Report of Fire detection using Machine learning algorithm

Aerospec project

Tridev Sudhakar
ASU ID:1222911898
Email: tsudhak2@asu.edu

Satwik Vivek Naik
ASU ID: 1219610457
Email: svivekna@asu.edu

Sree Pavan Goli
ASU ID: 1219385843
Email: sgoli6@asu.edu

Sulguti
ASU ID :1223482325
Email: ssulguti@asu.edu

Abstract:

A fire is a burning object that emits intense light, heat, and smoke. Fire can be hazardous and devastating at times, resulting in the death or destruction of people and property. Industrial fires are a common occurrence when machining operations like cutting, drilling, and milling are involved. A small fire in the scrap section or a short circuit in any of the equipment can lead to a catastrophic event if not taken care of. It is critical to notice such fires at the appropriate moment and to act quickly. As a result, we developed a system that detects when a fire starts and sends a signal to the user to alert them because of which the fire can then be put out and the necessary precautions are taken to prevent further harm. This is accomplished through the use of computer vision to detect fire. This system avoids false alarms by omitting flame and smoke sensors. We offer a fire detection solution in this work that is achieved through a Machine Learning algorithm using OpenCV and TensorFlow, a Raspberry Pi, and a Logitech C920e camera to detect fire and issue a warning.

Introduction:

In an industrial environment where there are very high rates of machining at high temperatures, there is a vulnerability to fire accidents. Continuous human monitoring is difficult in scenarios where the equipment is comprised of thousands of parts and so, it is very high time to involve computer supervision in flame detection. There are many ways to detect the flames in the industry using electronic gadgets, but they delay their approach of triggering a signal to

the operator which elapses the operation of fire extinguishment. For all these reasons it is necessary for an engineer to find a feasible solution that can solve each of them. Computer vision and machine learning is taking over the world in recent times. From surgical operations to home appliances computer vision has strong applications and it is fast growing. Machine learning, on the other hand, is also vastly used and had niche applications where it predicts what may happen. Furthermore, light-scale versions of these models are currently being rolled out and this is enabling users to deploy computer vision and machine learning on small-scale devices like mobile phones, Arduino, and Raspberry Pi. There are many ways of implementing a machine-learning algorithm to detect fire. Depending on the scale and temperature of the fire there are many ways of doing it. Amongst them, the most optimal way is to use the thermal camera for detecting fire through variations in temperature. But it is seen to have shortcomings in certain applications where the thermal frame has a restricted field of view or pixel ratio. For finding an optimal solution for fire detection in a region as explained previously we have followed the procedure mentioned in this paper wherein we have used Raspberry Pi, OpenCV, TensorFlow, and a Video Camera to tackle this issue.

Components and Setup:

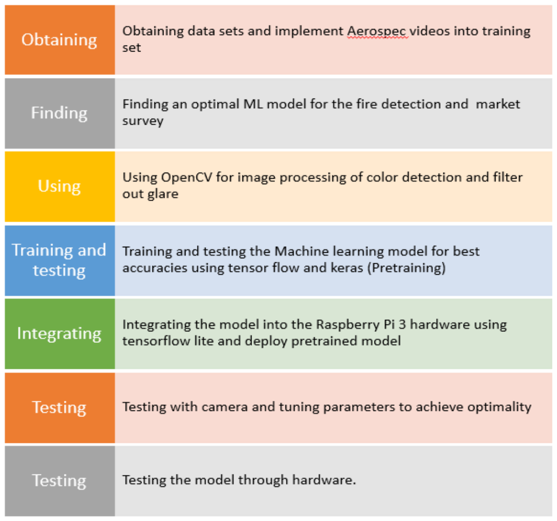
1) Raspberry Pi 3:
Specifications
- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU.
- 1GB RAM.
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board.
- 100 Base Ethernet.
- 40-pin extended GPIO.
- 4 USB 2 ports.
- 4 Pole stereo output and composite video port.

