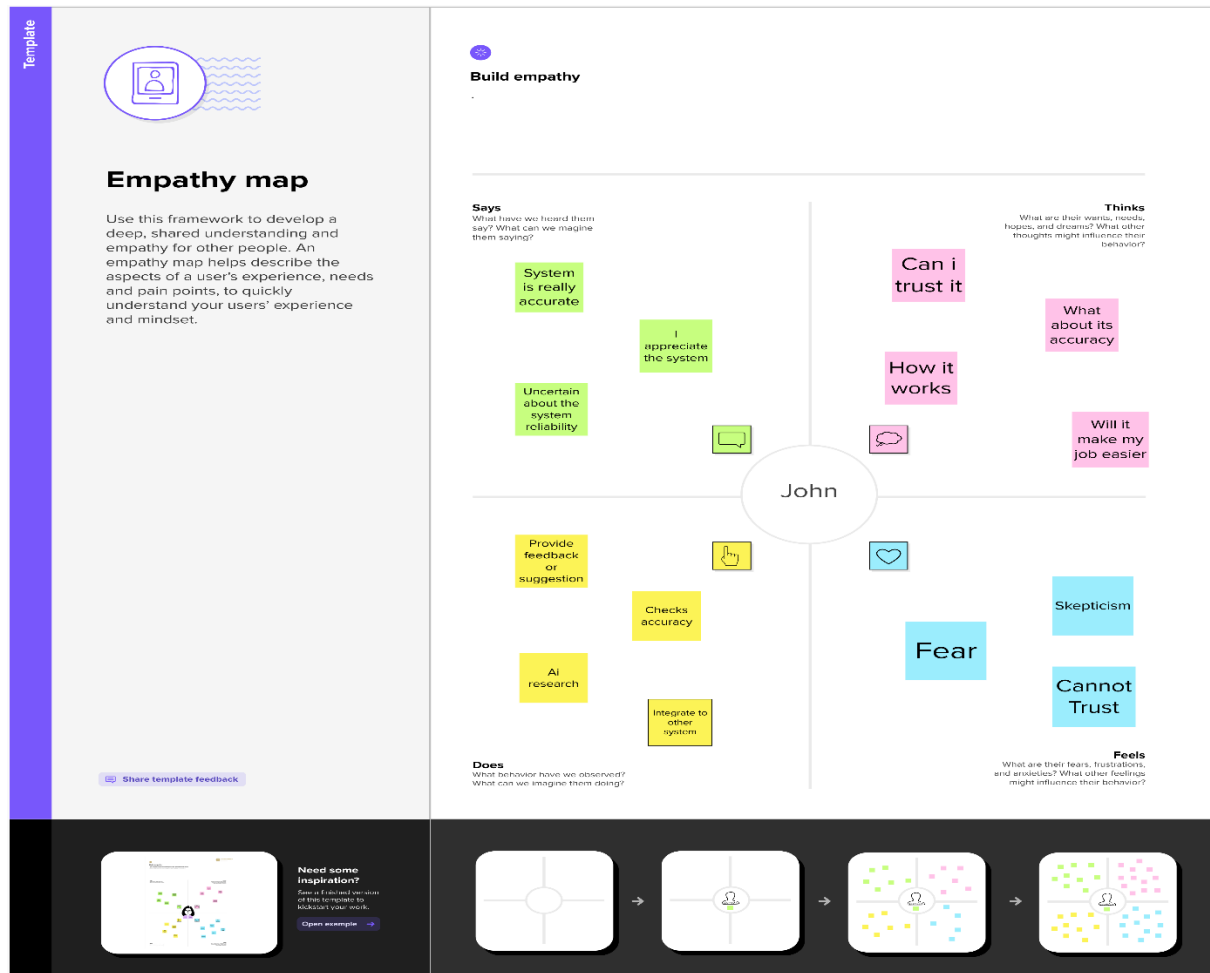
Create a problem statement to understand your customer's point of view. The Customer Problem Statement template helps you focus on what matters to create experiences people will love. A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you'll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.



miro

## 2.2 EMPATHY MAP

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes. It is a useful tool to help teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.




