

PROJECT REPORT

WEATHER

CLASSIFICATION

USING

DEEP LEARNING

TEAM ID: 591954

Table of Contents

1. INTRODUCTION.....	3
2. LITERATURE SURVEY.....	4
3. IDEATION & PROPOSED SOLUTION.....	5
4. REQUIREMENT ANALYSIS.....	6
5. PROJECT DESIGN.....	8
6. PROJECT PLANNING & SCHEDULING.....	10
7. CODING & SOLUTIONING.....	11
8. PERFORMANCE TESTING.....	14
9. RESULTS.....	15
10.ADVANTAGES & DISADVANTAGES.....	16
11.CONCLUSION.....	16
12.FUTURE SCOPE.....	17
13.APPENDIX.....	18

1. INTRODUCTION

Embarking on a transformative journey through the symphony of the atmosphere, deep learning weather classification is a pioneering field in meteorology. This innovative approach combines the complexity of weather models with the computing power of deep learning algorithms. Using massive meteorological data, this innovative method aims to unravel the enigmatic dance of clouds, winds and temperatures that reveal the complex tapestry of weather phenomena. Through the lens of neural networks, this effort seeks to transcend traditional weather forecasting and heralds a new era in which algorithms navigate the stormy seas of atmospheric dynamics, charting a course for unprecedented accuracy in forecasting and understanding the vagaries of nature.

1.1 Project Overview

The project aims to use deep learning techniques for weather classification and aims to create an advanced model that can accurately identify different weather patterns. By acquiring and pre-processing a variety of weather data that includes temperature, humidity, wind speed and more, the project delves into training and deep configuration of learning architectures. This includes experimenting with neural network settings, optimizing hyperparameters, and validating the model and accuracy with historical weather data. The ultimate goal is to develop a user-friendly user interface that enables real-time weather forecasting based on current weather data and offers a new type of solution for weather classification using state-of-the-art deep learning methods.

1.2 Purpose

Weather classification using deep learning serves the central purpose of revolutionizing meteorological forecasting by harnessing the power of advanced algorithms. Using neural networks to analyse and understand complex weather patterns, this approach aims to improve the accuracy and precision of weather forecasts. The use of deep learning makes it easier to identify and distinguish between different weather conditions, which provides more reliable and timely information to meteorologists, disaster response teams and various industries that rely on weather forecasts. Ultimately, this effort aims to improve early warning systems, optimize resource allocation and simplify decision-making processes for weather-related events.

2. LITERATURE SURVEY

2.1 Existing problems

- 1.The existing problem in weather classification using deep learning is the weather patterns exhibit high variability and complexity, making it challenging for models to capture all nuances accurately.
- 2.Weather data can be sparse, noisy, or have missing values, which can hinder the training and generalization of models.
- 3.This can lead to biased models favouring the majority class and performing poorly on minority classes.

