**CONVOLUTION NEURAL NETWORK DEEP LEARNING FOR NATURAL DISASTER IDENTIFICATIONS**

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **TITLE** | **PAGE.NO** |
| **01** | **ABSTRACT** |  |
| **02** | **EXISTING SYSTEM**  2.1 DRAWBACKS |  |
| **03** | **INTRODUCTION**  3.1 DATA SCIENCE  3.2 ARTIFICIAL INTELLIGENCE |  |
| **04** | **MACHINE LEARNING** |  |
| **05** | **PREPARING DATASET** |  |
| **06** | **PROPOSED SYSTEM**  6.1 ADVANTAGES |  |
| **07** | **LITERATURE SURVEY** |  |
| **08** | **SYSTEM STUDY**  8.1 OBJECTIVES  8.2 PROJECT GOAL  8.3 SCOPE OF THE PROJECT |  |
| **09** | **FEASIBILITY STUDY** |  |
| **10** | **LIST OF MODULES** |  |
| **11** | **PROJECT REQUIREMENTS**  11.1 FUNCTIONAL REQUIREMENTS  11.2 NON-FUNCTIONAL REQUIREMENTS |  |
| **12** | **ENVIRONMENT REQUIREMENT** |  |
| **13** | **SOFTWARE DESCRIPTION**  13.1 ANACONDA NAVIGATOR  13.2 JUPYTER NOTEBOOK |  |
| **14** | **PYTHON** |  |
| **15** | **SYSTEM ARCHITECTURE** |  |
| **16** | **WORKFLOW DIAGRAM** |  |
| **17** | **USECASE DIAGRAM** |  |
| **18** | **CLASS DIAGRAM** |  |
| **19** | **ACTIVITY DIAGRAM** |  |
| **20** | **SEQUENCE DIAGRAM** |  |
| **21** | **ER – DIAGRAM** |  |
| **22** | **MODULE DESCRIPTION**  22.1 MODULE DIAGRAM  22.2 MODULE GIVEN INPUT EXPECTED OUTPUT |  |
| **23** | **DEPLOYMENT** |  |
| **24** | **HTML** |  |
| **25** | **CSS** |  |
| **26** | **CODING** |  |
| **27** | **CONCLUSION** |  |
| **28** | **FUTURE WORK** |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **TITLE** | **PAGE.NO** |
| **01** | **SYSTEM ARCHITECTURE** |  |
| **02** | **WORKFLOW DIAGRAM** |  |
| **03** | **USECASE DIAGRAM** |  |
| **04** | **CLASS DIAGRAM** |  |
| **05** | **ACTIVITY DIAGRAM** |  |
| **06** | **SEQUENCE DIAGRAM** |  |
| **07** | **ER – DIAGRAM** |  |
| **08** | **MODULE DIAGRAM** |  |

**LIST OF SYMBOLS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NOTATION**  **NAME** | **NOTATION** | **DESCRIPTION** |
| 1. | Class | *Class Name*  *-attribute*  *-attribute*  *+operation*  *+operation*  *+operation*  *+ public*  *-private*  *# protected* | Represents a collection of similar entities grouped together. |
| 2. | Association | nAME  Class B  Class A    Class B  Class A | Associations represents static relationships between classes. Roles representsthe way the two classes see each other. |
| 3. | Actor | Class A  Class A  Class B  Class B | It aggregates several classes into a single classes. |
| 4. | Aggregation | Interaction between the system and external environment |

|  |  |  |  |
| --- | --- | --- | --- |
| 5. | Relation  (uses) | uses | Used for additional process communication. |
| 6. | Relation  (extends) | EXTENDS | Extends relationship is used when one use case is similar to another use case but does a bit more. |
| 7. | Communication |  | Communication between various use cases. |
| 8. | State | State | State of the processs. |
| 9. | Initial State |  | Initial state of the object |
| 10. | Final state |  | Final state of the object |
| 11. | Control flow |  | Represents various control flow between the states. |
| 12. | Decision box |  | Represents decision making process from a constraint |
| 13. | Usecase |  | Interact ion between the system and external environment. |

|  |  |  |  |
| --- | --- | --- | --- |
| 14. | Component |  | Represents physical modules which is a collection of components. |
| 15. | Node |  | Represents physical modules which are a collection of components. |
| 16. | Data Process/State |  | A circle in DFD represents a state or process which has been triggered due to some event or acion. |
| 17. | External entity |  | Represents external entities such as keyboard,sensors,etc. |
| 18. | Transition |  | Represents communication that occurs between processes. |
| 19. | Object Lifeline |  | Represents the vertical dimensions that the object communications. |
| 20. | Message | Message | Represents the message exchanged. |

**1. ABSTRACT:**

In recent years, the frequency and severity of natural disasters have escalated, necessitating advanced technologies for timely identification and mitigation. Deep learning, particularly Cataclysmic Convolutional Neural Networks (CCNNs), has emerged as a potent tool for automating disaster identification processes. This paper presents a novel approach utilizing Tensor Flow, a popular deep learning framework, for natural disaster identification. We propose a CNN architecture optimized for identifying various types of natural calamities, including earthquakes, floods, wildfires, and more, from Image classification. Our model harnesses the power of convolutional layers to extract spatial features and temporal dependencies, enabling accurate detection of disaster events. We demonstrate the efficacy of our approach through extensive experimentation on benchmark datasets, achieving state-of-the-art performance in terms of both accuracy and computational efficiency. Additionally, we discuss practical implications and potential applications of our methodology in real-world disaster response and management scenarios. This research represents a significant step forward in leveraging deep learning for enhancing early warning systems and disaster preparedness efforts.

**2. Existing System:**

In this paper, we study the deployment of K heterogeneous UAVs to monitor Points of Interest (PoIs) in a disaster zone, where a PoI may represent a school building or an office building, in which people are trapped. A UAV can take images/videos of PoIs and send its collected information back to a nearby rescue station for decision-making. Unlike most existing studies that focused on only homogeneous UAVs, we here study the scheduling of K heterogeneous UAVs, where different UAVs have different energy capacities and functionalities that lead to different monitoring qualities (monitoring rewards) of each PoI. For example, one type of UAVs can take only visual images while the other type of UAVs can take both visual and thermal infrared images. In this paper, we investigate a problem of scheduling K heterogeneous UAVs to monitor PoIs so that the sum of monitoring rewards received by all UAVs is maximized, subject to energy capacity on each UAV. We propose the very first 1 3 -approximation algorithm for this scheduling problem. We also evaluate the performance of the proposed algorithm, using real parameters of commercial UAVs. Experimental results show that the performance of the proposed algorithm is promising, which is improved by 25%, compared with existing algorithms.

**DEMERITS:**

* Energy capacities vary among the UAVs, which can lead to inefficient resource utilization. This limitation might result in shorter operational durations or restricted coverage areas, hindering comprehensive disaster monitoring.
* The existing system's reliance on manual decision-making processes may result in delays in data analysis and response, limiting the system's ability to provide real-time insights into disaster situations.
* They do not implement deploy and accuracy was low.

**3. INTRODUCTION**

Convolutional Neural Networks (CNNs) represent a groundbreaking advancement in deep learning, significantly enhancing our ability to identify and analyze natural disasters. Leveraging their powerful image processing capabilities, CNNs can discern intricate patterns and features within satellite imagery and other data sources, facilitating early detection and accurate classification of natural disaster events such as hurricanes, earthquakes, wildfires, and floods. By automatically extracting and learning from complex visual features, CNNs reduce the need for manual feature engineering and enable real-time monitoring of disaster scenarios. This technological prowess not only accelerates response times and improves situational awareness but also contributes to more effective disaster management and mitigation strategies, ultimately aiding in the protection of lives and infrastructure.

**3.1 Data Science:**

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains.

The term "data science" has been traced back to 1974, when Peter Naur proposed it as an alternative name for computer science. In 1996, the International Federation of Classification Societies became the first conference to specifically feature data science as a topic. However, the definition was still in flux.

The term “data science” was first coined in 2008 by D.J. Patil, and Jeff Hammer bacher, the pioneer leads of data and analytics efforts at LinkedIn and Facebook. In less than a decade, it has become one of the hottest and most trending professions in the market.

Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data.

Data science can be defined as a blend of mathematics, business acumen, tools, algorithms and machine learning techniques, all of which help us in finding out the hidden insights or patterns from raw data which can be of major use in the formation of big business decisions.

**Data Scientist:**

Data scientists examine which questions need answering and where to find the related data. They have business acumen and analytical skills as well as the ability to mine, clean, and present data. Businesses use data scientists to source, manage, and analyze large amounts of unstructured data.

**Required Skills for a Data Scientist:**

* **Programming**: Python, SQL, Scala, Java, R, MATLAB.
* **Machine Learning**: Natural Language Processing, Classification, Clustering.
* **Data Visualization**: Tableau, SAS, D3.js, Python, Java, R libraries.
* **Big data platforms**: MongoDB, Oracle, Microsoft Azure, Cloudera.

**3.2 ARTIFICIAL INTELLIGENCE**:

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

Artificial intelligence (AI) is [intelligence](https://en.wikipedia.org/wiki/Intelligence) demonstrated by [machines](https://en.wikipedia.org/wiki/Machine), as opposed to the natural intelligence [displayed by humans](https://en.wikipedia.org/wiki/Human_intelligence) or [animals](https://en.wikipedia.org/wiki/Animal_cognition). Leading AI textbooks define the field as the study of “[intelligent agents](https://en.wikipedia.org/wiki/Intelligent_agent)” any system that perceives its environment and takes actions that maximize its chance of achieving its goals.

Some popular accounts use the term “artificial intelligence” to describe machines that mimic “cognitive” functions that humans associate with the [human mind](https://en.wikipedia.org/wiki/Human_mind), such as “learning” and “problem solving”, however this definition is rejected by major AI researchers.

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision.

AI applications include advanced web search engines, recommendation systems (used by Youtube, Amazon and Netflix), Understanding human speech (such as Siri or Alexa), self-driving cars (e.g. Tesla), and competing at the highest level in strategic game systems (such as chess and Go), As machines become increasingly capable, tasks considered to require “intelligence” are often removed from the definition of AI, a phenomenon known as the AI effect. For instance, optical character recognition is frequently excluded from things considered to be AI, having become a routine technology.

Artificial intelligence was founded as an academic discipline in 1956, and in the years since has experienced several waves of optimism, followed by disappointment and the loss of funding (known as an “AI winter”), followed by new approaches, success and renewed funding.

AI research has tried and discarded many different approaches during its lifetime, including simulating the Skin cancer, modeling human problem solving, formal logic, large databases of knowledge and imitating animal behavior. In the first decades of the 21st century, highly mathematical statistical machine learning has dominated the field, and this technique has proved highly successful, helping to solve many challenging problems throughout industry and academia.

The various sub-fields of AI research are centered around particular goals and the use of particular tools. The traditional goals of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception and the ability to move and manipulate objects. General intelligence (the ability to solve an arbitrary problem) is among the field’s long-term goals.

To solve these problems, AI researchers use versions of search and mathematical optimization, formal logic, artificial neural networks, and methods based on statistics, probability and economics. AI also draws upon computer science, psychology, linguistics, philosophy, and many other fields.

The field was founded on the assumption that human intelligence “can be so precisely described that a machine can be made to simulate it”. This raises philosophical arguments about the mind and the ethics of creating artificial beings endowed with human-like intelligence.

These issues have been explored by myth, fiction and philosophy since antiquity. Science fiction and futurology have also suggested that, with its enormous potential and power, AI may become an existential risk to humanity.

As the hype around AI has accelerated, vendors have been scrambling to promote how their products and services use AI. Often what they refer to as AI is simply one component of AI, such as machine learning.

AI requires a foundation of specialized hardware and software for writing and training machine learning algorithms. No one programming language is synonymous with AI, but a few, including Python, R and Java, are popular.

In general, AI systems work by ingesting large amounts of labeled training data, analyzing the data for correlations and patterns, and using these patterns to make predictions about future states.

In this way, a chatbot that is fed examples of text chats can learn to produce life like exchanges with people, or an image recognition tool can learn to identify and describe objects in images by reviewing millions of examples.

AI programming focuses on three cognitive skills: learning, reasoning and self-correction.

**Learning processes.** This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called algorithms, provide computing devices with step-by-step instructions for how to complete a specific task.

**Reasoning processes.** This aspect of AI programming focuses on choosing the right algorithm to reach a desired outcome.

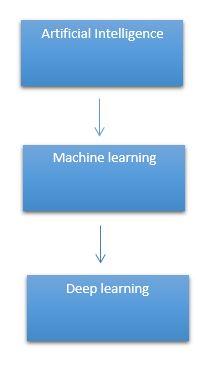
**Self-correction processes.** This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible.

AI is important because it can give enterprises insights into their operations that they may not have been aware of previously and because, in some cases, AI can perform tasks better than humans. Particularly when it comes to repetitive, detail-oriented tasks like analyzing large numbers of legal documents to ensure relevant fields are filled in properly, AI tools often complete jobs quickly and with relatively few errors.

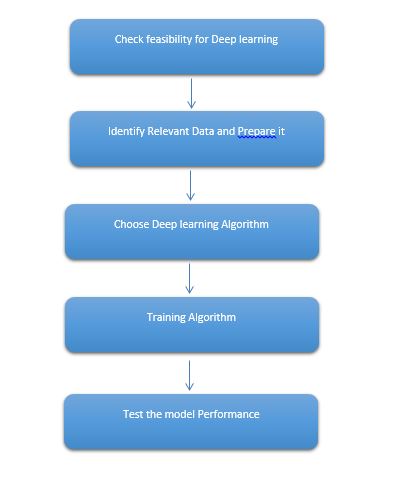
Artificial neural networks and deep learning artificial intelligence technologies are quickly evolving, primarily because AI processes large amounts of data much faster and makes predictions more accurately than humanly possible.

**4. DEEP LEARNING**

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human disease so deep learning is also a kind of mimic of human disease. It’s on hype nowadays because earlier we did not have that much processing power and a lot of data. A formal definition of deep learning is- neurons Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. In disease approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbors. The question here is how it recreates these neurons in a computer. So, it creates an artificial structure called an artificial neural net where we have nodes or neurons. It has some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.



# It need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). It needs to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Choose the Deep Learning Algorithm appropriately. Algorithm should be used while training the dataset. Final testing should be done on the dataset



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, deep reinforcement learning, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, 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.

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological disease. Specifically, neural networks tend to be static and symbolic, while the biological disease of most living organisms is dynamic (plastic) and analogue.

The adjective "deep" in deep learning refers to the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, but that a network with a non-polynomial activation function with one hidden layer of unbounded width can. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Deep learning is a class of [machine learning](https://en.wikipedia.org/wiki/Machine_learning) [algorithms](https://en.wikipedia.org/wiki/Algorithm) that uses multiple layers to progressively extract higher-level features from the raw input. For example, in [image processing](https://en.wikipedia.org/wiki/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.

**Interpretations:**

Deep neural networks are generally interpreted in terms of the universal approximation theorem or probabilistic inference.

The classic universal approximation theorem concerns the capacity of feed-forward neural networks with a single hidden layer of finite size to approximate continuous functions. In 1989, the first proof was published by George Cybenko for sigmoid activation functions and was generalised to feed-forward multi-layer architectures in 1991 by Kurt Hornik. Recent work also showed that universal approximation also holds for non-bounded activation functions such as the rectified linear unit.

The universal approximation theorem for deep neural networks concerns the capacity of networks with bounded width but the depth is allowed to grow proved that if the width of a deep neural network with ReLU activation is strictly larger than the input dimension, then the network can approximate any Lebesgue integrable function; If the width is smaller or equal to the input dimension, then deep neural network is not a universal approximator.