Fig. 1 Raspberry pi

2) Logitech C920e:
Specifications
- Image resolution: up to 15 Megapixels.
- Video Resolution: 1080p at 30 fps / 720p at 30 fps.
- Still Image Resolution: 1920x1080.
- Diagonal Field of View (FOV): 78°

Fig.2 Logitech camera

Methodology:



Obtaining	Obtaining data sets and implement Aerospec videos into training set
Finding	Finding an optimal ML model for the fire detection and market survey
Using	Using OpenCV for image processing of color detection and filter out glare
Training and testing	Training and testing the Machine learning model for best accuracies using tensor flow and keras (Pretraining)
Integrating	Integrating the model into the Raspberry Pi 3 hardware using tensorflow lite and deploy pretrained model
Testing	Testing with camera and tuning parameters to achieve optimality
Testing	Testing the model through hardware.

Table.1 Overview of methodology

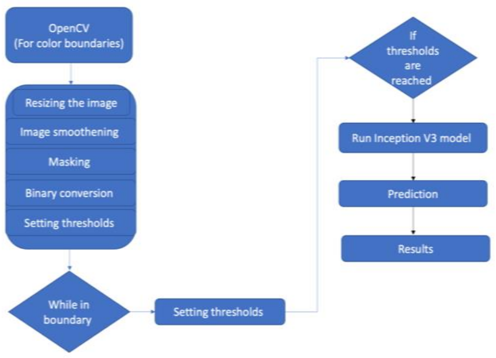


Fig3. Illustration of the model.

In the model of fire detection first Open CV is used to find the color boundaries and then they are specified. Then the images fed is resized before passing them to masking and converting them into binary. Then after getting the binary values the thresholds are set like in our case it is set to have 5000 white pixels. Then if the thresholds are reached by the image binary values inception v3 model runs. This model helps in predicting whether the fire is detected or not using the classification values. Then the results of fire or no fire is shown.

Open CV:

OpenCV is an excellent tool for image processing and computer vision. It's an open-source library for tasks including face detection, objection tracking, landmark detection, and more. Python, Java, and C++ are among the languages supported. Since we use python for everything in the project, we imported the OpenCV module in python.

Image Segmentation:

Image Segmentation is a prerequisite for more complex Computer Vision tasks like Object Classification and Object Detection, which use notions like contours and bounding boxes which are achieved by OpenCV.

Tensorflow:

Tensorflow API keras is the heart of our implemented model. Fire datasets are taken specifically for our project and the data is labeled using the Image data generator. Later, the datasets are used for training. Data augmentation is also used to get the various versions of an image in the dataset. The data augmentation methods we have utilized modifies an image by flipping it horizontally, rotating, and shifting its height.

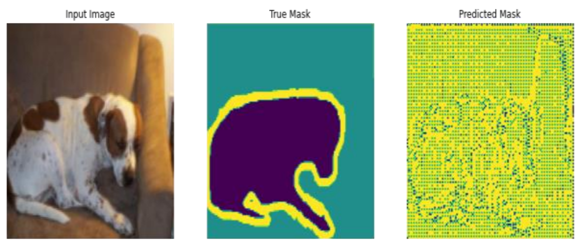


Fig.4 Illustration of image segmentation and applying masks.

Convolution Neural Network model:

The CNN model is initialized. It contains three sets of a convolution layer (Conv2D) and maxpooling layer (MaxPooling2D). These layers are followed by three dense layers. Dropout layers are added to avoid overfitting of the model. First two dense layers use ReLu activation, and third layer uses SoftMax activation.

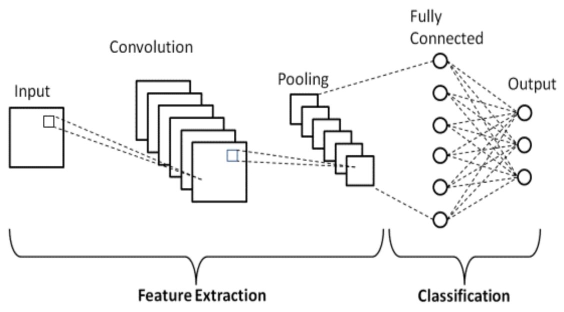


Fig.5 Illustration of Convolution neural network working.

Adam optimizer:

Adam optimizer is used with a learning rate of 0.0001. The Loss function is taken to be categorical cross-entropy. Training for 50 epochs we get the training accuracy of 95% and validation accuracy of 85%. Training loss and validation loss are respectively.