2.2 References

<https://www.kaggle.com/datasets/vijaygiitk/multiclass-weather-dataset>

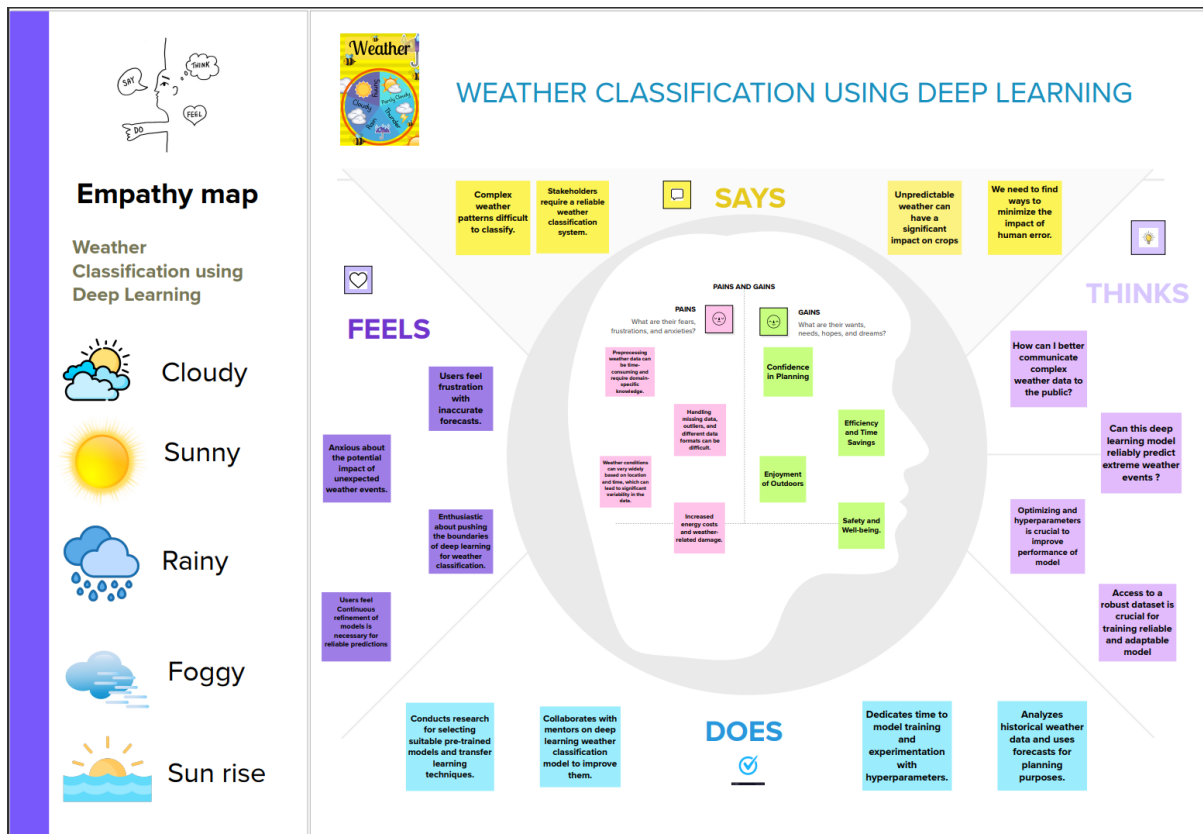
<https://www.sciencedirect.com/science/article/abs/pii/S0045790622004980>

2.3 Problem Statement Definition

How might we improve the accuracy and user-friendliness of weather forecasts to empower individuals to plan their activities with confidence and ensure their safety during changing weather conditions?

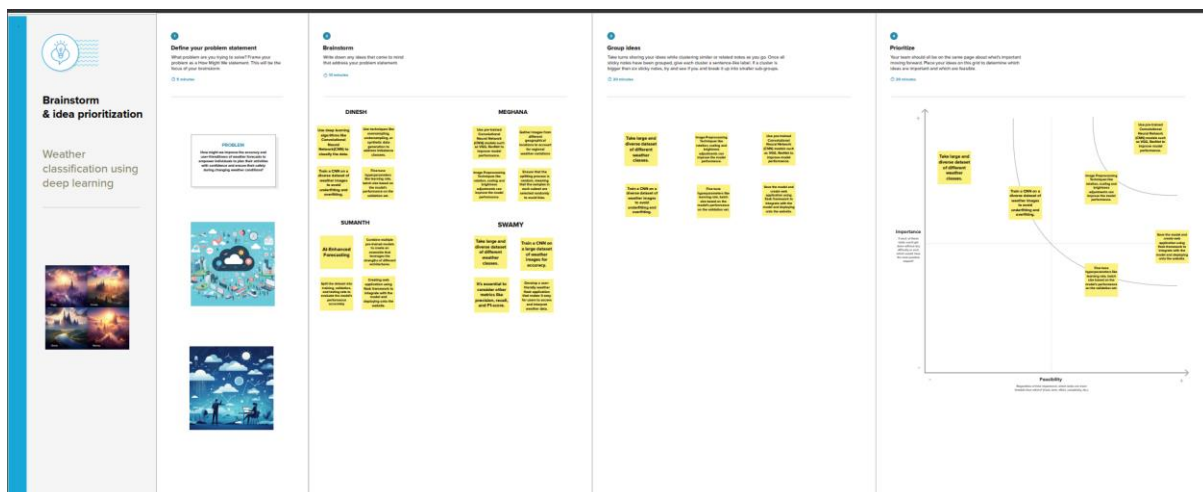
3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



<https://app.mural.co/t/aiml2389/m/aiml2389/1698753986350/481ad6e6b62622e1b70462b540a68b688232598f?sender=u3f6f08b6f9fcdc6d18192668>

3.2 Ideation & Brainstorming



<https://app.mural.co/t/aiml2389/m/aiml2389/1698927995836/02e8c5bf83efd845c451e4a837638a1fce855237?sender=u3f6f08b6f9fcdc6d18192668>

4. REQUIREMENT ANALYSIS

4.1 Functional requirements

1. **Data Acquisition and Preprocessing:** Gather diverse weather data from multiple sources like satellites, weather stations, or radar systems. Preprocess the data to handle missing values, normalize features, and potentially augment the dataset for balanced representation.