The probabilistic interpretation derives from the field of machine learning. It features inference, as well as the optimization concepts of training and testing, related to fitting and generalization, respectively. More specifically, the probabilistic interpretation considers the activation nonlinearity as a cumulative distribution function. The probabilistic interpretation led to the introduction of dropout as regularizer in neural networks. The probabilistic interpretation was introduced by researchers including Hopfield, Widrow and Narendra and popularized in surveys such as the one by Bishop.

### Deep learning revolution:

### In 2012, a team led by George E. Dahl won the "Merck Molecular Activity Challenge" using multi-task deep neural networks to predict the biomolecular target of one drug. In 2014, Hochreiter's group used deep learning to detect off-target and toxic effects of environmental chemicals in nutrients, household products and drugs and won the "Tox21 Data Challenge" of NIH, FDA and NCATS.

Significant additional impacts in image or object recognition were felt from 2011 to 2012. Although CNNs trained by back-propagation had been around for decades, and GPU implementations of NNs for years, including CNNs, fast implementations of CNNs on GPUs were needed to progress on computer vision. In 2011, this approach achieved for the first time superhuman performance in a visual pattern recognition contest. Also in 2011, it won the ICDAR Chinese handwriting contest, and in May 2012, it won the ISBI image contest. Until 2011, CNNs did not play a major role at computer vision conferences, but in June 2012, a paper by Ciresan et al. at the leading conference CVPR showed how max-pooling CNNs on GPU can dramatically improve many vision benchmark records.

In October 2012, a similar system by Krizhevsky et al. won the large-scale ImageNet competition by a significant margin over shallow machine learning methods. In November 2012, Ciresan et al.'s system also won the ICPR contest on analysis of large medical images for cancer detection, and in the following year also the MICCAI Grand Challenge on the same topic. In 2013 and 2014, the error rate on the ImageNet task using deep learning was further reduced, following a similar trend in large-scale speech recognition.

Image classification was then extended to the more challenging task of generating descriptions (captions) for images, often as a combination of CNNs and LSTMs.

Some researchers state that the October 2012 ImageNet victory anchored the start of a "deep learning revolution" that has transformed the AI industry.

In March 2019, Yoshua Bengio, Geoffrey Hinton and Yann LeCun were awarded the Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

**5. MACHINE LEARNING:**

It seems like your request is quite brief. "ML" typically stands for "Machine Learning." Machine learning is a subset of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform a specific task without being explicitly programmed for that task. If you have a specific question or topic related to machine learning that you'd like more information about, please provide more details, and I'll do my best to assist you!

Top of Form

**6. PROPOSED SYSTEM:**

The proposed system integrates cataclysmic convolutional deep learning models, leveraging Tensor Flow’s robust framework, for the precise identification and classification of natural disasters from various data sources. By harnessing the power of convolutional neural networks (CNNs) with catastrophic event detection mechanisms, the system can swiftly analyze vast amounts of data, including imagery classifications to detect events such as earthquakes, hurricanes, floods, and wildfires. Utilizing Django as the web framework provides a seamless interface for users to interact with the system, allowing for real-time monitoring, analysis, and visualization of disaster occurrences. Through an intuitive user interface, emergency responders and the general public can access timely and accurate information about ongoing disasters, enabling swift and effective response efforts. The system's modular architecture facilitates scalability and adaptability, ensuring its efficacy in diverse geographic and environmental contexts. Additionally, by incorporating Deep learning models trained on historical data, the system can provide predictive insights, aiding in disaster preparedness and mitigation strategies. Overall, the proposed system stands as a robust solution for natural disaster identification and management, integrating cutting-edge technology with user-friendly design for maximum impact and effectiveness.

**MERITS:**

* The proposed system utilizes convolutional deep learning models to precisely identify and classify various natural disasters from diverse data sources. This approach enables more accurate and rapid detection of events such as earthquakes, floods, and wildfires compared to traditional methods.
* Incorporating Django as the web framework provides a user-friendly interface for emergency responders and the general public. This allows for seamless interaction with the system, facilitating real-time monitoring, analysis, and visualization of disaster situations.
* The modular architecture of the proposed system ensures scalability and adaptability, making it effective in diverse geographic and environmental contexts. This flexibility allows for the system to be customized and expanded based on evolving needs and technological advancements.

**7. PREPARING THE DATASET:**

This dataset contains approximately train and image records of features extracted, which were then classified.

**8. LITERATURE SURVEY**

**General**

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. Loan default trends have been long studied from a socio-economic stand point.

Most economics surveys believe in empirical modeling of these complex systems in order to be able to predict the loan default rate for a particular individual. The use of machine learning for such tasks is a trend which it is observing now. Some of the survey’s to understand the past and present perspective of loan approval or not.

**Review of Literature Survey**

**Title** : A Large-Scale Virtual Dataset and Egocentric Localization for Disaster Responses.

**Author**: Hae-Gon Jeon , Sunghoon Im.

**Year** : 2023

With the increasing social demands of disaster response, methods of visual observation for rescue and safety have become increasingly important. However, because of the shortage of datasets for disaster scenarios, there has been little progress in computer vision and robotics in this field. With this in mind, we present the first large-scale synthetic dataset of egocentric viewpoints for disaster scenarios. We simulate pre- and post-disaster cases with drastic changes in appearance, such as buildings on fire and earthquakes. The dataset consists of more than 300K high-resolution stereo image pairs, all annotated with ground-truth data for the semantic label, depth in metric scale, optical flow with sub-pixel precision, and surface normal as well as their corresponding camera poses. To create realistic disaster scenes, we manually augment the effects with 3D models using physically-based graphics tools. We train various state-of-the-art methods to perform computer vision tasks using our dataset, evaluate how well these methods recognize the disaster situations, and produce reliable results of virtual scenes as well as real-world images. We also present a convolutional neural network-based egocentric localization method that is robust to drastic appearance changes, such as the texture changes in a fire, and layout changes from a collapse. To address these key challenges, we propose a new model that learns a shape-based representation by training on stylized images, and incorporate the dominant planes of query images as approximate scene coordinates. We evaluate the proposed method using various scenes including a simulated disaster dataset to demonstrate the effectiveness of our method when confronted with significant changes in scene layout. Experimental results show that our method provides reliable camera pose predictions despite vastly changed conditions

**Title**: Natural Disaster Discernment and Vigilance.

**Author** : N V Ganapathi Raju , Ajay Therala , and Venkat Yalla , Rohit Raju Ch , K Rajiv.

**Year**: 2021

Natural Disasters like cyclones and Earthquakes have a huge impact on the lives of people, results in damage to infrastructure, and lead to injuries and deaths. IoT Based detection systems are utilized for detecting disasters and performing subsequent rescue operations. The challenge with these IoT Based systems is that collecting data from sensors might be failed due to communication breakages or network congestions. To address this issue, this paper has come up with an idea of implementing Disaster Detection using Convolutional Neural Networks and sending SMS to people for making people alert. This paper aims to particularly detect Cyclones and Earthquakes. Data sets were collected from Kaggle. Convolutional Neural Network is a deep learning algorithm that takes an image as input, assigns weights/biases to a variety of aspects in the image for differentiating one from another image. Applications of this work includes disaster preparedness such as forecasts, warnings and predictions, disaster management and disaster relief operations. A comparative study has been performed on CNN and its variants.

**Title** : Disaster management using deep learning on social media.

**Author** : Bhavana, Parvathi Ramasubramanian.

**Year: 2021**

The main goal of this proposed work is to provide solutions for disaster management using deep learning algorithms on social media images. The MNIST dataset was used to initially build the deep learning models. The images were trained using LeNet5, VGG13, VGG 16 and LSTM deep learning models. Later a dataset containing 3460 images were taken from social media. The labels earthquake, wildfire and floods were used to achieve classification results. The images were trained and validated using LSTM, VGG13 and VGG16. The performance of the algorithms is compared and the disaster response technique is generated based on the image classification and disaster management strategies are provided based on classification.

**Title** : Incidents1M: A Large-Scale Dataset of Images With Natural Disasters, Damage, and Incidents.

**Author**: Ethan Weber , Dim P. Papadopoulos, Agata Lapedriza , Ferda Ofli.

**Year** : 2023

Natural disasters, such as floods, tornadoes, or wildfires, are increasingly pervasive as the Earth undergoes global warming. It is difficult to predict when and where an incident will occur, so timely emergency response is critical to saving the lives of those endangered by destructive events. Fortunately, technology can play a role in these situations. Social media posts can be used as a low-latency data source to understand the progression and aftermath of a disaster, yet parsing this data is tedious without automated methods. Prior work has mostly focused on text-based filtering, yet image and video-based filtering remains largely unexplored. In this work, we present the Incidents1M Dataset, a large-scale multi-label dataset which contains 977,088 images, with 43 incident and 49 place categories. We provide details of the dataset construction, statistics and potential biases; introduce and train a model for incident detection; and perform image-filtering experiments on millions of images on Flickr and Twitter. We also present some applications on incident analysis to encourage and enable future work in computer vision for humanitarian aid. Code, data, and models are available at http://incidentsdataset.csail.mit.edu.

**Title** : Natural Disasters Intensity Analysis and Classification Based on Multispectral Images Using Multi-Layered Deep Convolutional Neural Network.

**Author:** Muhammad Aamir , Tariq Ali , Muhammad Irfan , Ahmad Shaf , Muhammad Zeeshan Azam.

**Year** : 2021

Natural disasters not only disturb the human ecological system but also destroy the properties and critical infrastructures of human societies and even lead to permanent change in the ecosystem. Disaster can be caused by naturally occurring events such as earthquakes, cyclones, floods, and wildfires. Many deep learning techniques have been applied by various researchers to detect and classify natural disasters to overcome losses in ecosystems, but detection of natural disasters still faces issues due to the complex and imbalanced structures of images. To tackle this problem, we propose a multilayered deep convolutional neural network. The proposed model works in two blocks: Block-I convolutional neural network (B-I CNN), for detection and occurrence of disasters, and Block-II convolutional neural network (B-II CNN), for classification of natural disaster intensity types with different filters and parameters. The model is tested on 4428 natural images and performance is calculated and expressed as different statistical values: sensitivity (SE), 97.54%; specificity (SP), 98.22%; accuracy rate (AR), 99.92%; precision (PRE), 97.79%; and F1-score (F1), 97.97%. The overall accuracy for the whole model is 99.92%, which is competitive and comparable with state-of-the-art algorithms.

**STUDY**

**AIM:**

The aim of using Convolutional Neural Networks (CNNs) for natural disaster identification is to enhance the accuracy and efficiency of detecting and classifying various natural disasters through deep learning techniques. By leveraging CNNs, the goal is to analyze and interpret satellite imagery or real-time data to provide timely and precise information on disaster events. This approach aims to improve disaster response and management strategies by enabling quicker and more reliable identification of hazards.

**OBJECTIVE:**

1. Develop and train Convolutional Neural Networks (CNNs) to accurately identify and classify natural disaster events from satellite images and remote sensing data. Enhance model performance by incorporating advanced deep learning techniques and optimizing network architectures for improved detection sensitivity and specificity. Implement a real-time monitoring system to provide timely alerts and actionable insights for disaster management and response teams.

**Scope:**

The scope of the project titled "Convolutional Neural Network Deep Learning for Natural Disaster Identifications" encompasses the development and deployment of advanced machine learning models to accurately identify and classify natural disasters from various data sources, such as satellite imagery, aerial photographs, and remote sensing data. By leveraging Convolutional Neural Networks (CNNs), the project aims to enhance the precision and efficiency of natural disaster detection, including hurricanes, earthquakes, floods, wildfires, and landslides. The scope includes the collection and preprocessing of diverse datasets, the design and training of deep learning models, and the integration of these models into a real-time monitoring system. Additionally, the project involves validating the models' performance, optimizing their accuracy, and developing user-friendly interfaces for disaster response teams and decision-makers. Ultimately, the goal is to provide timely and reliable information to mitigate the impact of natural disasters and support effective disaster management and response strategies.

**OUTLINE OF THE PROJECT**

**Overview of the system:**

* Define a problem
* Gathering image data set
* Evaluating algorithms
* Detecting results

The steps involved in Building the data model is depicted below.

**Data collection** (Splitting Training set & Test) set)

**Pre Processing** (Sequential)

**Building classification Model**

**Detection** (**Natural Disaster Identifications)**

Fig: data flow diagram for CNN model

**PROJECT REQUIREMENTS**

**General:**

Requirements are the basic constrains that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Functional requirements

2. Non-Functional requirements

3. Environment requirements

A. Hardware requirements

B. software requirements

**Functional requirements:**

The software requirements specification is a technical specification of requirements for the software product. It is the first step in the requirements analysis process. It lists requirements of a particular software system. The following details to follow the special libraries like tensorflow, keras, matplotlib.

**Non-Functional Requirements:**

Process of functional steps,

1. Problem define
2. Preparing data
3. Evaluating algorithm
4. Improving results
5. Prediction the result

**Environment Requirements:**

**Framework :** Keras.

**Software Requirements:**

* Operating System : Windows / Linux
* Simulation Tool : Anaconda with Jupyter Notebook
* Language : Python

**Hardware requirements:**

* Processor : Intel i3
* Hard disk : minimum 400 GB
* RAM : minimum 4 GB

12. **FEASIBILITY STUDY**

**Splitting the dataset:**

The data use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. It has the test dataset (or subset) in order to test our models and it will do this using Tensor flow library in Python using the Keras method.

**Construction of a Detecting Model:**

## Deep learning needs data gathering have lot of past image data’s. Training and testing this model working and predicting correctly.

Data Gathering

CNN Algorithm

Train model

Test model

Prediction

Steps of dataflow diagram

**Data Flow Diagram:**

SKIN CANCER images

Test dataset

Preprocessing

**Natural Disaster Identifications**

CNN Architecture

Training dataset

Fig: Process of dataflow 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.”

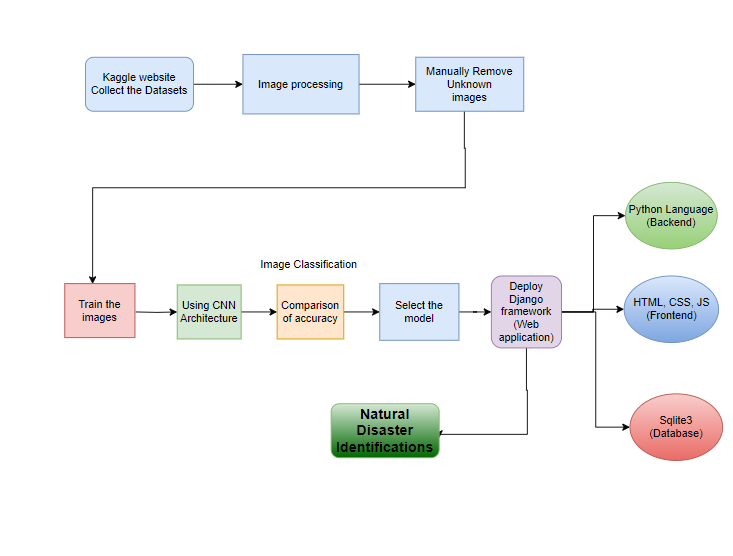
DFD levels and layers A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to 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 & developers

**13. DESIGN ARCHITECTURE**

**General**

Design is meaningful engineering representation of something that is to be built. Software design is a process design is the perfect way to accurately translate requirements in to a finished software product. Design creates a representation or model, provides detail about software data structure, architecture, interfaces and components that are necessary to implement a system.

