A Project Report

on

Fire and Smoke Detection using Convolutional Neural Networks

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE & ENGINEERING

by

19WH1A0556	Ms. Y. VARSHITHA
20WH5A0503	Ms. V. KAVYA

Under the esteemed guidance of

Dr. S. Ashok
Assistant Professor

Department of Computer Science & Engineering

BVRIT HYDERABAD
COLLEGE OF ENGINEERING FOR WOMEN
(NBA Accredited EEE.ECE.CSE.IT B.Tech Courses)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

June, 2023

DECLARATION

We hereby declare that the work describe in this report, entitled **"FIRE AND SMOKE DETECTION USING CONVOLUTIONAL NEURAL NETWORKS"** which is submitted by us in partial fulfillment for the award of the degree of **Bachelor of Technology** in the department of **Computer Science and Engineering** at **BVRIT HYDERABAD College of Engineering for Women**, affiliated to **Jawaharlal Nehru Technological University Hyderabad**, Kukatpally, Hyderabad – 500085 is the result of original work carried out under the guidance of **Dr. S. Ashok, Assistant Professor, Department of CSE.**

This work has not been submitted for any Degree / Diploma of this or any other institute/university to the best of our knowledge and belief.

Ms. Y. Varshitha

(19WH1A0556)

Ms. V. Kavya

(20WH5A0503)

Department of Computer Science and Engineering

BVRIT HYDERABAD
College of Engineering for Women

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with A Grade

Bachupally, Hyderabad – 500090

CERTIFICATE

This is to certify that the project work report, entitled **"Fire and Smoke Detection using Convolutional Neural Networks"** is a bonafide work carried out by **Ms. Y. Varshitha (19WH1A0556), Ms. V. Kavya (20WH5A503)** in partial fulfillment for the award of B.Tech degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to **Jawaharlal Nehru Technological University Hyderabad,** under my guidance and supervision.

The results embodied in the project work have not been submitted to any other university or institute for the award of any degree or diploma.

Head of Department	**Guide**
Dr. E. Venkateshwara Reddy	**Dr. S. Ashok**
Professor and HoD,	**Assistant Professor,**
Department of CSE	**Department of CSE**

External Examiner

ACKNOWLEDGEMENTS

The satisfaction that accompanies in successful completion of the task would be incomplete without the mention of the people who made it possible.

We would like to express our sincere thanks to **Dr. K. V. N. Sunitha, Principal, BVRIT HYDERABAD College of Engineering for Women**, for her support by providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. E. Venkateshwara Reddy, Head, Department of CSE, BVRIT HYDERABAD College of Engineering for Women**, for all timely support and valuable suggestions during the period of our project.

We are extremely thankful to our Internal Guide, **Dr. S. Ashok, Assistant Professor, CSE, BVRIT HYDERABAD College of Engineering for Women**, for his constant guidance and encouragement throughout the project.

Finally, we would like to thank our Major Project Coordinator, all Faculty and Staff of CSE department who helped us directly or indirectly. Last but not least, we wish to acknowledge our **Parents** and **Friends** for giving moral strength and constant encouragement.

Ms. Y. Varshitha
(19WH1A0556)

Ms. V. Kavya
(20WH5A0503)

CONTENTS

ABSTRACT

Fire outbreak is the common issue happening everywhere and the damage caused by this type of incidents is tremendous towards nature and human. Vision based fire detection system have recently gained popularity as compared to traditional sensor based fire detection system. However, the detection process by image processing technique is very tedious. We proposed a fire detection algorithm using Convolutional Neural Networks to achieve high-accuracy fire image detection, which is compatible in detection of fire by training with datasets. The model is trained by applying convolution, activation functions and max pooling operation. By using different batch sizes and epoch values, the model is trained. For better accuracy and high detection rate.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

• **CNN**	-	Convolutional Neural Networks
• **YOLO**	-	You Only Look Once
• **ANN**	-	Artificial Neural Network

1. INTRODUCTION

With rapid economic development, the increasing scale and complexity of constructions has introduced great challenges in fire control. Therefore, early fire detection and alarm with high sensitivity and accuracy is essential to reduce fire losses. However, traditional fire detection technologies, like smoke and heat detectors, are not suitable for large spaces, complex buildings, or spaces with many disturbances. Due to the limitations of above detection technologies, missed detections, false alarms, detection delays and other problems often occur, making it even more difficult to achieve early fire warnings. Recently, image fire detection has become a hot topic of research. The technique has many advantages such as early fire detection, high accuracy, flexible system installation, and the capability to effectively detect fires in large spaces and complex building structures. It processes image data from a camera by algorithms to determine the presence of a fire or fire risk in images. Therefore, the detection algorithm is the core of this technology, directly determining the performance of the image fire detector. There are three main stages in the process of image fire detection algorithms, including image preprocessing, feature extraction, and fire detection. Among, feature extraction is the core part in algorithms. Traditional algorithm depends on the manual selection of fire feature and machine learning classification. The shortcoming of algorithms is that manual feature selection should depend on professional knowledge. Though the researchers develop many studies in image features of smoke and flame, only simple image features, such as color, edges and simple textures, are discovered. However, because of complex fire types and scenes as well as many interference events in practical application, the algorithms extracting low and middle complex image features are difficult to distinguish fire and fire-like, thereby causing lower accuracy and weak generalization ability. Image recognition algorithms based on convolutional neural networks (CNNs) can automatically learn and extract complex image features effectively. This kind of algorithms has attracted great concerns and achieved excellent performance on visual search, automatic driving, medical diagnosis, etc. Therefore, some scholars introduce CNNs into the field of image fire detection, thereby developing the self-learned algorithm in collection of fire image features.

1.1. Objectives:

The main objective of the proposed system is to develop a fire detection technique in real time.

The main points of the proposed system objectives are given below:

- To develop a low-cost real-time fire detection system.
- To detect fire at an early stage

A system that detects a fire earlier and more accurately by using convolutional neural networks.

1.2. Methodology:

The methodology is proposed using CNN (Convolutional Neural Networks) model.

1.2.1. Dataset:

Proper dataset is required during the training and the testing phase. The dataset for the experiment is downloaded from the Kaggle which contains dataset is an extremely challenging set of over 7000+ original Fire and Smoke images captured and crowdsourced from over 400+ urban and rural areas, where each image is manually reviewed and verified by computer vision professionals at Data cluster Labs.

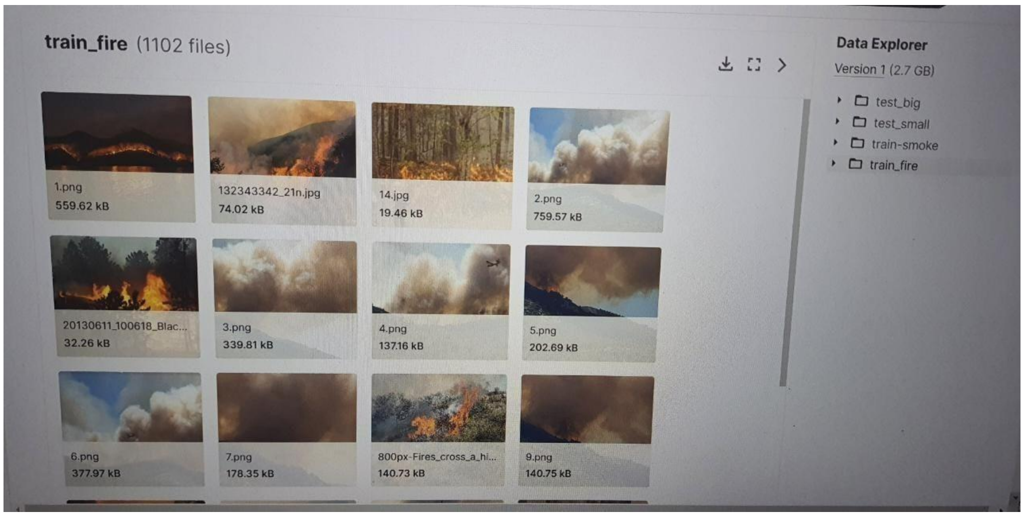


Fig-1: Sample Images in Dataset

Kaggle Dataset: https://www.kaggle.com/datasets/kutaykutlu/forest-fire?select=train_fire

1.2.2. The Proposed Model

On the basis of the literature review and subsequent findings, a deep learning-based model/method on the flow of image fire detection algorithms based on CNN is used to get better accuracy. CNN has functions of region proposals, feature extraction and classification.

1. Firstly, The CNN takes an image as input and outputs region proposals by convolution, pooling, etc.

2. Secondly, the region-based object detection CNN decides the presence or absence of fire in proposal regions through convolutional layers, pooling layers, fully-connected layers, etc.