2. **Model Training and Architecture:** Develop deep learning architectures suitable for weather classification, leveraging CNNs, RNNs, or hybrid models. Train these models on labelled data, optimizing hyperparameters, and employing transfer learning for improved performance.

3. **Accuracy and Performance Metrics:** Ensure the model achieves high accuracy, measured through metrics like precision, recall, F1-score, and accuracy, validated across different weather conditions and geographic regions.

4. **Real-Time Processing and Scalability:** Design the model for real-time processing, considering scalability for handling large volumes of data efficiently.

5. **Robustness and Adaptability:** Create models that are robust to noise and uncertainties in weather data, capable of adapting to varying temporal and spatial patterns.

6. **Interpretability and Explainability:** Incorporate features to interpret and explain model decisions, aiding in understanding how the model arrives at specific weather classifications.

7. **Integration and Deployment:** Deploy the trained model into production environments, integrating it with decision support systems or applications for practical usage in sectors like agriculture, transportation, or disaster management.

8. Monitoring and Maintenance: Establish a system for continuous monitoring and maintenance of the deployed model, ensuring its relevance and accuracy over time by updating it with new data and evolving weather patterns.

4.2 Non-Functional requirements

1. Performance: The model should exhibit high performance, with low inference time and efficient resource utilization to handle large-scale weather data processing in real-time or near-real-time scenarios.

2. Scalability: It should be scalable to accommodate increasing data volumes or additional computational demands, ensuring consistent performance as the workload grows.

3. Robustness and Reliability: The model must be robust against variations in weather patterns, handling noisy or incomplete data while maintaining consistent accuracy across diverse weather conditions and geographical regions.

4. Adaptability: Ability to adapt to evolving weather patterns and technological advancements, allowing seamless integration of new data sources or model enhancements without disrupting existing functionalities.

5. Interpretability: Provide clear insights into the model's decision-making process, ensuring transparency and interpretability for users to understand and trust the model's classifications.

6. Usability and Accessibility: Design a user-friendly interface for easy integration and interaction with the model, catering to users with varying levels of technical expertise.

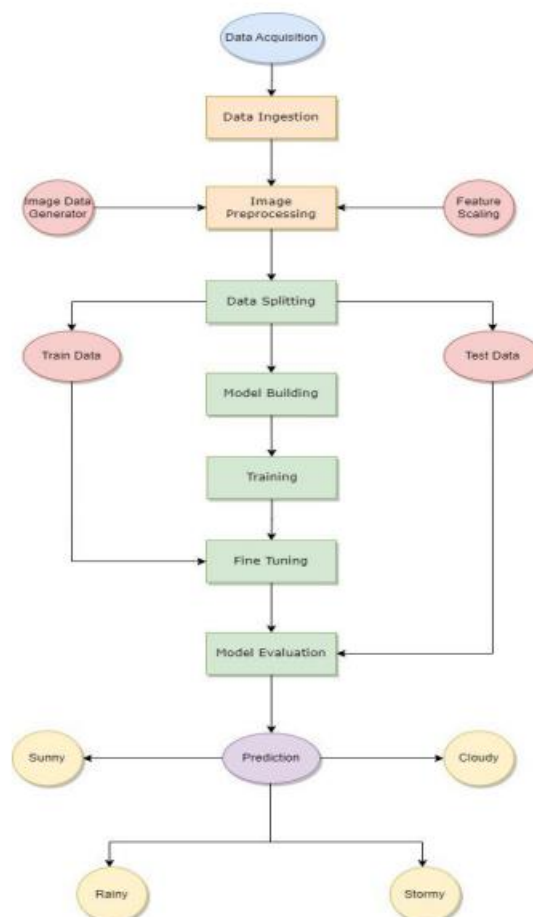
7.Security and Privacy: Implement measures to safeguard sensitive weather data, ensuring compliance with data protection regulations and preventing unauthorized access or misuse of information.

8.Maintainability and Upgradability: Facilitate easy maintenance and updates, allowing for the incorporation of new features, bug fixes, or model enhancements while minimizing downtime.

