

WEATHER IMAGE CLASSIFICATION USING CNN

Submitted in partial fulfillment of the requirements for the award of

Bachelor of Engineering Degree

in

Computer Science and

Engineering By

SAKTHIVEL.B (Reg.No : 37110668)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

MARCH 2021



SATHYABAMA



INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119
www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of SAKTHIVEL.B (Reg.No: 37110668) who carried out the project entitled "**WEATHER IMAGE CLASSIFICATION USING CNN**" under my supervision from November 2020 to March 2021.

Internal Guide

V.ULAGAMUTHALVI,M.E.,Ph.d.,
Associate Professor

Head of the Department

Dr. S.VIGNESHWARI, M.E., Ph.D.,
Dr.L.LAKSHMANAN, M., PhD.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I , **SAKTHIVEL.B (Reg.No: 37110668)** hereby declare that the Project Report entitled on **“WEATHER IMAGE CLASSIFICATION USING CNN”** done by me under the guidance of **Mrs.V.ULAGAMUTHALVI, M.E.,Ph.d.,** at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the Board of management of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

I convey our thanks to **Dr. T. SASIKALA, M.E., Ph.D., Dean, School of Computing and Dr.S.VIGNESHWARI, M.E., Ph.D., Head of the Department, Department of Computer Science And Engineering** for providing us the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Mrs.V.ULAGAMUTHALVI, M.E.,Ph.d.,** for her valuable guidance, suggestions which paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the department of **Computer Science And Engineering** who were helpful in many ways for the completion of project.

ABSTRACT

A real-world weather biometric system that detects and describes weather condition in image data is an emerging subject in machine vision. These systems develop computer vision approaches for the classification of weather condition. Advances in artificial intelligence mean applications increasingly can take on image recognition capabilities that allow them to identify given input image type of weather classification like sunny, rainy etc. Deep learning is a vast field and narrow focusing a bit and takes up the challenge of solving an Image Classification process. To propose a deep learning algorithms by tensor flow or keras by finding the image classification of weather reports by convolution neural network (CNN).

<u>TABLE OF CONTENTS</u>		
<u>CHAPTER NO.</u>	<u>TITLE</u>	<u>PAGE NO.</u>
	ABSTRACT	i
	TABLE OF CONTENTS	ii
	LIST OF FIGURES	iv
	LIST OF ABBREVIATIONS	iv
1.	INTRODUCTION	1
	OUTLINE OF THE PROJECT	1
	BACKGROUND	4
	DEEP LEARNING	5
	WORKING OF DEEP LEARNING	6
	DEEP LEARNING USE – Case	7
	DEEP LEARNING APPLICATIONS	7
	SETTING UP THE STRUCTURE OF OUR IMAGE DATA	8
	BREAKING DOWN THE PROCESS OF MODEL BUILDING	9
	LOADING AND PRE-PROCESSING THE DATA	9
	DEFINING THE MODEL’S ARCHITECTURE	10
	TRAINING THE MODEL	10
2.	LITERATURE SURVEY	12
	LITERATURE REVIEW	12
	REVIEW OF LITERATURE SURVEY	12
3.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	19
	AIM 19	
	SCOPE 19	
	EXISTING SYSTEM	19
	PROPOSED SYSTEM	20
	TRAINING THE DATASET	21
	TESTING THE DATASET	21
	SYSTEM ARCHITECTURE	22

4.	EXPERIMENTAL OR MATERIAL AND METHODS ALGORITHM USED	23
	SOFTWARE REQUIREMENTS	23
	HARDWARE REQUIREMENTS	23
	ALGORITHMS EXPLANATION	23
	CONVOLUTIONAL NEURAL NETWORK (CNN)	23
	METHODOLOGY	26
	ARCHITECTURE OF CNN	26
	CNN MODEL STEPS	28
	SOFTWARE DESCRIPTION	29
	TENSORFLOW	32
	KERAS	32
	DATA FLOW DIAGRAM	33
5.	RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS	37
	MODULES	37
	DATA VALIDATION PROCESS	37
	DATA ANALYSIS	37
	OUTLIER DETECTION PROCESS	38
	GUI BASED RESULTS	39
	RESULTS AND DISCUSSION	45
	RESULTS	47
6.	SUMMARY AND CONCLUSION	50
	CONCLUSION	50
	FUTURE WORK	50
	ADVANTAGES	50
	APPLICATION	50
	REFERENCES	51
	APPENDIX	52
	A. SOURCE CODE	52
	B. SCREEN SHOTS	64

<u>LIST OF FIGURES</u>		
<u>FIG. NO.</u>	<u>TITLE</u>	<u>PAGE NO.</u>
1.1	How deep learning is a subset of machine learning and how machine learning is a subset of artificial intelligence (AI)	5
1.2	Deep Learning Neural Network Layers	6
3.1	Architecture diagram of weather image classification	22
4.1	Training model	26
4.2	Testing model	26
4.3	TensorFlow model	32
5.1	Open the anaconda navigator	45
5.2	Launch the jupyter notebook platform	46
5.3	Open the correspondent result folder	46

<u>LIST OF ABBREVIATIONS</u>	
<u>ABBREVIATION</u>	<u>EXPANSION</u>
SML	SUPERVISED MACHINE LEARNING
DOA	DEVIATION OF ARCH
GUI	GRAPICAL USER INTERFACE
ML	MACHINE LEARNING
SVM	SUPPORT VECTOR MACHINES
CNN	CONVOLUTIONAL NEURAL NETWORK
RGB	RED GREEN BLUE
AUC	AREA UNDER CURVE
DL	DEEP LEARNING
AI	ARTIFICIAL INTELLIGENCE

CHAPTER 1

INTRODUCTION

OUTLINE OF THE PROJECT

Weather forecasting is the prediction of the state of atmosphere for a given location using the application science and technology. It includes temperature, rain, cloudiness, wind speed, and humidity. Weather warnings are a special kind of short range forecast carried out for the production of human life. Weather warnings are issued by the government throughout the world for all kind of threatening weather even including tropical storms and tropical and cyclones depending upon the location.

Weather prediction before weather models can predict the future state of the atmosphere they must know its present state very accurately. This is the primary purpose of a disparate array of weather measurement instruments and platforms that include surface weather stations, upper-air radiosonde observations, aircraft and ship observations, satellites and various other special-purpose devices. Twice daily at 00 and 12UTC, weather observations are fed into a global telecommunications network, and are then ingested by the national meteorological services around the world where they are checked for accuracy, and then projected onto a three-dimensional modeling grid. Computer-based forecast models then predict the weather using these observations to first initialize and then solve a set of mathematical equations that describe the laws governing atmospheric behavior. These laws include three equations of atmospheric motion, the thermodynamic equation (representing conservation of energy), the so-called gas law or equation of state, and the continuity equation (which express the conservation of mass). Today's sophisticated forecast models and vastly more capable computers have significantly improved the timeliness and accuracy of weather forecasts and warnings. However the accuracy of weather predictions is still inherently limited in four areas : (1) the quality, density and representativeness of atmospheric measurements; (2) the finite power of today's computers; (3) our imperfect understanding of and ability to model atmospheric processes, and (4) the inherent unpredictability of atmospheric processes on different scales due to their chaotic nature. Progress in the accuracy and extent of atmospheric prediction

will continue though.

Weather forecasts are made by collecting as much data as possible about the current state of the atmosphere (particularly the temperature, humidity and wind) and using understanding of atmospheric processes (through meteorology) to determine how the atmosphere evolves in the future. However, the chaotic nature of the atmosphere and incomplete understanding of the processes mean that forecasts become less accurate as the range of the forecast increases.

Traditional observations made at the surface of atmospheric pressure, temperature, wind speed, wind direction, humidity, precipitation are collected routinely from trained observers, automatic weather stations or buoys. During the data assimilation process, information gained from the observations is used in conjunction with a numerical model's most recent forecast for the time that observations were made to produce the meteorological analysis. Numerical weather prediction models are computer simulations of the atmosphere. They take the analysis as the starting point and evolve the state of the atmosphere forward in time using understanding of physics and fluid dynamics.

Python Imaging Library (PIL) is one of the popular libraries used for image processing. PIL can be used to display image, create thumbnails, resize, rotation, convert between file formats, contrast enhancement, filter.

Scikit : Scikit-image is an open source Python package that works with NumPy arrays. It implements algorithms and utilities for use in research, education, and industry applications. It is a fairly simple and straightforward library, even for those who are new to Python's ecosystem. The code is high-quality, peer-reviewed, and written by an active community of volunteers.

NumPy : NumPy is one of the core libraries in Python programming and provides support for arrays. An image is essentially a standard NumPy array containing pixels of data points. Therefore, by using basic NumPy operations, such as slicing, masking, and fancy indexing, you can modify the pixel values of an image. The image can be loaded using skimage and displayed using Matplotlib

SciPy : SciPy is another of Python's core scientific modules (like NumPy) and can be used for basic image manipulation and processing tasks. In particular, the submodule `scipy.ndimage` (in SciPy v1.1.0) provides functions operating on n-dimensional NumPy arrays. The package currently includes functions for linear

and non-linear filtering, binary morphology, B-spline interpolation, and object measurements.

PIL / Pillow : PIL/Pillow PIL (Python Imaging Library) is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. However, its development has stagnated, with its last release in 2009. Fortunately, there is Pillow, an actively developed fork of PIL, that is easier to install, runs on all major operating systems, and supports Python 3. The library contains basic image processing functionality, including point operations, filtering with a set of built-in convolution kernels, and color-space conversions.

OpenCV : OpenCV (Open Source Computer Vision Library) is one of the most widely used libraries for computer vision applications. OpenCV-Python is the Python API for OpenCV. OpenCV-Python is not only fast, since the background consists of code written in C/C++, but it is also easy to code and deploy (due to the Python wrapper in the foreground). This makes it a great choice to perform computationally intensive computer vision programs.

SimpleCV : SimpleCV is another open source framework for building computer vision applications. It offers access to several high-powered computer vision libraries such as OpenCV, but without having to know about bit depths, file formats, color spaces, etc. Its learning curve is substantially smaller than OpenCV's, and (as its tagline says), "it's computer vision made easy." Some points in favor of SimpleCV are: Even beginning programmers can write simple machine vision tests. Cameras, video files, images, and video streams are all interoperable.

Mahotas : Mahotas is another computer vision and image processing library for Python. It contains traditional image processing functions such as filtering and morphological operations, as well as more modern computer vision functions for feature computation, including interest point detection and local descriptors. The interface is in Python, which is appropriate for fast development, but the algorithms are implemented in C++ and tuned for speed. Mahotas' library is fast with minimalistic code and even minimum dependencies. Read its official paper for more insights.

ITK : ITK (Insight Segmentation and Registration Toolkit) is an "open source, cross-platform system that provides developers with an extensive suite of software tools for image analysis. SimpleITK is a simplified layer built on top of ITK,

intended to facilitate its use in rapid prototyping, education, [and] interpreted languages." It's also an image analysis toolkit with a large number of components

supporting general filtering operations, image segmentation, and registration. SimpleITK is written in C++, but it's available for a large number of programming languages including Python.

Pgmagick : pgmagick is a Python-based wrapper for the GraphicsMagick library.

The GraphicsMagick image processing system is sometimes called the Swiss Army Knife of image processing. Its robust and efficient collection of tools and libraries supports reading, writing, and manipulating images in over 88 major formats including DPX, GIF, JPEG, JPEG-2000, PNG, PDF, PNM, and TIFF.

Pycairo : Pycairo is a set of Python bindings for the Cairo graphics library. Cairo is a 2D graphics library for drawing vector graphics. Vector graphics are interesting because they don't lose clarity when resized or transformed. Pycairo can call Cairo commands from Python.

BACKGROUND

Weather forecasters use a mix of radar, satellite, forecast models, and observational data to forecast weather. Historically, weather observation networks have focused on atmospheric conditions and may ignore areas below the horizon where people live and drive vehicles.

Weather data acquired or collected by the historical observation networks is primarily tailored to atmospheric-based observation rather than ground-based observation. Based on such atmospheric-based weather data, forecasters can predict specific atmospheric conditions which are consistent with a specific weather event, but unfortunately cannot verify that the event is actually taking place or will take place on or near the ground in localized population centers. Instead, confirmation is left to haphazard ground-based reports from local storm-chasers and the like to confirm the weather event by visual means.

Thus, forecasters remain limited in their ability to provide long-lead forecasts for localized, small-scale, and high impact phenomena, such as individual thunderstorm cells, the location of the divide between rain and snow during major storms, flash floods, and fine-scale, short-lived variations in solar radiation and low-level winds. A weather control station receives from a communication network data packets transmitted wirelessly to the network by a

mobile platform separated from the control station. The data packets carry images captured by an image sensor aboard the mobile platform and show one or more

weather conditions local to the mobile platform. The control station recovers the images from the data packets and processes the recovered images according to one or more weather condition detection algorithms to detect the one or more weather conditions, respectively. The weather station reports the one or more detected weather conditions.