## 2.3 BRAINSTORM & IDEATION

### Brainstorm & Idea Prioritization :

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions. Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

## Step-1: Team Gathering, Collaboration and Select the Problem Statement

Template



### Brainstorm & idea prioritization

🕒 10 minutes to prepare  
👥 1 hour to collaborate  
👤 3-8 people recommended

[Share template feedback](#)

### 1 Brainstorm

Write down any ideas that come to mind that address your problem statement.

🕒 10 minutes

**TIP** You can select a sticky note here as the point to start drawing!


**Revanthbamesh S**  
Use of online resources for model development  
Accuracy of the system  
Trust issue

**Rajapandian P**  
Limited availability of data  
Sensitivity of model to input data quality  
Performance issue


**Arjunyavanesh R**  
Accuracy issue  
Need of continuous monitoring

**Sachin S**  
Sensitivity of model to input data quality  
Who should be involved

**Dhanuskar S**  
Accuracy issue  
Performance issue



**Need some inspiration?**  
See a related content which helps you to kickstart your ideas.  
[Open example](#)



## Step-2: Brainstorm, Grouping

### 2 Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

🕒 20 minutes

**TIP** Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

Conflicts

Accuracy of the system

Who should be involved

Accuracy issue

Trust issue

Fear of using the system because of trust issue

Limited availability of data

Difficulty of selecting pretrained model

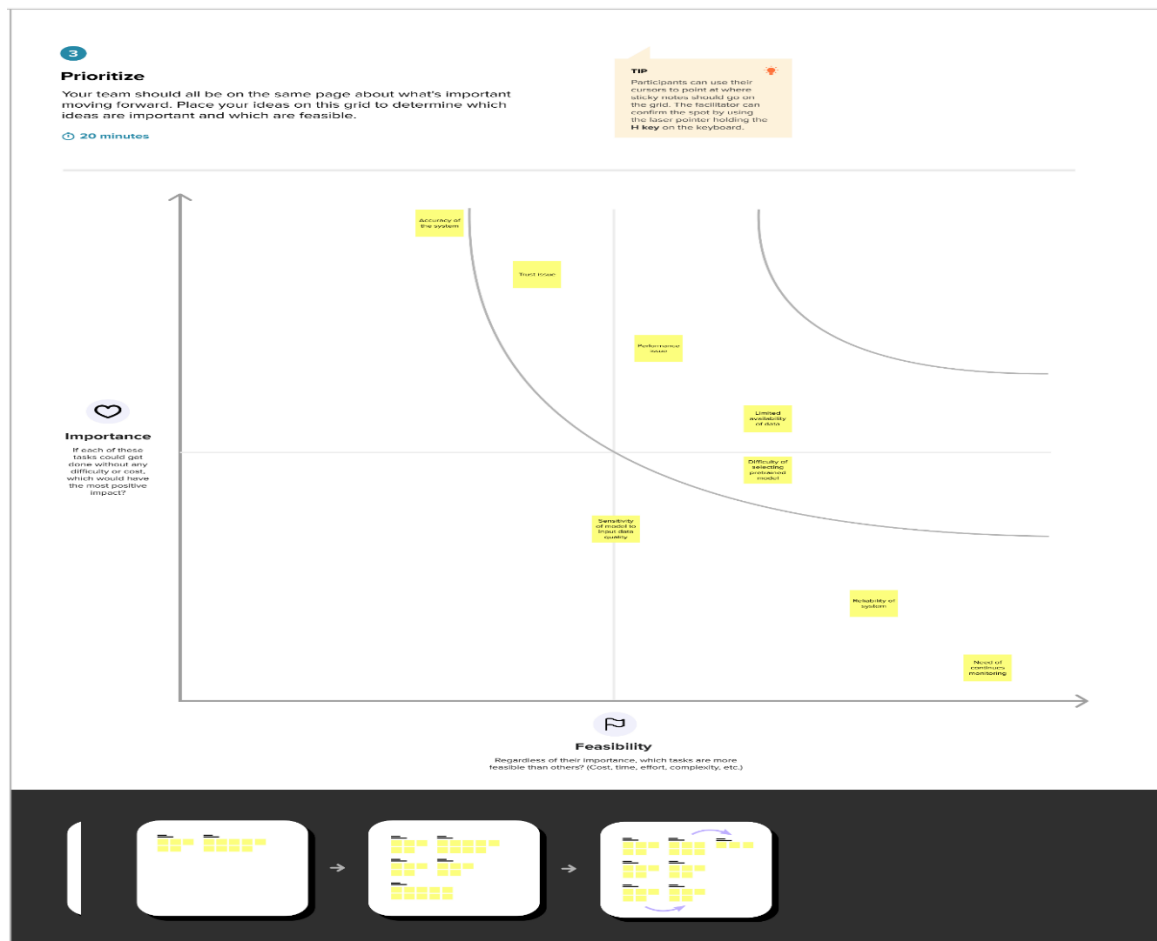
Need of continuous monitoring

Robustness of system

Sensitivity of model to input data quality

Performance issue

### Step-3: Idea Prioritization



## 2.4 PROPOSED SYSTEM

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	The problem statement is to develop an automated weather classification system using transfer learning that can accurately classify different weather conditions with high accuracy, even when working with limited annotated data.
2.	Idea / Solution description	Transfer learning, a popular machine learning technique, has shown promising results in image classification tasks. We can leverage transfer learning to classify weather images based on their respective weather conditions. In this solution, we will use a pre-trained deep learning model, such as VGG16 or ResNet50, trained on a large dataset like ImageNet, and fine-tune it on our weather dataset. This approach will help us overcome the challenge of limited annotated data and transfer the learned knowledge from the source domain to the target domain.