1.3. Organization of Project

The methodology is proposed using CNN (Convolutional Neural Networks) model. Input is given to the convolutional layer, Several kernels of different sizes are applied on the input data to generate feature maps. The model consists of 64 convolution filters of size 3x3 each. The feature maps go through a ReLU activation function. This function updates positive portion of the feature map rapidly. These features maps are input to the next operation known as max pooling. These feature maps are subjected again to convolution layer and pooling layer which has kernel size of 3x3. Then a flatten layer which converts 2D feature maps into a vector that can be fed to fully connected layer. Among these three main operations, the convolution and fully connected layers contain neurons whose weights are learnt and adjusted for better representation of the input data during training process. A dense layer represents a matrix vector multiplication. The values in the matrix are the trainable parameters, which are updated during back propagation. Therefore, you get an m dimensional vector as output. Finally, we have an activation function such as Softmax to classify the outputs as fire ,smoke and neutral. That’s why it is used as final layer of classifying model. The model is compiled using an Adam optimizer which provides an adaptive learning rate to find individual learning rate of each parameter. Categorical cross entropy loss function is used in this classification as only one result could be correct.

2. LITERATURE REVIEW

Title: Fire Detection Using Convolutional Deep Learning Algorithms

Authors: Rabah Nory University of Anbar Mustafa Aljumaili Nezar Ismat University of Anbar

Summary:

The fire image dataset since the average accuracy was 81% despite it has more layers in its topology which led to being slower than the ConvNet system. ConvNet, on the other hand, shows excellent results in a way that its accuracy reaches 99% in a binary classification of the fire image dataset. ConvNet is superior also in execution due to the very small layer number. The lack of an image dataset forces us to limit the system in the binary classification test. We plan to use categorical classification in the future to evaluate our system. The proposed model was tested on the dataset collected from the internet media freely available for evaluation and comparison An algorithm for fire detection has been introduced in this paper. A CNN-based system for binary classification of fire is proposed for real-time applications. The evaluation process shows that LeNet was too bad at classifying the systems developed.

Title: Wildfire Flame and Smoke Detection Using Static Image Features and Artificial Neural Network

Authors: F M Anim Hossain , Youmin Zhang, Chi Yuan, and Chun-Yi Su, 30 September 2019.

Summary:

If forest fires are not contained quickly, they can spread wide very fast and cause devastating environmental, social and economic damages. The best method to minimize wildfire loss is to be able to detect it in its early stages for rapid containment and suppression. Fire comes with some distinguishable signatures such as flame, smoke and heat that can be used for early detection using computer vision based remote sensing techniques. Each signature has its own merits and demerits that vary under different environmental conditions and circumstances. Therefore, it is not always enough to form a detection algorithm based on a single signature. Keeping that in mind, this paper presents a novel algorithm that is capable of detecting both flame and smoke from a single image using block-based color features, texture features and a single artificial neural network (ANN). Such an algorithm is capable of providing reliable, rapid, and continuous detection under any circumstances and can be incorporated into the existing unmanned aerial vehicle (UAV) based fire monitoring system.

A dataset containing images of different scenarios Each of the 240×320 resolution images has 300 16×16 blocks. Therefore, a total of 6,000 blocks were used to train the images using backpropagation. This means that if flame and smoke are present in the image, it has been able to accurately classify at least one block as flame or smoke. If every block in the image is Gradient descent intuition considered, the rate of correctly classified blocks, commonly known as the true positive rate, is 89% for flame, 80.7% for smoke, and 87.4% for background blocks. The average accuracy of the system is 84.8%. As it uses R-CNN it takes more time as one image containing 6000 blocks and each one is required to train.

Title: Computer Vision-Based Fire Detection System Using OpenCV

Authors: Aman Kumar Flavia D Gonsalves

Summary:

The conventional fire detection system was based on mechanic sensors for fire detection. The particles in the surrounding detected by sensors in the traditional fire detection system. For example, a person smoking in a room of can activate a general fire alarm system. In addition, these systems are expensive and ineffective if the fire is far away from the detector. An alternative detection system such as a system based on computer vision and Image/video Processing technology to manage false alarms from conventional fire detection. One of the most cost-effective ways is to use surveillance cameras to detect fires and alert affected parties. In the following proposed system proposes a dictate that will monitor the outburst of a fire anywhere within the camera range using a surveillance camera. In this paper, model detects fires and protect lives and property from fire hazards. This research describes a fire detection system that uses colour and motion models derived from video sequences. The proposed approach identifies colour changes and mobility in common areas to identify fires and can therefore be used both real-time and in dataset.

Title: A Real-Time Smoke Detection Model Based on YOLO-SMOKE Algorithm

Authors: It Weibin Cai, Cuiyu Wang, Hongan Huang, Tongzhen Wang 11 March 2021

Summary:

YOLO-SMOKE model which shows strong robustness and high accuracy, uses data augmentation to avoid model overfittings. But in this model they developed a model only for smoke. Bad performance when there are groups of small objects, because each grid can only detect one object. Main error of YOLO is from localization because the ratio of bounding box is totally learned from data and YOLO makes error on the unusual ratio bounding box. The major reason why you cannot proceed YOLO,with this problem by building a standard and can't recognise smoke images followed by a fully connected layer is that the length of the output layer is variable — not constant, this is because the number of occurrences of the objects of interest is not fixed. A naive approach to solve this problem would be to take different regions of interest from the image, and use a CNN to classify the presence of the object within that region. The problem with this approach is that the objects of interest might have different spatial locations within the image and different aspect ratios. Hence, you would have to select a huge number of regions and this could computationally blow up. Therefore, algorithms like R-CNN etc have been developed to find these occurrences and find them fast.

3. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

3.1. Requirements Gathering

3.1.1. Software Requirements

- Operating System: Windows 10
- Programming Language: Python 3.10
- Tools: Google Colab,Jupyter Notebook

3.1.2. Hardware Requirements

- Processor: Intel core i5
- Memory: RAM 8GB
- Storage: 1TB

3.2. Technological Description

1. **Programming Language:**

Python: Python is a versatile and powerful programming language that offers a wide range of abilities, making it highly popular among developers. One of Python's notable strengths is its simplicity and readability, allowing programmers to write clean and concise code. This simplicity, combined with its extensive standard library, enables developers to accomplish complex tasks with minimal effort. Python excels in various domains, including web development, data analysis, scientific computing, machine learning, and artificial intelligence. In web development, frameworks like Django and Flask provide robust tools for building scalable and secure web applications. Python's extensive support for data analysis and scientific computing, through libraries such as NumPy, Pandas, and Matplotlib, empowers researchers and data scientists to process, analyze, and visualize data efficiently. The language's focus on simplicity extends to machine learning and artificial intelligence. With libraries like TensorFlow, PyTorch, and scikit-learn, developers can create sophisticated machine learning models for tasks like image recognition, natural language processing, and predictive analytics. Python's versatility Department of Computer Science and Engineering 07 Stock Prediction Using LSTM and ML Methods also makes it a preferred choice for scripting, automation, and rapid prototyping. Another significant ability of Python is its cross-platform compatibility, enabling developers to write code that runs seamlessly on different operating systems such as Windows, macOS, and Linux. Moreover, Python's extensive community support and active development contribute to a rich ecosystem of libraries, frameworks, and tools, further enhancing its capabilities. Python's ability to integrate with other languages, such as C, C++, and Java, through its robust C-API opens up opportunities for leveraging existing codebases and libraries. This capability enables developers to combine the efficiency of lower-level languages with the ease and flexibility of Python. In summary, Python's abilities span a broad spectrum, from web development to data analysis, machine learning, and artificial intelligence. Its simplicity, readability, cross-platform compatibility, and extensive ecosystem of libraries make it a versatile and powerful language that continues to drive innovation and solve complex problems across various domains.

TensorFlow Keras: TensorFlow Keras is a powerful deep learning framework built on top of TensorFlow that provides a high-level API for developing neural networks. With its extensive abilities, TensorFlow Keras has become a popular choice for researchers and developers working on various machine learning tasks. One of the key strengths of TensorFlow Keras is its simplicity and ease of use. It offers a user-friendly interface that allows developers to quickly prototype and build deep learning models. Its modular design enables the creation of complex neural networks by easily combining pre-built layers, activation functions, and optimizers. TensorFlow Keras supports a wide range of network architectures, including sequential models, functional models, and custom models. This flexibility allows developers to design and implement models suited to their specific tasks, whether it's image classification, natural language processing, or time series analysis. Another notable ability of TensorFlow Keras is its seamless integration with the TensorFlow ecosystem. By leveraging the underlying TensorFlow framework, TensorFlow Keras benefits from its powerful computation capabilities and distributed training support. This integration enables developers to efficiently train and deploy models on a variety of hardware, including CPUs, GPUs, and TPUs. TensorFlow Keras also provides a rich set of tools and utilities for model evaluation and visualization. Developers can easily assess model performance using various metrics and visualize model architectures using intuitive plotting functions. Department of Computer Science and Engineering 09 Stock Prediction Using LSTM and ML Methods TensorFlow Keras supports checkpointing, which allows users to save and restore model weights during training or inference. It offers built-in support for transfer learning and pre-trained models, enabling developers to leverage pre-existing model architectures and weights for their own tasks. This ability speeds up model development and reduces the need for large-scale training. TensorFlow Keras is a versatile deep learning framework with a wide range of abilities. Its simplicity, flexibility, and integration with TensorFlow make it a powerful tool for developing and deploying neural networks. Whether you're a beginner or an experienced researcher, TensorFlow Keras provides the necessary tools and utilities to build, train, and evaluate deep learning models for various machine learning tasks.