DEEP LEARNING

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

Definition : *Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.*

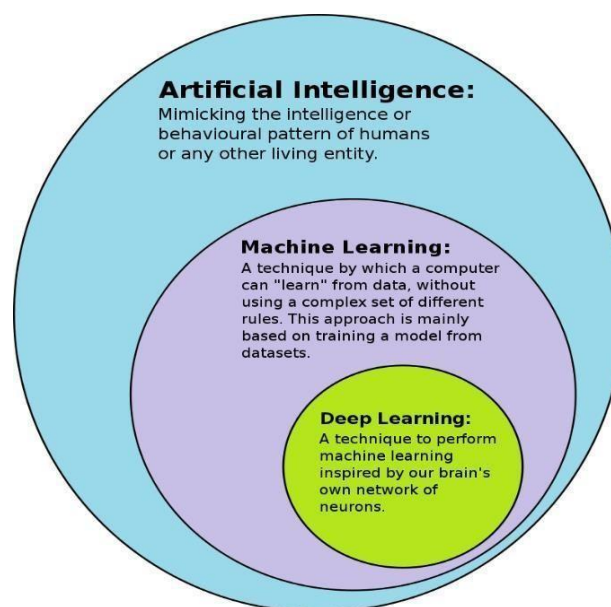
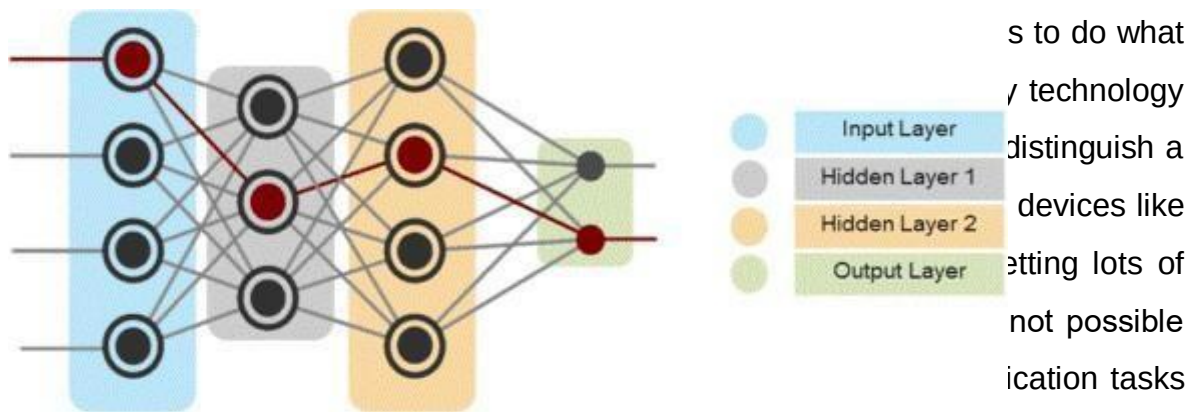


Fig 1.1 *How deep learning is a subset of machine learning and how machine learning is a subset of artificial intelligence (AI)*

WORKING OF DEEP LEARNING



directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance.

Fig 1.2 Deep Learning Neural Network Layers

Three types of layers :

Input Layer

Hidden Layer

Output Layer

The first layer is the input layer which receives all the inputs and the last layer is the output layer.

The layers in between these layers are called hidden layers. There can be 'n' number of hidden layers thanks to the high end resources.

The number of hidden layers and the number of perceptrons in each layer will entirely depend on the use-case.

DEEP LEARNING USE – Case

We are passing the high dimensional data to the input layer. To match the dimensionality of the input data, the input layer will contain multiple sub-layers of perceptron so that it can consume the entire input.

The output received from the input layer will contain patterns and will only be able to identify the edges of the images based on the contrast levels.

The output will be fed to the Hidden layer 1 where it will be able to identify various face features like eyes, nose, ears etc.

Now, this will be fed to the hidden layer 2 where it will be able to form the entire faces. Then, the output of layer 2 is sent to the output layer.

Finally, the output layer performs classification based on the result obtained from the previous dataset whether a pedestrian already present or not (Re-identification).

DEEP LEARNING APPLICATIONS

Pretrained deep neural network models can be used to quickly apply deep learning to your problems by performing transfer learning or feature extraction, some available models include AlexNet, VGG-16, and VGG-19, as well as Caffe imported using `importCaffeNetwork`.

Deep learning algorithms learn progressively more about the image as it goes through each neural network layer. Early layers learn how to detect low-level features like edges, and subsequent layers combine features from earlier layers into a more holistic representation. For example, a middle layer might identify edges to detect parts of an object in the photo like a leg or a branch, while a deep layer will be able to detect the full object like a dog or a tree.

Deep learning algorithms run data through several “layers” of neural network algorithms, each of which passes a simplified representation of the data to the next layer. Most machine learning algorithms work well on datasets that have up to a few hundred features, or columns. However, an unstructured dataset like one from an image has such a large number of features that this process becomes cumbersome or completely unfeasible. A single 800-by-1000-pixel image in RGB color has 2,400,000 features. Deep learning algorithms learn progressively more about the image as it goes through each neural network layer.

Early layers learn how to detect low-level features like edges, and subsequent layers combine features from earlier layers into a more holistic representation.

Why is Deep Learning important :

The ability to process large numbers of features makes deep learning very powerful when dealing with unstructured data. However, deep learning algorithms can be overkill for less complex problems because they require access to a vast amount of data to be effective. For instance, ImageNet, the common benchmark for training deep learning models for comprehensive image recognition, has access to over 14 million images.

If the data is too simple or there isn't enough of it, it is very easy for a deep learning model to become overfitted and fail to generalize well to new data. As a result, deep learning models don't perform as well as other techniques like boosted decision trees or linear models for most practical business problems when there are smaller datasets and fewer features, like understanding customer churn or detecting fraudulent transactions. In certain cases like multiclass classification, deep learning can work for smaller, structured datasets.

Image Classification :

For example, you will have instantly recognized it – it's a weather image/ car image. Take a step back and analyze how you came to this conclusion – you were shown an image and you classified the class it belonged to (a car, in this instance). And that, in a nutshell, is what image classification is all about.

There are potentially n numbers of categories in which a given image can be classified. Manually checking and classifying images is a very tedious process. The task becomes near impossible when we're faced with a massive number of images, say 10,000 or even 100,000. How useful would it be if we could automate this entire process and quickly label images per their corresponding class ?

Self-driving cars are a great example to understand where image classification is used in the real-world. To enable autonomous driving, we can build an image classification model that recognizes various objects, such as vehicles, people, moving objects, etc. on the road.

SETTING UP THE STRUCTURE OF OUR IMAGE DATA

Our data needs to be in a particular format in order to solve an image classification

problem. We will see this in action in a couple of sections but just keep these pointers in mind till we get there.

You should have 2 folders, one for the train set and the other for the test set. In the training set, you will have a .csv file and an image folder:

The .csv file contains the names of all the training images and their corresponding true labels. The image folder has all the training images.

The .csv file in our test set is different from the one present in the training set. This test set .csv file contains the names of all the test images, but they **do not** have any corresponding labels. Can you guess why? Our model will be trained on the images present in the training set and the label predictions will happen on the testing set images.

If your data is not in the format described above, you will need to convert it accordingly (otherwise the predictions will be awry and fairly useless).

BREAKING DOWN THE PROCESS OF MODEL BUILDING

Before we deep dive into the Python code, let's take a moment to understand how an image classification model is typically designed. We can divide this process broadly into 4 stages. Each stage requires a certain amount of time to execute :

Loading and pre-processing Data : 30% time

Defining Model architecture : 10% time

Training the model : 50% time

Estimation of performance : 10% time

Let me explain each of the above steps in a bit more detail. This section is crucial because not every model is built in the first go. You will need to go back after each iteration, fine-tune your steps, and run it again. Having a solid understanding of the underlying concepts will go a long way in accelerating the entire process.

LOADING AND PRE-PROCESSING THE DATA

Data is gold as far as deep learning models are concerned. Your image classification model has a far better chance of performing well if you have a good amount of images in the training set. Also, the shape of the data varies according to the architecture/framework that we use.

Hence, the critical data pre-processing step (the eternally important step in any project). I highly recommend going through the 'Basics of Image Processing in Python' to understand more about how pre-processing works with image data. But we are not quite there yet. In order to see how our model performs on unseen data (and before exposing it to the test set), we need to create a validation set. This is done by partitioning the training set data.

In short, we train the model on the training data and validate it on the validation data. Once we are satisfied with the model's performance on the validation set, we can use it for making predictions on the test data.

DEFINING THE MODEL'S ARCHITECTURE

This is another crucial step in our deep learning model building process. We have to define how our model will look and that requires answering questions like :
How many convolutional layers do we want ?

What should be the activation function for each layer ?

How many hidden units should each layer have ?

And many more. These are essentially the hyperparameters of the model which play a MASSIVE part in deciding how good the predictions will be.

How do we decide these values ? Excellent question ! A good idea is to pick these values based on existing research/studies. Another idea is to keep experimenting with the values until you find the best match but this can be quite a time consuming process.

Time required for this step : It should take around **1 minute** to define the architecture of the model.

TRAINING THE MODEL

For training the model, we require :

- ✓ Training images and their corresponding true labels.
- ✓ Validation images and their corresponding true labels (we use these labels only to validate the model and not during the training phase).

We also define the number of epochs in this step. For starters, we will run the model for 10 epochs (you can change the number of epochs later).

Time required for this step: Since training requires the model to learn structures,

we need around **5 minutes** to go through this step. And now time to make predictions!

Estimating the model's performance : Finally, we load the test data (images) and go through the pre-processing step here as well. We then predict the classes for these images using the trained model.

Time required for this step: ~ 1 minute.

Setting up the Problem Statement and Understanding the Data :

We will be picking up a really cool challenge to understand image classification. We have to build a model that can classify a given set of images according to the apparel (shirt, trousers, shoes, socks, etc.). It's actually a problem faced by many e-commerce retailers which makes it an even more interesting computer vision problem.

CHAPTER 2

LITERATURE SURVEY

LITERATURE REVIEW

A literature review is a body of text that aims to review the critical points of current knowledge on and/or methodological approaches to a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes information in a particular subject area within a certain time period. Its ultimate goal is to bring the reader up to date with current literature on a topic and forms the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources. Usually, it has an organizational pattern and combines both summary and synthesis.

A summary is a recap of important information about the source, but a synthesis is a re-organization, reshuffling of information. It might give a new interpretation of old material or combine new with old interpretations or it might trace the intellectual progression of the field, including major debates. Depending on the situation, the literature review may evaluate the sources and advise the reader on the most pertinent or relevant of them.

REVIEW OF LITERATURE SURVEY

Title 1: A Decision Support System for Wind Power Production

Author: Ruggero Donida Labati, Angelo Genovese, Vincenzo Piur, Fabio Scotti, Gianluca Sforza / Year:2018

It presented a novel decision support system that can predict the wind power production of a general WF. The system includes two main functions: long-term prediction of the power produced by a WF as well as computation of the related variability index and automatic tools that evaluate the production characteristics of WFs and the accuracy of weather forecasts. The main novelty presented in this paper consists of our system for predicting the power production of a WF based on NWP data. The proposed system is able to automatically learn the production characteristics of a WF. The system is composed of the following modules: data harmonization, feature extraction, calibration, thresholding, and

neural prediction. To achieve the best performance for heterogeneous WFs, we

proposed different configurations of the system obtained by using different combinations of its modules. Renewable energy production is constantly growing worldwide, and some countries produce a relevant percentage of their daily electricity consumption through wind energy. Therefore, decision support systems that can make accurate predictions of wind-based power production are of paramount importance for the traders operating in the energy market and for the managers in charge of planning the nonrenewable energy production. In this paper, we present a decision support system that can predict electric power production, estimate a variability index for the prediction, and analyze the wind farm (WF) production characteristics. The main contribution of this paper is a novel system for long-term electric power prediction based solely on the weather forecasts; thus, it is suitable for the WFs that cannot collect or manage the real-time data acquired by the sensors. Our system is based on neural networks and on novel techniques for calibrating and thresholding the weather forecasts based on the distinctive characteristics of the WF orography. We tuned and evaluated the proposed system using the data collected from two WFs over a two-year period and achieved satisfactory results. We studied different feature sets, training strategies, and system configurations before implementing this system for a player in the energy market. This company evaluated the power production prediction performance and the impact of our system at ten different WFs under real-world conditions and achieved a significant improvement with respect to their previous approach.

Title 2 : An Ensemble Framework For Day-Ahead Forecast of PV Output Power in Smart Grids

Author: Muhammad Qamar Raza, Mithulananthan, Jiaming Li, Kwang Y. Lee, Hoay Beng Gooi / Year: 2018

The performance of the proposed neural network ensemble framework is analyzed for the seasonal clear, partially cloudy and cloudy days. The predicted results of the proposed NNE forecast framework are compared with those of the comparative neural predictors and their wavelets. Results indicate that the proposed NNE framework outperforms other methods with minimum MAPE in all seasonal days. The forecast results indicate that the proposed NNE framework

achieves decreases in prediction error as compared to other models. The

proposed forecast framework gives less error variance for selected spring seasonal days. In addition, the forecast performance of the proposed forecast framework is also observed using different length of training dataset, and it is found that the 15-min training data produces better forecast performance than other training datasets. In order to analyze the impact of length and quality of data on training, 30, 60 and 90 days length of training data were applied. Different training and forecast performance is observed in the proposed framework with variable length of data. The uncertainty associated with solar photo-voltaic (PV) power output is a big challenge to design, manage and implement effective demand response and management strategies. Therefore, an accurate PV power output forecast is an utmost importance to allow seamless integration and a higher level of penetration. In this research, a neural network ensemble (NNE) scheme is proposed, which is based on particle swarm optimization (PSO) trained feedforward neural network (FNN). Five different FNN structures with varying network complexities are used to achieve the diverse and accurate forecast results. These results are combined using trim aggregation after removing the upper and lower forecast error extremes. Correlated variables namely wavelet transformed historical power output of PV, solar irradiance, wind speed, temperature and humidity are applied as inputs to the multivariate NNE. Clearness index is used to classify days into clear, cloudy and partial cloudy days. Test case studies are designed to predict the solar output for these days selected from all seasons. The performance of the proposed framework is analyzed by applying training data set of different resolution, length and quality from seven solar PV sites of the University of Queensland, Australia. The forecast results demonstrate that the proposed framework improves the forecast accuracy significantly in comparison with individual and benchmark models.

Advantage: It predicts accurate PV power output forecast is an utmost importance to allow seamless integration and a higher level of penetration.

Disadvantage : It has not done different clustering and data filtering techniques could be applied for smoothing the input data for better training.

