PROJECT REPORT:

OBJECTIVE:

The goal of this project is to create a fire detection system. The key objectives of the project is to identify fires from the images we can get from surveillance system or other resources.

DATASET DESCRIPTION:

The dataset comprises of Training and Testing Dataset images, which contain 3 categories.

The categories are:

1. Fire
2. Non Fire
3. Smoke

The Training Dataset contains approximately 30,000 images with 10k images of each category.

The Testing dataset contains approximately 10k images with approximately 3k images of each category.

CONSTRAINTS:

On briefly analyzing the dataset and problem statement, we can observe some constraints and preprocessing needs for our dataset.

i) The problem statement requires classification using a Deep Learning Neural Network.
ii) Hardware or Computational Resources can be an issue in Training models with more hyperparameters.
iii) High memory usage while loading the dataset.

PERFORMANCE METRICS:

- **Accuracy:** Measures the overall correctness of a model by calculating the ratio of correctly predicted instances to the total instances.
- **F1 Score:** Harmonic mean of precision and recall, providing a balanced metric for binary classification, particularly useful when classes are imbalanced.
- **Precision:** Proportion of true positive predictions among all positive predictions, indicating the model's ability to avoid false positives.
- **Recall:** Proportion of true positive predictions among all actual positives, revealing the model's ability to capture all relevant instances.
- **AUC-ROC Score:** Area Under the Receiver Operating Characteristic curve, quantifying the model's ability to distinguish between classes, with higher values indicating better performance.

DEEP LEARNING NEURONS:

1. **Sequential Model**: **Model** is defined as a sequential model, meaning that the layers are stacked sequentially on top of each other. This is a simple and common way to define neural networks in Keras.
2. **Convolutional Layers**: The network consists of four convolutional layers (`Conv2D`). Convolutional layers are responsible for extracting features from the input images. Each convolutional layer applies a set of filters to the input image, producing feature maps.
3. **Activation Functions**: The Rectified Linear Unit (ReLU) activation function (`activation='relu'`) is used after each convolutional layer. ReLU is a commonly used activation function that introduces non-linearity into the network and helps the model learn complex patterns in the data.
4. **Max Pooling Layers**: After each convolutional layer, a max pooling layer (`MaxPool2D`) is added. Max pooling reduces the spatial dimensions of the feature maps, which helps in reducing the computational complexity and controlling overfitting.
5. **Flatten Layer**: After the convolutional layers, a flatten layer is added. This layer reshapes the 2D feature maps into a 1D vector, which is required as input for the fully connected (Dense) layers.
6. **Dense Layers**: Two fully connected (Dense) layers are added after the flatten layer. The first dense layer has 512 units and uses the ReLU activation function. The second dense layer has 3 units (equal to the number of classes) and uses the softmax activation function. Softmax converts the raw output scores into probabilities, making it suitable for multi-class classification tasks.

Overall, this CNN architecture follows a common pattern for image classification tasks. It consists of alternating convolutional and max pooling layers for feature extraction, followed by fully connected layers for classification. The ReLU activation function is used for all layers except the output layer, which uses softmax for multi-class classification.

Model Training:

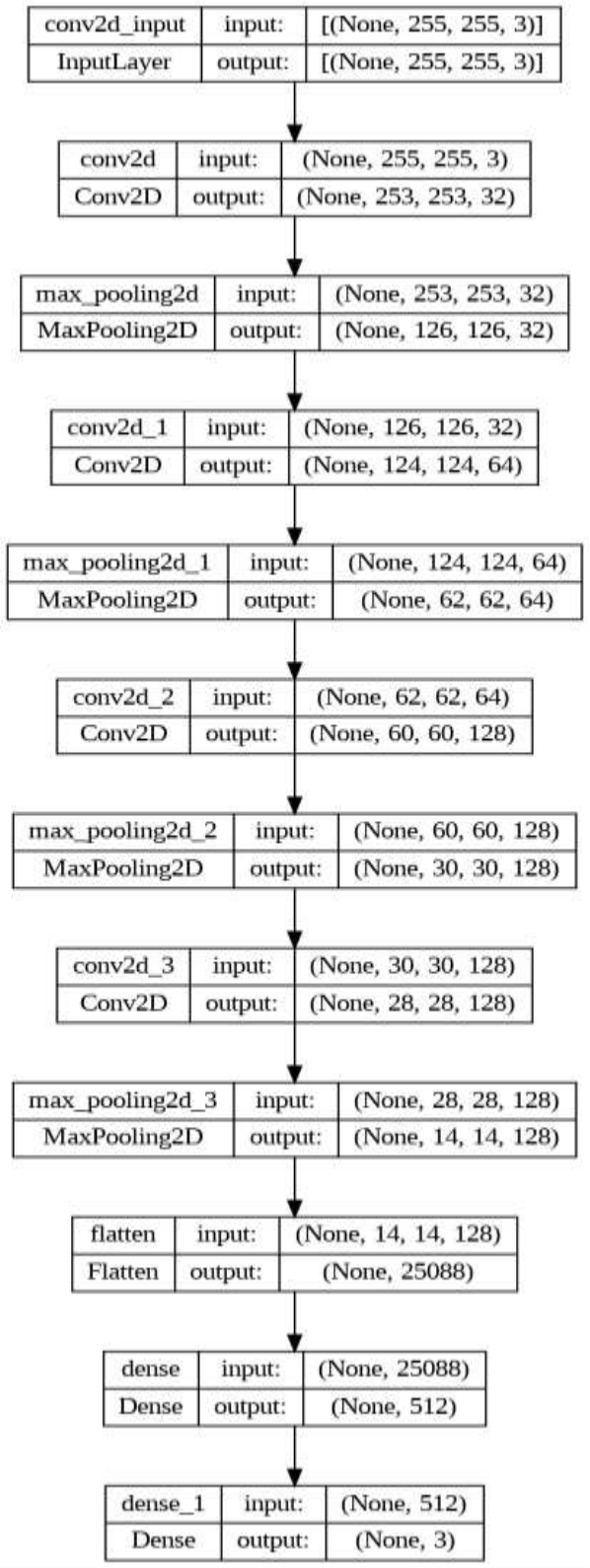
Procedure:

- **Preprocessing:**
  - Will be using Google Colab , for its Hardware acceleration resources.
  - Choose V100 as a gpu for Runtime Type, this will help in increased hardware acceleration.
  - Connect the google drive, and download and unzip the dataset in file location /content/data.
  - Define the Train and Test dataset paths :
    - train_path = '/content/data/train'
    - test_path = '/content/data/test'
  - Create **DirectoryIterator** variables of Train Dataset and Test Dataset, also employing data augmentation techniques for training dataset, ensuring better training.
  - The classes and their labels are:
    - `{'Smoke': 0, 'fire': 1, 'non fire': 2}`

- **Section 1:**
  - Design the architecture of CNN Keras model.
  - Compile with optimiser as 'adam' and using **'Accuracy'** as metric.
  - Save the model checkpoint this will ensure if there is any runtime error our trained weights are not lost, and employ the use of EarlyStopping monitoring the **'val loss'**.
  - Fit the data and visualize the
    - Loss
    - Accuracy
    - Validation Loss
    - Validation Accuracy
  - Save the model in .h5 and potrubuf format ensuring maximum support in future use cases.
- **Section 2:**
  - Loading the keras model from .h5 format file.
  - Creating Train and Test IterativeDirectories with keeping the **'shuffle=False'.**
  - Predict classes of Train and Test datasets, and name it as train_predictions and test_predictions respectively.
  - Plot confusions matrices of the true and predicted labels.
  - Create metrics report and tabulate the results, containing:
    - Accuracy
    - Precision
    - Recall
    - F1 Score
- **Model Testing:**
  - Testing the outcomes by giving new images apart from our datasets.