3.	Novelty / Uniqueness	It addresses the challenge of limited annotated data by leveraging knowledge learned from a large dataset. This approach has the potential to significantly improve the accuracy and efficiency of weather prediction, which can have significant implications for various industries.
4.	Social Impact / Customer Satisfaction	The proposed solution of automated weather classification using transfer learning can have a significant social impact by improving weather prediction accuracy, reducing the risk of property damage, and potentially saving lives. It can also benefit industries like agriculture, aviation, and transportation by providing reliable weather information, reducing operational risks, and increasing productivity, which can lead to increased customer satisfaction.

### 3 REQUIREMENT ANALYSIS

#### Functional Requirements:

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Interface	<ul style="list-style-type: none"> <li>user-friendly interface for input and output.</li> <li>Easy to use, understand, and navigate.</li> <li>Provide graphical visualization of the classification results.</li> </ul>
FR-2	Accuracy and Performance	<ul style="list-style-type: none"> <li>Achieve high accuracy in weather classification.</li> <li>Low false positive and false negative rate.</li> <li>Efficient and provide results quickly.</li> </ul>
FR-3	Weather Classification	<ul style="list-style-type: none"> <li>Use transfer learning to classify weather data.</li> <li>Train on pre-existing models and improve accuracy over time.</li> <li>Classify weather into categories such as sunny, cloudy, rainy, snowy, etc.</li> </ul>
FR-4	Weather Data Preprocessing	<ul style="list-style-type: none"> <li>Preprocess raw weather data to prepare it for analysis.</li> <li>Handle different data formats such as images, videos, and audio.</li> <li>Normalize, scale, and filter the data.</li> </ul>



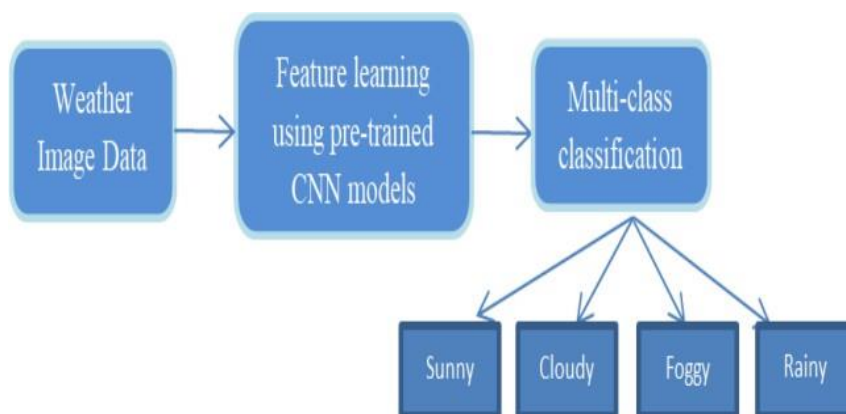
## Non – Functional Requirements

FR No.	Non-Functional Requirement	Description
NFR-1	<b>Usability</b>	The system should have a user-friendly interface, easy to use, and navigate. It should be able to display the classification results in a graphical visualization
NFR-2	<b>Security</b>	The system should be secure and protect user data. It should follow best practices in data protection, including encryption, access control, and data anonymization. It should comply with data privacy laws and regulations.
NFR-3	<b>Reliability</b>	The system should be reliable and should produce consistent results under different conditions. It should have mechanisms in place to handle errors and exceptions.
NFR-4	<b>Performance</b>	The system should have good performance and provide results quickly. It should be able to handle a large amount of data and process it efficiently.