2. **Data Processing and Analysis:**

- Tenserflow: TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.
- Keras: Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow. It was developed to make implementing deep learning models as fast and easy as possible for research and development.

3.Development Environment:

Google Collab: Google Colab is a cloud-based development environment that provides a range of powerful abilities for coding, experimentation, and collaboration. Built on the Google Cloud Platform, Colab offers a convenient and accessible platform for Python programming, data analysis, and machine learning tasks. One of the key strengths of Google Colab is its ability to provide a Jupyter Notebook-like interface directly in the browser. Users can write and execute Python code in interactive cells, making it easy to experiment and iterate on ideas. Colab also supports Markdown cells, allowing users to document and explain their code effectively. Colab comes pre-installed with popular Python libraries, including NumPy, Pandas, and TensorFlow, enabling users to perform data analysis, visualization, and machine learning tasks without the need for local installations. Additionally, Colab supports GPU and TPU acceleration, allowing for faster training and inference of machine learning models. Another notable ability of Colab is its seamless integration with Google Drive. Users can easily import and export data from their Google Drive, making it convenient for working with large datasets and sharing code with collaborators. Colab also provides version control capabilities through integration with Git, enabling collaborative development and easy project management. Colab offers powerful data access and storage capabilities. It allows users to upload, download, and stream data from various sources, including local files, Google Drive, and popular cloud storage services. Additionally, Colab provides free cloud-based computing resources, enabling users to leverage the power of Google's infrastructure without the need for expensive hardware. Collaboration is made easy with Colab's ability to share notebooks. Users can invite others to view or edit their notebooks, facilitating real-time collaboration and project sharing. Colab also supports inline commenting and discussion threads, allowing for effective communication among collaborators. Colab integrates with other Google services, such as BigQuery for large-scale data analysis and Google Sheets for data manipulation and visualization. This seamless integration enhances the capabilities of Colab and expands its potential for various data-driven tasks.

Jupyter Notebook: The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Its uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context.According to the official website of jupyter, Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

4.DESIGN

4.1 Architecture

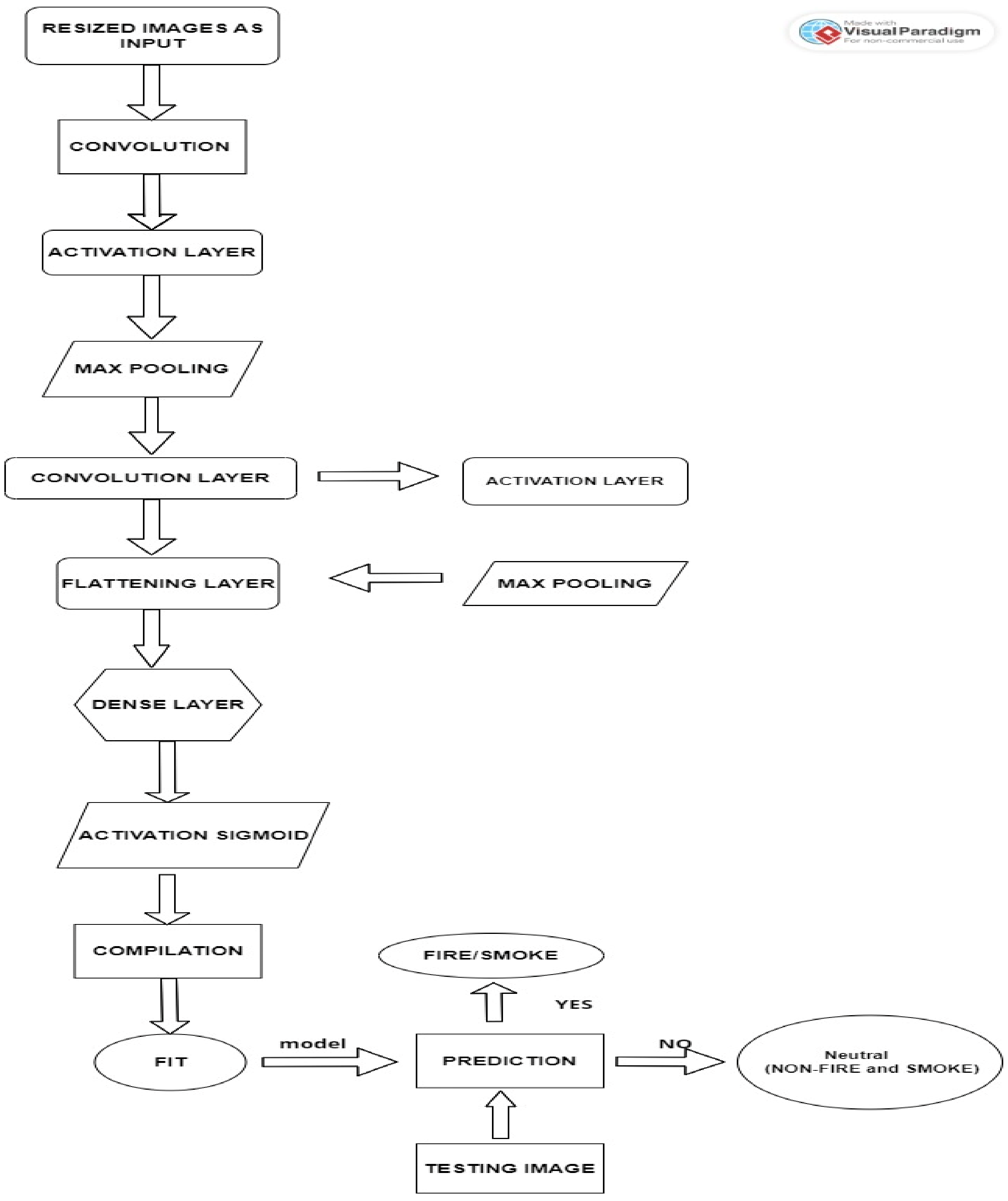


Fig 4.1: Architecture Diagram

4.2 Convolutional Neural Networks:

The dataset is trained using Convolutional Neural Networks. Here five convolutional layers are used to train the dataset. These layers were implemented using Sequential Model. In this model, we are using CNN classifier. The classifier is utilized for the classification of the Image Fusion and Recursive Filtering features. A convolution multiplies a matrix of pixels with a filter matrix or 'kernel' and sums up the multiplication values. Then the convolution slides over to the next pixel and repeats the same process until all the image pixels have been covered. The entire training process had completed with 100 epochs with a batch size of 64. Adam optimizer is used to compile the model. The trained model is then saved in json file.

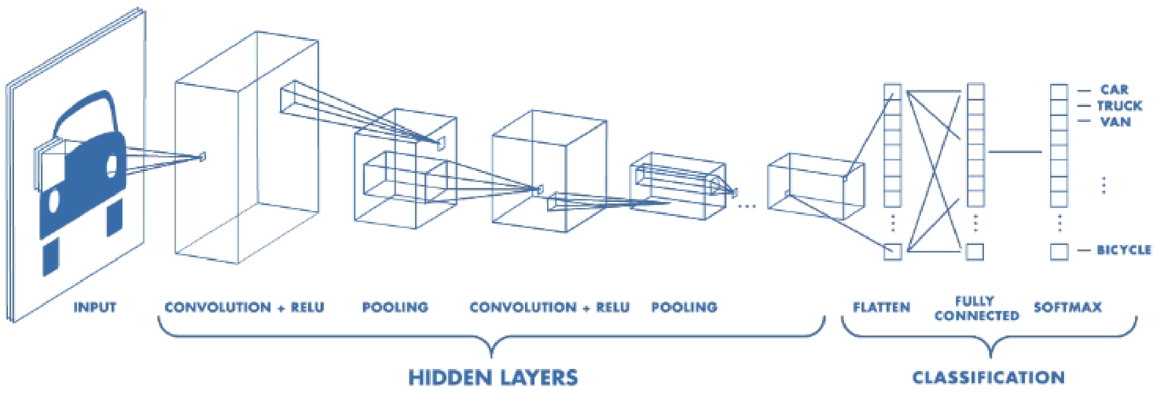


Figure 4.2.1: CNN Layer Architecture

A CNN is composed of several kinds of layers: The architecture of a CNN is a key factor in determining its performance and efficiency. The way in which the layers are structured, which elements are used in each layer and how they are designed will often affect the speed and accuracy with which it can perform various tasks. Convolution Layer: The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information and creates a feature map to predict the class probabilities for each feature by applying a filter that scans the whole image, few pixels at a time.

In the case of a CNN, the convolution is performed on the input data with the use of a filter or kernel to then produce a feature map.