**ARCHITECTURE DIAGRAM:**



**14. Work flow diagram:**

Data Collection

Pre-processing

Training Dataset

Testing Dataset

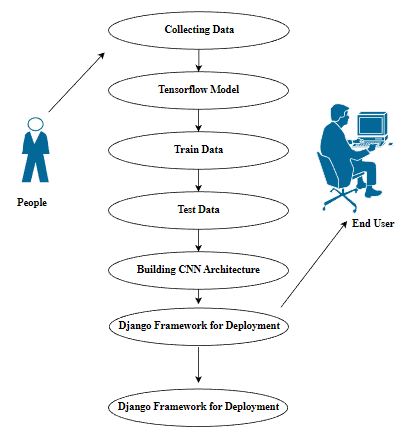
Architecture

Model

Accuracy

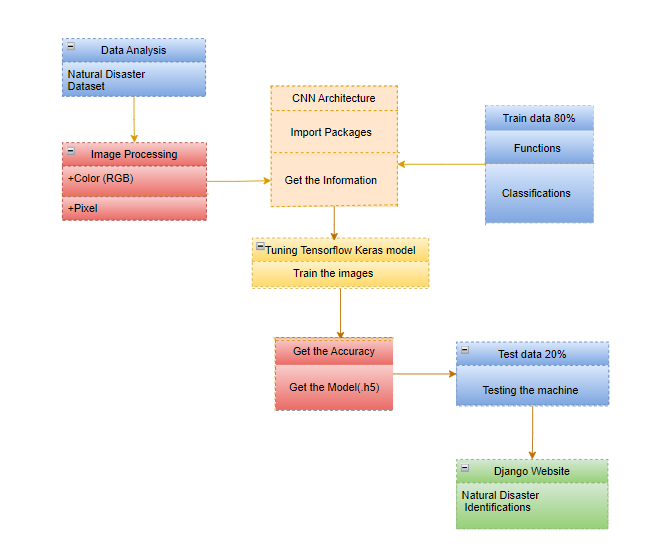
Workflow Diagram

**15. USECASE DIAGRAM:**



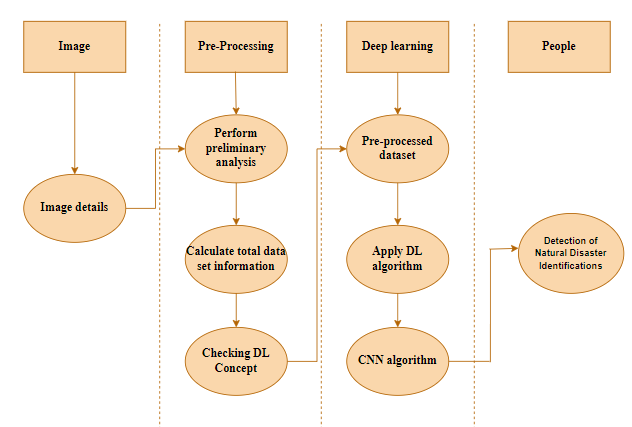
Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

**16. CLASS DIAGRAM:**



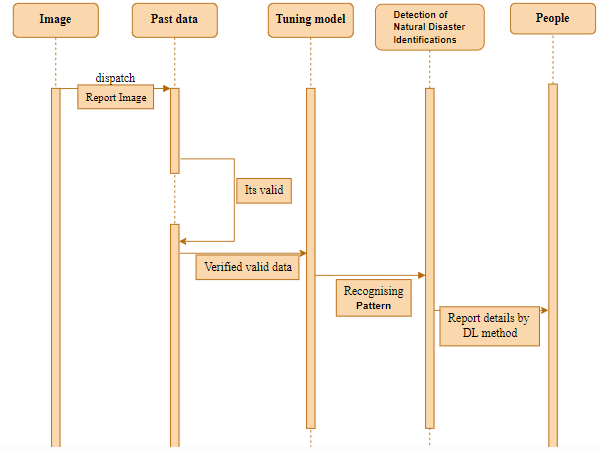
Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified for each class minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

17. **ACTIVITY DIAGRAM:**

****

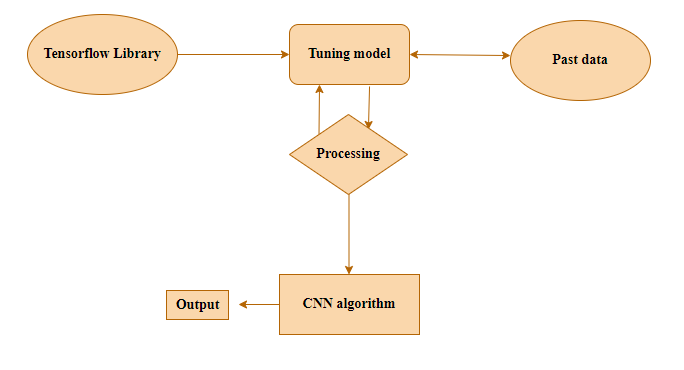
Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

**18. SEQUENCE DIAGRAM:**



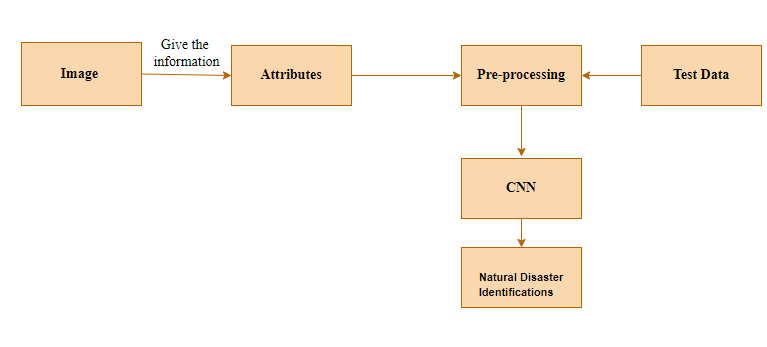
Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behaviour within your system. Other dynamic modelling techniques include [activity diagramming](http://agilemodeling.com/artifacts/activityDiagram.htm), [communication diagramming](http://agilemodeling.com/artifacts/communicationDiagram.htm), [timing diagramming](http://agilemodeling.com/artifacts/timingDiagram.htm), and [interaction overview diagramming](http://agilemodeling.com/artifacts/interactionOverviewDiagram.htm). Sequence diagrams, along with [class diagrams](http://agilemodeling.com/artifacts/classDiagram.htm) and [physical data models](http://agiledata.org/essays/dataModeling101.html) are in my opinion the most important design-level models for modern business application development.

**19. ER DIAGRAM:**

****

An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts or events within that system. An ERD is a data modeling technique that can help define business processes and be used as the foundation for a relational database. Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a referral point, should any debugging or business process re-engineering be needed later.

**20. COLLABORATION DIAGRAM:**

****

A collaboration diagram show the objects and relationships involved in an interaction, and the sequence of messages exchanged among the objects during the interaction.

The collaboration diagram can be a decomposition of a class, class diagram, or part of a class diagram.it can be the decomposition of a use case, use case diagram, or part of a use case diagram.

The collaboration diagram shows messages being sent between classes and object (instances). A diagram is created for each system operation that relates to the current development cycle (iteration).

**SOFTWARE DESCRIPTION**

Our software utilizes Convolutional Neural Networks (CNNs) to enhance natural disaster identification and management through deep learning. Leveraging advanced CNN architectures, the software is designed to analyze and interpret various types of data, including satellite imagery, meteorological data, and environmental sensors. By training on extensive datasets of natural disaster events, the CNN models are capable of accurately detecting and classifying disasters such as hurricanes, earthquakes, floods, and wildfires. The system offers real-time analysis and predictive capabilities, providing valuable insights and alerts to emergency responders, government agencies, and researchers. Its user-friendly interface allows for seamless integration with existing disaster management systems, facilitating timely and informed decision-making to mitigate the impact of natural disasters.

**ANACONDA NAVIGATOR:**

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.

Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

Anaconda is a 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.

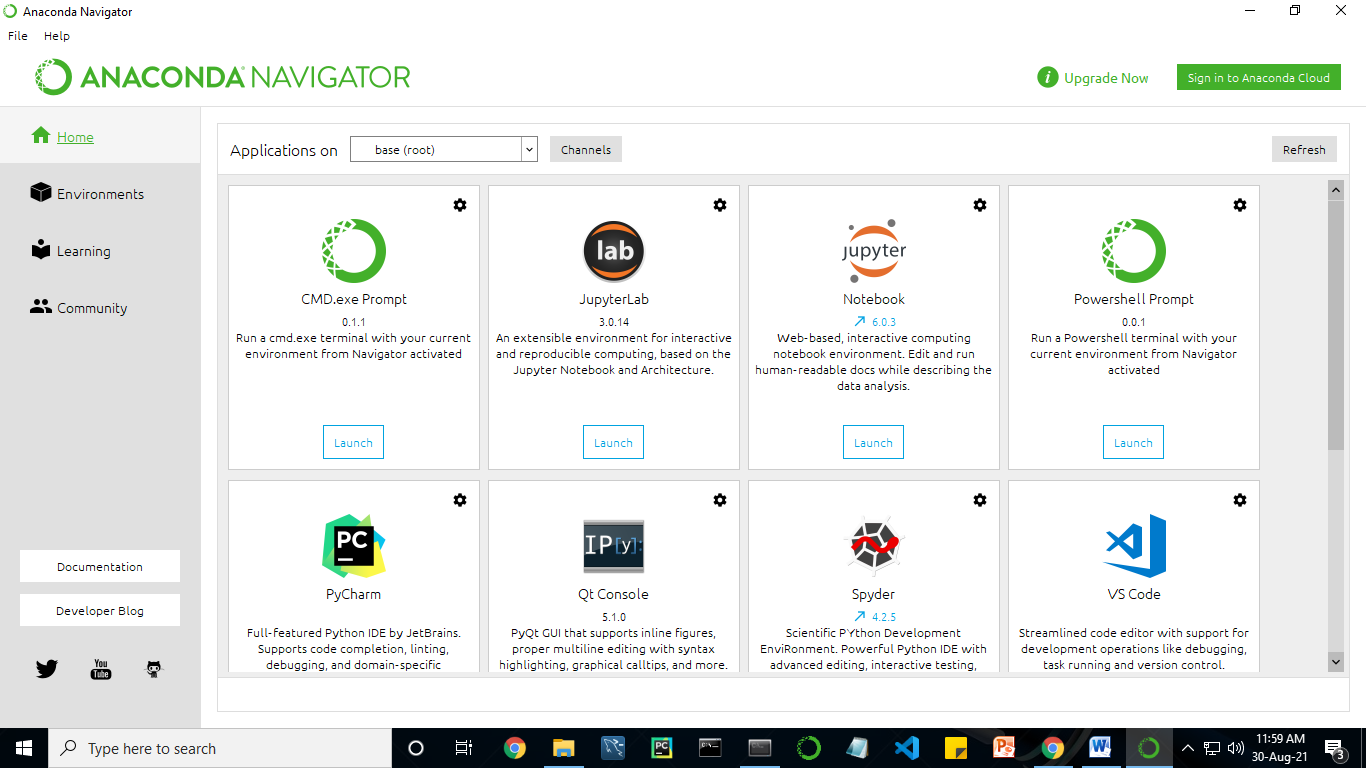
In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

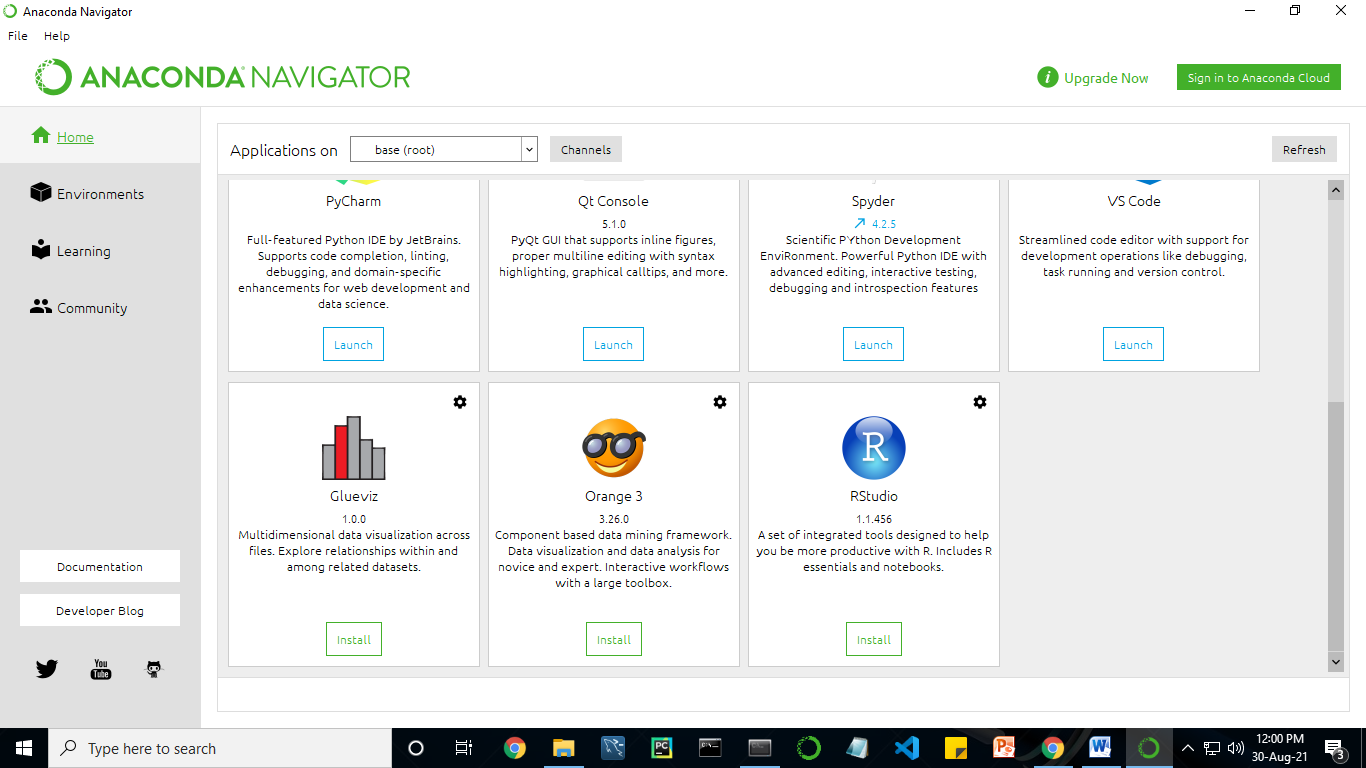
The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

The following applications are available by default in Navigator:

* [JupyterLab](https://jupyterlab.readthedocs.io/en/stable/)
* [Jupyter Notebook](https://jupyter.readthedocs.io/en/latest/)
* [Spyder](https://www.spyder-ide.org/)
* [VSCode](https://code.visualstudio.com/docs)
* [Glueviz](http://glueviz.org/en/stable/)
* [Orange 3 App](http://orange.biolab.si/docs/)
* [RStudio](http://docs.rstudio.com/)
* Anaconda Prompt (Windows only)
* Anaconda PowerShell (Windows only)





**PYTHON:**

**Introduction:**

**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [high-level](https://en.wikipedia.org/wiki/High-level_programming_language) [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). Its design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with its use of [significant indentation](https://en.wikipedia.org/wiki/Off-side_rule). Its [language constructs](https://en.wikipedia.org/wiki/Language_construct) as well as its [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aim to help [programmers](https://en.wikipedia.org/wiki/Programmers) write clear, logical code for small and large-scale projects.