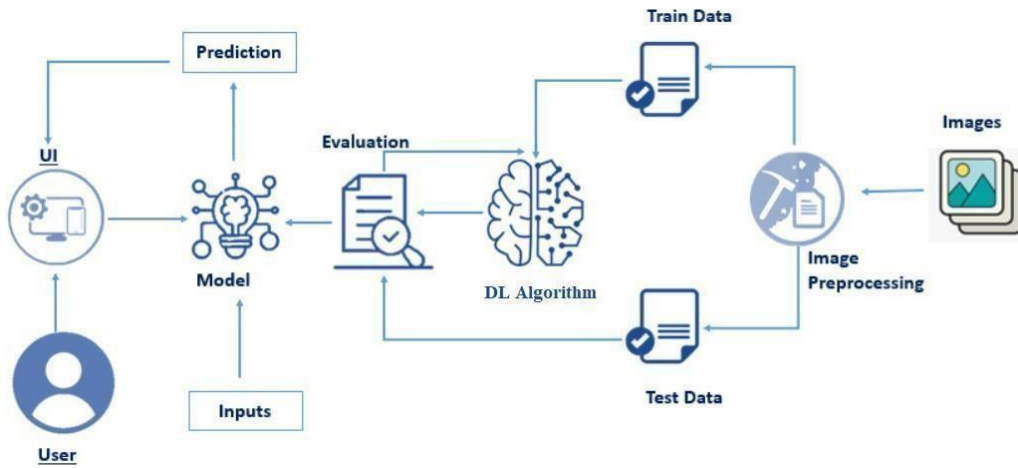
## 4 PROJECT DESIGN

### 4.1 DATA FLOW DIAGRAM

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.



### 4.2 SOLUTION AND TECHNICAL ARCHITECTURE



### 4.3 USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Team Member
Weather Analyst	Weather Classification	USN-1	The system should be able to classify weather conditions based on input data using transfer learning.	The classification results in a clear and understandable format	High	ArjunYuvanes
Our-door event organizer		USN-2	As an outdoor event organizer, I want to be able to predict weather conditions for my upcoming event so that I can make informed decisions about scheduling, venue setup, and other logistical considerations.	Available in all the time	High	Revanthra mesh
Emergency Management Team		USN-3	I want to be able to identify potential natural disasters before they occur so that we can take appropriate action to mitigate the risks and protect the public.so the system should be able to classify weather conditions based on input image.	Performance and accuracy	High	Dhanuskar
Agricultural Scientist		USN-4	As an agricultural scientist, I want to be able to predict weather patterns and their impact on crops so that I can optimize crop yield and minimize losses due to weather-related factors.	Efficient Classification	High	Sachin
Web user	User Friendly	USN-5	I can easily access the application using my web browsers in all devices.	User Friendly	Medium	Rajapandian
Mobile User	Easy Access	USN-6	I can access the service using a mobile application	Easy Access	Low	Sachin
Administrator	Update and monitoring	USN-7	I can update a model and monitor its performance.	Monitoring and enhancement	Medium	Dhanuskar