5. PROJECT DESIGN

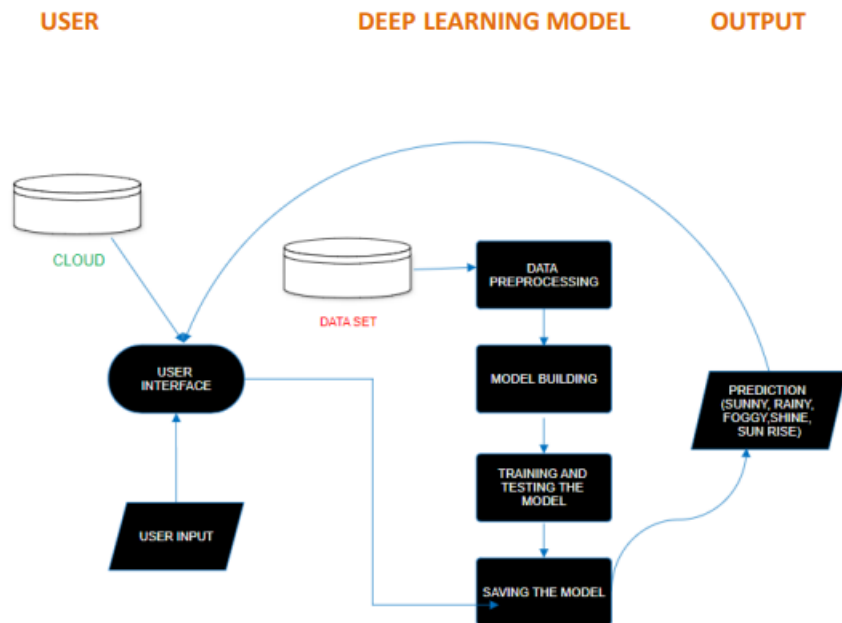
5.1 Data Flow Diagrams & User Stories

Data Flow Diagram:



User Stories:

- I want the option to contribute localized weather.



6.2 Sprint Planning & Estimation

Sprint	Total story points	Duration	Sprint Start Date	Sprint End Date	Sprint Goal
1	5	3 days	13 Nov 2023	15 Nov 2023	Data Collection
2	5	1 day	15 Nov 2023	15 Nov 2023	Data Preprocessing
3	10	2 days	15 Nov 2023	16 Nov 2023	Training Model
4	5	2 days	16 Nov 2023	17 Nov 2023	Model Evaluation
5	5	3 days	17 Nov 2023	19 Nov 2023	Model Testing
6	10	3 days	19 Nov 2023	21 Nov 2023	User interface

6.3 Sprint Delivery Schedule

Sprint	Sprint Release Date
1	16 Nov 2023
2	16 Nov 2023
3	17 Nov 2023
4	18 Nov 2023
5	20 Nov 2023
6	21 Nov 2023

7. CODING & SOLUTIONING

7.1 Data Augmentation

Data augmentation involves creating new training samples by applying transformations like rotation, flipping, zooming, etc., to the existing data. This helps in diversifying the dataset, reducing overfitting, and improving the model's generalization. The ImageDataGenerator from TensorFlow/Keras libraries, which performs various augmentation techniques on the fly while training the model.

```
from keras.preprocessing.image import ImageDataGenerator
```

```
#2.configure image data generator
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)
```

```
data generator functionality to train and test images
tagen.flow_from_directory(r"C:\Users\dines\Documents\AI_ML\PROJECT\Weather dataset\training",target_size=(224,224),batch_size=16)
gen.flow_from_directory(r"C:\Users\dines\Documents\AI_ML\PROJECT\Weather dataset\alien_test",target_size=(224,224),batch_size=16)
```

```
Found 1500 images belonging to 5 classes.
Found 30 images belonging to 5 classes.
```

```
print(train.class_indices)
```

```
{'cloudy': 0, 'foggy': 1, 'rainy': 2, 'shine': 3, 'sunrise': 4}
```

7.2 Transfer Learning

Transfer learning involves using a pre-trained neural network model on a related task and leveraging its learned features and weights to boost the performance of a new model for weather classification.