Here are the three elements that enter into the convolution operation:

● Input image

● Feature detector

● Feature map

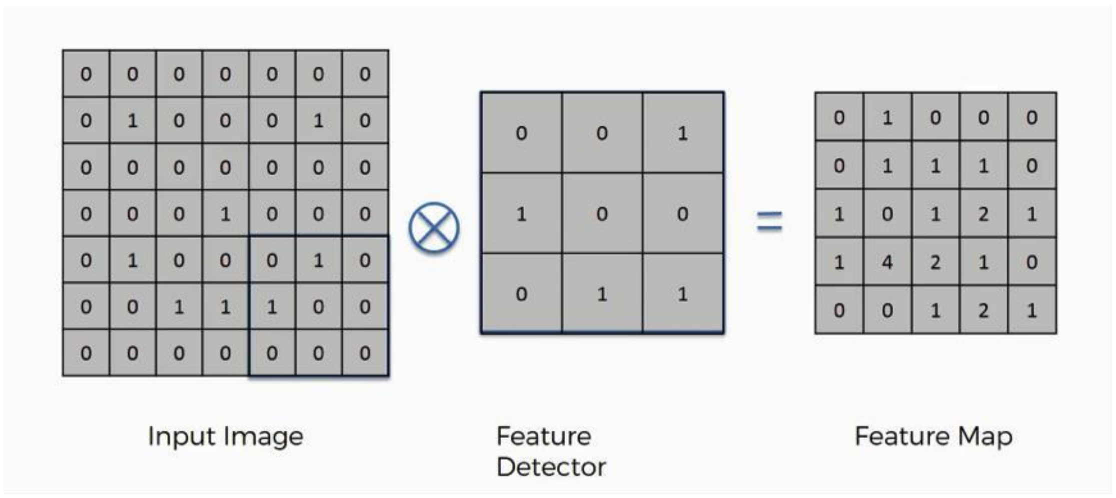


Fig 4.2.2: Convolution in CNN

Pooling Layer:

Max pooling is a sample-based discretization process. The objective is to down-sample an input representation. It scales down the amount of information the convolutional layer generated for each feature and maintains the most essential information (the process of the convolutional and pooling layers usually repeats several times).

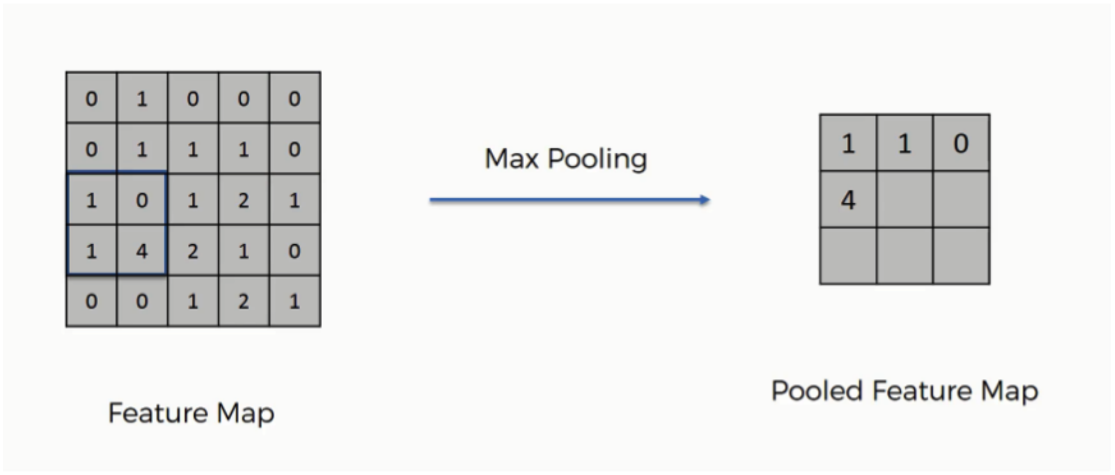


Fig 4.2.3: Max Pooling in CNN

Flattening:

Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector. The outputs generated by previous layers are flattened and turns into a single vector that can be used as an input for the next layer.

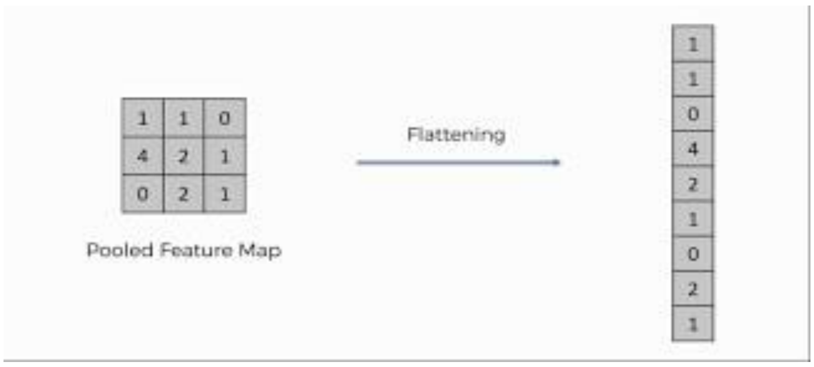


Fig 4.2.4: Flattening in CNN

Full Connection:

At the end of a CNN, the output of the last Pooling Layer acts as a input to the so called Fully Connected Layer. There can be one or more of these layers ("fully connected" means that every node in the first layer is connected to every node in the second layer).

As you see from the image below, we have three layers in the full connection step:

- Input layer
- Fully-connected layer
- Output layer

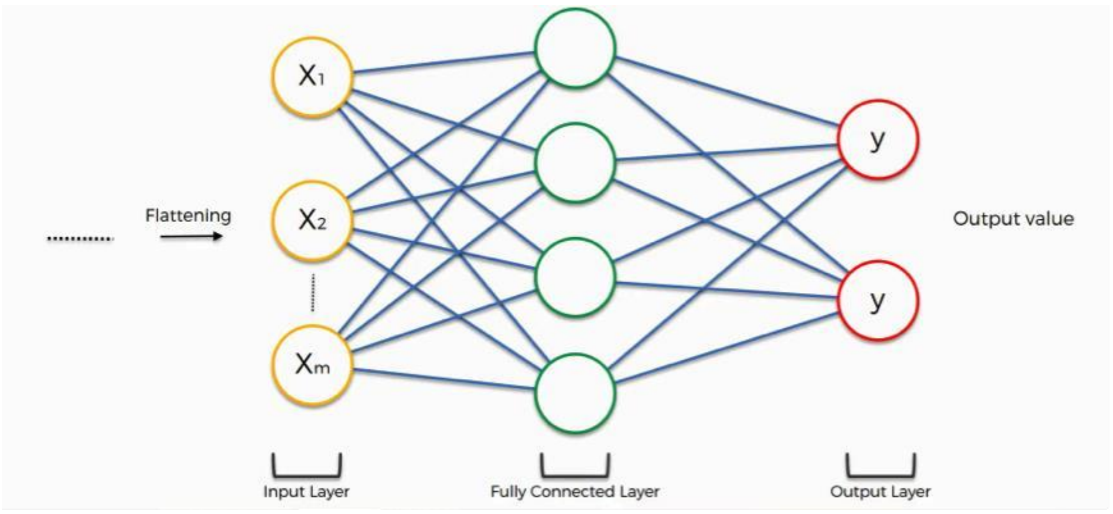


Fig 4.2.5: Full Connection in CNN

Advantages:

1. **Efficient image processing** – One of the key advantages of CNNs is their ability to process images efficiently. This is because they use a technique called convolution, which involves applying a filter to an image to extract features that are relevant to the task at hand. By doing this, CNNs can reduce the amount of information that needs to be processed, which makes them faster and more efficient than other types of algorithms.
2. **High accuracy rates** – Another advantage of CNNs is their ability to achieve high accuracy rates. This is because they can learn to recognize complex patterns in images by analyzing large datasets. This means that they can be trained to recognize specific objects or features with a high degree of accuracy, which makes them ideal for tasks like facial recognition or object detection.
3. **Robust to noise** – CNNs are also robust to noise, which means that they can still recognize patterns in images even if they are distorted or corrupted. This is because they use multiple layers of filters to extract features from images, which makes them more resilient to noise than other types of algorithms.
4. **Transfer learning** – CNNs also support transfer learning, which means that they can be trained on one task and then used to perform another task with little or no additional training. This is because the features that are extracted by CNNs are often generic enough to be used for a wide range of tasks, which makes them a versatile tool for many different applications.
5. **Automated feature extraction** – Finally, CNNs automate the feature extraction process, which means that they can learn to recognize patterns in images without the need for manual feature engineering. This makes them ideal for tasks where the features that are relevant to the task are not known in advance, as the CNN can learn to identify the relevant features through training.

Disadvantages:

1. **High computational requirements** – One of the main disadvantages of CNNs is their high computational requirements. This is because CNNs typically have a large number of layers and parameters, which require a lot of processing power and memory to train and run. This can make them impractical for use in some applications where resources are limited.
2. **Difficulty with small datasets** – CNNs also require large datasets to achieve high accuracy rates. This is because they learn to recognize patterns in images by analyzing many examples of

those patterns. If the dataset is too small, the CNN may overfit, meaning it becomes too specialized to the training dataset and performs poorly on new data.