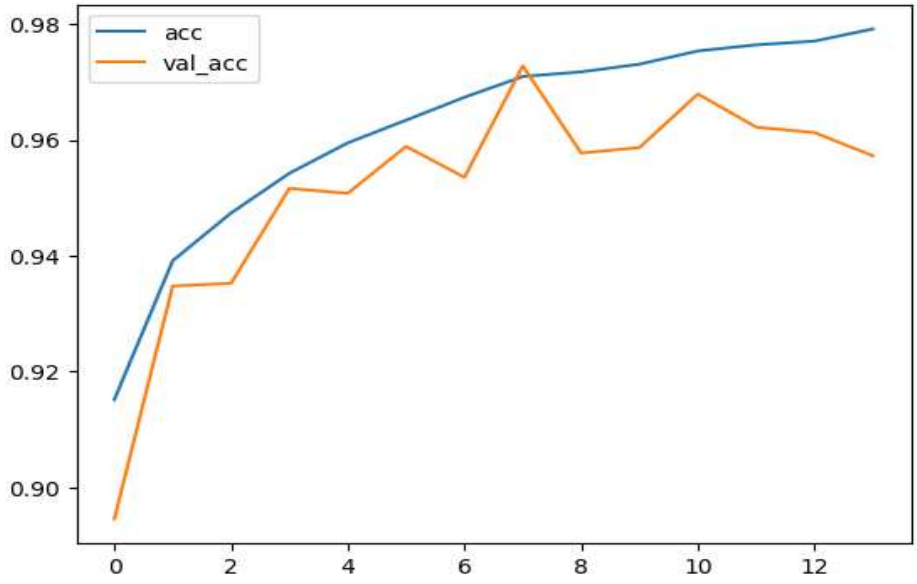
Outcomes:

- Keras Model: below figure describes the architecture of the keras model employed for model training.



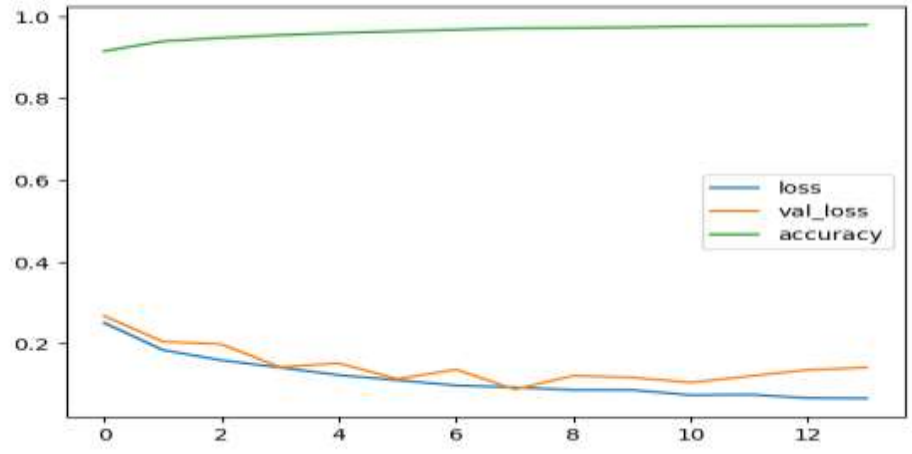
- **Metrics Graphs:**

Accuracy vs Validation Accuracy:



From the above graph we can notice the curve of accuracy and val_accuracy are showing a positive slope which is desired, suggesting the accuracy is increasing with more epochs.

LOSS VS VALIDATION LOSS:



From the above graph we can notice the val_loss and loss curves shows a negative slope suggesting that the Category Cross Entropy Loss asscosciated with the classification is decreasing with every epoch.

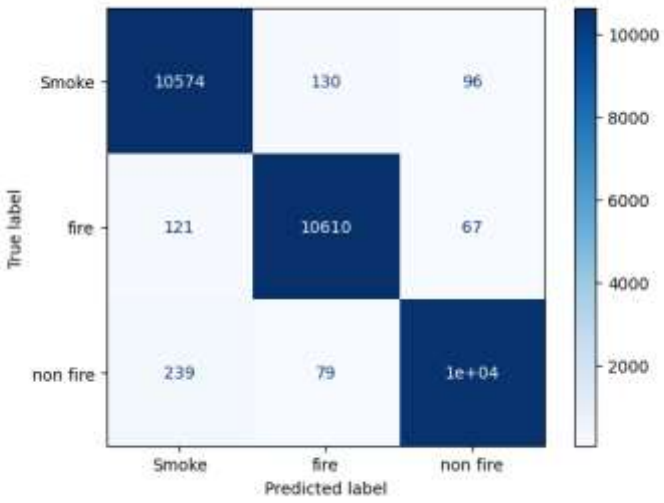
- As the epochs progress, both training accuracy and validation accuracy generally increase, indicating that the model is learning and improving its performance over time.
- The loss values (both training loss and validation loss) decrease as the epochs progress, which indicates that the model is converging towards a better fit to the training data.
- There are fluctuations in the accuracy and loss values from epoch to epoch, which is typical during the training process. These fluctuations might be due to factors like model complexity, learning rate, and dataset characteristics.