VGG 16

```
i]: #import model building libraries
    from tensorflow.keras.applications.vgg16 import VGG16
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras.layers import Flatten

r]: conv_base=VGG16(include_top=False,weights='imagenet',input_shape=(224,224,3))

}: conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080

block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

```

=====
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)

```

```

model=Sequential()
model.add(conv_base)
model.add(Flatten())
model.add(Dense(units=256,activation="relu"))#hidden layers
model.add(Dense(units=5,activation="softmax"))

```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6422784
dense_1 (Dense)	(None, 5)	1285

```

=====
Total params: 21138757 (80.64 MB)
Trainable params: 21138757 (80.64 MB)
Non-trainable params: 0 (0.00 Byte)

```

```
#let us freeze the conv_base because already trained
conv_base.trainable=False
```

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

```
model.fit(train,steps_per_epoch=len(train),epochs=10,validation_data=test,validation_steps=len(test))
```

8. PERFORMANCE TESTING

8.1 Performance Metrics

Using training and testing accuracy as performance metrics for weather classification via deep learning may provide a basic understanding of how well the model fits the data during training and how it generalizes to unseen data. Training accuracy measures the model's performance on the data it was trained on, indicating how well it's learning patterns within that dataset. Testing accuracy evaluates how accurately the model predicts unseen data, assessing its generalization capability. However, relying solely on these metrics might overlook potential issues like overfitting (high training accuracy but low testing accuracy), where the model memorizes training data rather than learning underlying patterns, leading to poor performance on new data. Incorporating additional metrics like precision, recall, or F1-score can offer a more comprehensive assessment of the model's performance, providing insights beyond just accuracy.

```
In [6]: # TRAINING ACCURACY
training_scores =model.evaluate(train)

# Print the training accuracy
print(f'Training Accuracy: {training_scores[1]*100:.2f}%')

94/94 [=====] - 255s 3s/step - loss: 0.0187 - accuracy: 0.9927
Training Accuracy: 99.27%
```

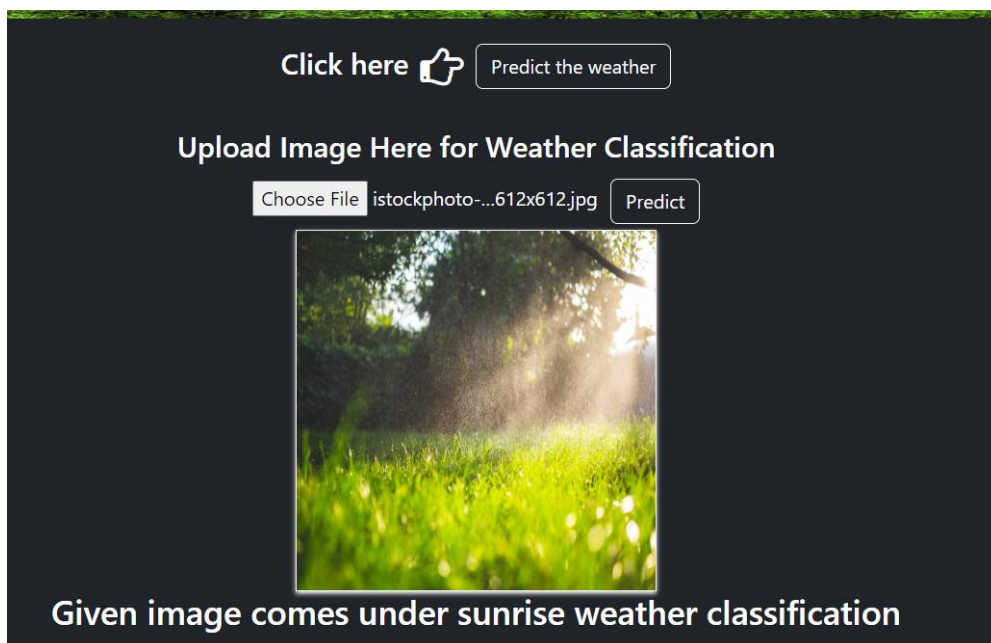
```
In [9]: # TESTING ACCURACY
testing_scores =model.evaluate(test)