Python is [dynamically-typed](https://en.wikipedia.org/wiki/Type_system#DYNAMIC) and [garbage-collected](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigms), including [structured](https://en.wikipedia.org/wiki/Structured_programming) (particularly, [procedural](https://en.wikipedia.org/wiki/Procedural_programming)), object-oriented and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). It is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

**METHODOLOGY**

In the methodology for using Convolutional Neural Networks (CNNs) in deep learning for natural disaster identification, a systematic approach is employed to achieve accurate classification and detection. The process begins with collecting and preprocessing diverse datasets, such as satellite imagery or remote sensing data, which may include various types of natural disasters like floods, hurricanes, or wildfires. These images are then labeled and segmented to create a robust training dataset. A CNN model is designed and trained using these labeled datasets, leveraging layers such as convolutional layers, pooling layers, and fully connected layers to extract and learn relevant features. Data augmentation techniques are often applied to enhance model robustness and generalization. The trained CNN model is evaluated using a separate validation set to fine-tune hyper parameters and improve accuracy. Finally, the model is tested on unseen data to assess its performance and reliability in real-world disaster detection scenarios, ensuring it can effectively identify and classify natural disasters with high precision.

CNN Weights

Natural Disaster Classification

CNN train

Raw image

Build a sequential model

The train dataset is used to train the model (CNN) so that it can identify the test image and the disease it has CNN has different layers that are Dense, Dropout, Activation, Flatten, Convolution2D, and MaxPooling2D. After the model is trained successfully, the software can identify natural disaster image contained in the dataset. After successful training and preprocessing, comparison of the test image and trained model takes place to predict.

**ARCHITECTURE OF CNN**

**CONVOLUTIONAL NEURAL NETWORK:**

A Convolutional neural network (CNN) is one type of Artificial Neural Network. A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, and also for other auto correlated data.

**Models API:**

**There are three ways to create Keras models:**

• The Sequential model, which is very straightforward (a simple list of layers), but is limited to single-input, single-output stacks of layers (as the name gives away).

• The Functional API, which is an easy-to-use, fully-featured API that supports arbitrary model architectures. For most people and most use cases, this is what you should be using. This is the Keras "industry strength" model.

• Model subclassing, where you implement everything from scratch on your own. Use this if you have complex, out-of-the-box research use cases.

**Types of Keras Models**

**Models in keras are available in two types:**

• Keras Sequential Model

• Keras Functional API

**1. Sequential Model in Keras**

It allows us to create models layer by layer in sequential order. But it does not allow us to create models that have multiple inputs or outputs.

It is best for simple stack of layers which have 1 input tensor and 1 output tensor.

This model is not suited when any of the layer in the stack has multiple inputs or outputs. Even if we want non-linear topology, it is not suited.

**2. Functional API in Keras**

It provides more flexibility to define a model and add layers in keras. Functional API allows us to create models that have multiple input or output.

It also allows us to share these layers. In other words. we can make graphs of layers using Keras functional API.

As functional API is a data structure, it is easy to save it as a single file that helps in recreating the exact model without having the original code. Also its easy to model the graph here and access its nodes as well.

**Model Subclassing in Keras**

Sequential model does not allow you much flexibility to create your models. Functional API also only has a little of customization available for you. But you may create your own fully-customizable models in Keras. This is done by subclassing the Model class and implementing a call method.

Input() is used to instantiate a Keras tensor.

A Keras tensor is a symbolic tensor-like object, which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model.

For instance, if a, b and c are Keras tensors, it becomes possible to do: model = Model(input=[a, b], output=c)

# **kernels:**

Each convolutional layer contains a series of filters known as convolutional kernels. The filter is a matrix of integers that are used on a subset of the input pixel values, the same size as the kernel. Each pixel is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid cell, like a pixel, in the output channel/feature map. These are linear transformations, each convolution is a type of affine function.

In computer vision the input is often a 3 channel RGB image. For simplicity, if we take a greyscale image that has one channel (a two dimensional matrix) and a 3x3 convolutional kernel (a two dimensional matrix). The kernel strides over the input matrix of numbers moving horizontally column by column, sliding/scanning over the first rows in the matrix containing the images pixel values. Then the kernel strides down vertically to subsequent rows. Note, the filter may stride over one or several pixels at a time, this is detailed further below.

In other non-vision applications, a one dimensional convolution may slide vertically over an input matrix.

**Padding:**

To handle the edge pixels there are several approaches:

* Losing the edge pixels
* Padding with zero value pixels
* Reflection padding

Reflection padding is by far the best approach, where the number of pixels needed for the convolutional kernel to process the edge pixels are added onto the outside copying the pixels from the edge of the image. For a 3x3 kernel, one pixel needs to be added around the outside, for a 7x7 kernel then three pixels would be reflected around the outside. The pixels added around each side is the dimension, halved and rounded down.

Traditionally in many research papers, the edge pixels are just ignored, which loses a small proportion of the data and this gets increasing worse if there are many deep convolutional layers. For this reason, I could not find existing diagrams to easily convey some of the points here without being misleading and confusing stride 1 convolutions with stride 2 convolutions.

With padding, the output from a input of width w and height h would be width w and height h (the same as the input with a single input channel), assuming the kernel takes a stride of one pixel at a time.

# **Strides:**

It is common to use a stride two convolution rather than a stride one convolution, where the convolutional kernel strides over 2 pixels at a time, for example our 3x3 kernel would start at position (1,1), then stride to (1,3), then to 1, 5) and so on, halving the size of the output channel/feature map, compared to the convolutional kernel taking strides of one.

With padding, the output from an input of width w, height h and depth 3 would be the ceiling of width w/2, height h/2 and depth 1, as the kernel outputs a single summed output from each stride.

# **Rectified Linear Unit (ReLU):**

A Rectified Linear Unit is used as a non-linear activation function. A ReLU says if the value is less than zero, round it up to zero.

# **Normalisation:**

Normalisation is the process of subtracting the mean and dividing by the standard deviation. It transforms the range of the data to be between -1 and 1 making the data use the same scale, sometimes called Min-Max scaling.

It is common to normalize the input features, standardising the data by removing the mean and scaling to unit variance. It is often important the input features are centred around zero and have variance in the same order. With some data, such as images the data is scaled so that it’s range is between 0 and 1, most simply dividing the pixel values by 255.

**Batch normalisation:**

Batch normalisation has the benefits of helping to make a network output more stable predictions, reduce overfitting through regularisation and speeds up training by an order of magnitude.

Batch normalisation is the process of carrying normalisation within the scope activation layer of the current batch, subtracting the mean of the batch’s activations and dividing by the standard deviation of the batches activations.

This is necessary as even after normalizing the input as some activations can be higher, which can cause the subsequent layers to act abnormally and makes the network less stable.

Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

Importantly, batch normalization works differently during training and during inference.

**During training** (i.e. when using fit() or when calling the layer/model with the argument training=True), the layer normalizes its output using the mean and standard deviation of the current batch of inputs. That is to say, for each channel being normalized, the layer returns gamma \* (batch - mean(batch)) / sqrt(var(batch) + epsilon) + beta, where:

* epsilon is small constant (configurable as part of the constructor arguments)
* gamma is a learned scaling factor (initialized as 1), which can be disabled by passing scale=False to the constructor.
* beta is a learned offset factor (initialized as 0), which can be disabled by passing center=False to the constructor.

**During inference** (i.e. when using evaluate() or predict() or when calling the layer/model with the argument training=False (which is the default), the layer normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training. That is to say, it returns gamma \* (batch - self.moving\_mean) / sqrt(self.moving\_var + epsilon) + beta.

self.moving\_mean and self.moving\_var are non-trainable variables that are updated each time the layer in called in training mode, as such:

* moving\_mean = moving\_mean \* momentum + mean(batch) \* (1 - momentum)
* moving\_var = moving\_var \* momentum + var(batch) \* (1 - momentum)

As such, the layer will only normalize its inputs during inference after having been trained on data that has similar statistics as the inference data.

**Arguments**

* **axis**: Integer, the axis that should be normalized (typically the features axis). For instance, after a Conv2D layer with data\_format="channels\_first", set axis=1 in BatchNormalization.
* **momentum**: Momentum for the moving average.
* **epsilon**: Small float added to variance to avoid dividing by zero.
* **center**: If True, add offset of beta to normalized tensor. If False, beta is ignored.
* **scale**: If True, multiply by gamma. If False, gamma is not used. When the next layer is linear (also e.g. nn.relu), this can be disabled since the scaling will be done by the next layer.
* **beta\_initializer**: Initializer for the beta weight.
* **gamma\_initializer**: Initializer for the gamma weight.
* **moving\_mean\_initializer**: Initializer for the moving mean.
* **moving\_variance\_initializer**: Initializer for the moving variance.
* **beta\_regularizer**: Optional regularizer for the beta weight.
* **gamma\_regularizer**: Optional regularizer for the gamma weight.
* **beta\_constraint**: Optional constraint for the beta weight.
* **gamma\_constraint**: Optional constraint for the gamma weight.

**Call arguments**

* **inputs**: Input tensor (of any rank).
* **training**: Python boolean indicating whether the layer should behave in training mode or in inference mode.

**TYPES OF CNN:**

* SUFFLENET
* RESNET

**LIST OF MODULES**

* + - Manual Architecture
    - SUFFLENET Architecture
    - RESNET Architecture
    - Deployment

**MODULE DESCRIPTION**

**IMPORT THE GIVEN IMAGE FROM DATASET:**

## We have to import our data set using keras preprocessing image data generator function also we create size, rescale, range, zoom range, horizontal flip. Then we import our image dataset from folder through the data generator function. Here we set train, test, and validation also we set target size, batch size and class-mode from this function we have to train using our own created network by adding layers of CNN.

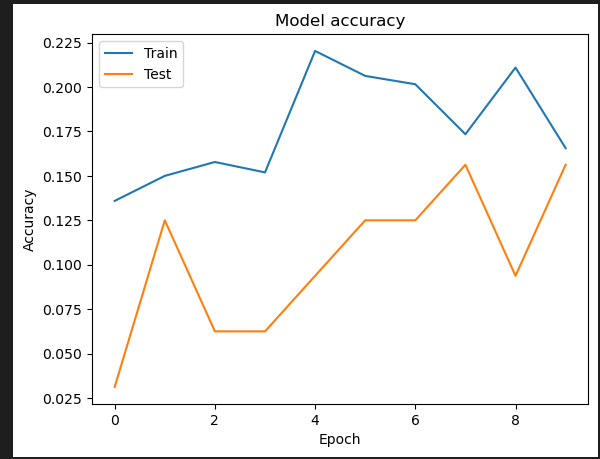
**TO TRAIN THE MODULE BY GIVEN 1. Data Analysis**

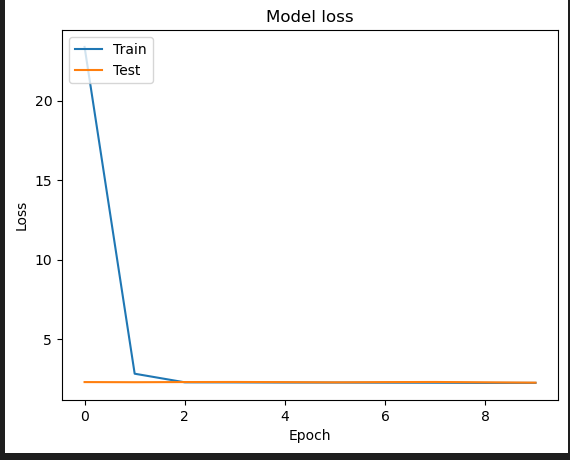
Data analysis is the process of cleaning, changing, and processing raw data, and extracting actionable, relevant information that helps businesses make informed decisions. The procedure helps reduce the risks inherent in decision-making by providing useful insights. The data analysis process, or alternately, data analysis steps, involves gathering all the information, processing it, exploring the data, and using it to find patterns and other insights.

In data analysis we analyse the data that how the image data is available. We analyse how many data are available and we check whether the normal data is available corresponding to the mask data.

**Manual Architecture :**

Skin cancer classification involves the categorization of skin lesions based on various visual and clinical features to aid in accurate diagnosis and treatment planning. In manual architecture, this process typically relies on the expertise of dermatologists who visually inspect and analyze skin lesions. Dermatologists employ their knowledge and experience to assess key characteristics such as asymmetry, border irregularity, color variation, diameter, and evolving changes in the lesion's appearance. The manual classification system often includes different types of skin cancer, such as melanoma, basal cell carcinoma, and squamous cell carcinoma, each exhibiting distinct visual cues. Dermatologists may also incorporate dermoscopy, a non-invasive technique using a handheld device with magnification and light, to enhance their examination. This meticulous manual examination, guided by established diagnostic criteria, plays a crucial role in early detection and effective management of skin cancer, highlighting the significance of human expertise in the classification process. However, the advent of computer-aided diagnostic tools and artificial intelligence has shown promise in augmenting these efforts by providing additional support through automated analysis





**SUFFLENET:**

Shuffle Net is a deep neural network architecture designed for efficient computation on resource-constrained devices, such as mobile phones and embedded systems. It was introduced to address the challenges of deploying large and computationally intensive models on devices with limited processing power and memory.

Key features of Shuffle Net include:

Channel Shuffle Operation:

One of the distinctive features of Shuffle Net is the channel shuffle operation, which enables communication between different groups of channels. This operation is crucial for achieving a balance between model complexity and computational efficiency.

Group Convolution:

ShuffleNet uses group convolution, where the input channels are divided into groups, and convolutional filters are applied separately to each group. This reduces the computational cost compared to standard convolutions.

Shuffle Unit:

The building block of ShuffleNet is the Shuffle Unit, which consists of grouped convolutions, channel shuffling, and pointwise convolutions. This unit facilitates efficient information exchange across channels while keeping the computational cost low.

Bottleneck Design:

Similar to other efficient neural network architectures, ShuffleNet employs bottleneck building blocks. These blocks consist of a combination of 1x1 pointwise convolutions, 3x3 depthwise convolutions, and another set of 1x1 pointwise convolutions, which helps reduce the number of parameters and computation.

Multiple ShuffleNet Versions:

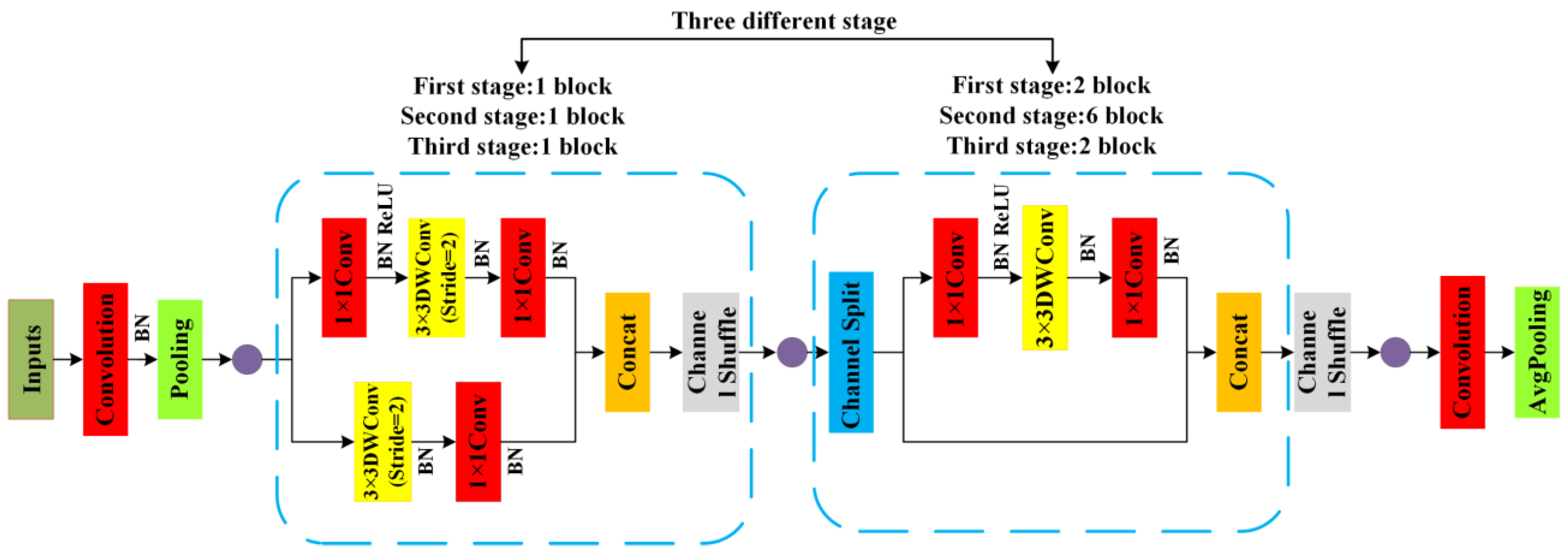
ShuffleNet has several versions, such as ShuffleNet, ShuffleNetV2, and ShuffleNetV2 with improved performance and efficiency. Each version introduces modifications to the original architecture to enhance its capabilities.