3. **CNNs also require large datasets to achieve high accuracy rates. This is because they learn to recognize patterns in images by analyzing many examples of those patterns. If the dataset is too small, the CNN may overfit, meaning it becomes too specialized to the training dataset and performs poorly on new data.** – Another disadvantage of CNNs is their lack of interpretability. This means that it is difficult to understand how the CNN makes its decisions. This can be problematic in applications where it is important to know why a certain decision was made.
4. **Vulnerability to adversarial attacks** – CNNs are also vulnerable to adversarial attacks, which involve intentionally manipulating the input data to fool the CNN into making incorrect decisions. This can be a serious problem in applications like autonomous vehicles, where safety is a critical concern.
5. **Limited ability to generalize** – Finally, CNNs have a limited ability to generalize to new situations. This means that they may perform poorly on images that are very different from those in the training dataset. This can be a problem in applications where the CNN needs to work with a wide variety of images.

5. IMPLEMENTATION

5.1.Coding

GitHub link: 19wh1a0556varshitha/Fire-and-Smoke-Detection-using-Convolutional-Neural-Networks (github.com)

5.2.Evaluation metrics

5.2.1.Confusion Matrix

In the context of gait process analysis for neuro diseases using ensemble techniques, a confusion matrix is a tool used to evaluate the performance and accuracy of a classification model. It provides a tabular representation of the model's predicted classifications compared tothe actual ground truth labels.

A confusion matrix is typically a square matrix that consists of four cells, representing the four possible outcomes of a binary classification problem:

- True Positive (TP): The model correctly predicts a positive class (e.g., presence ofa neurodegenerative disease) when the actual label is also positive.
- True Negative (TN): The model correctly predicts a negative class (e.g., absence ofa neurodegenerative disease) when the actual label is also negative.
- False Positive (FP): The model incorrectly predicts a positive class when the actuallabel is negative (also known as a Type I error).
- False Negative (FN): The model incorrectly predicts a negative class when the actual label is positive (also known as a Type II error).

The confusion matrix is organized as follows

	Predicted Positive	Predicted Negative
Actual Positive	TP (True Positive)	FN (False Negative)
Actual Negative	FP (False Positive)	TN (True Negative)

Table 1. Confusion Matrix

Using the confusion matrix, several evaluation metrics can be derived to assess the model's performance:

1. **Accuracy:** It measures the overall correctness of the model's predictions and is calculatedas **(TP + TN) / (TP + TN + FP + FN).**

2. **Precision:** It indicates the proportion of correctly predicted positive instances among all instances predicted as positive and is calculated as **TP / (TP + FP).**

3. **F1 Score:** It is the harmonic mean of precision and recall, providing a balance between thetwo metrics.

By analyzing the values in the confusion matrix and calculating these evaluation metrics, it becomes possible to assess the performance of the ensemble model for gait process analysis in neurodegenerative diseases. This information helps to understand the model's predictivecapabilities, identify potential areas for improvement, and make informed decisions about theaccuracy and reliability of the ensemble model.

```
Epoch 1/10
437/437 [==============================] - 1506s 3s/step - loss: 0.2068 - accuracy: 0.9
283 - val_loss: 0.1609 - val_accuracy: 0.9427
Epoch 2/10
437/437 [==============================] - 1000s 2s/step - loss: 0.1411 - accuracy: 0.9
467 - val_loss: 0.0904 - val_accuracy: 0.9697
Epoch 3/10
437/437 [==============================] - 652s 1s/step - loss: 0.1199 - accuracy: 0.95
66 - val_loss: 0.1208 - val_accuracy: 0.9523
Epoch 4/10
437/437 [==============================] - 458s 1s/step - loss: 0.1137 - accuracy: 0.95
76 - val_loss: 0.1175 - val_accuracy: 0.9555
Epoch 5/10
437/437 [==============================] - 432s 990ms/step - loss: 0.1090 - accuracy:
0.9591 - val_loss: 0.0952 - val_accuracy: 0.9607
Epoch 6/10
437/437 [==============================] - 482s 1s/step - loss: 0.0986 - accuracy: 0.96
15 - val_loss: 0.0983 - val_accuracy: 0.9626
Epoch 7/10
437/437 [==============================] - 572s 1s/step - loss: 0.0968 - accuracy: 0.96
31 - val_loss: 0.0648 - val_accuracy: 0.9787
Epoch 8/10
437/437 [==============================] - 446s 1s/step - loss: 0.0960 - accuracy: 0.96
39 - val_loss: 0.0968 - val_accuracy: 0.9729
Epoch 9/10
```

```
437/437 [==============================] - 971s 2s/step - loss: 0.0951 - accuracy: 0.96
29 - val_loss: 0.0843 - val_accuracy: 0.9646
Epoch 10/10
437/437 [==============================] - 637s 1s/step - loss: 0.0872 - accuracy: 0.96
62 - val_loss: 0.0884 - val_accuracy: 0.9755
```

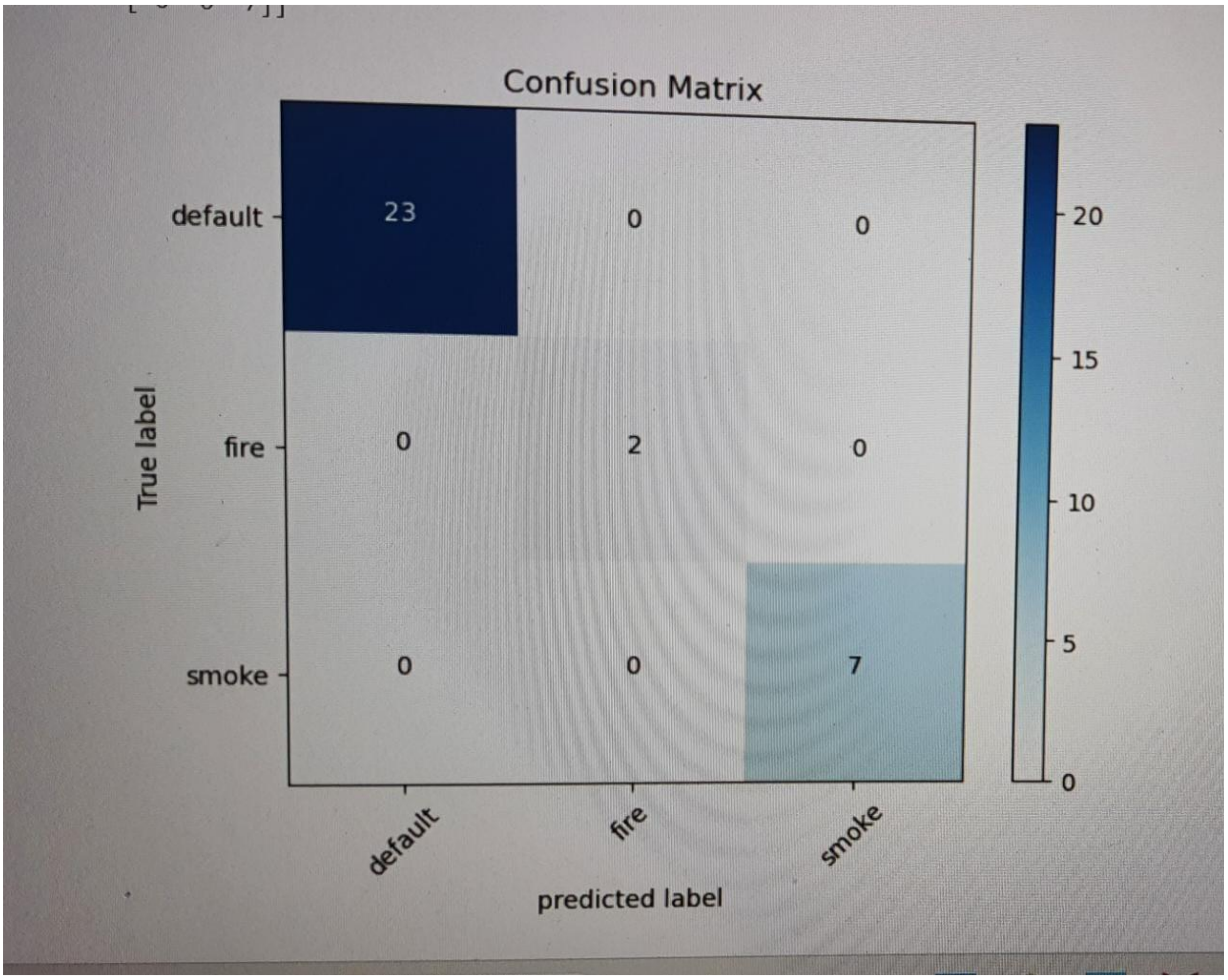
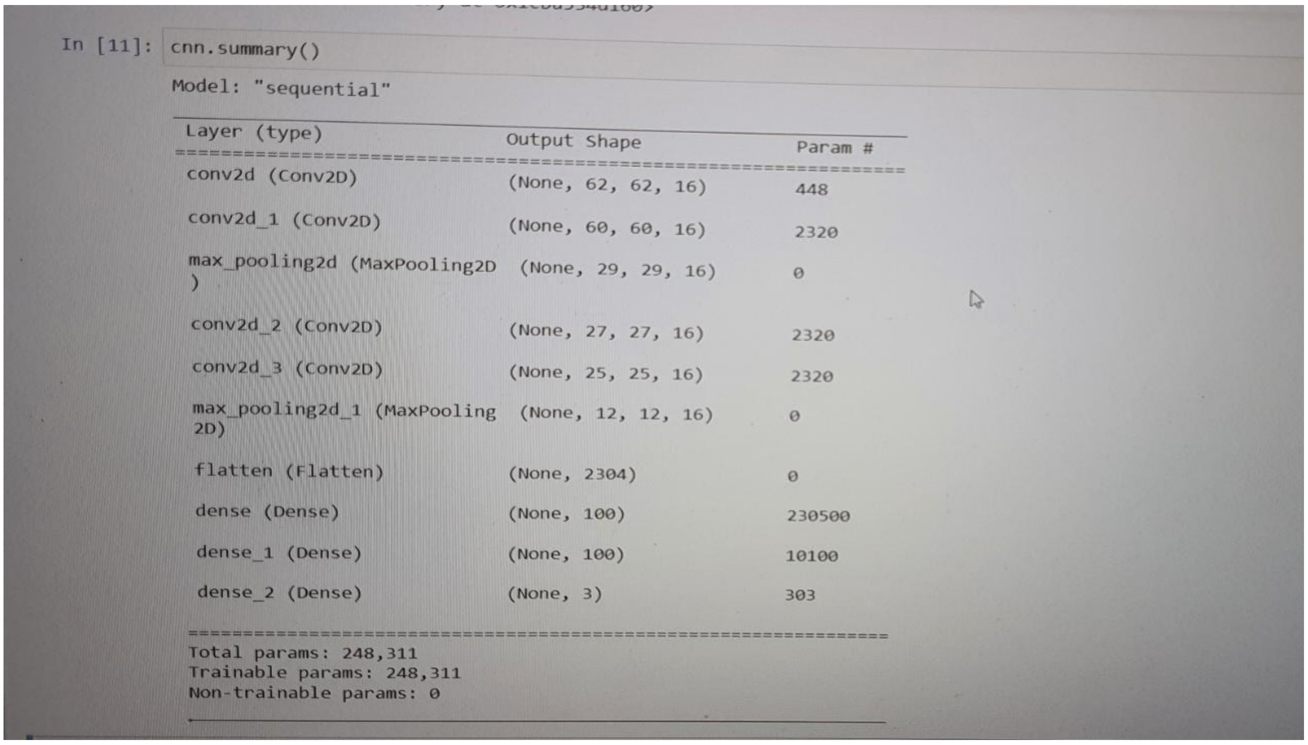