Overall, the trend suggests that the model is learning effectively, as both training and validation accuracy are increasing while training and validation loss are decreasing, which is desirable behavior during the training of a neural network.

- **Metrics:**

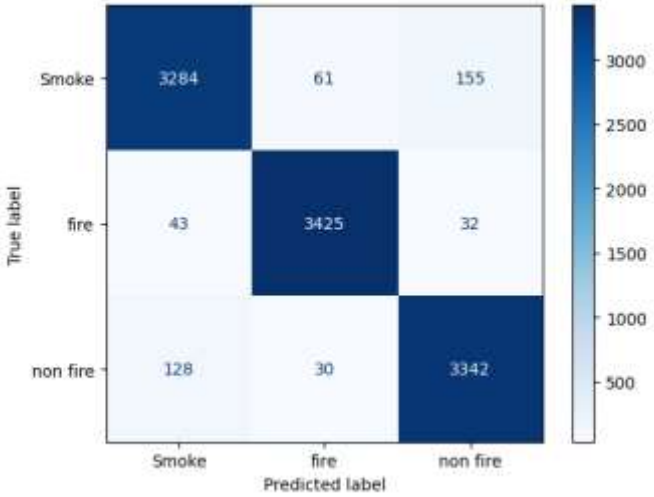
In this section we will try and understand the metrics of our keras model with our Training and Testing dataset.

Confusion Matrix (Train Dataset:



The confusion matrix suggests intensity of True Positive values is considerably more compared to than any other errors.

Confusion Matrix (Test Dataset):



The confusion matrix suggests that intensity of correct predictions is considerably more compared to errors.

Standard Metrics:

Dataset	accuracy	precision	recall	f1_score
Training Metrics	0.977406	0.977482	0.977406	0.977414
Test Metrics	0.957238	0.957212	0.957238	0.957205

Observations: The metrics percentage is more 95% in all the standards metrics for our test dataset , suggesting that the model is very accurate in classifying the images.

Deployment:

Objective:

Hosting flask server with the help of ngrok to host an api of the **Keras Model.**

Procedure:

Step 1:

Creating a flask server with /predict page in our homepage , which will handle POST and GET requests of our API.

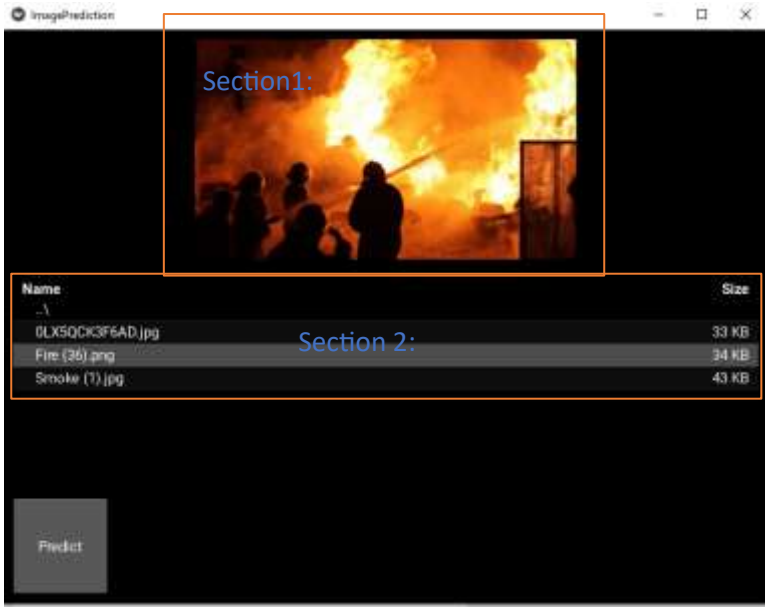
Step 2:

The Ngrok API url will be saved in the url.txt file once the flask server file is executed. Note : It is required for the flask server to always run for API to work properly.

Step 3:

Creating a Kivy App that will work as our Front End for requesting classification label of the image from our Ngrok API.