Applications:

Due to its efficiency, ShuffleNet is well-suited for deployment in scenarios where computational resources are limited, such as in mobile applications, edge devices, and other embedded systems.

The design principles of ShuffleNet aim to strike a balance between model accuracy and computational efficiency, making it a popular choice for real-time applications on devices with constrained resources. It has been widely adopted in computer vision tasks, including image classification, object detection, and semantic segmentation.

**ARCHITECTURE OF SUFFLENET:**



Architecture of SUFFLENET

**Convolutional layers:**

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

**Convolutional layers:**

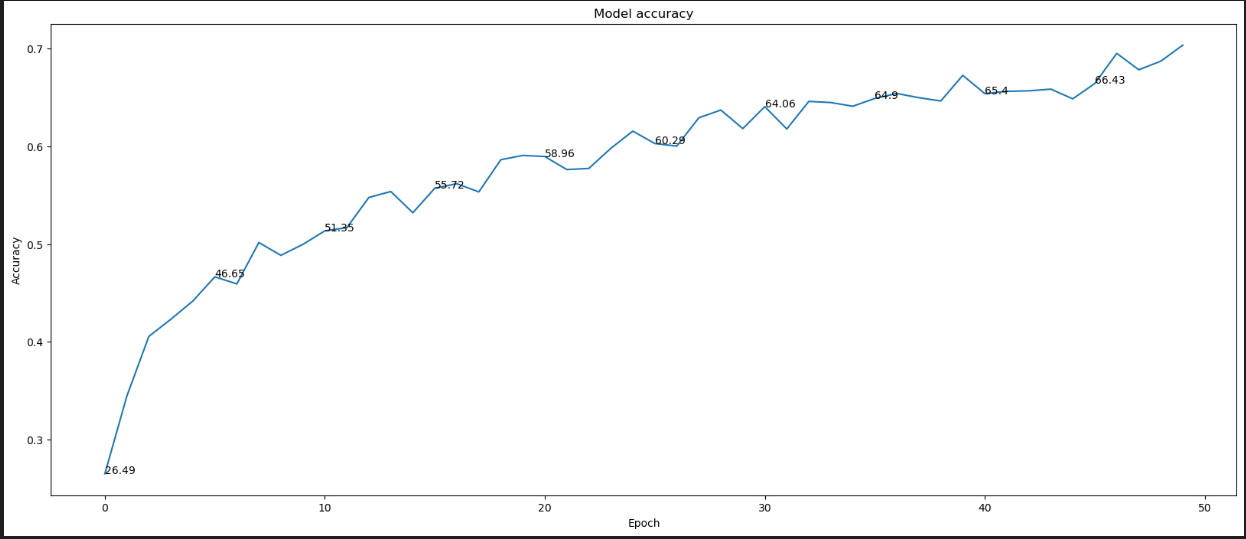
Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

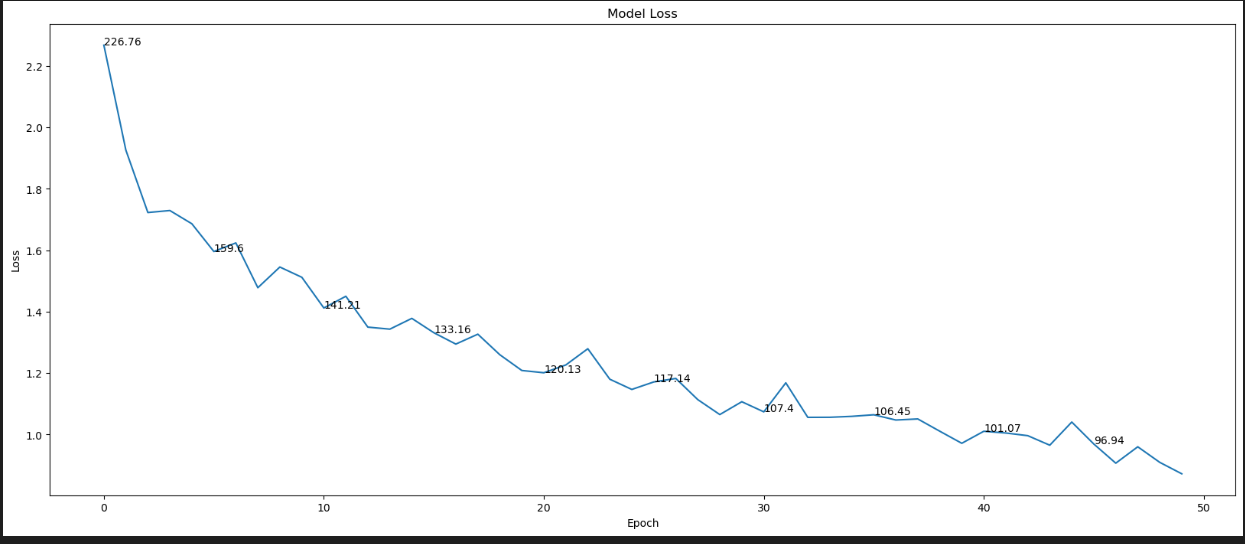
**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

****

****

**RESNET:**

ResNet, short for Residual Network, is a deep convolutional neural network architecture that was introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their 2015 paper titled "Deep Residual Learning for Image Recognition." ResNet is a groundbreaking architecture that has had a significant impact on the field of computer vision and deep learning. It was designed to address the problem of training very deep neural networks effectively by introducing a novel residual learning framework.

Here's a brief description of the key components and ideas behind the ResNet architecture:

Residual Block: The core innovation in ResNet is the residual block. Instead of trying to learn the desired underlying mapping (H(x)) directly, ResNet learns the residual mapping (F(x) = H(x) - x). The network is forced to learn the residual, which is easier, and then adds it back to the input, effectively "skipping" some layers. This helps mitigate the vanishing gradient problem and makes it easier to train very deep networks.

Skip Connections: Skip connections, also known as shortcut connections or identity mappings, are the fundamental architectural element in ResNet. These connections allow the network to skip one or more layers and pass the input directly to a deeper layer in the network. Skip connections enable the training of extremely deep networks while maintaining good performance.

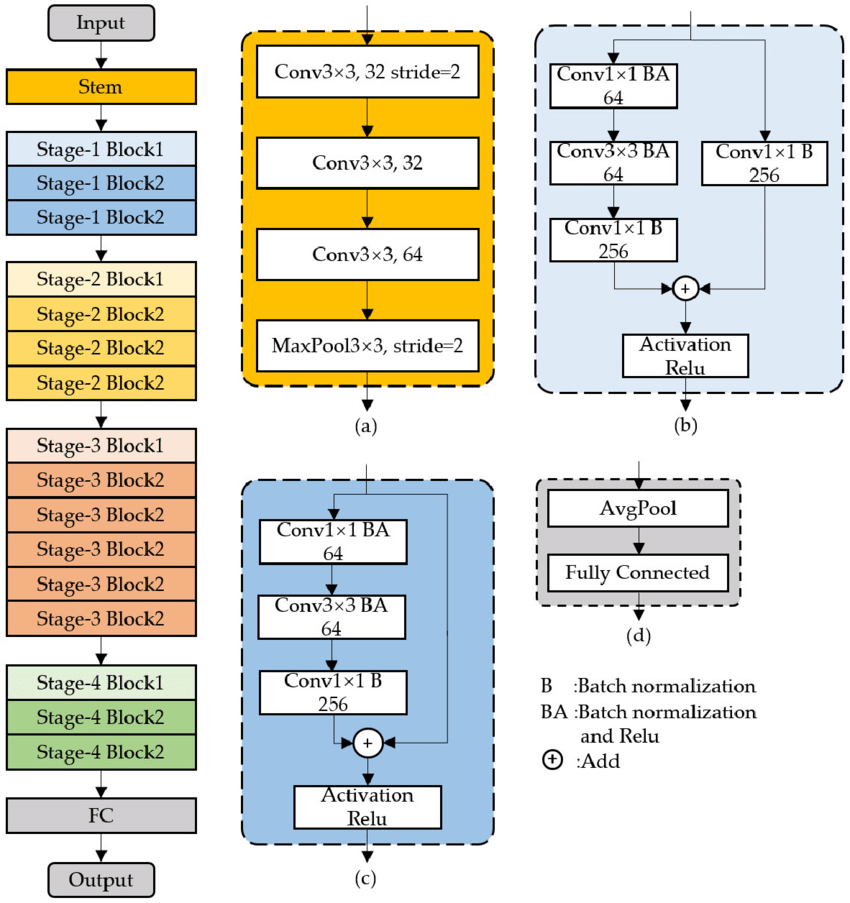
Network Depth: ResNets can be very deep, with hundreds of layers, thanks to the effectiveness of skip connections. Common versions of ResNet include ResNet-50, ResNet-101, ResNet-152, etc., which indicate the number of layers in the network.

Bottleneck Architecture: ResNet architectures often use a bottleneck design in the residual blocks to reduce the computational cost. In a bottleneck block, 1x1, 3x3, and 1x1 convolutions are used to transform the input, reducing the number of parameters and computational complexity.

Global Average Pooling: Instead of using fully connected layers at the end of the network, ResNet typically employs global average pooling, which computes the average of feature maps over spatial dimensions. This reduces overfitting and decreases the number of parameters.

ResNet has been widely adopted for various computer vision tasks, including image classification, object detection, and segmentation. The ability to train very deep networks with good performance has made ResNet a cornerstone of modern deep learning and a model architecture that serves as a basis for many subsequent innovations in neural network design.

**Architecture of ResNet:**



Architecture of ResNet

**Convolutional layers:**

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

**Convolutional layers:**

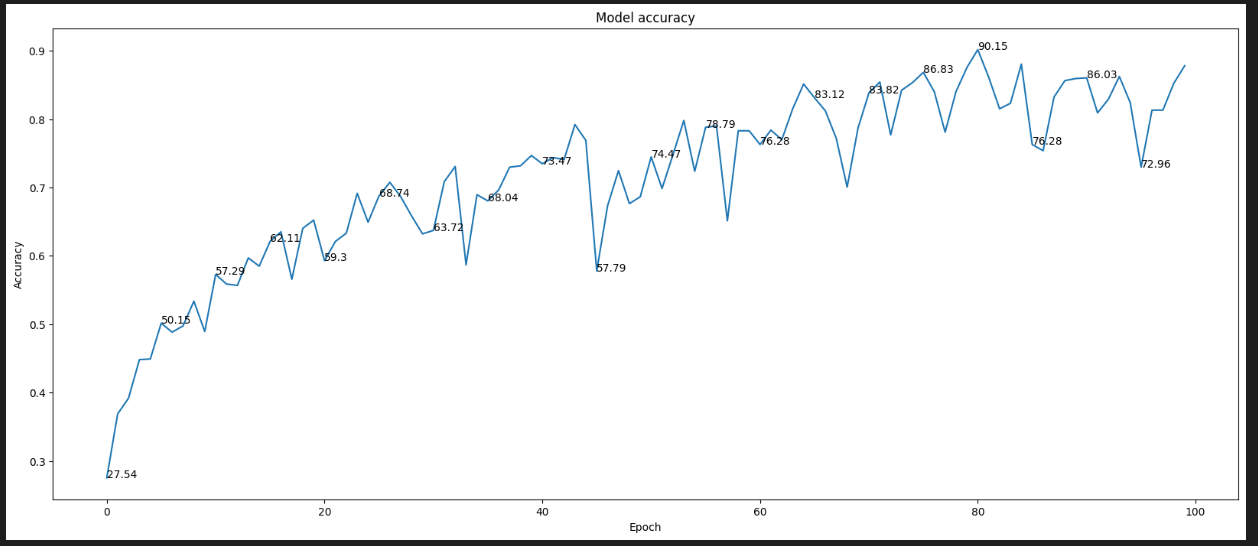
Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

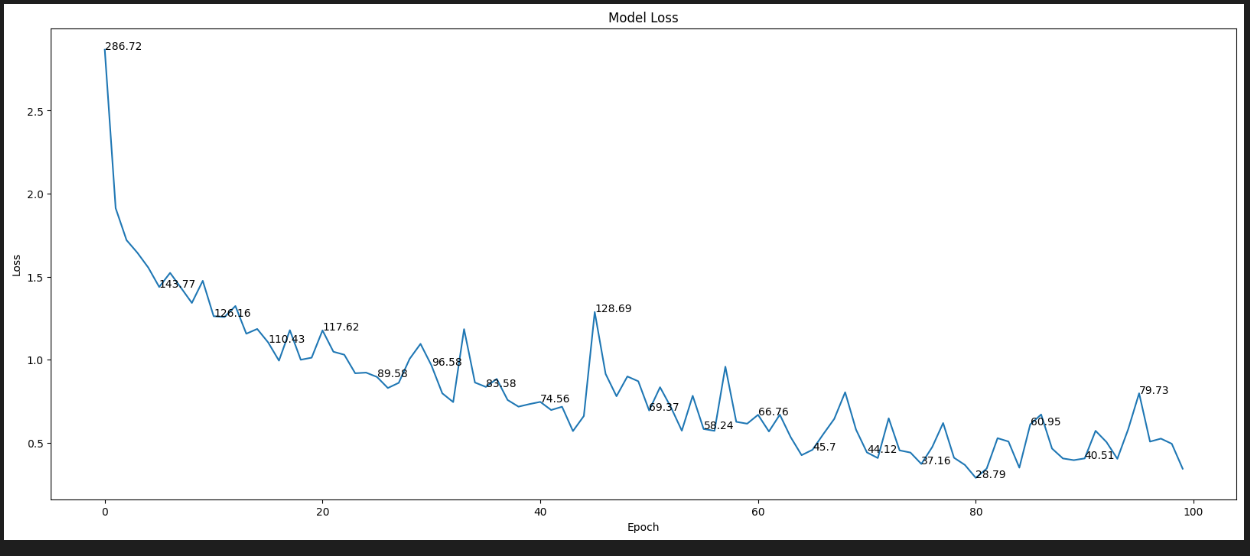
**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.