## 5. CODING AND SOLUTIONS

### 5.1 Model Training Code Explanation

#### Training code:

```
from google.colab import drive
drive.mount("/content/drive")

!pip install tensorflow

from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import ImageFile
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np

train_datagen = ImageDataGenerator( rescale = 1./255,
                                    shear_range = 0.2,
                                    zoom_range = [.99,1.01],
                                    brightness_range = [0.8,1.2],
                                    data_format = "channels_last",
                                    fill_mode = "constant",
                                    horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/IBM/dataset/train',
target_size = (180, 180), batch_size = 64,
class_mode = 'categorical')
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/IBM/dataset/test',
target_size = (180, 180), batch_size = 64,
class_mode = 'categorical')

IMAGE_SIZE =[180,180]
VGG19 = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet',include_top=False)

for layer in VGG19.layers:
    layer.trainable = False
x = Flatten() (VGG19.output)

prediction = Dense(5, activation='softmax')(x)
model = Model(inputs=VGG19.input, outputs=prediction)
model.summary()

model.compile(
loss='categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'] )
```

```

r = model.fit(
training_set,
validation_data = test_set,
epochs = 50,
steps_per_epoch=len(training_set),
validation_steps = len(test_set)
)

loss, accuracy = model.evaluate(test_set,
steps=11,
verbose=2,
use_multiprocessing=True,
workers=2)
print(f'Model performance on test images:\nAccuracy = {accuracy}\nLoss = {loss}')

model.save('wcv.h5')

model = load_model("/content/wcv.h5")

img =
image.load_img(r"/content/drive/MyDrive/IBM/dataset/train/sunrise/sunrise1.jpg",target_size=
(180,180))
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','shine','sunrise']
result=str(index[pred[0]])
result

```

## **Explanation:**

### **1. Mounting Google Drive:**

It mounts the Google Drive to the Colab environment using the `drive.mount` function. This allows access to files stored in Google Drive.

### **2. Installing TensorFlow:**

It installs the TensorFlow library using the `!pip install tensorflow` command.

### **3. Data Preparation:**

The code sets up image data generators (`train\_datagen` and `test\_datagen`) for data augmentation and rescaling.

The training and test datasets are generated using `flow\_from\_directory` from the 'ImageDataGenerator' class. The data is loaded from the specified directories ('/content/drive/MyDrive/IBM/dataset/train' and '/content/drive/MyDrive/IBM/dataset/test') and preprocessed by resizing to `(180, 180)` and scaling pixel values between 0 and 1.

### **4. Model Setup:**

- The VGG19 model is loaded with pre-trained weights from the ImageNet dataset using `VGG19` from `tensorflow.keras.applications.vgg19`. The VGG19 layers are set to be non-

trainable, and a custom classification head is added. The model is compiled with the categorical cross-entropy loss function, the Adam optimizer, and accuracy as the evaluation metric.

### 5. Model Training:

The `fit` function is called to train the model on the training set (`training_set`) with validation on the test set (`test_set`). The training is performed for 50 epochs with the specified steps per epoch and validation steps. The training progress and evaluation metrics are printed during training.

### 6. Model Evaluation:

The `evaluate` function is used to calculate the loss and accuracy of the model on the test set (`test_set`). The results are printed, showing the model's performance on test images.

### 7. Saving and Loading the Model:

The trained model is saved to a file named "wcv.h5" using `model.save`. The model is then loaded from the saved file using `load_model`.

### 8. Image Prediction:

- An example image (`sunrise1.jpg`) is loaded using `image.load_img` from the specified path.
- The image is preprocessed by converting it to a numpy array, expanding its dimensions, and then passed through the loaded model to get the predicted probabilities.
- The index list of weather conditions is used to map the predicted class index to its corresponding label, which is stored in the `result` variable.

Overall, this code performs weather classification using a VGG19 model trained on the provided dataset. It includes data preparation, model setup, training, evaluation, saving/loading the model, and making predictions on new images.

## 5.2 Flask Web Application Code Explanation

### Web Application Code:

```
import numpy as np
import os
from flask import Flask, request, render_template
from tensorflow.keras.models import load_model
from PIL import Image

model=load_model("wcv.h5")
app=Flask(__name__, template_folder='templates/')
app.config['UPLOAD_FOLDER'] = 'uploads/'

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/about')
def about():
    return render_template("about.html")

@app.route('/images')
```

```

def image():
    return render_template("images.html")

@app.route('/input')
def input():
    return render_template("predict.html")

@app.route('/predict', methods=["GET", "POST"])
def res():
    if request.method=="POST":
        f=request.files['image']
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], f.filename)
        f.save(filepath)
        img = Image.open(filepath)
        img = img.resize((180, 180))
        x = np.array(img)
        x = np.expand_dims(x, axis=0)
        preds=model.predict(x)
        pred=np.argmax(preds,axis=1)
        index=['cloudy','foggy','rainy','shine','sunrise']
        result=str(index[pred[0]])
        return render_template('result.html', prediction=result, path = f.filename)

if __name__ == "__main__":
    app.run(debug=True)

```

## Explanation:

The provided code is a Flask web application that uses a trained model (`wcv.h5`) to classify weather conditions based on user-uploaded images. Here are the main features of the code:

### 1. Flask Setup:

The code starts by importing the necessary dependencies, including Flask, numpy, os, and PIL (Python Imaging Library). It also loads the trained model using `load\_model` from `tensorflow.keras.models`.