Fig.5.2.1. Confusion Matrix

5.3.Results



```
In [11]: cnn.summary()

         Model: "sequential"
         _________________________________________________________________
          Layer (type)                Output Shape              Param #
         =================================================================
          conv2d (Conv2D)             (None, 62, 62, 16)        448

          conv2d_1 (Conv2D)           (None, 60, 60, 16)        2320

          max_pooling2d (MaxPooling2D  (None, 29, 29, 16)       0
          )

          conv2d_2 (Conv2D)           (None, 27, 27, 16)        2320

          conv2d_3 (Conv2D)           (None, 25, 25, 16)        2320

          max_pooling2d_1 (MaxPooling  (None, 12, 12, 16)       0
          2D)

          flatten (Flatten)           (None, 2304)              0

          dense (Dense)               (None, 100)               230500

          dense_1 (Dense)             (None, 100)               10100

          dense_2 (Dense)             (None, 3)                 303

         =================================================================
         Total params: 248,311
         Trainable params: 248,311
         Non-trainable params: 0
         _________________________________________________________________
```

5.3.1 CNN Summary

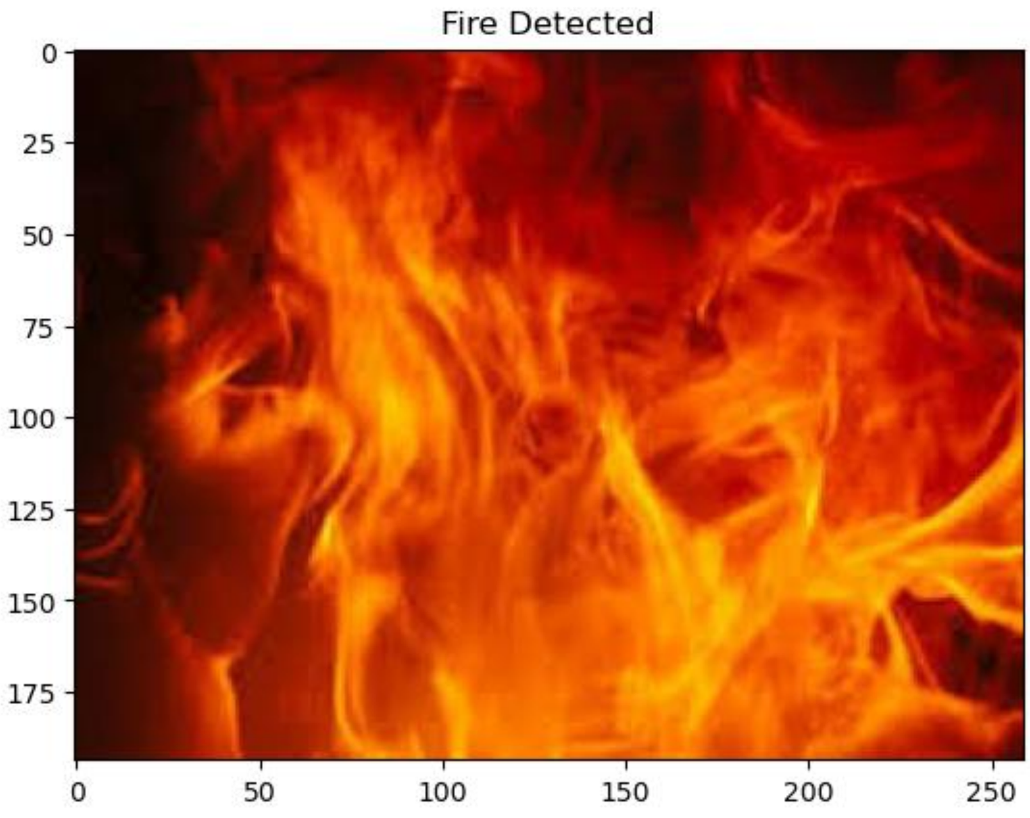
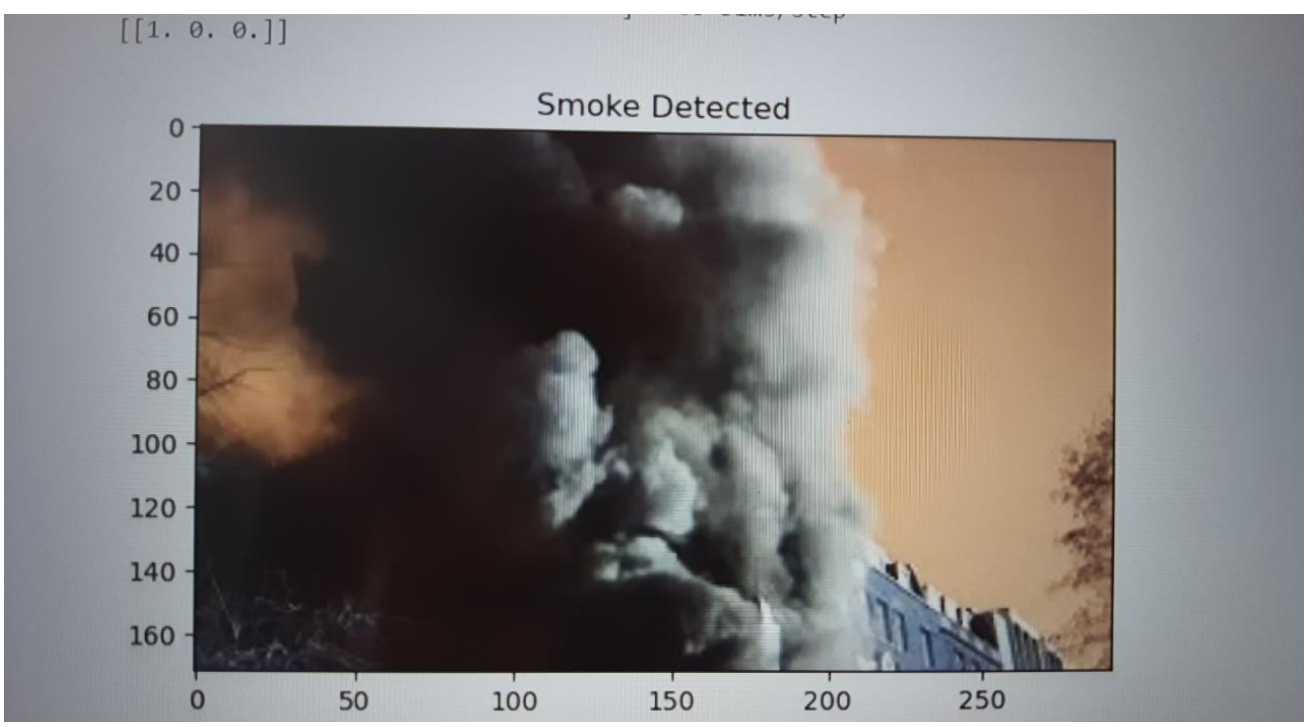
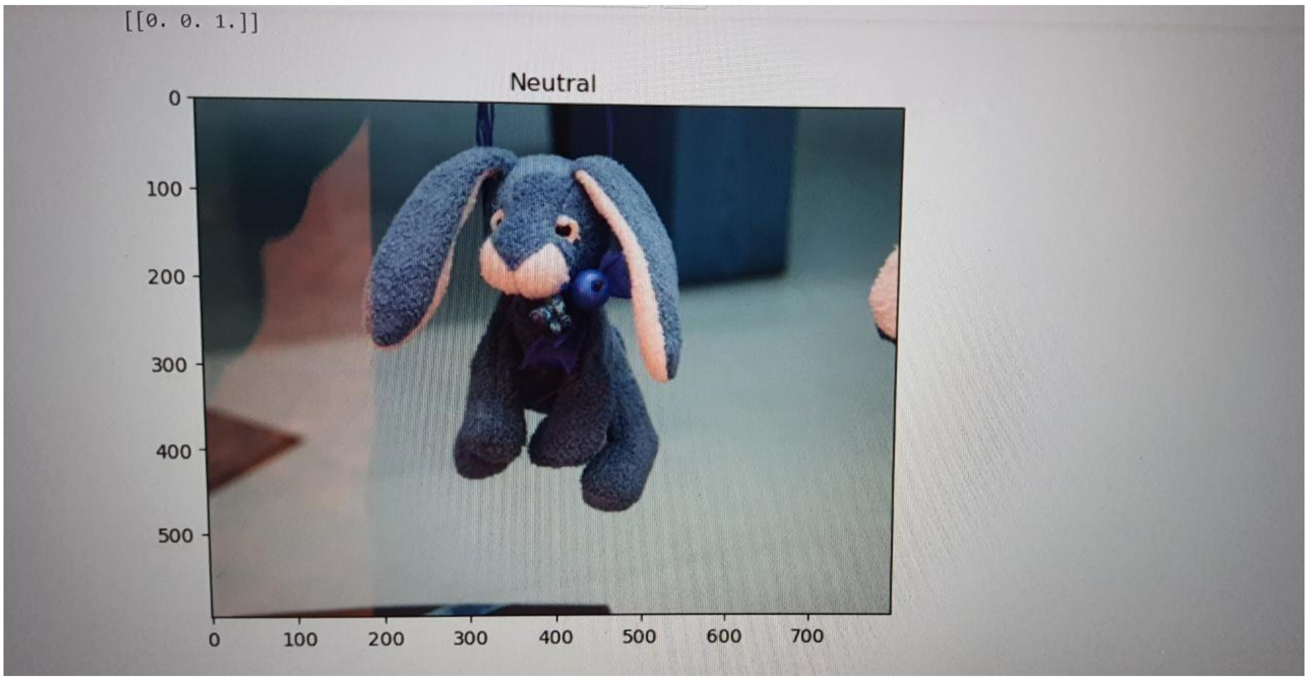