Inception V3 model:

Different datasets are used in this model. They contain both outdoor and indoor fire images. Customized image data generator is required to use the following datasets because the previous CNN model tend to succumb to overfitting. Some noise is added to the dataset.

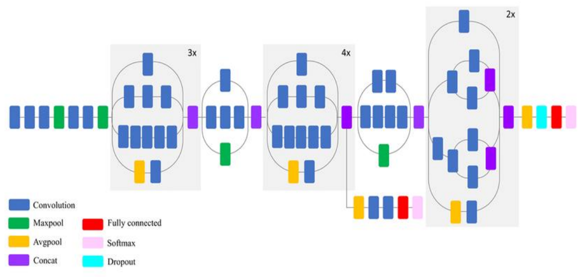


Fig. 6 Illustration of a general Inception V3 model.

Data Augmentation:

In this model, different data augmentation methods are used namely horizontal flipping and zooming. The Inception V3 model is imported from the keras API. The Global spacial average pooling layer is used followed by two dense and two dropout layers to avoid overfitting of the model. At last, we will add a

SoftMax activated dense layer for 2 classes. Using RMSprop optimizer the model is initiated.

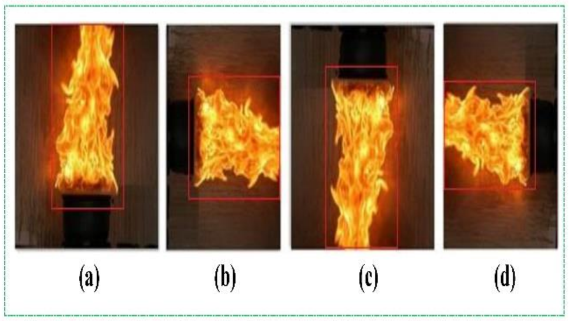


Fig.7 Illustration of fire image data augmentation.

Stochastic Gradient Descent:

Stochastic gradient descent optimizer is used with learning rate set to 0.0001. Momentum is set to 0.9 and the lost function used is categorical cross-entropy. As usual, the metrics calculated are accurate. A callback function is used to let us know when the model reaches a certain threshold accuracy or validation loss. The threshold validation loss is set as 0.1.

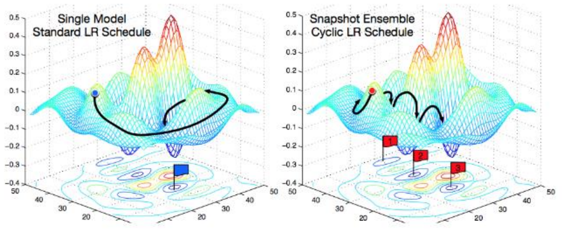


Fig.8 Illustration of a general gradient descent performed on data.

Next the model is fitted with the dataset number of epochs Isa to 10 with steps per epoch set to 14. Validation steps are set to 14. The results and logs are stored in a h5 file.

Results:

Color Separation and Detection:

Color separation is achieved using OpenCV. The video is split into frames. The frame obtained is then resized and mask is applied to filter out the image and detect only fire. Reflections are caught as false positives and can be tuned and filtered based on setup location. The frame is converted into binary (black and white). The white pixels are counted to determine chance of fire. This process can be seen in the video below.

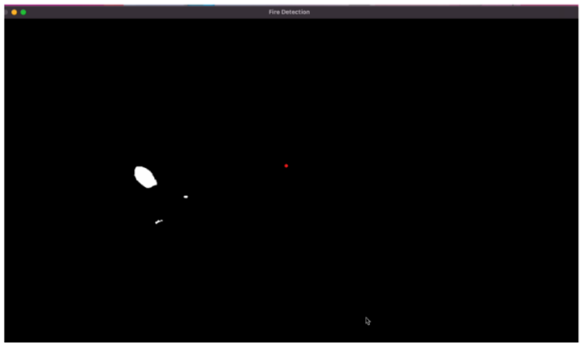


Fig.9 Illustration of color separation and detection of flame of a lighter.