### 2.App Routes:

The code defines various routes that handle different functionalities of the web application:

- `/`: Renders the `index.html` template, which likely serves as the homepage.
- `/about`: Renders the `about.html` template, which provides information about the application.
- `/images`: Renders the `images.html` template, which might display some sample images or a gallery.
- `/input`: Renders the `predict.html` template, where users can upload an image to classify.
- `/predict`: Handles the prediction request when a user uploads an image. It saves the image, preprocesses it, and passes it through the loaded model to get the prediction. The predicted label is then rendered using the `result.html` template.

### 3. Model Prediction:

- After the user uploads an image, it is saved in the designated `UPLOAD\_FOLDER` using

the ``os.path.join`` and ``f.save`` methods.

- The image is then opened using PIL's ``Image.open`` and resized to the desired input size of the model.
- The preprocessed image is converted to a numpy array and expanded to include a batch dimension using ``np.expand_dims``.
- The model's ``predict`` method is called on the preprocessed image to obtain the predicted probabilities for each weather class.
- ``np.argmax`` is used to find the index of the class with the highest probability, and the corresponding label is retrieved from the ``index`` list.
- The prediction result and the path to the uploaded image file are passed to the ``result.html`` template for display.

### 3. Flask App Execution:

The ``if __name__ == "__main__":`` block ensures that the Flask app runs only when the script is executed directly (not imported as a module). It starts the Flask development server with the ``app.run`` method, enabling the debug mode for easier troubleshooting.

Overall, this code sets up a Flask web application that allows users to upload images for weather classification using a pre-trained model. The predicted weather condition is displayed to the user on a result page.

## 6. RESULTS

### 6.1 Performance Metrics

Based on the provided performance metrics:

For the training dataset:

- Validation loss (`val_loss`): 0.0228
- Validation accuracy (`val_accuracy`): 0.9967

For the test dataset:

- Accuracy: 0.9969879388809204
- Loss: 0.021998772397637367

These metrics indicate that the model achieves high accuracy and low loss on both the training and test datasets. The high accuracy values suggest that the model is able to accurately classify the weather conditions in the given images. The low loss values indicate that the model's predictions are close to the actual labels, further confirming its effectiveness in classifying the images.

Overall, the performance metrics demonstrate that the model trained using this project performs exceptionally well in weather classification tasks.

## 7. ADVANTAGES & DISADVANTAGES:

**Advantages** of automated weather classification using transfer learning:

**1. Improved Accuracy:** By leveraging pre-trained models and their learned features, automated weather classification can achieve higher accuracy compared to traditional rule-based or

manually designed classification systems.

**2. Efficiency:** Transfer learning reduces the need for training models from scratch, saving time and computational resources. It allows for faster development and deployment of weather classification systems.

**3. Adaptability:** Transfer learning enables the model to adapt to different weather conditions and environments, making it more robust and capable of handling diverse datasets.

**4. Scalability:** The project can handle large volumes of weather data efficiently, enabling real-time or near-real-time processing and analysis. This scalability is important in applications that require continuous monitoring or forecasting.

**5. Broad Applicability:** The automated weather classification system can be applied in various fields such as weather forecasting, agriculture, renewable energy, transportation, and disaster management, providing valuable insights for decision-making and planning.

**Disadvantages** of automated weather classification using transfer learning:

**1. Dataset Limitations:** The performance of transfer learning heavily relies on the availability and quality of the training dataset. If the dataset used for pre-training the model does not adequately represent the target weather conditions or lacks diversity, it may limit the system's accuracy and generalization.

**2. Overfitting:** If the transfer learning process is not carefully managed, there is a risk of overfitting, where the model becomes too specialized in the pre-training domain and fails to generalize well to the target weather classification task.