Title 3 : Considering the Differentiating Health Impacts of Fuel Emissions in Optimal Generation Scheduling

Author : Mingfei Ban, Jilai Yu, Danyang Guo, Yiyun Yao / Year: 2018

It develops an environment-based SCUC model, which aims at reducing AAPC and associated health impacts in populated regions rather than limiting the total emission. The proposed model integrates differentiating marginal health impacts of fuel emissions into optimal generation scheduling. The scheduling problem is formulated as MILP and solved by stochastic optimization using Benders decomposition. Case studies suggest that a differentiating regulatory system is more effective in reducing fuel emission associated AAPC and corresponding health impacts. By modulating electricity generation among units with differentiating marginal impacts on AAPC and human health, the ambient air quality of populated regions will improve significantly at a minute incremental cost. The results demonstrate that wind power generation and energy storage applications in the proposed approach will deliver more environmental benefits. The paper introduces potentials for flexible emission abatements that can reduce the probability of high AAPC hours in populated regions. It contributes to a broader research effort that manages emissions more intelligently. Future work will involve the modeling of population flows, traffic flows and interactions among multiple air pollutants. Thermal power generation could exhibit a considerable level of fuel emissions with a significant threat to human health. Due to variations in weather and atmospheric conditions, the time, the location, and the level of fuel emissions could determine emissions' differentiating impacts on ambient air pollutant concentration (AAPC). The densely populated regions are more vulnerable to changes in AAPC because the associated health impacts can scale up log-linearly with AAPC and linearly with affected population sizes. This paper proposes an approach to evaluating the differentiating impacts of fuel emissions on AAPC and human health. The proposed approach applies the day-ahead hourly generation scheduling for reducing AAPC in densely populated regions. We employ a Gaussian puff model to address the source-receptor relationship among fuel emissions and resulting AAPC increments (Δ AAPC). We formulate a penalty cost for fuel emissions which reflects the marginal health impacts of individual fuel emissions and differentiates the emission dispatch by shifting the electricity production away from power plants with high marginal health impacts. The proposed approach captures the Δ AAPC response to changes in generation

scheduling and incorporates the marginal health impacts of fuel emissions into daily decision-making processes. Case studies demonstrate the effectiveness of

the proposed model in which differentiating environmental benefits of wind power generation and energy storage are illustrated in power generation scheduling

Advantage: Weather and atmospheric conditions the level of fuel emissions could determine emissions' differentiating impacts on ambient air pollutant concentration.

Disadvantage: It has not involved the modeling of population flows, traffic flows and interactions among multiple air pollutants.

Title 4: Power Generation Forecast of Hybrid PV-Wind System

Author: M.J. Sanjari, H.B. Gooi, Nirmal-Kumar C. Nair / Year: 2019

The higher-order multivariate Markov Chain was developed to forecast the PV and wind generated power. The time-adaptive stochastic correlation between the wind and PV output power was taken into consideration in the proposed method, which works based on the observation window of the last measured data samples to follow the pattern of PV / wind power variations. Moreover, to enhance the forecast accuracy, two issues are considered: the HI index was utilized as an additional meteorological variable and the procedure of dynamic update of the model parameters was applied to the PV/wind power forecast. Compared to scenario 1, in which the PV and wind power interdependency is not considered, the HMMC-based method models the correlation between PV and wind output power and HI and leads to higher forecast accuracy (scenario 2). The other advantage of applying the proposed framework to forecast the PV/wind power is that the dimension of the input vector of the forecast algorithm can be increased in a flexible manner to reach better performance of the forecast algorithm. This issue is simulated in scenario 3 with adding one of the variables, i.e. ambient temperature, relative humidity, atmospheric pressure and heat index, in different steps to the list of forecast algorithm input. Another advantage of using the proposed method is its adaptability, which was simulated in scenario 4 such that lower forecast error is achieved, compared to the condition in which the forecast model parameters cannot be updated dynamically (scenario 2). Finally, scenario 4 is the result of the proposed method, which matches well with the measurement data recorded from the actual PV and wind plants. To make the proposed method more applicable to real conditions in scenario 5, the proposed method was applied to the incomplete set of historical data to show the proposed method capability to

model the PV/wind power variation in the case some samples are missed.. Due to

their intermittency and unpredictability, increasing the penetration level of renewable energy (RE) resources to the power system leads to difficulties in operation. Reliable system operation requires a precise forecast of generated power by RE units. Photovoltaic (PV) and wind units are the significant portion of RE resources integrated into the power system. This paper proposes a forecast method for PV and wind generated power to achieve good prediction accuracy in different weather conditions. Not only is the relation between the wind and PV output power modeled, but the heat index (HI) is also taken into consideration as a useful meteorological variable to achieve the 15-min ahead precise expectation of PV / wind output power. Moreover, the input data is discretized in such a way that the best accuracy for the PV and wind power forecast is achieved. Comparing the results of the proposed method with the historical data recorded at actual PV and wind plants shows that the proposed forecast method results in high accuracy in PV and wind output power forecast. Moreover, the forecast model performance with HI consideration is compared with the model not using HI as an input variable.

Advantage: Reliable system operation requires a precise forecast of generated power by RE units.

Disadvantage: It has not capability in modeling the dynamic variations, and the stochastic nature of the output power.

Title 5 : The Use of Extreme Value Theory for Forecasting Long-Term Substation Maximum Electricity Demand

Author: Yun Li, Ben Jones / Year: 2019

The major contribution of this paper is proposing a new methodology for forecasting substation annual maximum demand as a function of other forecast variables using extreme value theory. For a given substation, the times of daily load exceedances above a given high threshold and the exceedance values were combined into a two-dimensional point process model by a nonhomogeneous Poisson process. To account for the influence of the customer count, average demand, and PV capacity, we have employed a PP model whose location and scale parameters that depend on the trends in these factors. The resulting PP model is linked to a non stationary GEV, which governs the behaviors of block maxima (i.e., annual maximum demand). As such, with trends in forecasted values

of the customer count, average demand, and PV capacity in the future period,

substation MD in terms of 10%, 50%, and 90% POE forecasts as respective 90%, 50%, and 10% quantiles of the fitted GEV distribution of the PP model. The expression of probabilistic MD using these designated POE levels allows the forecasts to meet industry specifications for capacity planning; allowing an efficient tradeoff between network investments and the risk of customer outages. Substation annual maximum electricity demand events are extreme, as customers respond to infrequent and extreme weather. Despite the extreme nature of annual maximum demand, the statistical theory of extreme values has only rarely, if ever, been applied. To support long term planning, utilities typically complete energy consumption and maximum demand forecasts, which are often conducted separately through two different process, leading to inconsistent trends and messages. To address these shortcomings, a point process approach from extreme value theory is proposed to model substation maximum demand as a function of trends in three common factors already required by utilities including customer count, average demand, and installed photovoltaic capacity. The point process model can be parameterized as a non stationary generalized extreme value distribution with location and scale parameters dependent on the trends of these factors. As the generalized extreme value distribution governs the behaviors of block maxima (annual maximum demand) with forecast trends of three common factors, substation maximum demand can be estimated as per quantiles required by planning standards. Therefore, the proposed approach is not only realistic and flexible to forecast maximum demand but also ensures consistent outcomes and messaging between the two outputs from energy consumption and maximum demand forecasts.

Advantage: To support long term planning, utilities typically complete energy consumption and maximum demand forecasts.

Disadvantage: It has not investigated the application of these techniques to account for the influence of new technologies.

CHAPTER 3

AIM AND SCOPE OF THE PRESENT INVESTIGATION

AIM

- Data validation
- Data cleaning / preparing
- Data image prediction

SCOPE

The scope of this project is the prediction of the weather image classification for a given location using the application science and technology i.e, deep learning. It includes temperature, rain, cloudiness, wind speed, and humidity. Weather warnings are a special kind of short range forecast carried out for the production of human life. Weather prediction before weather models can predict the future state of the atmosphere they must know its present state very accurately.

EXISTING SYSTEM

It has described briefly our newly created night video dataset, namely, TU-VDN, for moving object detection in thermal infrared images. The dataset consists of degraded atmospheric night outdoor scenes under low-light, dusty, rainy, and foggy conditions. It also presented a video salient feature-based background segmentation technique that uses both spatial features and thermal intensity for the robust investigation of thermal frames. It summarize the findings regarding this proposed technique. It handles various key challenges in thermal outdoor scenes, such as dynamic background, flat cluttered background, and thermal intensity adjustment during the maiden appearance of a moving object in the video sequence. in terms of accuracy, F1-score, and MCC, the results of the comparative experiments on the TU-VDN dataset has demonstrated the superior performance of our proposed method. The results of our analysis on the CDnet-2014 dataset over the night, thermal, and bad Weather category sequences have also demonstrated the superior performance of the approach in terms of MCC value and error rate. Foreground segmentation of moving objects in adverse

atmospheric conditions such as fog, rain, low light and dust is a challenging task in computer vision. The advantages of thermal infrared imaging at night time under

adverse atmospheric conditions have been demonstrated, which are due to the long wavelength. However, existing state-of-the-art object detection techniques have not been useful in such scenarios. It proposes an improved background model that utilizes both thermal pixel intensity features and spatial video salient features. The proposed spatial video salient features are represented as an Akin-based per-pixel Boolean string over a local region block, and depend on the effect of neighboring pixels on a centre pixel. It results of this Boolean procedure is referred to as the - 'Akin-based Local Whitening Boolean Pattern (ALWBP),' which differentiates foreground and background region accurately, even against a cluttered background. The background model is controlled via the automatic adaptation of parameters such as the decision threshold RT and, learning parameter L , and the updating of background samples B_{sample_int} and, B_{sample_ALWBP} to minimize (a) the effect of the background dynamics of outdoor scenes, and (b) the temperature polarity changes during the maiden appearance of a moving object in thermal frame sequences. The performance of this model is evaluated using nine existing standard segmentation performance metrics on our newly created - 'Tripura University Video Dataset at Night time (TU- VDN)' and on the publicly available CDnet-2014 dataset. Our newly created weather degraded video dataset, namely, TU-VDN, consists of sixty video sequences that represent four atmospheric conditions, namely, low light, dust, rain, and fog. The results of a performance comparison with fourteen state-of-the- art detection techniques also demonstrate the high accuracy of the proposed technique.

Drawback:

The existing state-of-the-art object detection techniques have not been useful.

It is only done for the weather concept not done the machine learning concept and deep learning concept.

PROPOSED SYSTEM

To predict the changes in weather patterns by using image dataset. We apply this information to create educated predict about the future. The data collected from web service. I will use the requests library to interact with the once collected, the data will need to be process and aggregated into a format that is

suitable for data analysis, and then cleaned. Then it will focus on analyzing the trends in the data with the goal of selecting appropriate features for building a Linear Regression model using the stats models and scikit-learn Python libraries. It will discuss the importance of understanding the assumptions necessary for using a Linear Regression model and demonstrate how to evaluate the features to build. It is discussion of Linear Regression model testing and validation also focuses on using Neural Networks. I will compare the process of building a Neural Network model, interpreting the results and, overall accuracy between the Linear Regression models built in the prior article and the Neural Network model.

Advantages:

These reports are to the investigation of applicability of deep learning techniques for weather forecasting in operational conditions.

Finally, it highlights some observations on future research issues, challenges, and needs.

TRAINING THE DATASET

- We are taking the weather image data set from kaggle website. After reviewing the image datasets we consider 4 types of datasets. They are : rainy, shiny, cloudy and sunrise.
- For example, to import any algorithm and train_test_split class from sklearn and numpy module for use in this program.
- Then we encapsulate load_data() method in data_dataset variable. Further we divide the dataset into training data and test data using train_test_split method. The X prefix in variable denotes the feature values and Y prefix denotes target values.
- This method divides dataset into training and test data randomly in ratio of 67:33. Then we encapsulate any algorithm.
- In the next line, we fit our training data into this algorithm so that computer can get trained using this data. Now the training part is complete.

TESTING THE DATASET

- Now we are testing the 4 types of image datasets namely rainy, shiny, cloudy and sunrise. We do this using the predict method which takes this

array as input and spits out predicted target value as output.

- So the predicted target value comes out to be 0. Finally we find the test score which is the ratio of no. of predictions found correct and total predictions made. We do this using the score method which basically compares the actual values of the test set with the predicted values.

SYSTEM ARCHITECTURE

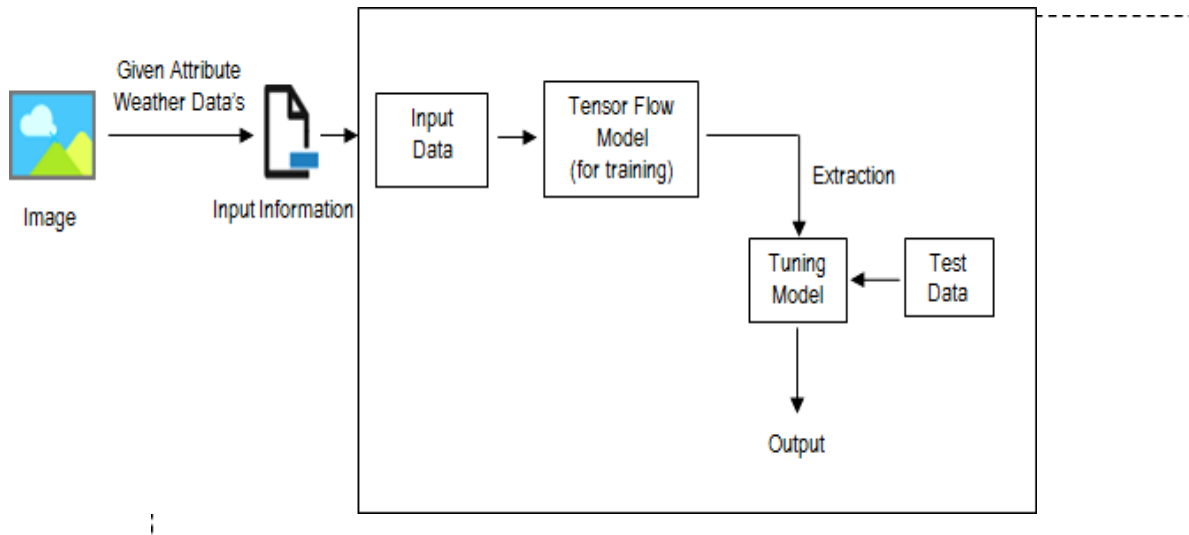


Fig 3.1 Architecture diagram of weather image classification

CHAPTER 4

EXPERIMENTAL OR MATERIALS AND METHODS ALGORITHM

SOFTWARE REQUIREMENTS

Operating System	:	Windows
Tool	:	Anaconda with Jupyter
Notebook Framework	:	TensorFlow / Keras

HARDWARE REQUIREMENTS

Processor	:	Pentium IV / III
Hard disk	:	Minimum 80 GB
RAM	:	Minimum 2 GB

ALGORITHMS EXPLANATION

In deep learning or machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, bio metric identification, document classification etc. In Supervised Learning, algorithms learn from labeled data. After understanding the data, the algorithm determines which label should be given to new data based on pattern and associating the patterns to the unlabeled new data.

