

# PROJECT MANUAL

## Weather Classification Using Deep Learning

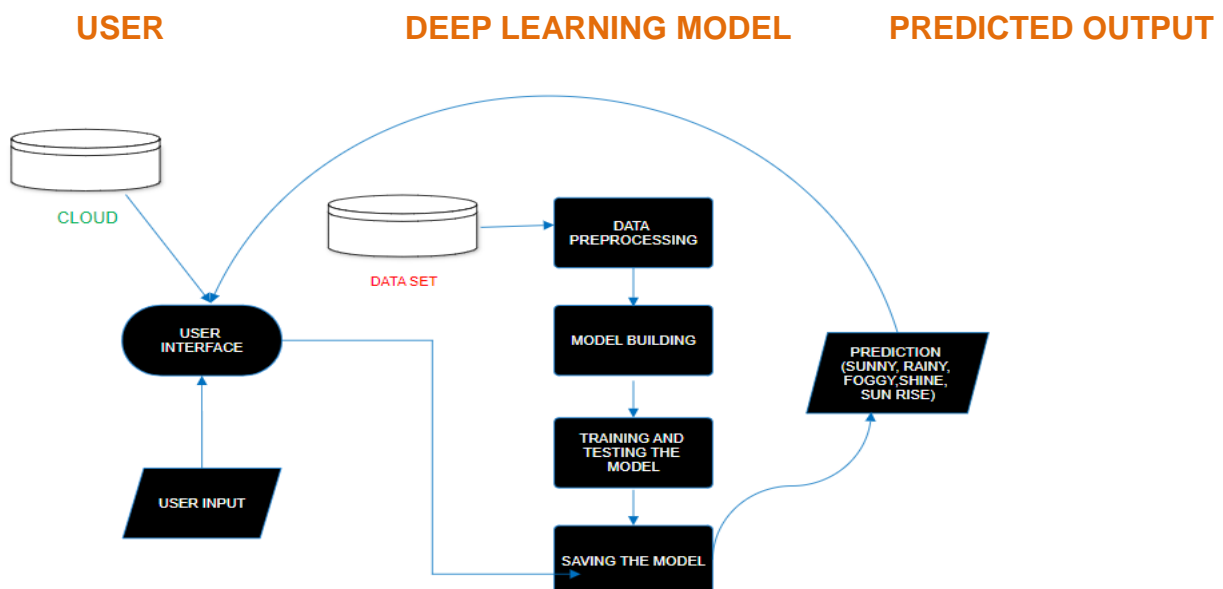
Date	18 November 2023
Team ID	591954
Project Name	Weather Classification Using Deep Learning

### Introduction:

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.

In this project we are classifying various types of weather. These weathers are majorly classified into 5 categories namely Cloudy, Shine, Rain, Foggy, Sunrise . We used Transfer Learning Technique **VGG16** model for better performance of model.

### Technical Architecture:



## **Project Flow:**


- The user interacts with the UI to choose an image.
- The chosen image is processed by a VGG16 deep learning model.
- The VGG16 model is integrated with a Flask application.
- The VGG16 model analyzes the image and generates predictions.
- The predictions are displayed on the Flask UI for the user to see.
- This process enables users to input an image and receive accurate predictions quickly.


## **Steps Involved:**

- **Data Collection**
  1. Collected Diverse Weather Dataset from various Resources
- **Image Preprocessing**
  1. Import ImageDataGenerator library
  2. Configure ImageDataGenerator class
  3. Applying Techniques like Rotation, Flipping, Brightness, Zoom Range etc to both training and testing data.
- **Model Building**
  1. Import all necessary model building libraries.
  2. Initializing the Model.
  3. Adding pre-trained model VGG16
  4. Adding Flatten layer (Input of ANN)
  5. Adding Dense layer (Hidden layers)
  6. Adding Dense layer (Output layer)
  7. Compile the Model
  8. Fit the Model
  9. Checking the Accuracy of model
- **Saving the Model**
- **Testing the Model**
- **Application Building**
  1. Creating index.html file for user interface and necessary CSS and Java Script files to send request to model.
  2. Integrating model with flask application
  3. Deploying Model on cloud platform (AWS)

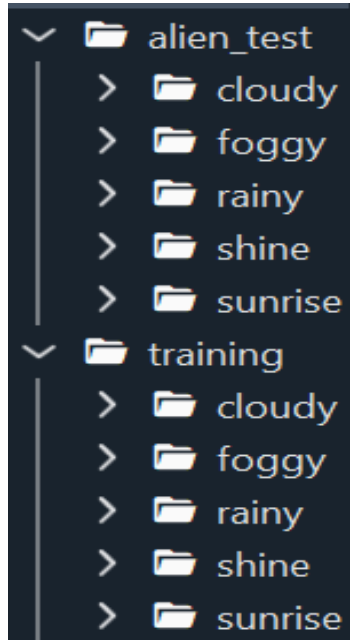
## PROJECT STRUCTURE:

### The Project Folders

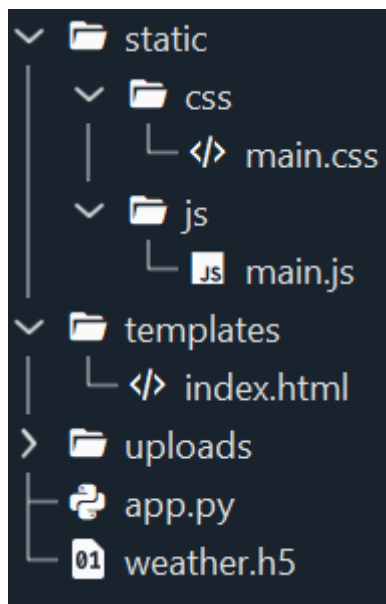
 Weather dataset

 Weather\_flask

### Under Weather\_dataset



### Under Weather\_flask



## Data Collection:

- Collected the weather images from various platforms available on internet example Keras etc.
- The downloaded dataset is organized into 5 categorical classes in both training and testing data
  - 1) Cloudy 2) Rainy 3) Foggy 4) Shine 5) Sunrise

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

## Importing Necessary Libraries

- Basic Data manipulation libraries like Numpy, Pandas and for better visualization matplotlib and seaborn are imported.

### 1.Importing Necessary Libraries

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

## Image Preprocessing:

- Import the ImageDataGenerator class from tensorflow framework.
- **flow\_from\_directory** method reads images from the specified directory and applies the specified preprocessing, including resizing, during the training process.
- **shear\_range** is a parameter that determines the maximum angle or range of angles by which the image can be sheared.
- **zoom\_range** parameter specifies the range of zooming factors that can be randomly applied to the images during training.
- **horizontal\_flip=True** involves the transformation of an image by reversing it from left to right.
- **rescale=1./255** This normalizes the pixel values of the image to the range of [0, 1], which is a common practice in deep learning models.
- **target\_size=(224, 224)** specifies that the input images should be resized to have dimensions of 224 pixels in height and 224 pixels in width.
- **batch\_size** defines the number of samples that will be propagated through the neural network at each training iteration.

## 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),batch_size=16)
test=test_datagen.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.

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

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

## Model Building:

- Import the necessary libraries from tensorflow for model building
- Import the pre-trained model **VGG16**
- **include\_top=False** specifies that we are not including the fully connected layers at the top of the network of VGG16.
- **weights="imagenet"** parameter initialize the model with pre-trained weights from the ImageNet dataset.
- **input\_shape=(224, 224, 3)** correspond to the height, width, and channels of the input images.

```
> #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

> conv_base=VGG16(include_top=False,weights='imagenet',input_shape=(224,224,3))
```

### 1) Adding Layers to Model:

- **model = Sequential()** initializes a sequential model. In a sequential model, we add layers one by one in a linear stack.
- **model.add(conv\_base)** adds a pre-trained convolutional base (conv\_base) to our model.
- **model.add(Flatten())** flattens the output of the convolutional base into a 1D array, which is necessary before feeding it into densely connected layers.

- **model.add(Dense(units=256, activation="relu"))** adding a dense (fully connected) hidden layer with 256 units and a rectified linear unit (ReLU) activation function. The ReLU activation function is commonly used in hidden layers to introduce non-linearity.
- **model.add(Dense(units=5, activation="softmax"))** output layer with 5 classes and a softmax activation function. Softmax is often used in multi-class classification problems to convert the model's raw output into probability scores for each class.

```

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)		

## 2) Configure the Learning Process

- **conv\_base.trainable=False** freeze the conv\_base because it is already trained in VGG16
- **Loss Function (loss='categorical\_crossentropy')** measure of how well the neural network is performing. For a classification task with multiple classes (as indicated by 'categorical\_crossentropy'), this is a common choice. It computes the cross-entropy loss between the true labels and the predicted probabilities.
- **Optimizer (optimizer='adam')** is responsible for updating the weights of the neural network in order to minimize the loss function.

- **Metrics (metrics=['accuracy'])** used to evaluate the performance of the model during training and testing. 'Accuracy' is a common metric for classification problems. It represents the ratio of correctly predicted instances to the total instances.

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

### 3) Train the Model

- The model is trained on train dataset of weather classification
- **epoch** is one complete pass through the entire training dataset. Setting epochs to 10 means that the model will iterate over the entire training dataset 10 times during the training process
- **steps\_per\_epoch=len(train)** specifies the number of batches to process before moving to the next epoch. In this case, it's set to the length of the training dataset (train)
- **validation\_data=test** parameter specifies the validation dataset, which is used to evaluate the model's performance on data it hasn't seen during training
- **validation\_steps=len(test)** this parameter specifies the number of batches to process before completing one validation epoch. In this case, it's set to the length of the validation dataset (test).