# Print the training accuracy
print(f'Testing Accuracy: {testing_scores[1]*100:.2f}%')

2/2 [=====] - 4s 2s/step - loss: 0.2170 - accuracy: 0.9667
Testing Accuracy: 96.67%
```

9. RESULTS

9.1 Output Screenshots



10. ADVANTAGES & DISADVANTAGES

ADVANTAGES:

1. **Better accuracy:** Deep learning models can identify complex patterns in large and varied meteorological data, resulting in more accurate weather forecasts compared to traditional methods.

2. **Better prediction of extreme events:** These models are better able to capture and predict sudden changes or extreme weather events, which are crucial for early warning systems and disaster preparedness.

3. **Adaptability and learning:** Deep learning models can continuously learn and adapt to new data, improving their predictions over time as they are exposed to more data, improving predictive capabilities.

4. **Complex pattern recognition:** They excel at identifying complex, non-linear relationships in weather data, enabling the detection of subtle patterns that may be missed by conventional methods.

DISADVANTAGES:

1. **Data intensity:** Deep learning models often require massive amounts of high-quality data for training, which can be difficult and expensive to obtain, especially for specific weather phenomena.

2. **Computing requirements:** Training and maintaining deep learning models requires significant computing resources, including high-performance hardware (such as GPUs or TPUs), which may be unavailable or expensive for some applications.

3. **Interpretability:** Deep learning models can be considered "black boxes" and therefore, it is difficult to understand how they arrive at certain predictions, limiting the interpretation of the results by meteorologists and scientists.

4. **Overfitting and generalization:** Models can overfit certain data sets, which degrades performance when applied to unseen or slightly different conditions, affecting their generalization.

11. CONCLUSION

In summary, the use of deep learning models for weather classification offers enormous potential for accurate detection and forecasting of various weather patterns. Using convolutional neural networks (CNNs),

recurrent neural networks (RNNs), or hybrid architectures, these models demonstrate the ability to handle large and diverse data sources, including satellite images, radar data, and historical climate data. Despite the inherent complexity of weather models, deep learning models show promising opportunities for capturing temporal and spatial dependencies, handling data variability, and achieving commendable accuracy in weather classification. However, challenges remain, such as the need for robustness to noise, interpretability of decisions, scalability and integration of evolving data sources. Continued research and development is needed to fully exploit the potential of deep learning in weather classification. Addressing these challenges will not only improve the accuracy and reliability of weather forecasts, but also advance applications in fields as diverse as agriculture, disaster management and energy, ultimately aiding in informed decision-making and improving resilience to changing weather events. The continuous development of deep learning methods and the integration of industry knowledge promise a future where weather classification models will become indispensable tools for understanding and adapting to changing weather patterns.

12. FUTURE SCOPE

The future of weather classification using deep learning models has enormous potential in various dimensions. The advances aim to strengthen the robustness of the model against noisy data and rare events, while removing uncertainty, which is crucial for real-world applications. Integrating different data sources, such as satellite images, climate models and ground sensors, improves forecast accuracy. Key steps include improving interpretability, ensuring transparency of model decisions and increasing trust. Advances in transfer learning allow for adaptable models across geographic regions and time periods. Integration into climate models allows an overview of long-term weather conditions and the effects of climate change. Additionally, optimizing computer performance facilitates real-time forecasting, enhanced disaster preparedness, precision agriculture, renewable energy management and more. Ethical considerations and social implications such as fairness and equal access will also shape further development. These collective advances mark a promising landscape for weather classification, contributing to informed decision-making in sectors that depend on accurate weather forecasts.

13.APPENDIX

Source Code

Weather model building.ipynb

1.Importing Necessary Libraries

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2.Import Model Building Libraries

```
In [5]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
```

Image Preprocessing

```
In [2]: from keras.preprocessing.image import ImageDataGenerator
```

```
In [3]: #2.configure image data generator
train_datagen=ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)
```

```
In [4]: #3.Apply image data generator functionality to train and test images
train=train_datagen.flow_from_directory(r"C:\Users\dines\Documents\AI_ML\PROJECT\Weather dataset\training",target_size=(224,224),
test=test_datagen.flow_from_directory(r"C:\Users\dines\Documents\AI_ML\PROJECT\Weather dataset\alien_test",target_size=(224,224),
```

Found 1500 images belonging to 5 classes.
Found 30 images belonging to 5 classes.