CONVOLUTIONAL NEURAL NETWORK (CNN)

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the shared-weight architecture of the convolution kernels that scan the hidden layers and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, Image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in

one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include varying the weights as the loss function gets minimized while randomly trimming connectivity. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in the filters. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters or convolution kernels that in traditional algorithms are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

Defintion : The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Convolution : The primary purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. The convolution layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a CNN might have size $3 \times 5 \times 5$ (i.e. images have depth 3 i.e. the color channels, 5 pixels width and height). During the forward pass, each filter is convolved across the width and height of the input volume and compute dot products between the entries of the filter and the

input at any position. As the filter convolve over the width and height of the input volume it produces a 2-dimensional activation map that gives the responses of that

filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as a 3 edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, there will be an entire set of filters in each convolution layer (e.g. 20 filters), and each of them will produce a separate 2-dimensional activation map.

The size of the Feature Map (Convolved Feature) is controlled by three parameters:

Depth corresponds to the number of filters we use for the convolution operation.

Stride is the size of the filter, if the size of the filter is 5x5 then stride is 5.

Zero-padding : Sometimes, it is convenient to pad the input matrix with zeros around the border, so that filter can be applied to bordering elements of input image matrix. Using zero padding size of the feature map can be controlled.

Pooling (sub-sampling) :

Spatial Pooling (also called subsampling or down sampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. In case of Max Pooling, a spatial neighborhood (for example, a 2x2 windows) is defined and the largest element is taken from the rectified feature map within that window. In case of average pooling the average or sum of all elements in that window is taken. In practice, Max Pooling has been shown to work better. The function of Pooling is to progressively reduce the spatial size of the input representation. In particular, pooling.

Makes the input representations (feature dimension) smaller and more manageable.

Reduces the number of parameters and computations in the network, therefore, controlling over-fitting.

Makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling).

Helps us arrive at an almost scale invariant representation. This is very

powerful since objects can be detected in an image no matter where they are located.

METHODOLOGY

eathe system d usil neural network. The block diagram of the system is shown in following figures:

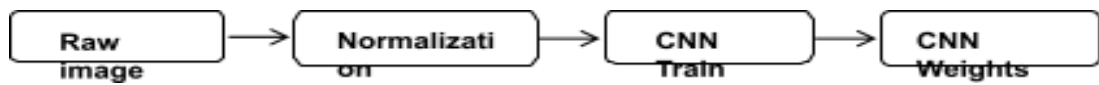


Fig 4.1 Training model

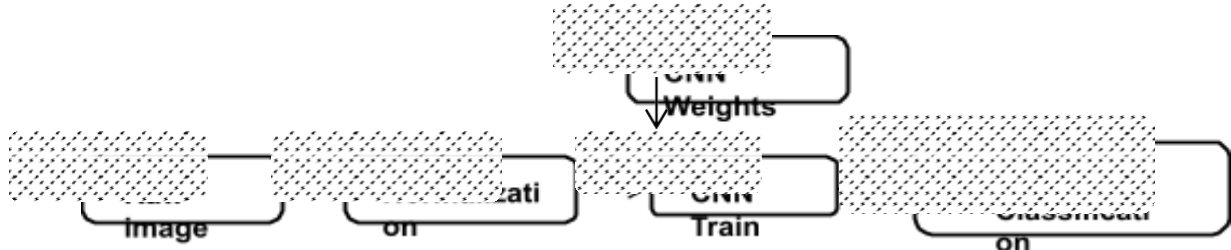


Fig 4.2 Testing model

During training, the system received a training data comprising gray scale images of faces with their respective expression label and learns a set of weights for the network. The training step took as input an image with a face. Thereafter, intensity normalization is applied to the image. The normalized images are used to train the Convolutional Network. To ensure that the training performance is not affected by the order of presentation of the examples, validation dataset is used to choose the final best set of weights out of a set of trainings performed with samples presented in different orders. The output of the training step is a set of weights that achieve the best result with the training data. During test, the system received a gray scale image of a face from test dataset, and output the predicted expression by using the final network weights learned during training. Its output is a single number that represents one of the seven basic expressions.

ARCHITECTURE OF CNN

A typical architecture of a convolutional neural network contains an input layer, some convolutional layers, some fully-connected layers, and an output layer. CNN is designed with some modification on LeNet Architecture. It has 6 layers without considering input and output.

Input Layer :

The input layer has pre-determined, fixed dimensions, so the image must be pre-processed before it can be fed into the layer. Normalized gray scale images of size 48 X 48 pixels from given dataset are used for training, validation and testing. For

testing propose laptop webcam images are also used, in which face is detected and cropped using OpenCV Haar Cascade Classifier and normalized.

Convolution and Pooling (ConvPool) Layers :

Convolution and pooling is done based on batch processing. Each batch has N images and CNN filter weights are updated on those batches. Each convolution layer takes image batch input of four dimension N x Color-Channel x width x height. Feature map or filter for convolution is also four dimensional (Number of feature maps in, number of feature maps out, filter width, filter height). In each convolution layer, four dimensional convolutions are calculated between image batch and feature maps. After convolution only parameter that changes is image width and height.

- ✓ New image width = old image width – filter width + 1
- ✓ New image height = old image height – filter height + 1

After each convolution layer down sampling / subsampling is done for dimensionality reduction. This process is called Pooling. Max pooling and Average Pooling are two famous pooling methods. In this project max pooling is done after convolution. Pool size of (2x2) is 12 taken, which splits the image into grid of blocks each of size 2x2 and takes maximum of 4 pixels. After pooling only height and width are affected. Two convolution layer and pooling layer are used in the architecture. At first convolution layer size of input image batch is Nx1x48x48. Here, size of image batch is N, number of color channel is 1 and both image height and width are 48 pixel. Convolution with feature map of 1x20x5x5 results image batch is of size Nx20x44x44. After convolution pooling is done with pool size of 2x2, which results image batch of size Nx20x22x22. This is followed by second convolution layer with feature map of 20x20x5x5, which results image batch of size Nx20x18x18. This is followed by pooling layer with pool size 2x2, which results image batch of size Nx20x9x9.

ReLU layer :

ReLU is the abbreviation of rectified linear unit, which applies the non saturating activation function $f(x) = \max(0, x)$. It effectively removes negative values from an activation map by setting them to zero. It increases the non-linear properties of the decision function and of the overall network without

affecting the receptive fields of the convolution layer.

Fully Connected Layer :

This layer is inspired by the way neurons transmit signals through the brain. It takes a large number of input features and transforms features through layers connected with trainable weights. Two hidden layers of size 500 and 300 unit are used in fully-connected layer. The weights of these layers are trained by forward propagation of training data then backward propagation of its errors. Back propagation starts from evaluating the difference between prediction and true value, and back calculates the weight adjustment needed to every layer before. We can control the training speed and the complexity of the architecture by tuning the hyper-parameters, such as learning rate and network density. Hyper-parameters for this layer include learning rate, momentum, regularization parameter, and decay. The output from the second pooling layer is of size $N \times 20 \times 9 \times 9$ and input of first hidden layer of fully-connected layer is of size $N \times 500$. So, output of pooling layer is flattened to $N \times 1620$ size and fed to first hidden layer. Output from first hidden layer is fed to second hidden layer. Second hidden layer is of size $N \times 300$ and its output is fed to output layer of size equal to number of facial expression classes.

Output Layer :

Output from the second hidden layer is connected to output layer having seven distinct classes and output is obtained using the probabilities for each of the seven classes. The class with the highest probability is the predicted class.

CNN MODEL STEPS

Conv2D : It is the layer to convolve the image into multiple images activation is the activation function.

MaxPooling2D : It is used to max pool the value from the given size matrix and same is used for the next 2 layers.

Flatten : It is used to flatten the dimensions of the image obtained after convolving it. **Dense** : It is used to make this a fully connected model and is the hidden layer.

Dropout : It is used to avoid over fitting on the dataset and dense is the output layer contains only one neuron which decide to which category image belongs.

Image Data Generator : It is that rescales the image, applies shear in some range, zooms the image and does horizontal flipping with the image. This Image

Data Generator includes all possible orientation of the image.

Training Process : `train_datagen.flow_from_directory` is the function that is used to prepare data from the `train_dataset` directory. `Target_size` specifies the target size of the image. `test_datagen.flow_from_directory` is used to prepare test data for the model and all is similar as above. `fit_generator` is used to fit the data into the model made above, other factors used are `steps_per_epochs` tells us about the number of times the model will execute for the training data.

Epochs : It tells us the number of times model will be trained in forward and backward pass.

Validation process : `validation_data` is used to feed the validation/test data into the model. `validation_steps` denotes the number of validation/test samples.

SOFTWARE DESCRIPTION

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system "Conda".

The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS. So, Anaconda distribution comes with more than 1,400 packages as well as the Conda package and virtual environment manager called Anaconda Navigator and it eliminates the need to learn to install each library independently. The open source packages can be individually installed from the Anaconda repository with the `conda install` command or using the `pip install` command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in most cases they can work together. Custom packages can be made using the `conda build` command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.

Anaconda Navigator :

Anaconda Navigator is a desktop graphical user interface (GUI) included in

Anaconda distribution that allows users to launch applications and manage conda

packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

- (i) JupyterLab
- (ii) Jupyter Notebook
- (iii) QtConsole
- (iv) Spyder
- (v) Glueviz
- (vi) Orange
- (vii) Rstudio
- (viii) Visual Studio Code

Conda :

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-languages. The Conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

The Jupyter Notebook :

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Notebook document :

Notebook documents (or “notebooks”, all lower case) are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc...). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents

which can be run to perform data analysis.

Jupyter Notebook App :

The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet. In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a “Dashboard” (Notebook Dashboard), a “control panel” showing local files and allowing opening notebook documents or shutting down their kernels.

Kernel :

A notebook kernel is a “computational engine” that executes the code contained in a Notebook document. The ipython kernel, referenced in this guide, executes python code. Kernels for many other languages exist (official kernels). When you open a Notebook document, the associated kernel is automatically launched. When the notebook is executed (either cell-by-cell or with menu Cell -> Run All), the kernel performs the computation and produces the results. Depending on the type of computations, the kernel may consume significant CPU and RAM. Note that the RAM is not released until the kernel is shut-down.

Notebook Dashboard :

The Notebook Dashboard is the component which is shown first when you launch Jupyter Notebook App. The Notebook Dashboard is mainly used to open notebook documents, and to manage the running kernels (visualize and shutdown). The Notebook Dashboard has other features similar to a file manager, namely navigating folders and renaming/deleting files.

Working Process :

Download and install anaconda and get the most useful package for machine learning in Python.

- Load a dataset and understand its structure using statistical summaries and data visualization.

- Machine learning models, pick the best and build confidence that the accuracy is reliable.

Python is a popular and powerful interpreted language. Unlike R, Python is a

complete language and platform that you can use for both research and development and developing production systems. There are also a lot of modules

and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

The best way to get started using Python for machine learning is to complete a project.

- It will force you to install and start the Python interpreter (at the very least).
- It will give you a bird's eye view of how to step through a small project.
- It will give you confidence, maybe to go on to your own small projects.

TENSORFLOW

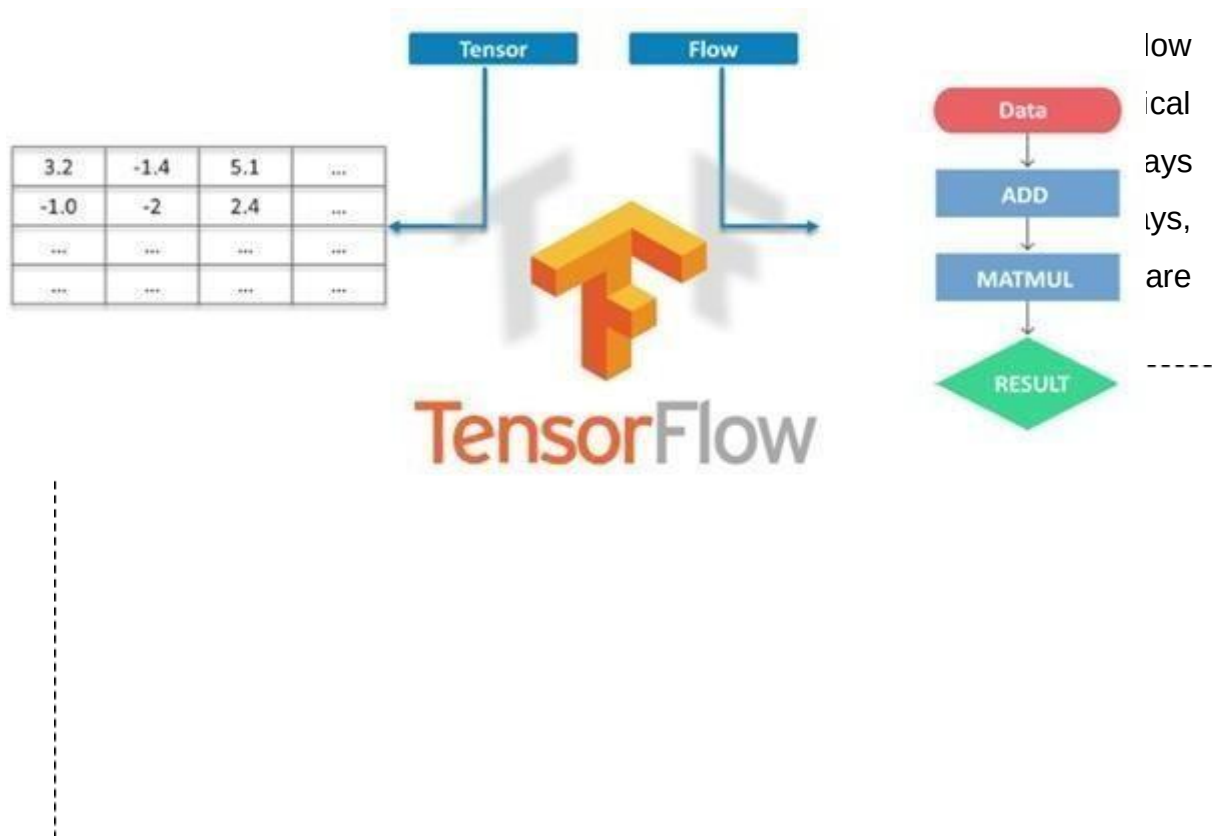


Fig 4.3 TensorFlow model

Keras : Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and

a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model.

Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top down approach to Systems Design. Symbols and Notations Used in DFDs Using any convention's DFD rules or guidelines, the symbols depict the four components of data flow diagrams.

External entity : an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

Process : any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.

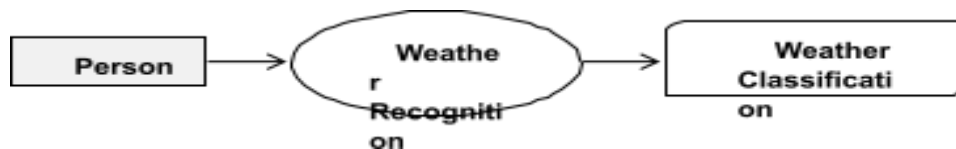
Data store : files or repositories that hold information for later use, such as a database table or a membership form.

Data flow : the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown

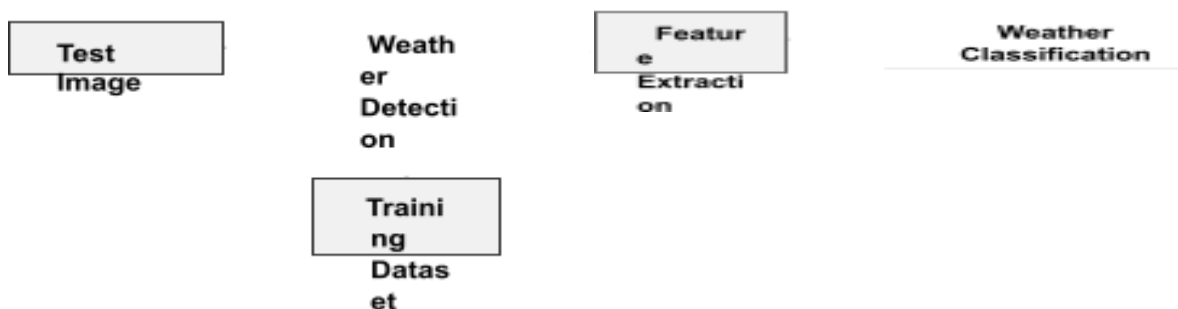
with arrows, typically labeled with a short data name, like “Billing details.”

DFD levels and layers : A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish. DFD Level 0 is also called a Context Diagram. It's a basic overview of the whole system or process being analyzed or modeled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

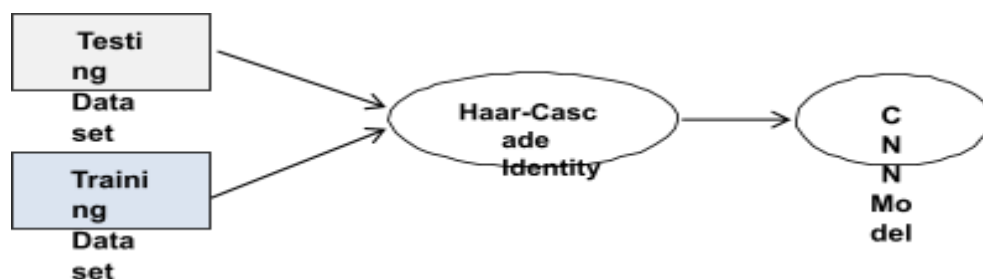
Level 0 :



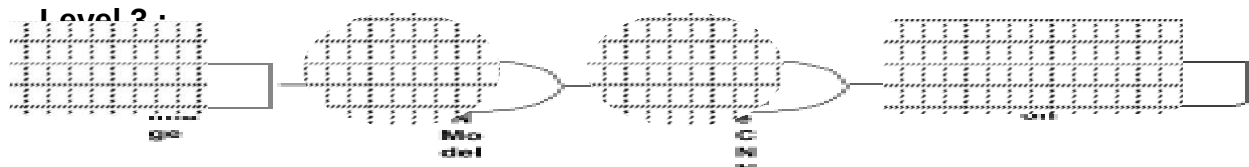
Level 1 :



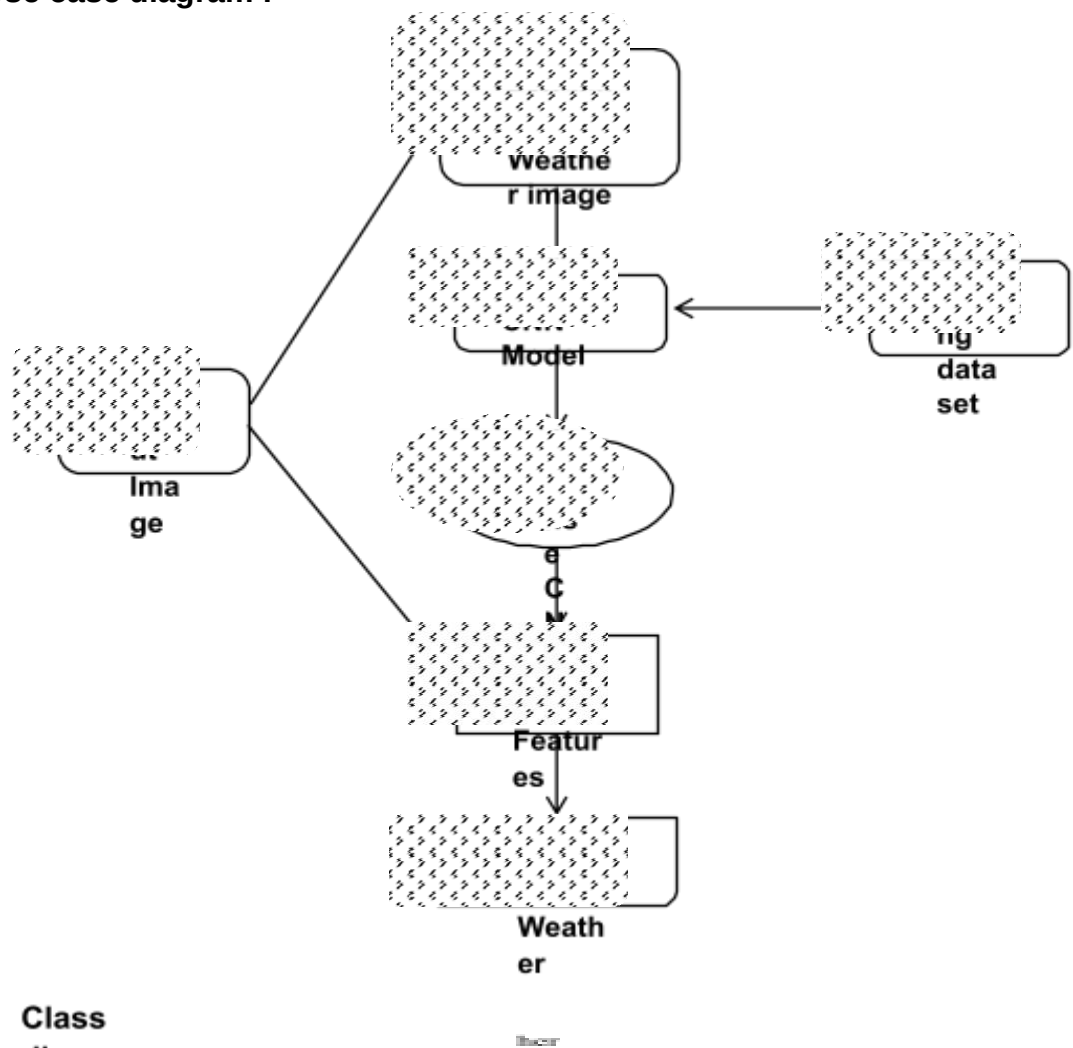
Level 2 :



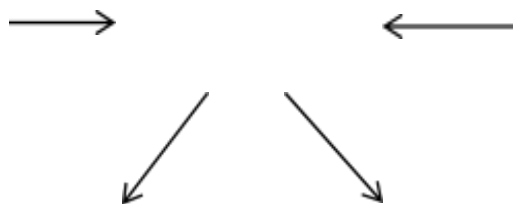
Level 3 :



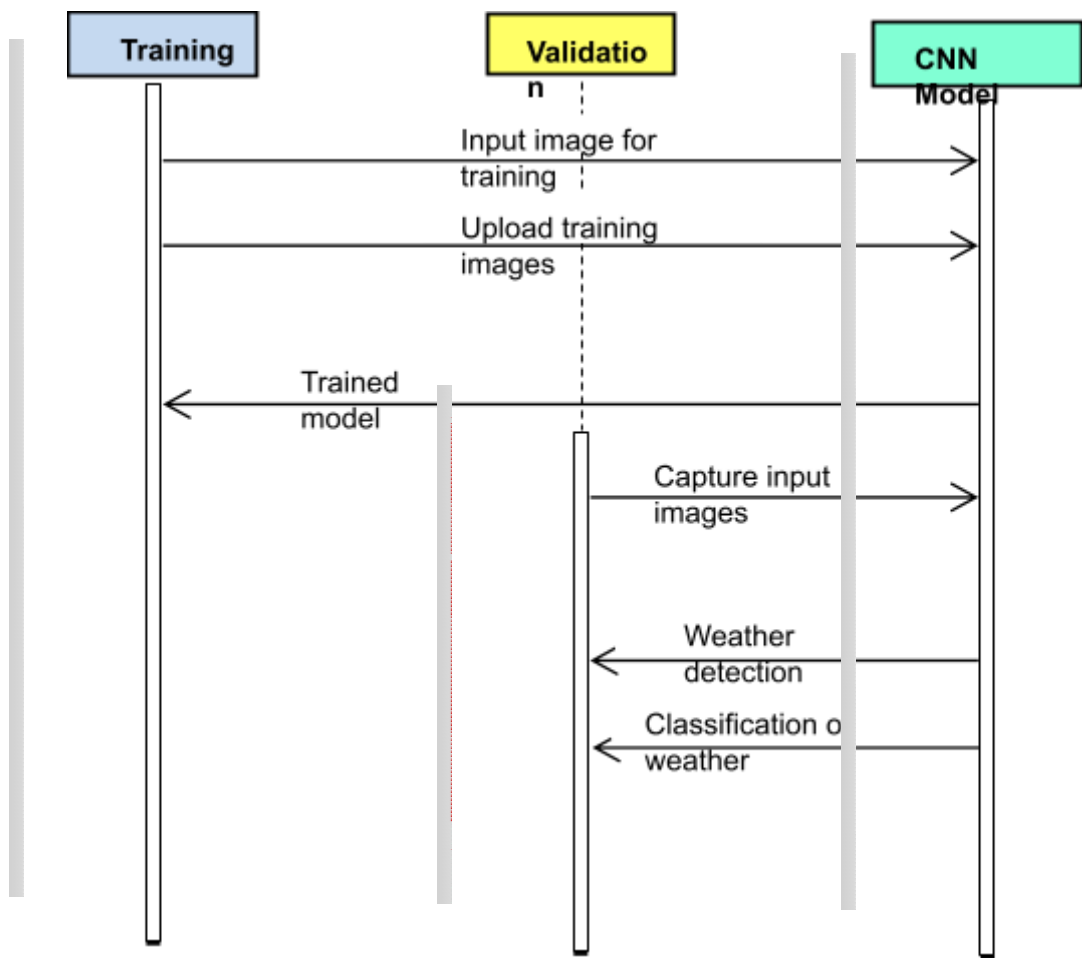
Use case diagram :



Class diagram :



Sequence diagram :



CHAPTER 5

RESULT AND DISCUSSION PERFORMANCE ANALYSIS

MODULES

Data validation process

Exploration data analysis of pre-processing

To training a model by given dataset using sklearn package

GUI based prediction results of weather image

DATA VALIDATION PROCESS

Validation techniques in machine learning are used to get the error rate of the Machine Learning (ML) model, which can be considered as close to the true error rate of the dataset. If the data volume is large enough to be representative of the population, you may not need the validation techniques. However, in real-world scenarios, to work with samples of data that may not be a true representative of the population of given dataset. To finding the missing value, duplicate value and description of data type whether it is float variable or integer. The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyper parameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. The validation set is used to evaluate a given model, but this is for frequent evaluation. It as machine learning engineers uses this data to fine-tune the model hyper parameters. Data collection, data analysis, and the process of addressing data content, quality, and structure can add up to a time-consuming to-do list. During the process of data identification, it helps to understand your data and its properties; this knowledge will help you choose which algorithm to use to build your model. For example, time series data can be analyzed by regression algorithms; classification algorithms can be used to analyze discrete data. (For example to show the data type format of given dataset).