ML Training and testing:

A dataset containing fire and non-fire images are taken and trained using Inception V3 model in. tflite format. Once trained the program is run on the test set and real time video feed. The fire was detected with an accuracy of 90% when run on the test set and real-time feed. False-positive values had a low prediction value at around 55% and can be eliminated using threshold marking.

Fire Detection:

Fire is detected accurately as shown in the video. This model is run on a regular laptop and image processing is quick due to the processing power of the CPU. The average accuracy for this case is around 93%. (Please notice the terminal window in the video).

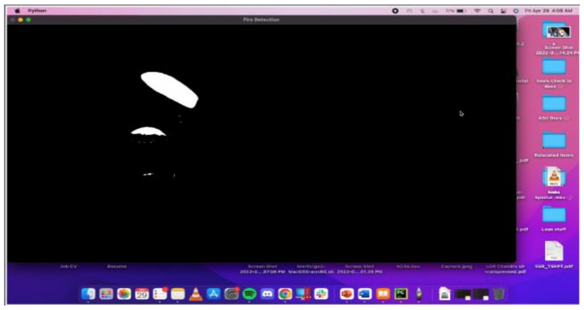
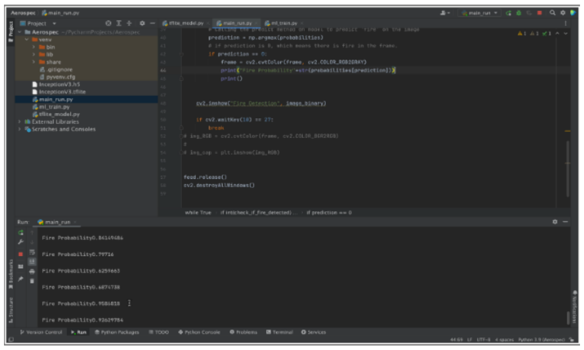


Fig.10 Illustration of fire detection of the lighter flame.



```
        prediction = np.argmax(probabilities)
        # if prediction is 0, which means there is fire in the frame.
        if prediction == 0:
            frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
            print("Fire Probability"+str(probabilities[prediction]))
            print()


    cv2.imshow("Fire Detection", image_binary)

    if cv2.waitKey(10) == 27:
        break
# img_RGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
#
# img_cop = plt.imshow(img_RGB)


feed.release()
cv2.destroyAllWindows()
```

```
Fire Probability0.86149686

Fire Probability0.79716

Fire Probability0.6259663

Fire Probability0.6874738

Fire Probability0.9586818

Fire Probability0.92629784
```

Fig.11 Illustration of program detecting the flame of the lighter.

Raspberry pi implementation:

The model setup containing both the Raspberry Pi and Logitech camera is shown below. The camera is connected to a USB 3.0 port for faster data transfer.

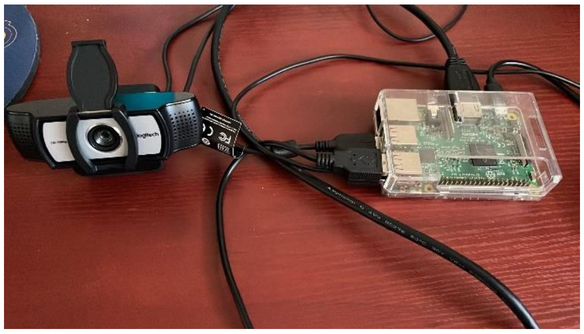
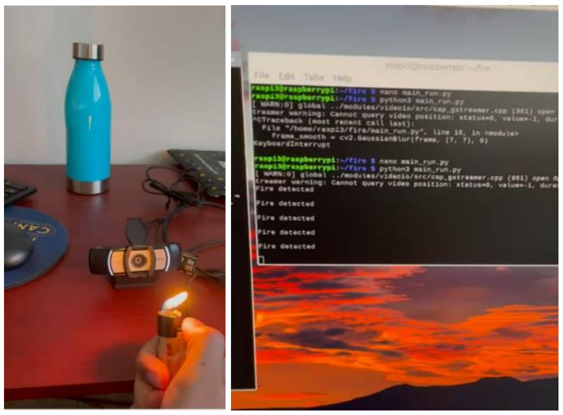