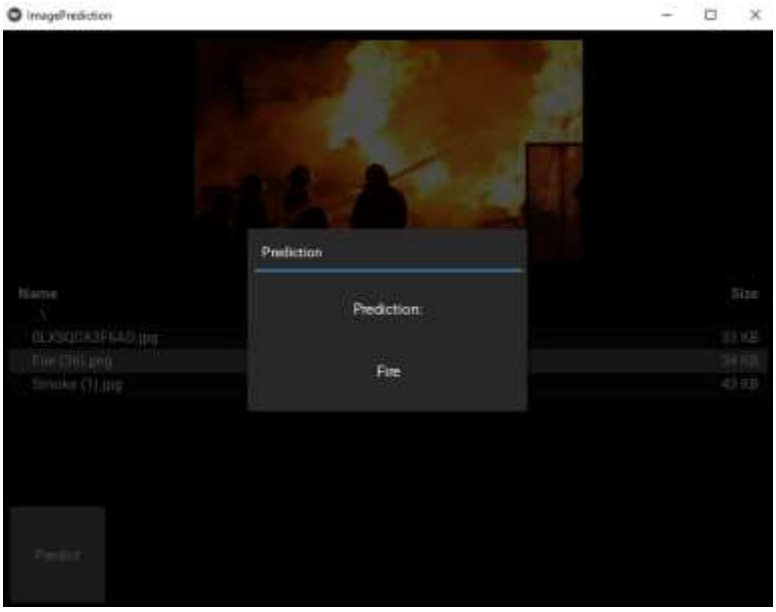
Structure:



Section 1: Gives a preview of the selected image.

Section 2: Shows the available files in our local disk.

Predict : Clicking on it will invoke the API and will fetch the class.



Prediction Pop up:

A prediction pop up will show up , showing the classification of image.

Resources:

GitHub Repository: https://github.com/mohammad1774/project-forest-fire

Dataset : https://kh3-ls-storage.s3.us-east-1.amazonaws.com/Updated%20Project%20guide%20data%20set/FOREST_FIRE_SMOKE_AND_NON_FIRE_DATASET.zip

File structure:

MyDrive: It is a copy of the drive folder from Google Colab notebook, containing training logs and trained keras models.

Graphs&tables: it contains all the images of the graphs and tables visualized in our training and observations.

Python-files: This folder contains our flask_server.py and main.py which are flask server and kivy app python files respectively.

Conclusion:

The capstone project aimed to develop a fire detection system with the objective of identifying fires from images obtained from surveillance systems or other sources. Through the project, a comprehensive solution was created which included the development of a Kivy app integrated with a Keras trained model, as well as the deployment of a Flask server to provide an API for fire detection.

Key Achievements:

1. Model Development: A robust fire detection model was developed using Keras, a high-level neural networks API. The model was trained on a dataset of fire and non-fire images to accurately classify images as containing fire or not.
2. Kivy App Development: A user-friendly Kivy app was created to provide an interface for users to upload images and receive real-time fire detection predictions. The app utilized the trained Keras model to perform inference on uploaded images.
3. Flask API Deployment: A Flask server was deployed to provide an API endpoint for fire detection. This allowed integration with other systems or applications, enabling seamless access to the fire detection functionality.
4. Integration of Components: The Kivy app and Flask server were successfully integrated, providing a complete end-to-end solution for fire detection. Users could interact with the app to upload images, which were then processed by the Flask API to provide fire detection predictions.
5. Scalability and Accessibility: The solution was designed to be scalable and accessible, allowing it to be deployed on various platforms and accessed from different devices. This ensured that the fire detection system could be utilized effectively in diverse environments.

Future Directions:

1. Enhancing Model Performance: Continuously improving the accuracy and efficiency of the fire detection model through ongoing training with larger and more diverse datasets, and exploring advanced techniques such as transfer learning.
2. User Interface Refinement: Iteratively refining the user interface of the Kivy app to enhance usability and user experience, incorporating user feedback and usability testing.
3. Deployment Optimization: Optimizing the deployment of the Flask server for improved performance and scalability, such as utilizing containerization technologies like Docker and deploying on cloud platforms for increased reliability and flexibility.

4. Integration with Surveillance Systems: Integrating the fire detection system with existing surveillance systems to provide real-time monitoring and early detection of fires in various environments, such as industrial facilities, forests, and urban areas.

In conclusion, the capstone project successfully achieved its objective of creating a fire detection system, laying the foundation for further advancements and applications in fire prevention and safety.