```
In [5]: print(train.class_indices)

{'cloudy': 0, 'foggy': 1, 'rainy': 2, 'shine': 3, 'sunrise': 4}
```

VGG 16

```
In [6]: #import model building libraries
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
```

```
In [7]: conv_base=VGG16(include_top=False,weights='imagenet',input_shape=(224,224,3))
```

```
In [8]: conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0

block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

=====
 Total params: 14714688 (56.13 MB)
 Trainable params: 14714688 (56.13 MB)
 Non-trainable params: 0 (0.00 Byte)

```
In [9]: model=Sequential()
model.add(conv_base)
model.add(Flatten())
model.add(Dense(units=256,activation="relu"))#hidden layers
model.add(Dense(units=5,activation="softmax"))
```

```
In [10]: model.summary()

Model: "sequential"

```

Layer (type)	Output Shape	Param #
=====		
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6422784
dense_1 (Dense)	(None, 5)	1285

=====
 Total params: 21138757 (80.64 MB)
 Trainable params: 21138757 (80.64 MB)
 Non-trainable params: 0 (0.00 Byte)

```
In [11]: #let us freeze the conv_base because already trained
conv_base.trainable=False
```

```
In [13]: model.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics =['accuracy'])
```

```
In [15]: model.fit(train,steps_per_epoch=len(train),epochs=10,validation_data=test,validation_steps=len(test))

Epoch 1/10
94/94 [=====] - 249s 3s/step - loss: 0.0943 - accuracy: 0.9680 - val_loss: 0.2681 - val_accuracy: 0.9333
Epoch 2/10
94/94 [=====] - 255s 3s/step - loss: 0.0717 - accuracy: 0.9780 - val_loss: 0.3111 - val_accuracy: 0.9000
Epoch 3/10
94/94 [=====] - 267s 3s/step - loss: 0.0399 - accuracy: 0.9840 - val_loss: 0.0526 - val_accuracy: 0.9600
```

```

67
Epoch 4/10
94/94 [=====] - 267s 3s/step - loss: 0.0879 - accuracy: 0.9640 - val_loss: 0.4652 - val_accuracy: 0.90
00
Epoch 5/10
94/94 [=====] - 264s 3s/step - loss: 0.0630 - accuracy: 0.9773 - val_loss: 0.1654 - val_accuracy: 0.96
67
Epoch 6/10
94/94 [=====] - 261s 3s/step - loss: 0.0409 - accuracy: 0.9860 - val_loss: 0.3730 - val_accuracy: 0.93
33
Epoch 7/10
94/94 [=====] - 254s 3s/step - loss: 0.0215 - accuracy: 0.9947 - val_loss: 0.2123 - val_accuracy: 0.96
67
Epoch 8/10
94/94 [=====] - 253s 3s/step - loss: 0.0432 - accuracy: 0.9853 - val_loss: 0.5932 - val_accuracy: 0.83
33
Epoch 9/10
94/94 [=====] - 298s 3s/step - loss: 0.0417 - accuracy: 0.9847 - val_loss: 0.0361 - val_accuracy: 1.00
00
Epoch 10/10
94/94 [=====] - 255s 3s/step - loss: 0.0309 - accuracy: 0.9880 - val_loss: 0.2170 - val_accuracy: 0.96
67

Out[15]: <keras.src.callbacks.History at 0x29d0104d8d0>

In [18]: model.save("weather.h5")

In [41]: #checking model Accuracy
model.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics =['accuracy'])
loss, accuracy = model.evaluate(test)

2/2 [=====] - 5s 2s/step - loss: 0.2170 - accuracy: 0.9667

```

Testing the Model

```


In [16]: from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

In [20]: import tensorflow as tf

In [21]: model=tf.keras.models.load_model(r"C:\Users\dines\Documents\AI_ML\PROJECT\weather.h5",compile=False)

In [22]: img=image.load_img(r"C:\Users\dines\Documents\AI_ML\PROJECT\Weather dataset\alien_test\rainy\rain_1.jpg",target_size=(224,224))

In [23]: img

Out[23]: 

In [24]: x=image.img_to_array(img)