DATA ANALYSIS

Data visualization is an important skill in applied statistics and machine learning. Statistics does indeed focus on quantitative descriptions and estimations of data. Data visualization provides an important suite of tools for gaining a

qualitative understanding. This can be helpful when exploring and getting to know

a dataset and can help with identifying patterns, corrupt data, outliers, and much more. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral and stakeholders than measures of association or significance. Data visualization and exploratory data analysis are whole fields themselves and it will recommend a deeper dive into some the books mentioned at the end.

Sometimes data does not make sense until it can look at in a visual form, such as with charts and plots. Being able to quickly visualize of data samples and others is an important skill both in applied statistics and in applied machine learning. It will discover the many types of plots that you will need to know when visualizing data in Python and how to use them to better understand your own data.

How to chart time series data with line plots and categorical quantities with bar charts.

How to summarize data distributions with histograms and box plots.

How to summarize the relationship between variables with scatter plots.

OUTLIER DETECTION PROCESS

Many machine learning algorithms are sensitive to the range and distribution of attribute values in the input data. Outliers in input data can skew and mislead the training process of machine learning algorithms resulting in longer training times, less accurate models and ultimately poorer results.

Even before predictive models are prepared on training data, outliers can result in misleading representations and in turn misleading interpretations of collected data. Outliers can skew the summary distribution of attribute values in descriptive statistics like mean and standard deviation and in plots such as histograms and scatterplots, compressing the body of the data. Finally, outliers can represent examples of data instances that are relevant to the problem such as anomalies in the case of fraud detection and computer security.

It couldn't fit the model on the training data and can't say that the model will work accurately for the real data. For this, we must assure that our model got the correct patterns from the data, and it is not getting up too much noise. Cross-validation is a technique in which we train our model using the subset of the data-

set and then evaluate using the complementary subset of the data-set.

The three steps involved in cross-validation are as follows :

1. Reserve some portion of sample data-set.
2. Using the rest data-set train the model.
3. Test the model using the reserve portion of the data-set.

GUI BASED RESULTS

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis. To achieving better results from the applied model in Deep Learning method of the data has to be in a proper manner. Some specified Deep Learning model needs information in a specified format; for example, AlexNet algorithm does not support null values. Therefore, to execute AlexNet algorithm null values have to be managed from the original raw data set. And another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in given dataset.

A way to do this is to use different visualization methods to show the average accuracy, variance and other properties of the distribution of model accuracies. In the next section you will discover exactly how you can do that in Python with scikit-learn. The key to a fair comparison of deep learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data and it can achieve this by forcing each algorithm to be evaluated on a consistent test harness.

In the example below 4 different algorithms are compared :

Manual CNN

AlexNet

LeNet

GUI Deployment

Module-01 : Manual CNN

Real-time weather detection involves detection of weather from a series of frames from a video capturing device. While the hardware requirements for such a system

are far more stringent, from a computer vision stand point, real-time face

detection is actually a far simpler process than detecting a face in a static image.

Framework : Tensorflow

Pre-requisites :

- ✓ Opencv installed
- ✓ A working webcam

Opencv (Open Source Computer Vision) is a computer vision library that contains various functions to perform operations on Images or videos. Opencv library can be used to perform multiple operations on videos. It comes with many powerful video editing functions. In current scenario, techniques such as image scanning, face recognition can be accomplished using opencv. This library can be used to perform multiple operations on videos. Let's try to do interesting using cv2 or cv based on python version package. Take a video as input by webcam and break the video into frame by frame and save that frame. Now, number of operations can be performed on these frames like reversing the video file or crop the video etc.

Step by step procedure of capture the given image :

- ✓ To import the necessary libraries.
- ✓ Then 'cv2.VideoCapture(0)' starts capturing from the default camera(here 0 means default camera).
- ✓ Capturing frames in an infinite loop will be next step. After that, we convert the captured video into a gray-scale via 'cv2.COLOR_BGR2GRAY' command(Open -CV captures in Blue-Green-Red as opposed to Red-Green-Blue).
- ✓ We output both the frames(original video and gray-scaled video) by 'cv2.imshow' command.
- ✓ To quit from the setup, we need to define a key. In our case "s" is the key we have defined to break the loop.
- ✓ Lastly we release all the cameras from the open-CV and destroy all the windows opened in the background. This is done in order to release all the background process happening after quitting the setup.

Gray Scale Base : Gray information within a face can also be treating as important features. Facial features such as eyebrows, pupils, and lips appear generally darker than their surrounding facial regions. Various recent feature

extraction algorithms search for local gray minima within segmented facial regions. The input images are first enhanced by contrast-stretching and gray-scale

morphological routines to improve the quality of local dark patches and thereby make detection easier. The extraction of dark patches is achieved by low-level gray-scale threshold. Based method and consist three levels. Faces gray scale behavior in pyramid images and it utilizes hierarchical Face location consist three levels. Higher two level based on images at different resolution and the lower level, edge detection method.

Edge Base : This work was based on analyzing line drawings of the faces from photographs, aiming to locate facial features. To initially the images are enhanced by applying median filter for noise removal and histogram equalization for contrast adjustment.

Module-02 : AlexNet

AlexNet : AlexNet is the name of a convolutional neural network (CNN), designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D advisor. AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. The network achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of graphics processing units (GPUs) during training.

Network design of AlexNet : AlexNet contained eight layers; the first five were convolutional layers, some of them followed by max-pooling layers, and the last three were fully connected layers. It used the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid.

Module-3 : LeNet

LeNet is a convolutional neural network structure proposed by Yann LeCun et al. in 1989. In general, LeNet refers to LeNet-5 and is a simple convolutional neural network. Convolutional neural networks are a kind of feed-forward neural network whose artificial neurons can respond to a part of the surrounding cells in the coverage range and perform well in large-scale image processing.

Structure of LeNet : As a representative of the early convolutional neural

network, LeNet possesses the basic units of convolutional neural network, such as convolutional layer, pooling layer and full connection layer, laying a foundation for

the future development of convolutional neural network.

Layer C1 is a convolution layer with six convolution kernels of 5×5 and the size of feature mapping is 28×28 , which can prevent the information of the input image from falling out of the boundary of convolution kernel.

Layer S2 is the subsampling/pooling layer that outputs 6 feature graphs of size 14×14 . Each cell in each feature map is connected to 2×2 neighborhoods in the corresponding feature map in C1.

Layer C3 is a convolution layer with 16 5×5 convolution kernels. The input of the first six C3 feature maps is each continuous subset of the three feature maps in S2, the input of the next six feature maps comes from the input of the four continuous subsets, and the input of the next three feature maps comes from the four discontinuous subsets. Finally, the input for the last feature graph comes from all feature graphs of S2.

Layer S4 is similar to S2, with size of 2×2 and output of 16 5×5 feature graphs.

Layer C5 is a convolution layer with 120 convolution kernels of size 5×5 . Each cell is connected to the 5×5 neighborhood on all 16 feature graphs of S4. Here, since the feature graph size of S4 is also 5×5 , the output size of C5 is 1×1 . So S4 and C5 are completely connected. C5 is labeled as a convolutional layer instead of a fully connected layer, because if LeNet-5 input becomes larger and its structure remains unchanged, its output size will be greater than 1×1 , i.e. not a fully connected layer.

F6 layer is fully connected to C5, and 84 feature graphs are output.

Features of LeNet :

- Every convolutional layer includes three parts: convolution, pooling, and nonlinear activation functions

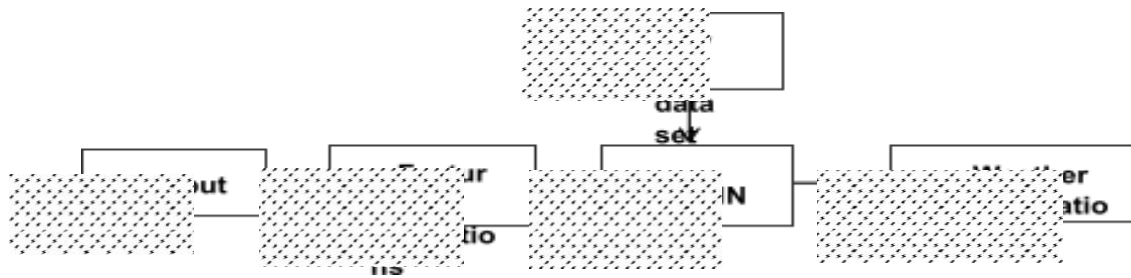
- Using convolution to extract spatial features (Convolution was called receptive fields originally)

- Subsampling average pooling layer

- tanh activation function

- Using MLP as the last classifier

- Sparse connection between layers to reduce the complexity of computation



Module-4 : GUI Deployment

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python. The name Tkinter comes from Tk interface. Tkinter is free software released under a Python license.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps :

- Import the Tkinter module.

- Create the GUI application main window.

- Add one or more of the above-mentioned widgets to the GUI application.

- Enter the main event loop to take action against each event triggered by the user.

Neural networks gaining much more attention in many pattern recognition problems, object recognition, and autonomous robot driving. Since face detection with class pattern recognition problem, various neural network algorithms have been proposed. The advantage of using neural networks for face detection is the feasibility of training a system to capture the complex class conditional density of face patterns. However, one demerit is that the network architecture has to be extensively tuned (number of layers, number of nodes, learning rates, etc.) to get exceptional performance. The first stage having two parallel sub networks in which the inputs are filtered intensity values from an original image. The inputs to the second stage network consist of the outputs from the sub networks and extracted feature values. An output at the second stage shows the presence of a face in the input region. Their network consists of four layers with 1,024 input units, 256 units in the first hidden layer, eight units in the second hidden layer, and two output units.

Libraries Required :

- ✓ **numpy** : To process the image matrices
- ✓ **open-cv** : To process the image like converting them to grayscale and etc.
- ✓ **os** : To access the file system to read the image from the train and test directory from our machines
- ✓ **random** : To shuffle the data to overcome the biasing
- ✓ **matplotlib** : To display the result of our predictive outcome
- ✓ **tensorflow** : Just to use the tensor board to compare the loss and adam curve our result data or obtained log

OpenCv : OpenCv (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. Opencv was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, Opencv makes it easy for businesses to utilize and modify the code. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. Opencv has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Numpy : NumPy is an acronym for "Numeric Python" or "Numerical Python". It is an open source extension module for Python, which provides fast precompiled functions for mathematical and numerical routines. Furthermore, NumPy enriches the programming language Python with powerful data structures for efficient computation of multi-dimensional arrays and matrices. The implementation is even aiming at huge matrices and arrays. Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and

arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones,

- ✓ A powerful N-dimensional array object
- ✓ Sophisticated (broadcasting) functions
- ✓ Tools for integrating C/C++ and Fortran code
- ✓ Useful linear algebra, Fourier Transform, and random number capabilities

Numpy Array : A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array. The shape of an array is a tuple of integers giving the size of the array along each dimension.

RESULTS AND DISCUSSION

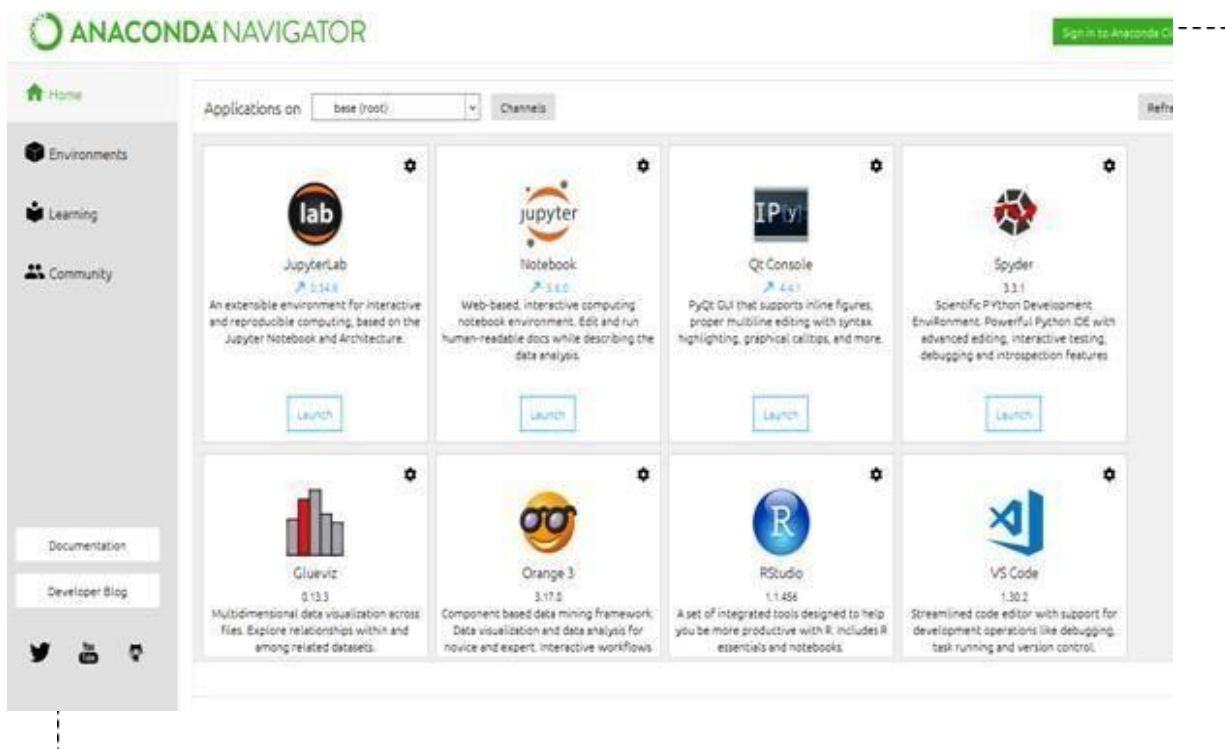


Fig 5.1 : Open the anaconda navigator



	Normalization + Raw image	
	CNN Model + Dense + Weight	
	Training Model + Given image	
	Weathe r Detecti on + Harr-Cascade	
	Weather Classification + Features	