```
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:
0.9667
Epoch 4/10
94/94 [=====] - 267s 3s/step - loss: 0.0879 - accuracy: 0.9640 - val_loss: 0.4652 - val_accuracy:
0.9000
Epoch 5/10
94/94 [=====] - 264s 3s/step - loss: 0.0630 - accuracy: 0.9773 - val_loss: 0.1654 - val_accuracy:
0.9667
Epoch 6/10
94/94 [=====] - 261s 3s/step - loss: 0.0409 - accuracy: 0.9860 - val_loss: 0.3730 - val_accuracy:
0.9333
Epoch 7/10
94/94 [=====] - 254s 3s/step - loss: 0.0215 - accuracy: 0.9947 - val_loss: 0.2123 - val_accuracy:
0.9667
Epoch 8/10
94/94 [=====] - 253s 3s/step - loss: 0.0432 - accuracy: 0.9853 - val_loss: 0.5932 - val_accuracy:
0.8333
Epoch 9/10
94/94 [=====] - 298s 3s/step - loss: 0.0417 - accuracy: 0.9847 - val_loss: 0.0361 - val_accuracy:
1.0000
Epoch 10/10
94/94 [=====] - 255s 3s/step - loss: 0.0309 - accuracy: 0.9880 - val_loss: 0.2170 - val_accuracy:
0.9667

<keras.src.callbacks.History at 0x29d0104d8d0>
```

We are getting good accuracy at epoch=10 with Training Accuracy: 98.8 and Testing Accuracy=96.67

#### 4) Checking the Accuracy of Model

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

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

2/2 [=====] - 5s 2s/step - loss: 0.2170 - accuracy: 0.9667
Testing Accuracy: 96.67%
```

The Accuracy of model on testing data is 96.67% with loss=0.2170

#### Saving the Model:

```
model.save("weather.h5")
```

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

#### Testing the Model:

- Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data. Load the saved model using load\_model.

#### Testing the Model

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

import tensorflow as tf

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

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

img
```

3]:





```
▶ x=np.expand_dims(x,axis=0) # changing the shape as input contain 4 dimensions
```

```
▶ x.shape
```

```
]: (1, 224, 224, 3)
```

```
▶ pred=model.predict(x)
```

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

```
▶ pred
```

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

```
▶ pred.argmax()
```

```
]: 2
```

```
▶ Index=['cloudy', 'foggy', 'rainy', 'shine', 'sunrise']
```

```
▶ result=Index[pred.argmax()]  
result
```

```
]: 'rainy'
```

As we see the model is predicting Correctly of Rainy Image.

## Application Building

- Here we integrate the our deep learning model with python flask framework. Later we will deploy on AWS (EC2 Service) Cloud platform
- This section contain three phases
  1. Building HTML pages
  2. Building python flask code
  3. Running the application

### 1) Building HTML Pages:

The interface of our application looks like given below

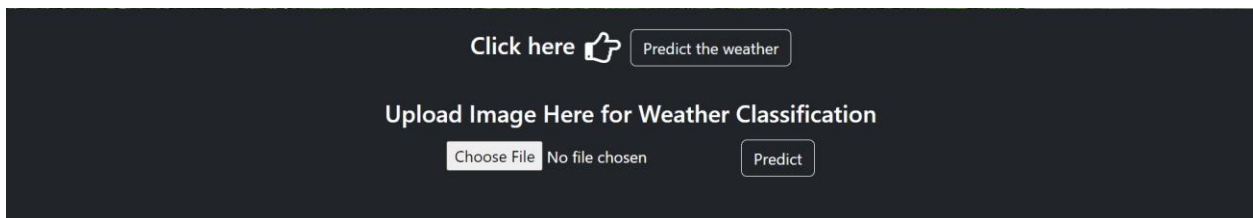
#### Index.html

## Weather Classification using Deep Learning

Weather classification is an essential tool for meteorologists and weather forecasters to predict weather patterns and communicate them to the public. 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.



- When we click on Predict the weather button the interface opens like given below



In static folder JS folder main.js code will help to post request to the server

Code:

toggleWeatherPrediction function called when we click on Predict the weather button in above image.

```
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 });

```

## 2) Building python flask code:

### Importing Necessary Libraries

```

3 from flask import Flask, render_template, request, jsonify
4 from tensorflow.keras.models import load_model
5 from tensorflow.keras.preprocessing import image
6 from PIL import Image
7 import numpy as np
8 import os

```

### Loading the Model

```

app = Flask(__name__)

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

```

### Defining the function to preprocess the input image

```

4 # Define a function to preprocess the image
5 def preprocess_image(image):
6     image = image.resize((224, 224))
7     image = np.array(image) / 255.0
8     image = np.expand_dims(image, axis=0)
9     return image

```

## Handling the routing

```
# 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})
```

## Main function

```
47
48 if __name__ == '__main__':
49     app.run(host='0.0.0.0',port=8080,debug = False, threaded = False)
50
```

## 3) Running the Application

We Deployed our deep learning model on **AWS** cloud platform

**DEPLOYMENT LINK:** <http://ec2-13-54-253-56.ap-southeast-2.compute.amazonaws.com:8080/>

**INPUT 01:**



INPUT 02:

**Upload Image Here for Weather Classification**

Choose File sunrise\_3.jpg Predict



**Given image comes under sunrise weather classification**

INPUT 03:

**Upload Image Here for Weather Classification**

Choose File rain\_4.jpg Predict



**Given image comes under rainy weather classification**