```

```
In [25]: x
```

```
Out[25]: array([[19., 18., 23.],
               [10.,  9., 14.],
               [ 9.,  8., 13.],
               ...,
               [17., 12., 19.],
               [28., 21., 29.],
               [18., 11., 19.]],

               [[19., 18., 23.],
               [12., 11., 16.],
               [ 9.,  8., 13.],
               ...,
               [25., 20., 26.],
               [24., 19., 25.],
               [19., 12., 19.]],

               [[22., 21., 26.],
               [17., 16., 21.],
               [15., 14., 19.],
               ...,
               [30., 28., 33.],
               [22., 17., 23.],
               [16.,  9., 16.]],

               ...,

               [[49., 59., 50.],
               [48., 58., 49.],
               [44., 54., 46.],
               ...,
               [54., 43., 57.],
               [37., 27., 38.],
               [36., 25., 41.]],
```

```
               [[46., 56., 45.],
               [47., 54., 46.],
               [45., 51., 47.],
               ...,
               [49., 38., 52.],
               [35., 25., 36.],
               [38., 27., 43.]],

               [[42., 53., 39.],
               [45., 52., 44.],
               [44., 49., 45.],
               ...,
               [51., 40., 54.],
               [39., 29., 40.],
               [38., 27., 43.]]], dtype=float32)
```

```
In [26]: x=np.expand_dims(x,axis=0)
```

```
In [27]: x.shape
```

```
Out[27]: (1, 224, 224, 3)
```

```
In [28]: pred=model.predict(x)
```

```
1/1 [=====] - 0s 325ms/step
```

```
In [29]: pred
```

```
Out[29]: array([[0., 0., 1., 0., 0.]], dtype=float32)
```

```
In [30]: pred.argmax()
```

```
Out[30]: 2
```

```
In [31]: Index=['cloudy', 'foggy', 'rainy', 'shine', 'sunrise']
```

```
In [34]: result=Index[pred.argmax()]
result
```

```
Out[34]: 'rainy'
```

```
# TRAINING ACCURACY
```

```
training_scores =model.evaluate(train)
```

```
# Print the training accuracy
```

```
print(f'Training Accuracy: {training_scores[1]*100:.2f}%')
```

```
94/94 [=====] - 255s 3s/step - loss: 0.0187 - accuracy: 0.9927
```

```
Training Accuracy: 99.27%
```

```
# TESTING ACCURACY
testing_scores = model.evaluate(test)

# Print the training accuracy
print(f'Testing Accuracy: {testing_scores[1]*100:.2f}%')

2/2 [=====] - 4s 2s/step - loss: 0.2170 - accuracy: 0.9667
Testing Accuracy: 96.67%
```

main.js

```
1 function toggleWeatherPrediction() {
2   var weatherDiv = document.getElementById("weatherPrediction");
3   if (weatherDiv.style.display === "none" || weatherDiv.style.display === "") {
4     weatherDiv.style.display = "block";
5   } else {
6     weatherDiv.style.display = "none";
7   }
8 }
9
```

```
10 $(document).ready(function(){
11   $('#imageUpload').change(function(){
12     const file = $(this)[0].files[0];
13     if (file) {
14       const reader = new FileReader();
15       reader.onload = function (e) {
16         $('#imagePreview').css('background-image', 'url(' + e.target.result + ')');
17         $('.image-section').show();
18       }
19       reader.readAsDataURL(file);
20     }
21   });
22   $('form').on('submit', function(event){
23     event.preventDefault();
24     var formData = new FormData($('form')[0]);
25     $.ajax({
26       type: 'POST',
27       url: '/predict',
28       data: formData,
29       contentType: false,
30       cache: false,
31       processData: false,
32       success: function(response){
33         $('#result').text(response.weather);
34       }
35     });
36   });
37 });
38
```

app.py

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

```
app = Flask(__name__)

model = load_model('weather.h5', compile=False)
```

```
# Define a function to preprocess the image
def preprocess_image(image):
    image = image.resize((224, 224))
    image = np.array(image) / 255.0
    image = np.expand_dims(image, axis=0)
    return image
```

```
# Define the route for the home page
@app.route('/')
def home():
    return render_template('index.html')

# Define the route to handle the image upload and make predictions
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        f = request.files['image']
        print("current path")
        basepath = os.path.dirname(__file__)
        print("current path", basepath)
        filepath = os.path.join(basepath, 'uploads', f.filename)
        print("upload folder is ", filepath)
        f.save(filepath)

        img = Image.open(filepath)
        processed_img = preprocess_image(img)
        pred = model.predict(processed_img)
        dict={'cloudy': 0, 'foggy': 1, 'rainy': 2, 'shine': 3, 'sunrise': 4}
        pred_class=np.argmax(pred,axis=1)
        key_with_value = [key for key,value in dict.items() if value ==pred_class[0] ]
        weather = "Given image comes under " + str(key_with_value[0])+" weather classification"
        return jsonify({'weather': weather})
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0',port=8080,debug = False, threaded = False)
```

GitHub & Project Demo Link

<https://github.com/smartinternz02/SI-GuidedProject-612430-1698989121>

https://drive.google.com/file/d/1XPmL63Mdd_QBVUUevki2q2woSAyUvV8h/view?usp=drive_link