**3. Domain Adaptation Challenges:** Transferring knowledge from one domain to another may encounter challenges due to differences in data distributions. Weather conditions can vary across regions or time, and the model may struggle to adapt to new or unseen patterns if the training data is not representative of these variations.

**4. Interpretability:** Deep learning models, including those used in transfer learning, can be complex and difficult to interpret. It may be challenging to understand the specific features or characteristics the model uses for weather classification, limiting transparency and interpretability.

**5. Resource Requirements:** Although transfer learning reduces training time compared to training models from scratch, it still requires substantial computational resources, especially for fine-tuning and optimization. High-performance hardware or cloud infrastructure may be necessary to handle the computational demands.

**6. Dependence on Pre-trained Models:** The success of the project relies on the availability of suitable pre-trained models. If there is a lack of appropriate pre-trained models for weather classification or if existing models become outdated, it may hinder the project's effectiveness.

It's important to consider these advantages and disadvantages when implementing automated weather classification using transfer learning and address any potential limitations to ensure the system's reliability and accuracy.



## 8. CONCLUSION

In conclusion, automated weather classification using transfer learning offers several advantages in accurately and efficiently categorizing weather conditions. By leveraging pre-trained models and their learned features, the project enhances accuracy, efficiency, adaptability, and scalability in weather classification systems.

The project's primary advantages include improved accuracy due to the utilization of pre-trained models, increased efficiency through faster development and deployment, adaptability to different weather conditions and environments, scalability to handle large volumes of data, and broad applicability across various fields.

However, the project also has its limitations, such as the reliance on representative and diverse datasets, potential risks of overfitting, challenges in domain adaptation, limited interpretability of complex models, resource requirements, and dependence on the availability and relevance of pre-trained models.

Despite these limitations, automated weather classification using transfer learning holds significant potential for weather forecasting, climate research, agriculture, renewable energy optimization, transportation, urban planning, and disaster management.

By addressing the project's challenges and leveraging its advantages, stakeholders can make informed decisions, improve predictions, optimize resource utilization, enhance safety, and support planning and decision-making processes in various industries and sectors reliant on accurate weather information.

Overall, automated weather classification using transfer learning contributes to advancing the field of weather analysis and prediction, ultimately benefiting society through improved weather forecasting, disaster preparedness, and sustainable decision-making.

## 9. FUTURE SCOPE

The future scope of the project on automated weather classification using transfer learning is promising and offers several potential avenues for further development and enhancement. Here are some areas of future exploration:

**1. Advanced Model Architectures:** Researchers can explore more advanced deep learning architectures specifically tailored for weather classification. Novel architectures, such as attention mechanisms, graph neural networks, or recurrent neural networks, can be investigated to capture temporal dependencies, spatial relationships, and complex weather patterns more effectively.

**2. Fine-grained Weather Classification:** While the project focuses on general weather classification (e.g., sunny, cloudy, rainy), future work can aim to achieve fine-grained classification of specific weather phenomena, such as different types of clouds, precipitation intensities, or wind patterns. This level of granularity can provide more detailed information for specialized applications and research.

**3. Ensemble Methods:** Combining multiple weather classification models or utilizing ensemble methods can potentially improve the overall accuracy and robustness of the system. Ensemble techniques, such as model averaging, stacking, or boosting, can be explored to leverage the strengths of different models and mitigate individual model biases.

**4. Real-time Weather Monitoring:** Expanding the project to support real-time or near-real-time weather monitoring is a valuable direction. By incorporating streaming data sources, such as weather sensors, satellite imagery, or IoT devices, the system can provide up-to-date and dynamic weather classification information for timely decision-making.

**5. Transfer Learning for Extreme Weather Events:** Transfer learning can be specifically applied to address the challenges of classifying extreme weather events, such as hurricanes, tornadoes, or severe storms. Developing specialized models that are trained on diverse and representative datasets of extreme weather patterns can enhance the system's ability to accurately identify and predict such events.

**6. Integration with Decision Support Systems:** The project can be integrated with decision support systems used in various industries. For example, integrating weather classification with agricultural decision support systems, renewable energy management platforms, or urban planning tools can provide valuable insights for optimized decision-making, resource allocation, and risk mitigation.