**DEPLOY**

**Deploying the model in Django Framework and predicting output**

In this module the trained deep learning model is converted into hierarchical data format file (.h5 file) which is then deployed in our django framework for providing better user interface and predicting the output whether the given material image is output

**Django (Web FrameWork) :**

Django is a micro web framework written in Python.

It is classified as a micro-framework because it does not require particular tools or libraries.

It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

However, Django supports extensions that can add application features as if they were implemented in Django itself.

Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Django was created by [Armin Ronacher](https://en.wikipedia.org/wiki/Armin_Ronacher) of Pocoo, an international group of Python enthusiasts formed in 2004. According to Ronacher, the idea was originally an [April Fool’s](https://en.wikipedia.org/wiki/April_Fool%27s) joke that was popular enough to make into a serious application. The name is a play on the earlier [Bottle](https://en.wikipedia.org/wiki/Bottle_(web_framework)) framework.

When Ronacher and Georg Brand created a bulletin board system written in Python, the Pocoo projects Werkzeug and [Jinja](https://en.wikipedia.org/wiki/Jinja_(template_engine)) were developed.

In April 2016, the Pocoo team was disbanded and development of Django and related libraries passed to the newly formed Pallets project.

Django has become popular among Python enthusiasts. As of October 2020, it has second most stars on [GitHub](https://en.wikipedia.org/wiki/GitHub) among Python web-development frameworks, only slightly behind Django, and was voted the most popular web framework in the Python Developers Survey 2018.

The micro-framework Django is part of the Pallets Projects, and based on several others of them.

Django **is** based on Werkzeug, [Jinja2](http://quintagroup.com/cms/python/jinja2) and inspired by Sinatra Ruby framework, available under BSD licence. It was developed at pocoo by Armin Ronacher. Although Django is rather young compared to most [Python](https://quintagroup.com/services/python) frameworks, it holds a great promise and has already gained popularity among Python web developers. Let’s take a closer look into Django, so-called “micro” framework for Python.

**FEATURES:**

Django was designed to be **easy to use and extend**.  The idea behind Django is to build a solid foundation for web applications of different complexity. From then on you are free to**plug in any extensions** you think you need. Also you are free to build your own modules. Django is great for all kinds of projects.  It's especially good for prototyping. Django depends on two external libraries: the Jinja2 template engine and the Werkzeug WSGI toolkit.

Still the question remains why use Django as your web application framework if we have immensely powerful [Django](https://quintagroup.com/services/python/django), [Pyramid,](https://quintagroup.com/cms/python/pyramid) and don’t forget web mega-framework [Turbo-gears](https://quintagroup.com/cms/python/turbogears)? Those are supreme[Python web frameworks](https://quintagroup.com/services/python/python-web-development.png) BUT out-of-the-box Django is pretty impressive too with it’s:

* Built-In Development server and Fast debugger
* integrated support for unit testing
* RESTful request dispatching
* Uses [Jinja2](https://quintagroup.com/cms/python/jinja2) Templating
* support for secure cookies
* Unicode based
* Extensive Documentation
* Google App Engine Compatibility
* Extensions available to enhance features desired

Plus Django gives you so much more **CONTROL** on the development stage of **your project**. It follows the principles of minimalism and let you decide how you will build your application.

* Django has a lightweight and modular design, so it easy to transform it to the web framework you need with a few extensions without weighing it down
* ORM-agnostic: you can plug in your favourite ORM e.g. [SQLAlchemy](https://quintagroup.com/cms/python/sqlalchemy).
* Basic foundation API is nicely shaped and coherent.
* Django documentation is comprehensive, full of examples and well structured. You can even try out some sample application to really get a feel of Django.
* It is super easy to deploy Django in production (Django is 100%WSGI 1.0 compliant”)
* HTTP request handling functionality
* High Flexibility

The configuration is even more flexible than that of Django, giving you plenty of solution for every production need.

To sum up, Django is one of the most polished and feature-rich micro frameworks, available. Still young, Django has a thriving community, first-class extensions, and an **elegant API**.  Django comes with all the benefits of fast templates, strong WSGI features, **thorough unit testability** at the web application and library level, **extensive documentation**. So next time you are starting a new project where you need some good features and a vast number of extensions, definitely check out Django.

Django is an API of Python that allows us to build up web-applications. It was developed by Armin Ronacher. Django's framework is more explicit than Django framework and is also easier to learn because it has less base code to implement a simple web-Application

Django is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

Overview of Python Django Framework Web apps are developed to generate content based on retrieved data that changes based on a user’s interaction with the site. The server is responsible for querying, retrieving, and updating data. This makes web applications to be slower and more complicated to deploy than static websites for simple applications.

Django is an excellent web development framework for REST API creation. It is built on top of Python which makes it powerful to use all the python features.

Django is used for the backend, but it makes use of a templating language called Jinja2 which is used to create HTML, XML or other markup formats that are returned to the user via an HTTP request.

Django is considered to be more popular because it provides many out of box features and reduces time to build complex applications. Django is a good start if you are getting into web development. Django is a simple, un-opinionated framework; it doesn't decide what your application should look like developers do.

Django is a web framework. This means Django provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, and a wiki or go as big as a web-based calendar application or a commercial website.

**Advantages of Django:**

* Higher compatibility with latest technologies.
* Technical experimentation.
* Easier to use for simple cases.
* Codebase size is relatively smaller.
* High scalability for simple applications.
* Easy to build a quick prototype.
* Routing URL is easy.
* Easy to develop and maintain applications.

Framework Django is a web framework from Python language. Django provides a library and a collection of codes that can be used to build websites, without the need to do everything from scratch. But Framework Django still doesn't use the Model View Controller (MVC) method.

Django-RESTful is an extension for Django that provides additional support for building REST APIs. You will never be disappointed with the time it takes to develop an API. Django-Restful is a lightweight abstraction that works with the existing ORM/libraries. Django-RESTful encourages best practices with minimal setup.

Django Restful is an extension for Django that adds support for building REST APIs in Python using Django as the back-end. It encourages best practices and is very easy to set up. Django restful is very easy to pick up if you're already familiar with Django.

Django is a web framework for Python, meaning that it provides functionality for building web applications, including managing HTTP requests and rendering templates and also we can add to this application to create our API.

**Start Using an API**

1. Most APIs require an API key. ...
2. The easiest way to start using an API is by finding an HTTP client online, like REST-Client, Postman, or Paw.
3. The next best way to pull data from an API is by building a URL from existing API documentation.

The Django object implements a WSGI application and acts as the central object. It is passed the name of the module or package of the application. Once it is created it will act as a central registry for the view functions, the URL rules, template configuration and much more.

The name of the package is used to resolve resources from inside the package or the folder the module is contained in depending on if the package parameter resolves to an actual python package (a folder with an \_\_init\_\_.py file inside) or a standard module (just a .py file).

For more information about resource loading, see [open resource()](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Flask.open_resource).

Usually you create a [Django](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Flask) instance in your main module or in the \_\_init\_\_.py file of your package.

**Parameters**

* **rule** ([str](https://docs.python.org/3/library/stdtypes.html#str)) – The URL rule string.
* **endpoint** (Optional[[str](https://docs.python.org/3/library/stdtypes.html#str)]) – The endpoint name to associate with the rule and view function. Used when routing and building URLs. Defaults to view\_func.\_\_name\_\_.
* **view\_func** (Optional[Callable]) – The view function to associate with the endpoint name.
* **provide\_automatic\_options** (Optional[bool]) – Add the OPTIONS method and respond to OPTIONS requests automatically.
* **options** (Any) – Extra options passed to the [Rule](https://werkzeug.palletsprojects.com/en/2.0.x/routing/#werkzeug.routing.Rule) object.

Return type -- [None](https://docs.python.org/3/library/constants.html#None)

After\_Request(f)

Register a function to run after each request to this object.

The function is called with the response object, and must return a response object. This allows the functions to modify or replace the response before it is sent.

If a function raises an exception, any remaining after request functions will not be called. Therefore, this should not be used for actions that must execute, such as to close resources. Use [teardown\_request()](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Flask.teardown_request) for that.

**Parameters:**

**f** (Callable[[[Response](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Response)], [Response](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Response)])

Return type

Callable[[[Response](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Response)], [Response](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Response)]

after\_request\_funcs: t.Dict[AppOrBlueprintKey,

t.List[AfterRequestCallable]]

A data structure of functions to call at the end of each request, in the format {scope: [functions]}. The scope  key is the name of a blueprint the functions are active for, or None for all requests.

To register a function, use the [after\_request()](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Flask.after_request) decorator.

This data structure is internal. It should not be modified directly and its format may change at any time.

app\_context()

Create an [AppContext](https://flask.palletsprojects.com/en/2.0.x/api/#flask.ctx.AppContext). Use as a with block to push the context, which will make [current\_app](https://flask.palletsprojects.com/en/2.0.x/api/#flask.current_app) point at this application.

An application context is automatically pushed by [RequestContext.push()](https://flask.palletsprojects.com/en/2.0.x/api/#flask.ctx.RequestContext.push) when handling a request, and when running a CLI command. Use this to manually create a context outside of these situations.

With app.app\_context():

Init\_db()

**HTML**

HTML stands for Hyper Text Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

#### Basic Construction of an HTML Page

These tags should be placed underneath each other **at the top of every HTML page** that you create.



<!DOCTYPE html> — This tag**specifies the language** you will write on the page. In this case, the language is HTML 5.

<html> — This tag signals that from here on we are going to write in HTML code.

<head> — This is where all the **metadata for the page** goes — stuff mostly meant for search engines and other computer programs.

<body> — This is where the**content of the page** goes.

#### Further Tags

Inside the <head> tag, there is one tag that is always included: <title>, but there are others that are just as important:

<title>

This is where we**insert the page name** as it will appear at the top of the browser window or tab.

<meta>

This is where information about the document is stored: character encoding, name (page context), description.

**Head Tag**  
<head>

<title>My First Webpage</title>

<meta charset="UTF-8">

<meta name="description" content="This field contains information about your page. It is usually around two sentences long.">.

<meta name="author" content="Conor Sheils">

</header>

### Adding Content

Next, we will make<body> tag.

The HTML <body> is where we add the content which is designed for viewing by human eyes.

This includes **text, images, tables, forms**and everything else that we see on the internet each day.

#### Add HTML Headings To Web Page

In HTML, [headings](https://html.com/tags/heading/) are written in the following elements:

* <h1>
* <h2>
* <h3>
* <h4>
* <h5>
* <h6>

As you might have guessed <h1> and <h2> should be used for the most important titles, while the remaining tags should be used for sub-headings and less important text.

**Search engine bots use this order**when deciphering which information is most important on a page.

##### Creating Your Heading

Let’s try it out. On a new line in the HTML editor, type:

<h1> Welcome To My Page </h1>

And hit save. We will save this file as “index.html” in a new folder called “my webpage.”

**Add Text In HTML**

Adding text to our HTML page is simple using an element opened with the tag <p> which **creates a new paragraph**. We place all of our regular text inside the element <p>.

When we write text in HTML, we also have a number of other elements we can use **to control the text or make it appear in a certain way.**

#### Add Links In HTML

As you may have noticed, the internet is made up of lots of [links](https://html.com/anchors-links/).

Almost everything you click on while surfing the web is a link **takes you to another page** within the website you are visiting or to an external site.

Links are included in an attribute opened by the [**<a>**](https://html.com/tags/a/) tag. This element is the first that we’ve met which uses an attribute and so it**looks different to previously mentioned tags.**

<a href=<http://www.google.com>>Google</a>

**Image Tag**

In today’s modern digital world, [images](https://html.com/blog/100-legal-sources-free-stock-images/) are everything. The [**<**img**>**](https://html.com/tags/img/) tag has everything you need to display images on your site. Much like the <a> anchor element, <img> also contains an attribute.

The attribute features information for your computer regarding the source, height, width and alt text of the image

<img src=”yourimage.jpg” alt=”Describe the image” height=“X” width=“X”>

**CSS**

CSS stands for Cascading Style Sheets. It is the language for describing the presentation of Web pages, including colours, layout, and fonts, thus making our web pages presentable to the users.CSS is designed to make style sheets for the web. It is independent of HTML and can be used with any XML-based markup language. Now let’s try to break the acronym:

* Cascading: Falling of Styles
* Style: Adding designs/Styling our HTML tags
* Sheets: Writing our style in different documents

## **CSS Syntax**

Selector {

Property 1 : value;

Property 2 : value;

Property 3 : value;

}

For example:

h1

{

Color: red;

Text-align: center;

}

#unique

{

color: green;

}

* Selector: selects the element you want to target
* Always remains the same whether we apply internal or external styling
* There are few basic selectors like tags, id’s, and classes
* All forms this key-value pair
* Keys: properties(attributes) like color, font-size, background, width, height,etc
* Value: values associated with these properties

## **CSS Comment**

* Comments don’t render on the browser
* Helps to understand our code better and makes it readable.
* Helps to debug our code
* Two ways to  comment:
  + Single line

## **CSS How-To**

* There are 3 ways to write CSS in our HTML file.
  + Inline CSS
  + Internal CSS
  + External CSS
* Priority order
  + Inline > Internal > External

**Inline CSS**

* Before CSS this was the only way to apply styles
* Not an efficient way to write as it has a lot of redundancy
* Self-contained
* Uniquely applied on each element
* The idea of separation of concerns was lost
* Example:

<h3 style = “color:red”> Have a great day </h3>

<p style = “color:green”> I did this, I did that </p>

**Internal CSS**

* With the help of style tag, we can apply styles within the HTML file
* Redundancy is removed
* But the idea of separation of concerns still lost
* Uniquely applied on a single document
* Example:

<style>

H1{

Color:red;

}

</style>

<h3> Have a great day </h3>

**External CSS**

* With the help of <link> tag in the head tag, we can apply styles
* Reference is added
* File saved with .css extension
* Redundancy is removed
* The idea of separation of concerns is maintained
* Uniquely applied to each document
* Example:

<head>

<link rel= “stylesheet” type= “text/css” href= “name of the CSS file”>

</head>

h1{

color:red; //.css file

}

## **CSS Selectors**

* The selector is used to target elements and apply CSS
* Three simple selectors
  + Element Selector
  + Id Selector
  + Class Selector
* Priority of Selectors

## **CSS Colors**

* There are different colouring schemes in CSS
* **RGB**-This starts with RGB and takes 3 parameter
* **HEX**-Hex code starts with # and comprises of 6 numbers which are further divided into 3 sets
* **RGBA**-This starts with RGB and takes 4 parameter

**CSS Background**

* There are different ways by which CSS can have an effect on HTML elements
* Few of them are as follows:
  + Color – used to set the color of the background
  + Repeat – used to determine if the image has to repeat or not and if it is repeating then how it should do that
  + Image – used to set an image as the background
  + Position – used to determine the position of the image
  + Attachment – It basically helps in controlling the mechanism of scrolling.