Fig:5.3.2. Image with Fire



```
[[1. 0. 0.]]
```

Fig.5.3.2. Image with Smoke



```
[[0. 0. 1.]]
```

Fig:5.3.4.neutral Image

6. CONCLUSION AND FUTURE SCOPE

The objective of implementing this work is that it should be capable of generating real-time information about the fire. The aim behind doing this work is to overcome the drawbacks of traditional firefighting systems.

Future Scope : There is a large scope for future advancements in early fire detection using deep learning neural networks. There are some areas in which further research and development can be done: Improvement in detection accuracy of the individual models: Deep learning algorithms can be further optimized to give better accuracy results where there are minor differences between fire and smoke. Sometimes models detect smoke as fire and that false information can be costly sometimes or lead to waste of resources.

Development of IOT based systems: An IOT based systems can be developed like drones, deep learning models can be fitted inside a drone and then the drone can detect fire and smoke at high altitudes where it's not possible for a human to reach. Real-time analysis of video and image data: Cameras can be fitted at the forest location on a tower and then they capture images at a periodic time and send those images to a model to detect. The future scope of the deep learning field is very vast, fire detection can be very much optimized with accuracy and can be designed in such a way that it takes less time to detect and save time. We have tried a lot of models but this research can be further advanced with some new neural networks being developed to use ensemble techniques other than the models we have tried.

7. REFERENCES

[1] Jareerat Seebamrungsat, Suphachai Praising, and Panomkhawn Riyamongkol in (2014)proposed a paper entitled Fire Detection in the Buildings Using Image Processing in Third ICT International Journal of IEEE Access in Department of Electrical and Computer Engineering.

[2] Khan Muhammad and Jamil Ahmad in (2018) proposed a paper entitled Efficient deep CNN based fire detection system and localization in video surveillance system at IEEE Transactions on Systems, Man, and Cybernetics Systems.

[3] Abdulaziz and Young Im CHO in (2018) proposed a paper entitled An Efficient Deep Learning Algorithm for Fire and Smoke Detection with Limited Data in IEEE Explore Advances in Electrical and Computer Engineering.

[4] Sebastien Frizzi1, Rabeb Kaabi, Moez Bouchouicha, Jean-Marc Ginoux in (2017) proposed a paper entitled Convolutional Neural Network for Video Fire and Smoke Detection in IEEE Transactions.

[5] Muhammad Khan and Jamil Ahmad in (2017) proposed a paper entitled Early fire detection using convolutional neural networks during surveillance for effective disaster management.

APPENDIX

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator

training_data_gen= ImageDataGenerator(rescale=1./255,horizontal_flip=True,
    shear_range=0.2,
    zoom_range=0.2,
    validation_split=0.1)

training_set=training_data_gen.flow_from_directory('D:/Major Project/Fire and Smoke and
Neutral Dataset/Training Data',
                                        target_size=(64, 64),
                               class_mode='categorical',
                               batch_size=32,
                               subset='training')

testing_set=training_data_gen.flow_from_directory('D:/Major Project/Fire and Smoke and Neutral
Dataset/Training Data',
                                        target_size=(64, 64),
                               class_mode='categorical',
                               batch_size=32,
                               subset='validation')

print(training_set[0][1])

#OUTPUT
[[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]
```

```
training_set.class_indices
```

```
#OUTPUT
{'train-smoke': 0, 'train_fire': 1, 'train_neutral': 2}
```

```
cnn=tf.keras.models.Sequential()

# adding first layer

cnn.add(tf.keras.layers.Conv2D(filters=16,kernel_size=3,activation='relu',input_shape=[64,64,3])
)

cnn.add(tf.keras.layers.Conv2D(filters=16,kernel_size=3,activation='relu'))

cnn.add(tf.keras.layers.MaxPool2D(pool_size=3,strides=2))

cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 16)        448

 conv2d_1 (Conv2D)           (None, 60, 60, 16)        2320

 max_pooling2d (MaxPooling2D  (None, 29, 29, 16)       0
 )

=================================================================
Total params: 2,768
Trainable params: 2,768
Non-trainable params: 0
```

```
# adding second layer

cnn.add(tf.keras.layers.Conv2D(filters=16,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=16,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=3,strides=2))

cnn.summary()
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 16)        448

 conv2d_1 (Conv2D)           (None, 60, 60, 16)        2320

 max_pooling2d (MaxPooling2D  (None, 29, 29, 16)       0
 )

 conv2d_2 (Conv2D)           (None, 27, 27, 16)        2320

 conv2d_3 (Conv2D)           (None, 25, 25, 16)        2320

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 16)       0
 2D)

=================================================================
Total params: 7,408
Trainable params: 7,408
Non-trainable params: 0

# Flattening Layer

cnn.add(tf.keras.layers.Flatten())

cnn.summary()
```

```
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 16)        448

 conv2d_1 (Conv2D)           (None, 60, 60, 16)        2320

 max_pooling2d (MaxPooling2D  (None, 29, 29, 16)       0
 )

 conv2d_2 (Conv2D)           (None, 27, 27, 16)        2320

 conv2d_3 (Conv2D)           (None, 25, 25, 16)        2320

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 16)       0
 2D)

 flatten (Flatten)           (None, 2304)              0

=================================================================
Total params: 7,408
Trainable params: 7,408
Non-trainable params: 0
```

```
# Fully connected layer

cnn.add(tf.keras.layers.Dense(units=100,activation='relu'))

cnn.add(tf.keras.layers.Dense(units=100,activation='relu'))

cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 16)        448       
                                                                 
 conv2d_1 (Conv2D)           (None, 60, 60, 16)        2320      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 29, 29, 16)       0         
 )                                                               
                                                                 
 conv2d_2 (Conv2D)           (None, 27, 27, 16)        2320      
                                                                 
 conv2d_3 (Conv2D)           (None, 25, 25, 16)        2320      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 12, 12, 16)       0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 2304)              0         
                                                                 
 dense (Dense)               (None, 100)               230500    
                                                                 
 dense_1 (Dense)             (None, 100)               10100     
                                                                 
=================================================================
Total params: 248,008
Trainable params: 248,008
Non-trainable params: 0
_________________________________________________________________
```