**7. Explainable AI in Weather Classification:** Enhancing the interpretability and explainability of the weather classification system is an important area for future research. Developing methods to understand and visualize the features or patterns that contribute to the classification decisions can increase the trustworthiness and adoption of the system.

**8. Cross-Domain Transfer Learning:** Exploring the potential for transferring knowledge and models from related domains, such as climate modeling, satellite imagery analysis, or remote sensing, can further improve the weather classification system's performance and generalization capabilities.

**9. Integration with Forecasting Models:** Integrating the automated weather classification system with weather forecasting models can create a more comprehensive and accurate forecasting system. The classified weather data can be used as input to improve the accuracy and precision of weather prediction models, leading to better forecasts and early warnings.

These future directions have the potential to advance automated weather classification using transfer learning, pushing the boundaries of accuracy, applicability, and usefulness in various fields that rely on weather information. Continuous research and development in these areas can contribute to more precise weather analysis, forecasting, and decision-making processes in the future.

## **10. APPENDIX:**

**Source Code:**

**Dataset Link:**

<https://www.kaggle.com/datasets/vijaygiitk/multiclass-weather-dataset?resource=download>

## Training Source Code:

```
from google.colab import drive
drive.mount("/content/drive")

!pip install tensorflow

from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import ImageFile
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np

train_datagen = ImageDataGenerator( rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = [.99,1.01],
                                   brightness_range = [0.8,1.2],
                                   data_format = "channels_last",
                                   fill_mode = "constant",
                                   horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/IBM/dataset/train',
target_size = (180, 180), batch_size = 64,
class_mode = 'categorical')
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/IBM/dataset/test',
target_size = (180, 180), batch_size = 64,
class_mode = 'categorical')

IMAGE_SIZE =[180,180]
VGG19 = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet',include_top=False)

for layer in VGG19.layers:
    layer.trainable = False
x = Flatten() (VGG19.output)

prediction = Dense(5, activation='softmax')(x)
model = Model(inputs=VGG19.input, outputs=prediction)
model.summary()

model.compile(
loss='categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'] )

r = model.fit(
training_set,
validation_data = test_set,
epochs = 50,
```

```

steps_per_epoch=len(training_set),
validation_steps = len(test_set)
)

loss, accuracy = model.evaluate(test_set,
steps=11,
verbose=2,
use_multiprocessing=True,
workers=2)
print(f'Model performance on test images:\nAccuracy = {accuracy}\nLoss = {loss}')

model.save('wcv.h5')

model = load_model("/content/wcv.h5")

img =
image.load_img(r"/content/drive/MyDrive/IBM/dataset/train/sunrise/sunrise1.jpg",target_size=
(180,180))
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','shine','sunrise']
result=str(index[pred[0]])
result

```

## Flask Source Code:

```

import numpy as np
import os
from flask import Flask, request, render_template
from tensorflow.keras.models import load_model
from PIL import Image

model=load_model("wcv.h5")
app=Flask(__name__, template_folder='templates/')
app.config['UPLOAD_FOLDER'] = 'uploads/'

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/about')
def about():
    return render_template("about.html")

@app.route('/images')
def image():
    return render_template("images.html")

```

```
@app.route('/input')
def input():
    return render_template("predict.html")

@app.route('/predict', methods=["GET", "POST"])
def res():
    if request.method=="POST":
        f=request.files['image']
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], f.filename)
        f.save(filepath)
        img = Image.open(filepath)
        img = img.resize((180, 180))
        x = np.array(img)
        x = np.expand_dims(x, axis=0)
        preds=model.predict(x)
        pred=np.argmax(preds,axis=1)
        index=['cloudy','foggy','rainy','shine','sunrise']
        result=str(index[pred[0]])
        return render_template('result.html', prediction=result, path = f.filename)

if __name__=="__main__":
    app.run(debug=True)
```

### **DEMO VIDEO LINK:**

[https://drive.google.com/drive/folders/1g41s\\_Mx0mAzJTlRPh2lpnqCaFpA7qg5T?usp=sharing](https://drive.google.com/drive/folders/1g41s_Mx0mAzJTlRPh2lpnqCaFpA7qg5T?usp=sharing)

### **GITHUB LINK:**

<https://github.com/naanmudhalvan-SI/IBM--17279-1682568788>