Fig.12 illustration of model setup

The program [1] is uploaded onto Raspberry pi and code modified to cater to raspberry pi functions. The whole setup is connected to a monitor to see the results on the serial monitor of Raspberry pi. The testing of the setup is showcased in the video below. The image processing seems a little slow and this is because of the raspberry pi's processing power. With a 4GB ram, this issue can be resolved.



```
raspi3@raspberrypi:~/fire $ nano main_run.py
raspi3@raspberrypi:~/fire $ python3 main_run.py
[ WARN:0] global ../modules/videoio/src/cap_gstreamer.cpp (961) open
treamer warning: Cannot query video position: status=0, value=-1, dur
^CTraceback (most recent call last):
  File "/home/raspi3/fire/main_run.py", line 16, in <module>
    frame_smooth = cv2.GaussianBlur(frame, (7, 7), 0)
KeyboardInterrupt

raspi3@raspberrypi:~/fire $ nano main_run.py
raspi3@raspberrypi:~/fire $ python3 main_run.py
[ WARN:0] global ../modules/videoio/src/cap_gstreamer.cpp (961) open O
treamer warning: Cannot query video position: status=0, value=-1, dura
Fire detected

Fire detected

Fire detected

Fire detected

Fire detected
```

Fig.13 Raspberry pi predicting and detecting the flame of lighter.

Conclusion:

An effective and error-proof system is designed for fire detection. Successful implementation of the program into Raspberry Pi is achieved. Integration of Raspberry Pi and Logitech camera is implemented resulting in seamless transfer of data between the two components. The video fed into the raspberry pi is accurately analyzed by the program and conversion is performed on it. The ML predicts fire with an accuracy of 90% when a flame is detected. This accuracy is maintained regardless of the lighting condition.

Future Work:

A Thermal camera has always more optimality in increasing the accuracy of the prediction model. Hence, in the future, a thermal camera can be used instead of the Logitech camera. Once the proposed model is finalized, a monitoring facility can be added like giving voice feedback (speaker) or visual feedback (LED). Ensemble learning can be implemented to improve accuracy. Three or more algorithms can be used simultaneously and determine if a fire is detected based on votes. This will however require a more powerful processor than raspberry pi (version 3) which is slow and cannot handle large amounts of data at a particular instance.

Appendix:

[1] Main – The Program that is fed into the Raspberry Pi is inserted below:

```
import cv2 as cv2
import tflite_runtime.interpreter as tflite
import numpy as np
from PIL import Image

feed = cv2.VideoCapture(0)
lower_bound = np.array([25, 55, 230])
upper_bound = np.array([255, 255, 255])
model = tflite.Interpreter('InceptionV3.tflite')
model.allocate_tensors()

while True:
    ret, frame = feed.read()
    frame = cv2.resize(frame, (1280, 720))
    frame_smooth = cv2.GaussianBlur(frame, (7, 7), 0)
    mask = np.zeros_like(frame)
    mask [0:720, 0:1280] = [255, 255, 255]
    img_roi = cv2.bitwise_and (frame_smooth, mask)
    frame_hsv = cv2.cvtColor(img_roi, cv2.COLOR_BGR2HSV)
    image_binary = cv2.inRange(frame_hsv, lower_bound, upper_bound)

    check_if_fire_detected = cv2.countNonZero(image_binary)
    threshold = 5000
    if int(check_if_fire_detected) >= threshold:

        input_details = model.get_input_details()
        output_details = model.get_output_details()
        im = Image.fromarray(frame, 'RGB')
        im = im.resize((224, 224))
        img_array = np.array(im)
        img_array = np.expand_dims(img_array, axis=0) / 255
        img_array = np.float32(img_array)

        model.set_tensor(input_details[0]['index'], img_array)
        model.invoke()

        probabilities = model.get_tensor(output_details[0]['index'])
        prediction = np.argmax(probabilities)
        if prediction == 0:
            frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
            print("Fire detected")
            print()
    cv2.imshow("Fire Detection", image_binary)

    if cv2.waitKey(10) == 27:
        break

feed.release()
cv2.destroyAllWindows()
```