Fig 5.2 : Launch the jupyter notebook platform

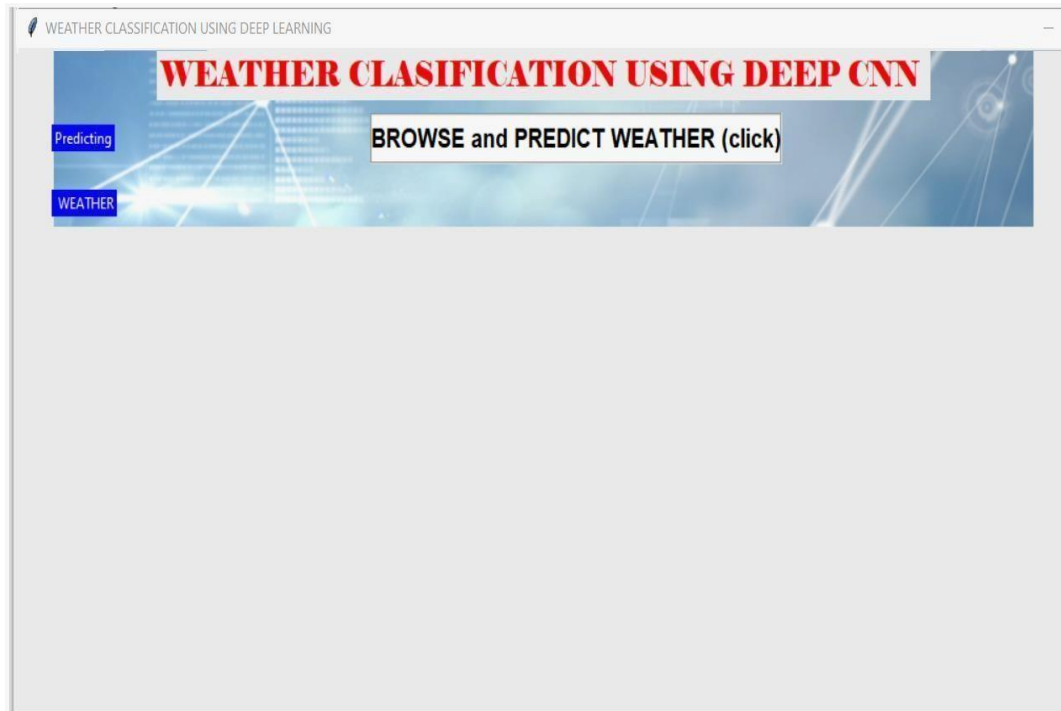


Fig 5.3 : Open the correspondent result folder

RESULTS

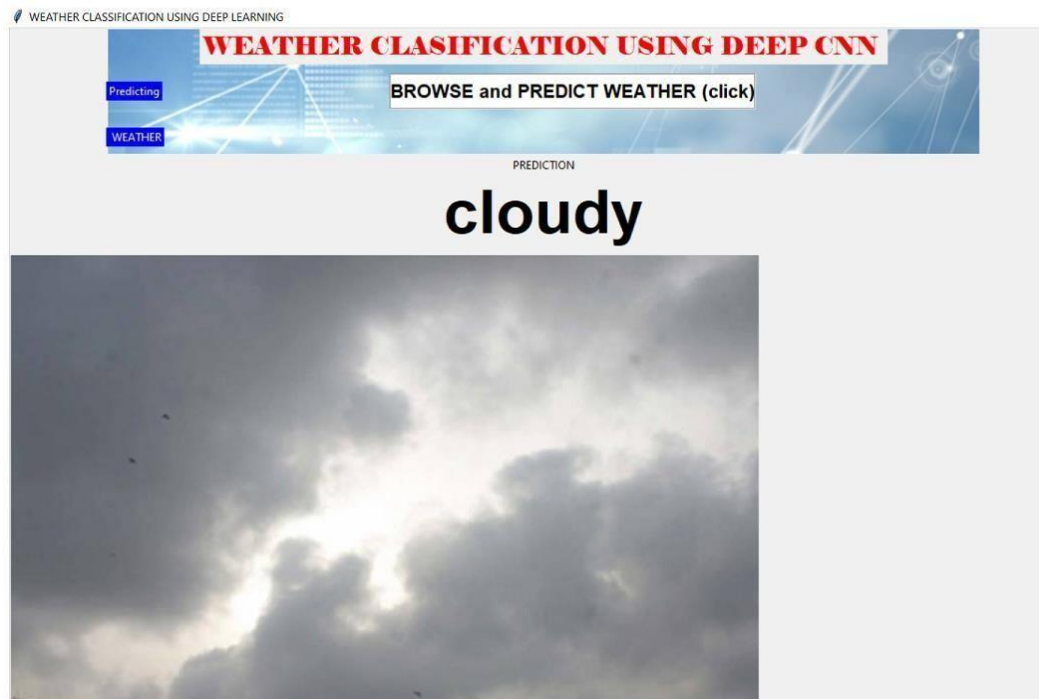
Before getting result :

Output (Offline GUI application)

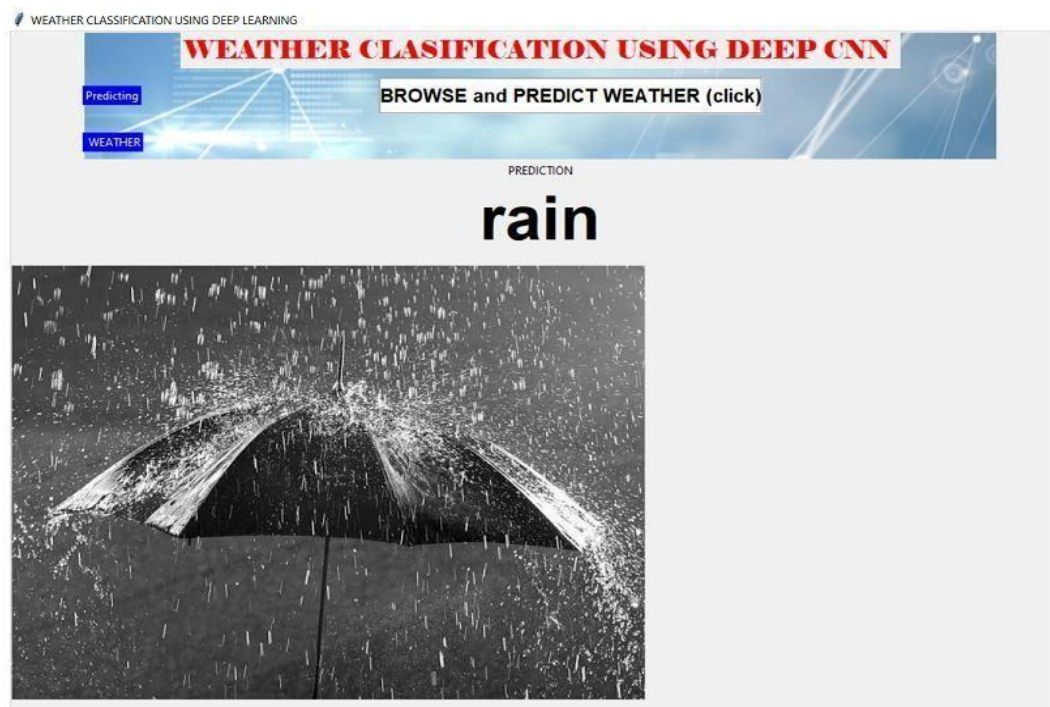


After getting results :

Final Output 1 :



Final Output 2



Final Output 3



Final Output 4



CHAPTER 6

SUMMARY AND CONCLUSION

CONCLUSION

The analytical process started from data cleaning and processing, missing value, exploratory analysis and finally model building and evaluation. Finally we predict the weather image using deep learning algorithm with different results. The best accuracy on public test set is higher accuracy score by deep learning method from calculating cross validation checking, precision, recall and F1score in future. This brings some of the following insights about weather prediction.

FUTURE WORK

India Meteorological Department wants to automate in detecting the air quality is good or not from eligibility process (real time).

To automate this process by show the prediction result in web application or desktop application or mobile application.

To optimize the work to implement in Artificial Intelligence environment.

ADVANTAGES

These reports and all available data sources offers significant advantages of weather image classification using deep learning.

For example, rainfall measurement is useful on its own, but for the best level of flood prevention and detection, it is essential to also look at drainage information and local sewer systems due to versatile meteorological infrastructure.

Using the same infrastructure, the radar system that can detect oncoming storms will also be useful for gathering general rainfall data for the farming sector.

It highlights some observations on future research issues, challenges, and needs.

APPLICATION

This process by show the prediction result in web application or desktop application or mobile application.

To optimize the work to implement in Artificial Intelligence environment.

REFERENCES

- [1] Ruggero Donida Labati, Angelo Genovese, Vincenzo Piur, Fabio Scotti, Gianluca Sforza ; A Decision Support System for Wind Power Production ; Year : 2018
- [2] Muhammad Qamar Raza, Mithulananthan, Jiaming Li, Kwang Y. Lee, Hoay Beng Gooi ; An Ensemble Framework For Day-Ahead Forecast of PV Output Power in Smart Grids ; Year : 2018
- [3] Mingfei Ban, Jilai Yu, Danyang Guo, Yiyun Yao ; Considering the Differentiating Health Impacts of Fuel Emissions in Optimal Generation Scheduling ; Year : 2018
- [4] M.J. Sanjari, H.B. Gooi, Nirmal-Kumar C. Nair ; Power Generation Forecast of Hybrid PV-Wind System ; Year : 2019
- [5] Yun Li, Ben Jones ; The Use of Extreme Value Theory for Forecasting Long-Term Substation Maximum Electricity Demand ; Year : 2019
- [6] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012
- [7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun "Deep Residual Learning for Image Recognition" Computer Vision and Pattern Recognition (cs.CV). 2015
- [8] Chih-Wei Hsu, Chih-Jen Lin 'A Comparison of Methods for Multiclass Support Vector Machines "IEEE Transaction on Neural Networks.2002
- [9] Cosmin Ancuti, Codruta O. Ancuti, Christophe De Vleeschouwer "DHAZY: A Dataset to Evaluate Quantitatively Dehazing Algorithms" IEEE International Conference on Image Processing 2016
- [10] C. Lu, D. Lin, J. Jia, and C.-K. Tang, "Two-class weather classification," IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017
- [11] Yun-Fu Liu, Da-Wei Jaw, Shih-Chia Huang, Jenq-Neng Hwang "DesnowNet: Context-Aware Deep Network for Snow Removal" Computer Vision and Pattern Recognition 2017

APPENDIX

A. SOURCE CODE

```
import os
import numpy as np
import pandas as pd
import random
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense, Flatten, Dropout,
BatchNormalization
from tensorflow.keras.layers import Conv2D, SeparableConv2D, MaxPool2D,
LeakyReLU, Activation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau,
EarlyStopping
from IPython.display import display
from os import listdir
from os.path import isfile, join
from PIL import Image
import glob
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
import warnings
warnings.filterwarnings('ignore')
dir_name_train_cloudy = 'dataset/train/cloudy'
```

```
dir_name_train_rain = 'dataset/train/rain'
```

```

dir_name_train_shine = 'dataset/train/shine'
dir_name_train_sunrise = 'dataset/train/sunrise'
def plot_images(item_dir, n=6):
    all_item_dir = os.listdir(item_dir)
    item_files = [os.path.join(item_dir, file) for file in all_item_dir][:n]

    plt.figure(figsize=(35, 10))
    for idx, img_path in enumerate(item_files):
        plt.subplot(2, n, idx+1)
        img = plt.imread(img_path)
        plt.imshow(img, cmap='gray')
        plt.axis('off')

    plt.tight_layout()
def Images_details_Print_data(data, path):
    print(" ===== Images in: ", path)
    for k, v in data.items():
        print("%s:\t%s" % (k, v))

def Images_details(path):
    files = [f for f in glob.glob(path + "**/*.jpg", recursive=True)]
    data = {}
    data['images_count'] = len(files)
    data['min_width'] = 10**100 # No image will be bigger than that
    data['max_width'] = 0
    data['min_height'] = 10**100 # No image will be bigger than that
    data['max_height'] = 0

    for f in files:
        im = Image.open(f)
        width, height = im.size
        data['min_width'] = min(width, data['min_width'])

```

```
data['max_width'] = max(width, data['max_height'])  
data['min_height'] = min(height, data['min_height'])
```

```

        data['max_height'] = max(height, data['max_height'])
    Images_details_Print_data(data, path)
print("")
print("Training data for Cloudy images:")
print("")
Images_details(dir_name_train_cloudy)
print("")
plot_images(dir_name_train_cloudy, 5)
print("")
print("Training data for Rainy images:")
print("")
Images_details(dir_name_train_rain)
print("")
plot_images(dir_name_train_rain, 5)
print("")
print("Training data for Shine images:")
print("")
Images_details(dir_name_train_shine)
print("")
plot_images(dir_name_train_shine, 5)
print("")
print("Training data for Sunrise images:")
print("")
Images_details(dir_name_train_sunrise)
print("")
plot_images(dir_name_train_sunrise, 5)
Classifier=Sequential()
Classifier.add(Convolution2D(32,(3,3),input_shape=(128,128,3),activation='relu'))
Classifier.add(MaxPooling2D(pool_size=(2,2)))
Classifier.add(Flatten())
Classifier.add(Dense(4, activation='relu'))
Classifier.add(Dense(4, activation='softmax'))

```

```
Classifier.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])
```

```

train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)
training_set=train_datagen.flow_from_directory('dataset/train',target_size=(128,128),batch_size=32,class_mode='categorical')
test_set=test_datagen.flow_from_directory('dataset/test',target_size=(128,128),batch_size=32,class_mode='categorical')
img_dims = 150
epochs = 20
batch_size = 32
history = Classifier.fit_generator(
    training_set, steps_per_epoch=training_set.samples // batch_size,
    epochs=epochs,
    validation_data=test_set,validation_steps=test_set.samples // batch_size)
def graph():

    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
import os