```
# Output layers

#cnn.add(tf.keras.layers.Dense(units=1,activation='sigmoid'))
```

```
cnn.add(tf.keras.layers.Dense(units=3,activation='softmax'))


cnn.summary()


cnn.compile(optimizer='Adam',loss='categorical_crossentropy', metrics=['accuracy'])


cnn.fit(x=training_set,validation_data=testing_set,epochs=10)
#output
Epoch 1/10
437/437 [==============================] - 1506s 3s/step - loss: 0.2068 - accuracy: 0.9
283 - val_loss: 0.1609 - val_accuracy: 0.9427
Epoch 2/10
437/437 [==============================] - 1000s 2s/step - loss: 0.1411 - accuracy: 0.9
467 - val_loss: 0.0904 - val_accuracy: 0.9697
Epoch 3/10
437/437 [==============================] - 652s 1s/step - loss: 0.1199 - accuracy: 0.95
66 - val_loss: 0.1208 - val_accuracy: 0.9523
Epoch 4/10
437/437 [==============================] - 458s 1s/step - loss: 0.1137 - accuracy: 0.95
76 - val_loss: 0.1175 - val_accuracy: 0.9555
Epoch 5/10
437/437 [==============================] - 432s 990ms/step - loss: 0.1090 - accuracy:
0.9591 - val_loss: 0.0952 - val_accuracy: 0.9607
Epoch 6/10
437/437 [==============================] - 482s 1s/step - loss: 0.0986 - accuracy: 0.96
15 - val_loss: 0.0983 - val_accuracy: 0.9626
Epoch 7/10
437/437 [==============================] - 572s 1s/step - loss: 0.0968 - accuracy: 0.96
31 - val_loss: 0.0648 - val_accuracy: 0.9787
Epoch 8/10
437/437 [==============================] - 446s 1s/step - loss: 0.0960 - accuracy: 0.96
39 - val_loss: 0.0968 - val_accuracy: 0.9729
Epoch 9/10
437/437 [==============================] - 971s 2s/step - loss: 0.0951 - accuracy: 0.96
29 - val_loss: 0.0843 - val_accuracy: 0.9646
Epoch 10/10
437/437 [==============================] - 637s 1s/step - loss: 0.0872 - accuracy: 0.96
62 - val_loss: 0.0884 - val_accuracy: 0.9755


  cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 16)        448

 conv2d_1 (Conv2D)           (None, 60, 60, 16)        2320

 max_pooling2d (MaxPooling2D  (None, 29, 29, 16)       0
 )

 conv2d_2 (Conv2D)           (None, 27, 27, 16)        2320

 conv2d_3 (Conv2D)           (None, 25, 25, 16)        2320

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 16)       0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 100)               230500

 dense_1 (Dense)             (None, 100)               10100

 dense_2 (Dense)             (None, 3)                 303

=================================================================
Total params: 248,311
Trainable params: 248,311
Non-trainable params: 0
_________________________________________________________________
```

```
scores = cnn.evaluate(training_set)
print(scores[1]*100)
```

```
scores = cnn.evaluate(testing_set)
print(scores[1]*100)

testing_set_label = testing_set[0][1]
testing_set_sample = testing_set[0][0]

predictions = cnn.predict(testing_set_sample,batch_size=32,verbose=0)

predictions
#output
array([[3.5119356e-05, 4.3141558e-03, 9.9565071e-01],
       [9.9993908e-01, 6.0609756e-05, 3.7889424e-07],
       [1.0000000e+00, 5.5738496e-08, 3.5004486e-12],
       [1.0000000e+00, 4.9730327e-09, 5.4129010e-12],
       [9.9612373e-01, 3.8669149e-03, 9.3557483e-06],
       [9.9999917e-01, 7.9765880e-07, 6.9139139e-11],
       [1.0000000e+00, 4.2989948e-12, 4.9202386e-15],
       [1.1830502e-05, 9.9795270e-01, 2.0355477e-03],
       [9.9999952e-01, 4.4585059e-07, 3.3801770e-10],
       [1.0373498e-03, 2.4689781e-02, 9.7427285e-01],
       [9.8341137e-01, 1.6427761e-02, 1.6088302e-04],
       [9.9999797e-01, 2.0288298e-06, 3.8488857e-08],
       [8.2485043e-03, 2.4515188e-03, 9.8930001e-01],
       [4.4484178e-08, 3.1004247e-04, 9.9968994e-01],
       [1.0000000e+00, 7.3933069e-13, 1.2931832e-15],
       [6.7005885e-06, 1.0973702e-03, 9.9889588e-01],
       [9.9999857e-01, 1.4275726e-06, 2.5046194e-08],
       [1.3004376e-04, 2.0897265e-03, 9.9778026e-01],
       [1.0000000e+00, 7.0435333e-12, 7.5778278e-15],
       [9.9999654e-01, 3.4185671e-06, 3.2096261e-09],
       [9.9999964e-01, 3.7457730e-07, 4.0334444e-11],
       [1.0000000e+00, 2.0626903e-10, 4.2077026e-14],
       [1.0000000e+00, 1.4709520e-08, 1.2817035e-11],
       [1.0000000e+00, 7.2420361e-11, 1.4207838e-13],
       [1.0000000e+00, 9.1316496e-11, 2.4695326e-13],
       [1.0000000e+00, 4.8528781e-09, 3.3943611e-13],
       [1.0000000e+00, 8.6171337e-09, 4.7375728e-11],
       [2.2925232e-03, 8.6543638e-01, 1.3227116e-01],
       [1.0000000e+00, 2.1620130e-08, 1.1921074e-10],
       [2.9432600e-05, 8.1984786e-04, 9.9915075e-01],
       [9.9999988e-01, 6.3976699e-08, 2.4637753e-12],
       [9.9999738e-01, 2.5764318e-06, 1.0840859e-08]], dtype=float32)

rounded_predictions=[]
```

```
 for arr in predictions:
    if arr[0]>arr[1] and arr[0]>arr[2]:
        rounded_predictions.append([1,0,0])
    elif arr[1]>arr[0] and arr[1]>arr[2]:
        rounded_predictions.append([0,1,0])
    else:
        rounded_predictions.append([0,0,1])
rounded_predictions
#output
[[0, 0, 1],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [0, 1, 0],
 [1, 0, 0],
 [0, 0, 1],
 [1, 0, 0],
 [1, 0, 0],
 [0, 0, 1],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [1, 0, 0],
 [0, 1, 0],
 [1, 0, 0],
 [0, 0, 1],
 [1, 0, 0],
 [1, 0, 0]]

rounded_predictions=np.array(rounded_predictions)
```

```
%matplotlib inline
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt

cm=confusion_matrix(testing_set_label.argmax(axis=1),rounded_predictions.argmax(axis=1))
#output
Confusion matrix, without normalization
[[23  0  0]
 [ 0  2  0]
 [ 0  0  7]]

defplot_confusion_matrix(cm,classes,normalize=False,title='Confusion
matrix',cmap=plt.cm.Blues):
    plt.imshow(cm,interpolation='nearest',cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks,classes,rotation=45)
    plt.yticks(tick_marks,classes)

    if normalize:
        cm = cm.astype('float')/cm.sum(axis=1)[:,np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)
    thresh = cm.max()/2.
    for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
                    plt.text(j,i,cm[i,j],horizontalalignment = "center",color ="white" if cm[i,j] >
thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('predicted label')
```

```
cm_plot_labels=['default','fire','smoke']
plot_confusion_matrix(cm,cm_plot_labels,title="Confusion Matrix")

pip install opencv-python
cnn.save('fire_smoke_detection_model.h5')
from keras.models import load_model

cnn = load_model('fire_smoke_detection_model.h5')

import numpy as np
from tensorflow.keras.preprocessing import image
from PIL import Image
%matplotlib inline
from matplotlib import pyplot as plt
import cv2
# im = cv2.imread("D:\Major Project\Fire and Smoke and Neutral Dataset\Test\Smoke\image_0.jpg",cv2.IMREAD_UNCHANGED)
im = cv2.imread("D:\Major Project\Fire and Smoke and Neutral Dataset\Test\Smoke\image_1.jpg",cv2.IMREAD_UNCHANGED)
# im = cv2.imread("D:/Major Project/Fire and Smoke and Neutral Dataset/Test/Neutral/image_64.jpg",cv2.IMREAD_UNCHANGED)
# im = im.astype(np.uint8)
plt.imshow(im)

# plt.show()
test_image = image.load_img("D:\Major Project\Fire and Smoke and Neutral Dataset\Test\Smoke\image_0.jpg",target_size = (64,64))

test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
# result = (cnn.predict(test_image) > 0.5).astype("int32")



result = cnn.predict(test_image)
```

```
print(result)
if result[0][0] == 1:
   plt.title("Smoke Detected")
   plt.show()
elif result[0][1] == 1:
   plt.title("Fire Detected")
   plt.show()
elif result[0][2] == 1:
   plt.title("Neutral")
   plt.show()
```