## **CSS BoxModel**

* Every element in CSS can be represented using the BOX model
* It allows us to add a border and define space between the content
* It helps the developer to develop and manipulate the elements
* It consists of 4 edges
  + Content edge – It comprises of the actual content
  + Padding edge – It lies in between content and border edge
  + Border edge – Padding is followed by the border edge
  + Margin edge – It is an outside border and controls the margin of the element

**CODE:**

**MODULE 1:**

**# MANUAL NET ARCHITECTURE**

import warnings

warnings.filterwarnings('ignore')

import os

import glob

import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from PIL import Image

from tensorflow.keras.layers import Convolution2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Activation

from keras.callbacks import ModelCheckpoint

import matplotlib.pyplot as plt

Cyclone = 'DATASET/TRAIN/Cyclone'

Flood = 'DATASET/TRAIN/Flood'

Earthquake = 'DATASET/TRAIN/Earthquake'

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=(80, 40))

for idx, img\_path in enumerate(item\_files):

plt.subplot(3, n, idx+1)

img = plt.imread(img\_path)

plt.imshow(img, cmap='gray')

plt.axis('off')

plt.tight\_layout()

def image\_details\_print(data,path):

print('======== Images in: ', path)

for key,values in data.items():

print(key,':\t', values)

def images\_details(path):

files=[f for f in glob.glob(path + "\*\*/\*.\*", recursive=True)]

data={}

data['Images\_count']=len(files)

data['Min\_width']=10\*\*100

data['Max\_width']=0

data['Min\_height']=10\*\*100

data['Max\_height']=0

for f in files:

img=Image.open(f)

width,height=img.size

data['Min\_width']=min(width,data['Min\_width'])

data['Max\_width']=max(width, data['Max\_width'])

data['Min\_height']=min(height, data['Min\_height'])

data['Max\_height']=max(height, data['Max\_height'])

image\_details\_print(data,path)

print("")

print("TRAINING DATA FOR Cyclone:")

print("")

images\_details(Cyclone)

print("")

plot\_images(Cyclone, 10)

print("")

print("TRAINING DATA FOR Flood:")

print("")

images\_details(Flood)

print("")

plot\_images(Flood, 10)

print("")

print("TRAINING DATA FOR Earthquake:")

print("")

images\_details(Earthquake)

print("")

plot\_images(Earthquake, 10)

train\_datagen=ImageDataGenerator(rescale=1./255,shear\_range=0.2,zoom\_range=0.2,horizontal\_flip=True)

training\_set=train\_datagen.flow\_from\_directory('dataset/train',target\_size=(224,224),batch\_size=32,class\_mode='categorical')

test\_datagen=ImageDataGenerator(rescale=1./255)

test\_set=test\_datagen.flow\_from\_directory('dataset/test',target\_size=(224,224),batch\_size=32,class\_mode='categorical')

Classifier=Sequential()

Classifier.add(Convolution2D(32,(3,3),input\_shape=(224,224,3),activation='relu'))

Classifier.add(MaxPooling2D(pool\_size=(2,2)))

Classifier.add(Flatten())

Classifier.add(Dense(38, activation='relu'))

Classifier.add(Dense(10, activation='softmax'))

Classifier.compile(optimizer='rmsprop',loss='categorical\_crossentropy',metrics=['accuracy'])

model\_path = "MANUAL.h5"

callbacks = [

ModelCheckpoint(model\_path, monitor='accuracy', verbose=1, save\_best\_only=True)

]

epochs = 10

batch\_size = 512

#### Fitting the model

history = Classifier.fit(

training\_set, steps\_per\_epoch=training\_set.samples // batch\_size,

epochs=epochs,

validation\_data=test\_set,validation\_steps=test\_set.samples // batch\_size,

callbacks=callbacks)

import matplotlib.pyplot as plt

def graph():

#Plot training & validation accuracy values

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()

graph()

import matplotlib.pyplot as plt

def graph():

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()

graph()

**MODULE 2:**

import warnings

warnings.filterwarnings('ignore')

import tensorflow

import tensorflow as tf

print(tf.\_\_version\_\_)

import keras

import keras.backend as K

from keras.models import Model

from keras.layers import Input, Dense, Conv2D, Conv3D, DepthwiseConv2D, SeparableConv2D, Conv3DTranspose

from keras.layers import Flatten, MaxPool2D, AvgPool2D, GlobalAvgPool2D, UpSampling2D, BatchNormalization

from keras.layers import Concatenate, Add, Dropout, ReLU, Lambda, Activation, LeakyReLU, PReLU

from time import time

import numpy as np

from keras.callbacks import ModelCheckpoint

from tensorflow.keras.callbacks import EarlyStopping

import warnings

warnings.filterwarnings('ignore')

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train=ImageDataGenerator(rescale=1./255,shear\_range=0.2,zoom\_range=0.2,horizontal\_flip=True,validation\_split = 0.2)

train\_data=train.flow\_from\_directory(directory = 'DATASET/TRAIN',target\_size=(224,224),

batch\_size=32,class\_mode='categorical')

test=ImageDataGenerator(rescale=1./255)

test\_data=test.flow\_from\_directory(directory = 'DATASET/TEST',target\_size=(224,224),

batch\_size=32,class\_mode='categorical')

def shufflenet(input\_shape, n\_classes, g=8):

channels = 384, 769, 1536

repetitions = 3, 7, 3

def ch\_shuffle(x, g):

# 1 2 3 4 5 6 7 8 9 -reshape-> 1 2 3 -permute dims-> 1 4 7 -reshape-> 1 4 7 2 5 8 3 6 9

# 4 5 6 2 5 8

# 7 8 9 3 6 9

\_, w, h, ch = K.int\_shape(x)

ch\_g = ch // g

def shuffle\_op(x):

x = K.reshape(x, [-1, w, h, ch\_g, g])

x = K.permute\_dimensions(x, [0, 1, 2, 4, 3])

x = K.reshape(x, [-1, w, h, ch])

return x

x = Lambda(shuffle\_op)(x)

return x

def gconv(tensor, ch, g):

\_, \_, \_, in\_ch = K.int\_shape(tensor)

ch\_g = in\_ch // g

out\_ch = ch // g

group = []

for i in range(g):

# x = tensor[:, :, :, i\*ch\_g:(i+1)\*ch\_g]

x = Lambda(lambda x: x[:, :, :, i\*ch\_g: (i+1)\*ch\_g])(tensor)

x = Conv2D(out\_ch, 1)(x)

group.append(x)

x = Concatenate()(group)

return x

def shufflenet\_block(tensor, ch, s, g):

x = gconv(tensor, ch // 4, g)

x = BatchNormalization()(x)

x = ReLU()(x)

x = ch\_shuffle(x, g)

x = DepthwiseConv2D(3, strides=s, padding='same')(x)

x = BatchNormalization()(x)

x = gconv(x, ch if s==1 else ch-K.int\_shape(tensor)[-1], g)

x = BatchNormalization()(x)

if s == 1:

x = Add()([tensor, x])

else:

avg = AvgPool2D(3, strides=2, padding='same')(tensor)

x = Concatenate()([avg, x])

output = ReLU()(x)

return output

def stage(x, ch, r, g):

x = shufflenet\_block(x, ch, 2, g)

for i in range(r):

x = shufflenet\_block(x, ch, 1, g)

return x

input = Input(input\_shape)

x = Conv2D(24, 3, strides=2, padding='same')(input)

x = BatchNormalization()(x)

x = ReLU()(x)

x = MaxPool2D(3, strides=2, padding='same')(x)

for ch, r in zip(channels, repetitions):

x = stage(x, ch, r, g)

x = GlobalAvgPool2D()(x)

output = Dense(n\_classes, activation='softmax')(x)

model = Model(input, output)

model.compile(optimizer='Adam',loss='categorical\_crossentropy',metrics=['accuracy',tensorflow.keras.metrics.Precision()])

return model

input\_shape = 224, 224, 3

n\_classes = 10

K.clear\_session()

model = shufflenet(input\_shape, n\_classes)

model.summary()

model\_path = "SHUFFLE.h5"

from keras.callbacks import ModelCheckpoint

M = ModelCheckpoint(model\_path, monitor='accuracy', verbose=1, save\_best\_only=True)

epochs = 50

batch\_size = 128

#### Fitting the model

history = model.fit(

train\_data, steps\_per\_epoch=train\_data.samples // batch\_size,

epochs=epochs,

validation\_data=test\_data,validation\_steps=test\_data.samples // batch\_size,

callbacks=[M])

history.history.keys()

import matplotlib.pyplot as plt

import numpy as np

plt.figure(figsize=(20, 8))

plt.plot(history.history['accuracy'])

for i in range(epochs):

if i%5 == 0:

plt.annotate(np.round(history.history['accuracy'][i]\*100,2),xy=(i,history.history['accuracy'][i]))

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.show()

plt.figure(figsize=(20, 8))

plt.plot(history.history['loss'])

for i in range(epochs):

if i%5 == 0:

plt.annotate(np.round(history.history['loss'][i]\*100,2),xy=(i,history.history['loss'][i]))

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.show()

MODULE 3:

## RESNET ARCHITECTURE

import warnings

warnings.filterwarnings('ignore')

import tensorflow

import tensorflow as tf

print(tf.\_\_version\_\_)

import keras

import keras.backend as K

from keras.models import Model

from keras.layers import Input, Dense, Conv2D, Conv3D, DepthwiseConv2D, SeparableConv2D, Conv3DTranspose

from keras.layers import Flatten, MaxPool2D, AvgPool2D, GlobalAvgPool2D, UpSampling2D, BatchNormalization

from keras.layers import Concatenate, Add, Dropout, ReLU, Lambda, Activation, LeakyReLU, PReLU

from time import time

import numpy as np

from keras.callbacks import ModelCheckpoint

from tensorflow.keras.callbacks import EarlyStopping

import warnings

warnings.filterwarnings('ignore')

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train=ImageDataGenerator(rescale=1./255,shear\_range=0.2,zoom\_range=0.2,horizontal\_flip=True,validation\_split = 0.2)

train\_data=train.flow\_from\_directory(directory = 'DATASET/TRAIN',target\_size=(224,224),

batch\_size=32,class\_mode='categorical')

test=ImageDataGenerator(rescale=1./255)

test\_data=test.flow\_from\_directory(directory = 'DATASET/TEST',target\_size=(224,224),

batch\_size=32,class\_mode='categorical')

def resnet(input\_shape, n\_classes):

def conv\_bn\_rl(x, f, k=1, s=1, p='same'):

x = Conv2D(f, k, strides=s, padding=p)(x)

x = BatchNormalization()(x)

x = ReLU()(x)

return x

def identity\_block(tensor, f):

x = conv\_bn\_rl(tensor, f)

x = conv\_bn\_rl(x, f, 3)

x = Conv2D(4\*f, 1)(x)

x = BatchNormalization()(x)

x = Add()([x, tensor])

output = ReLU()(x)

return output

def conv\_block(tensor, f, s):

x = conv\_bn\_rl(tensor, f)

x = conv\_bn\_rl(x, f, 3, s)

x = Conv2D(4\*f, 1)(x)

x = BatchNormalization()(x)

shortcut = Conv2D(4\*f, 1, strides=s)(tensor)

shortcut = BatchNormalization()(shortcut)

x = Add()([x, shortcut])

output = ReLU()(x)

return output

def resnet\_block(x, f, r, s=2):

x = conv\_block(x, f, s)

for \_ in range(r-1):

x = identity\_block(x, f)

return x

input = Input(input\_shape)

x = conv\_bn\_rl(input, 64, 7, 2)

x = MaxPool2D(3, strides=2, padding='same')(x)

x = resnet\_block(x, 64, 3, 1)

x = resnet\_block(x, 128, 4)

x = resnet\_block(x, 256, 6)

x = resnet\_block(x, 512, 3)

x = GlobalAvgPool2D()(x)

output = Dense(n\_classes, activation='softmax')(x)

model = Model(input, output)

model.compile(optimizer='Adam',loss='categorical\_crossentropy',metrics=['accuracy',tensorflow.keras.metrics.Precision()])

return model

input\_shape = 224, 224, 3

n\_classes = 10

K.clear\_session()

model = resnet(input\_shape, n\_classes)

model.summary()

model\_path = "RESENT.h5"

from keras.callbacks import ModelCheckpoint

M = ModelCheckpoint(model\_path, monitor='accuracy', verbose=1, save\_best\_only=True)

epochs = 100

batch\_size = 32

#### Fitting the model

history = model.fit(

train\_data, steps\_per\_epoch=train\_data.samples // batch\_size,

epochs=epochs,

validation\_data=test\_data,validation\_steps=test\_data.samples // batch\_size,

callbacks=[M])

history.history.keys()

import matplotlib.pyplot as plt

import numpy as np

plt.figure(figsize=(20, 8))

plt.plot(history.history['accuracy'])

for i in range(epochs):

if i%5 == 0:

plt.annotate(np.round(history.history['accuracy'][i]\*100,2),xy=(i,history.history['accuracy'][i]))

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.show()

plt.figure(figsize=(20, 8))

plt.plot(history.history['loss'])

for i in range(epochs):

if i%5 == 0:

plt.annotate(np.round(history.history['loss'][i]\*100,2),xy=(i,history.history['loss'][i]))

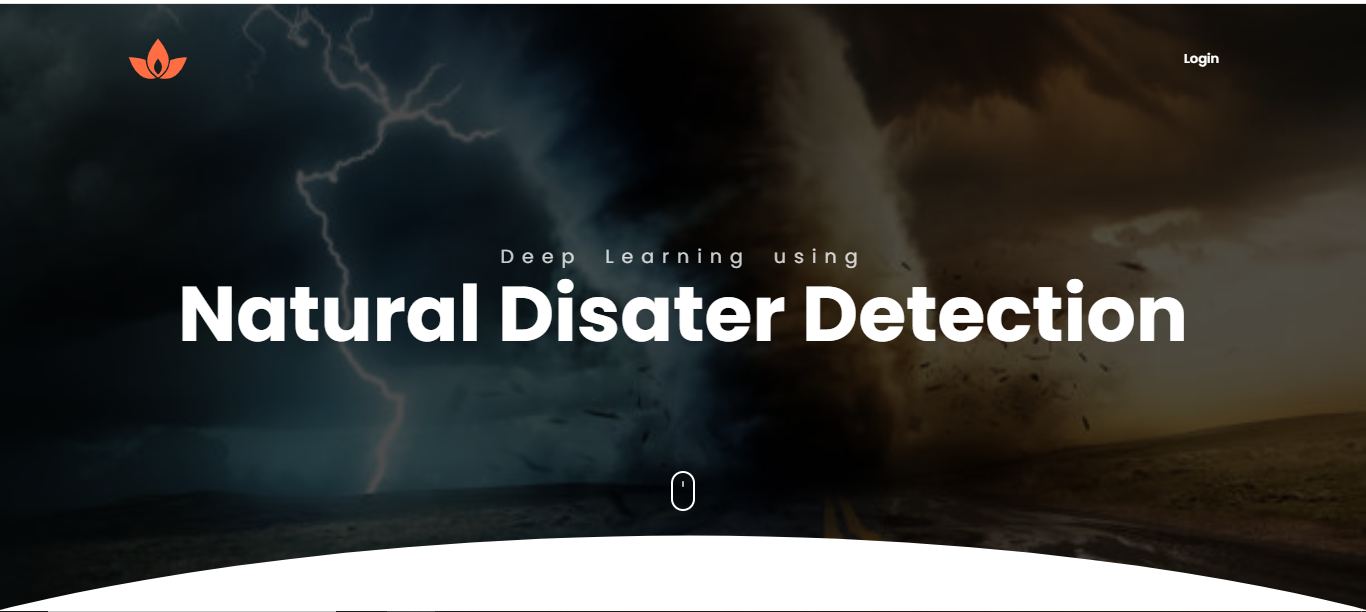
plt.title('Model Loss')