```

```
import numpy as np
import pandas as pd
```

```
import random
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense, Flatten, Dropout,
BatchNormalization
from tensorflow.keras.layers import Conv2D, SeparableConv2D, MaxPool2D,
LeakyReLU, Activation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau,
EarlyStopping
from IPython.display import display
from os import listdir
from os.path import isfile, join
from PIL import Image
import glob

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
import warnings
warnings.filterwarnings('ignore')
model = Sequential()
model.add(Conv2D(filters = 6,
                 kernel_size = 5,
                 strides = 1,
                 activation = 'relu',
                 input_shape = (225,225,3)))
model.add(MaxPooling2D(pool_size = 2, strides = 2))
model.add(Conv2D(filters = 16,
```

kernel_size = 5,

```

        strides = 1,
        activation = 'relu',
        input_shape = (14,14,6)))

model.add(MaxPooling2D(pool_size = 2, strides = 2))
model.add(Flatten())
model.add(Dense(units = 32, activation = 'relu'))
model.add(Dense(units = 16, activation = 'relu'))
model.add(Dense(units = 4, activation = 'softmax'))

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
['accuracy','AUC','Precision','Recall','TruePositives','TrueNegatives','FalsePositives',
'FalseNegatives'])

train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)

training_set=train_datagen.flow_from_directory('dataset/train',target_size=(225,225),batch_size=32,class_mode='categorical')
test_set=test_datagen.flow_from_directory('dataset/test',target_size=(225,225),batch_size=32,class_mode='categorical')

img_dims = 150
epochs = 30
batch_size = 32

history = model.fit_generator(
    training_set,steps_per_epoch=training_set.samples // batch_size,
    epochs=epochs,
    validation_data=test_set,validation_steps=test_set.samples // batch_size)

import h5py
model.save('weather2.h5')

def graph():

    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy Curve')

```

```
plt.ylabel('Accuracy')  
plt.xlabel('Epoch')
```

```

plt.legend(['Train', 'Valid'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss or Log loss Curve')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Valid'], loc='upper left')
plt.show()

plt.plot(history.history['AUC'])
plt.plot(history.history['val_AUC'])
plt.title('Model AUC Curve')
plt.ylabel('AUC')
plt.xlabel('Epoch')
plt.legend(['Train', 'Valid'], loc='upper left')
plt.show()

print("[INFO] Calculating model accuracy")
scores = model.evaluate(test_set)
precision=scores[3]
recall=scores[3]
true_positives=scores[5]
true_negatives=scores[6]
false_positives=scores[7]
false_negatives=scores[8]
confusion_matrix=[[true_positives,false_positives],[false_negatives,true_negatives]]
print(f"Accuracy          : {scores[1]*100}")
print("")
print("Confusion Matrix      : ",confusion_matrix)
print("")
print("precision Score       : ",precision)

```

```
print("")  
print("Recall or Sensitivity Score : ",recall)
```

```

print("")
f1=(precision*recall)/(precision+recall)
F1_Score=2*f1
print("F1 Score          :
",F1_Score) print("")
Specificity=true_negatives/true_negatives+false_positive
s print("Specificity Score      : ",Specificity)
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Activation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import EarlyStopping
import warnings
warnings.filterwarnings('ignore')
model = Sequential()

model.add(Conv2D(filters=96, input_shape=(224,224,3), kernel_size=(11,11),
strides=(4,4), padding='valid'))
model.add(Activation('relu'))

```

```
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))  
model.add(Flatten())
```

```
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(96))
```

```
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(32))
```

```
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(4))
model.add(Activation('softmax'))
model.summary()
```

```
model.compile(loss = 'categorical_crossentropy', optimizer='adam',
metrics=['accuracy','AUC','Precision','Recall','TruePositives','TrueNegatives','False
Positives','FalseNegatives'])
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,
horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)
training_set=train_datagen.flow_from_directory('dataset/train',target_size=(224,224),
batch_size=32,class_mode='categorical')
test_set=test_datagen.flow_from_directory('dataset/test',target_size=(224,224),
batch_size=32,class_mode='categorical')
img_dims = 150
epochs = 30
batch_size = 50
history = model.fit(
    training_set, steps_per_epoch=training_set.samples // batch_size,
    epochs=epochs,
    validation_data=test_set,validation_steps=test_set.samples // batch_size)
```

```
import matplotlib.pyplot as plt  
def graph():
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
plt.plot(history.history['auc'])
plt.plot(history.history['val_auc'])
plt.title('Model AUC Curve')
plt.ylabel('AUC')
plt.xlabel('Epoch')
plt.legend(['Train', 'Valid'], loc='upper left')
plt.show()
```

```
print("[INFO] Calculating model accuracy")
scores = model.evaluate(test_set)
print(f"Test Accuracy: {scores[1]*100}")
precision=scores[3]
recall=scores[3]
true_positives=scores[5]
true_negatives=scores[6]
false_positives=scores[7]
```

```
false_negatives=scores[8]
```

```
confusion_matrix=[[true_positives,false_positives],[false_negatives,true_negatives]]
```

```

import h5py
model.save("weather.h5")
print(f'Accuracy          : {scores[1]*100}')
print("")
print("Confusion Matrix      : ",confusion_matrix)
print("")
print("precision Score        : ",precision)
print("")
print("Recall or Sensitivity Score : ",recall)
print("")
f1=(precision*recall)/(precision+recall)
F1_Score=2*f1
print("F1 Score              :
",F1_Score) print("")
Specificity=true_negatives/true_negatives+false_positive
s print("Specificity Score    : ",Specificity)
import h5py
import tensorflow as tf
from tensorflow.keras.models import load_model
classifierLoad=tf.keras.models.load_model('weather5.h5')
import numpy as np
from tensorflow.keras.preprocessing import image
test_image=image.load_img('5.jpg',target_size=(225,225))
import matplotlib.pyplot as plt
img = plt.imshow(test_image)
test_image=image.img_to_array(test_image)
test_image=np.expand_dims(test_image,axis=0)
test_image=image.img_to_array(test_image)
prediction = result[0]
classes=training_set.class_indices
classes
{'cloudy': 0, 'rain': 1, 'shine': 2, 'sunrise': 3}

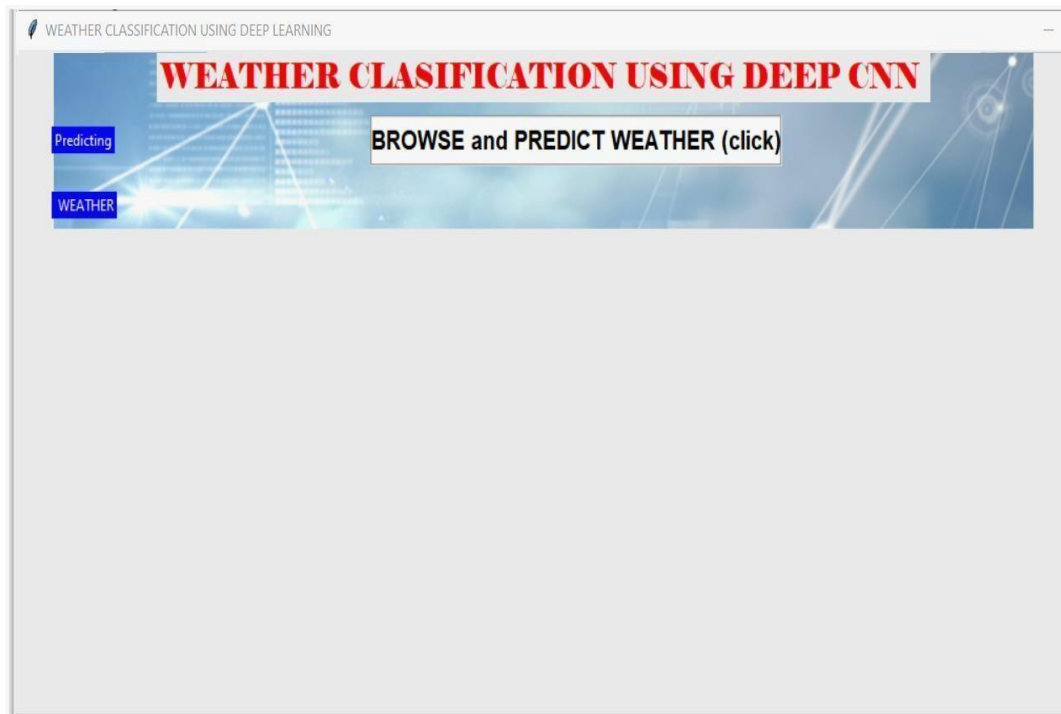
```

```
prediction=list(prediction)  
prediction
```

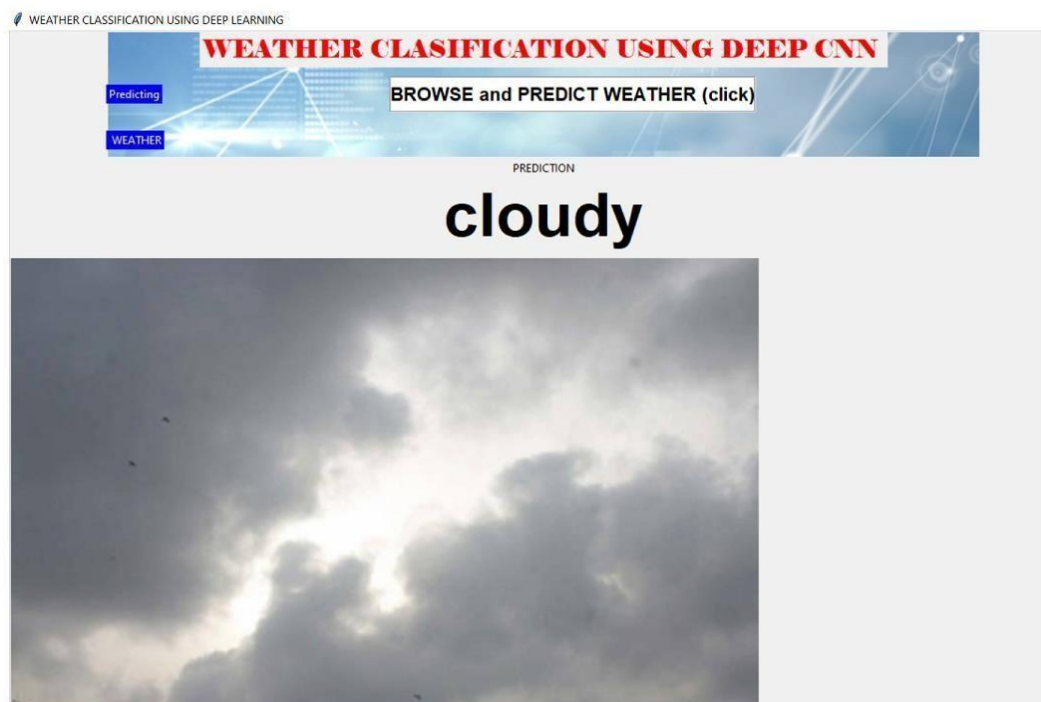
```
classes=['cloudy', 'rain', 'shine', 'sunrise']
output=zip(classes,prediction)
output=dict(output)
output
{'cloudy': 0.0, 'rain':0.0, 'shine':0.0, 'sunrise':1.0}
if output['cloudy']==1.0 :
    print("cloudy")
elif output['rain']==1.0:
    print("rain")
elif output['shine']==1.0:
    print("shine")
elif output['sunrise']==1.0:
    print("sunrise")
```

B. SCREENSHOTS

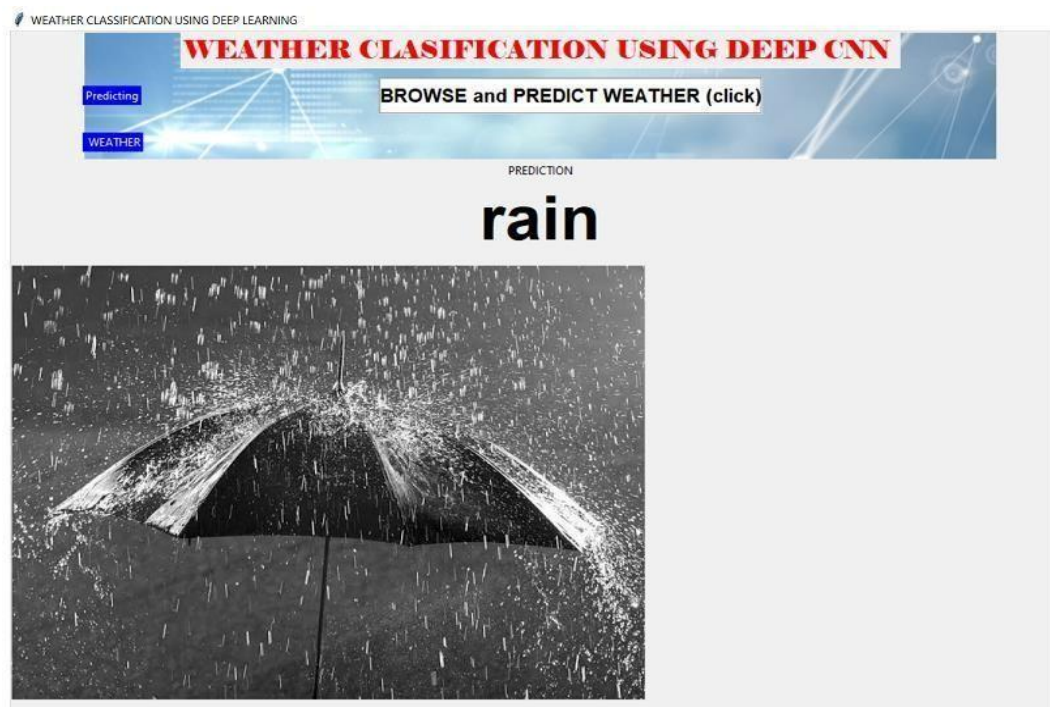
Output (Offline GUI application)



Final Output 1 :



Final Output 2



Final Output 3



Final Output 4



~~~~~END OF PROJECT REPORT~~~~~