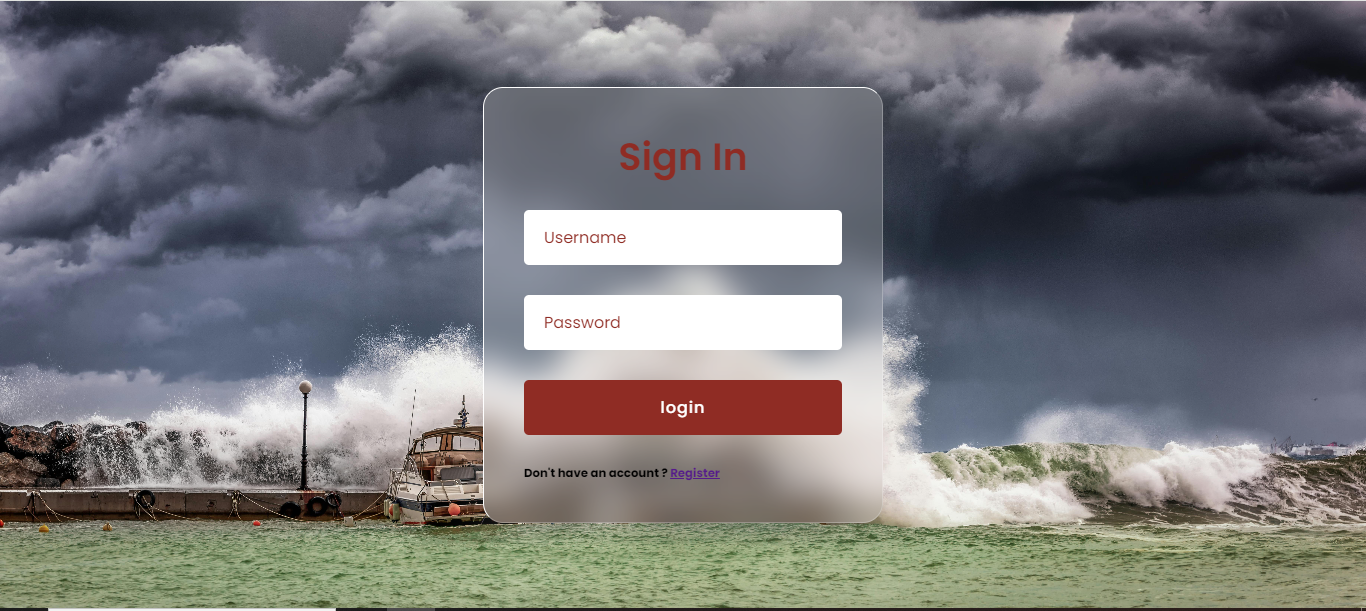
plt.ylabel('Loss')

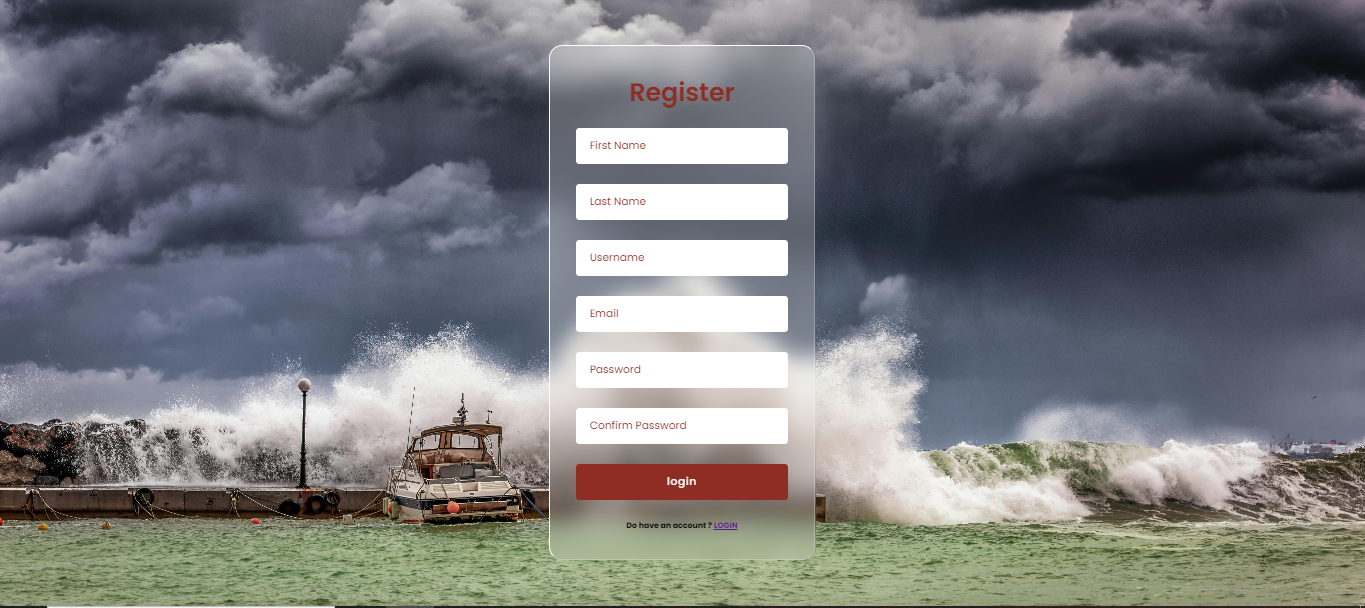
plt.xlabel('Epoch')

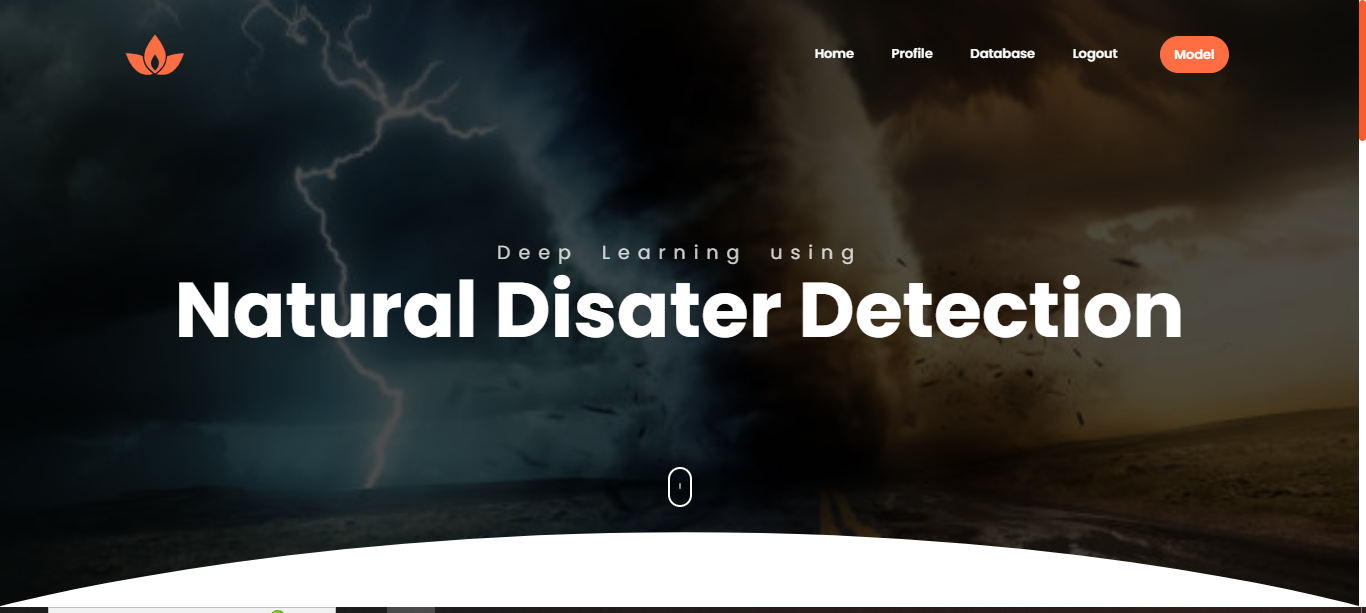
plt.show()

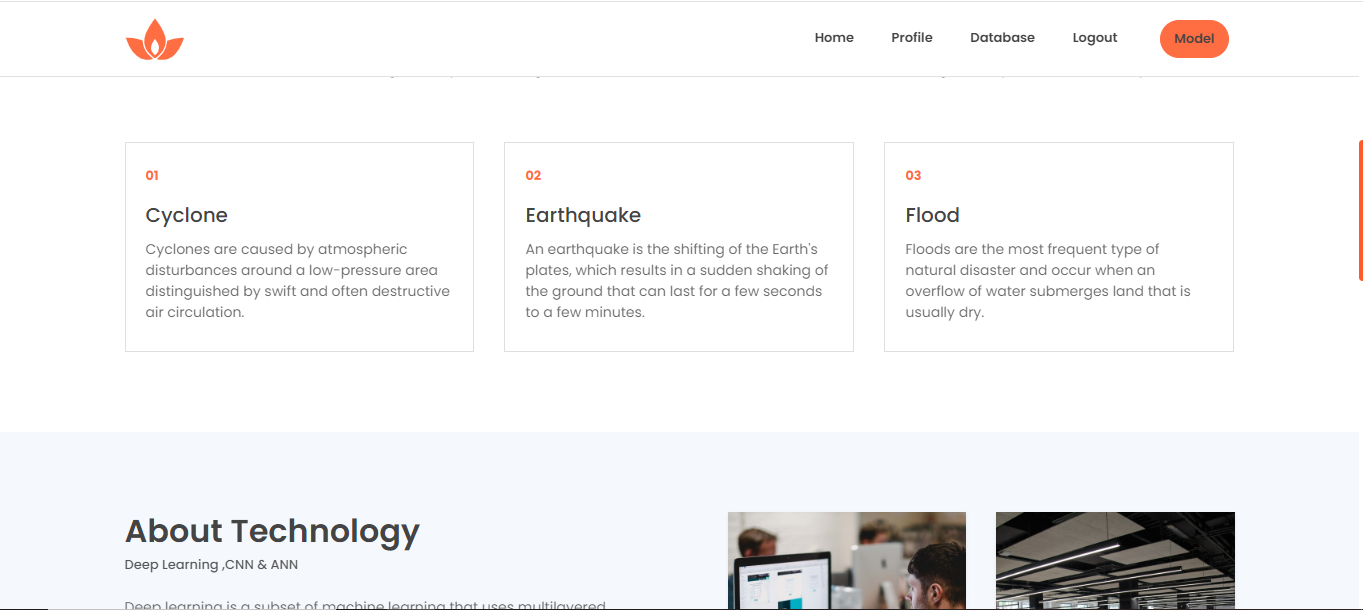
**OUTPUT SCREENSHOT:**

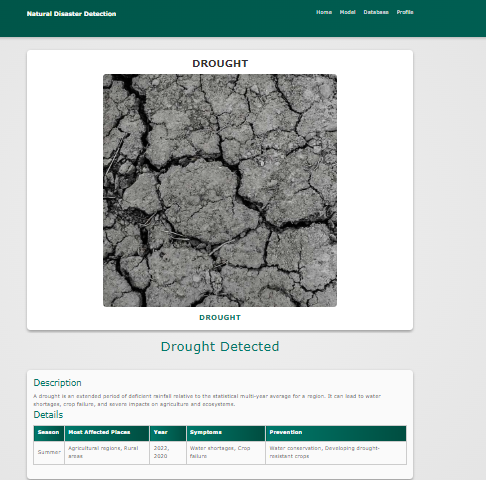
****

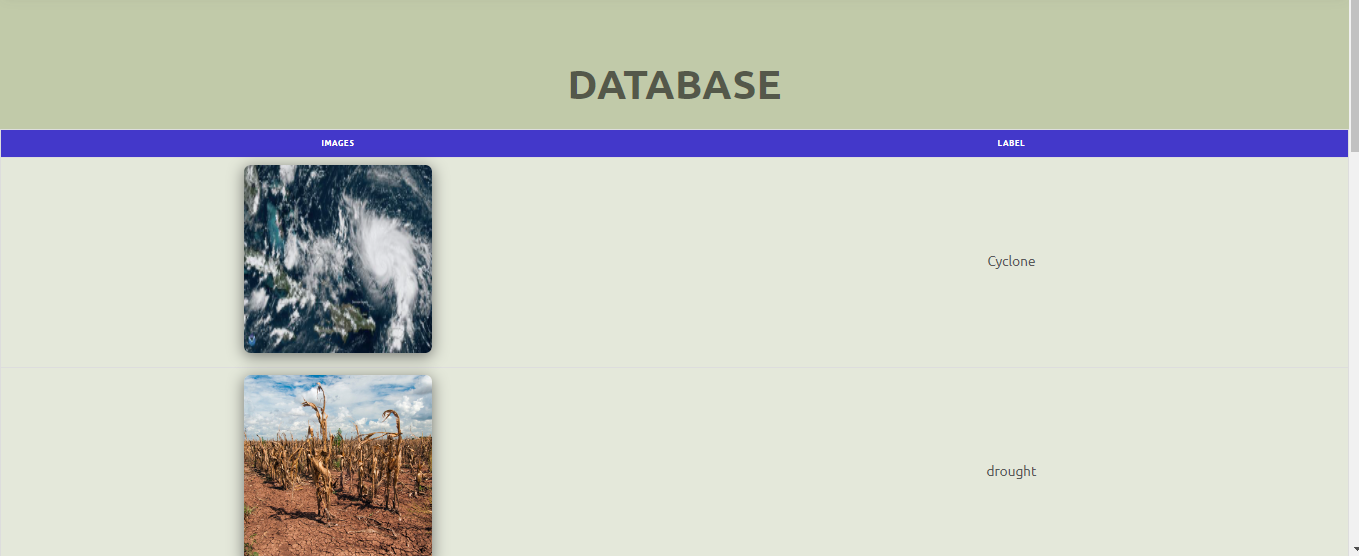
****

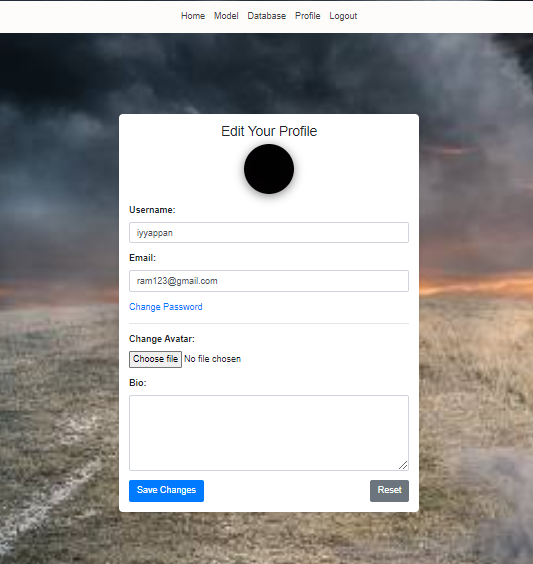
****

****

**** ****

****

****

****

**Conclusion:**

In the context of identifying natural disasters, Convolutional Neural Networks (CNNs) have proven to be a powerful tool, offering significant advancements in deep learning for image classification and pattern recognition. The application of CNNs in this domain allows for the accurate and efficient analysis of satellite images, aerial photographs, and other visual data, enabling the automated detection and classification of disasters such as floods, wildfires, hurricanes, and earthquakes. Through the extraction of hierarchical features from raw input data, CNNs can recognize complex patterns associated with different types of natural disasters, making it possible to predict their occurrence with higher accuracy and speed. This capability is crucial in early warning systems, disaster response planning, and mitigation strategies, ultimately contributing to saving lives and reducing economic losses. As deep learning techniques continue to evolve, the integration of CNNs in natural disaster identification systems promises even greater precision, reliability, and real-time application, paving the way for more robust and responsive disaster management solutions.

**FUTURE WORK:**

1. Enhance Data Diversity: Expand and diversify datasets with a wider range of skin conditions and demographics to improve model robustness and address class imbalances.

2. Develop Interpretability Methods: Implement techniques to make CNN decision-making more transparent and understandable, ensuring that clinicians can trust and effectively use the models in practice.