**FINAL PROJECT REPORT**

**USING MACHINE LEARNING ALGORITHMS FOR FIRE DETECTION IN IMAGES**

**Student Name:** Antony Garcia

**WPI Student ID:** 901004615

**WPI Email:** agarcia3@wpi.edu

# Introduction

For my final project I am proposing an algorithm to detect fire on images. The main reason for the development of this algorithm is to use it in a forest fire detection equipment based on the Internet of Things paradigm.

The idea would be to build equipment equipped with cameras to detect fire or smoke through image processing. The Machine learning algorithm will be used to classify the images and determine if there is a presence of fire or not.



Figure 1. A concept image of the proposed solution installed in the middle of a forest.

The proposed solution for the detection of wildfires is the development and deployment of a network of monitoring stations equipped with cameras and machine learning algorithms. These stations will be installed in strategic locations within forests and will continuously scan the surrounding area for signs of fire.

The monitoring stations will be powered by a combination of solar panels and batteries, which allows them to operate independently of external power sources. They will be designed to be easy to install and deploy, with a single pole that supports all the system components. The system is also designed to be energy efficient and reliable, with a focus on long-term sustainability.

Diagram

Description automatically generated

Figure 2. General concept and components of the proposed solution

To improve the accuracy of the fire detection algorithm, the cameras used in the monitoring stations will be equipped with IR filters. These filters will allow the cameras to more accurately detect the infrared radiation emitted by fires, which is a key indicator of their presence. By using cameras with IR filters, we can increase the chances of correctly detecting fires as soon as they start, enabling rapid response to prevent them from spreading. In addition to the IR filters, we will also take additional measures to ensure the reliability of the fire detection algorithm, such as training it on a large dataset of images of fires and non-fires.

The methodology for data acquisition in the proposed solution involves the continuous capture of images by the cameras at each monitoring station. These images are processed by the microcontroller using machine learning algorithms specifically trained to detect the presence of a wildfire.

In this document I am going to focus on describing some previous work that we have already done, specifically the development of a Machine Learning classification algorithm. For the design of this algorithm, there are some images with flames that were captured by some students as part of a final project for Dr. James Urban of the Fire Protection Engineering department of WPI.

The images available images look like this:

Imagen que contiene edificio, verde, ladrillo, cuarto

Descripción generada automáticamente

Figure 3. An example of images with burning objects

From these images I hope to build a training and testing dataset, which I can use to develop a Machine Learning model to determine if there is fire or not in a picture.

# Machine Learning classification model

## Building datasets

Back in November Dr. James Urban provided a collection of images of burning objects in controlled environments. There were 271 images available, in a format like that of **Figure *1***. For me the first task was to separate both images and keep the filtered images (left side). That part was achieved with a Python script.

After cropping images, now we have a resolution of 1920x1440. Using such a big image with a Machine Learning algorithm could imply a huge computational cost. That said, I decided to split images in smaller frames, specifically 256x256 images.

My idea would be to separate images in frames and then classify each frame as positive or negative for fire. Then, using MNIST dataset as reference, which is one of the standard Machine Learning datasets. My first ever experience with Machine Learning was with MNIST, so I decided to convert all images to pixels in a CSV file with a column with “ones” or “zeros” as labels for supervised training, just like MNIST.

To classify images and build datasets I developed a Java Application where you can select the frames for of each image as positive or negatives. Then, the program can separate images in “training” and “testing” images, make the cuts and save all frames in a folder.

Interfaz de usuario gráfica, Gráfico de rectángulos

Descripción generada automáticamente

The image above shows the Java interface where you can select frames with fire. This application can also perform data augmentation on the available frames, as there are so many negative frames compared to the positives. Data augmentation is done by flipping and rotating positive images. It can also detect black frames based on custom parameters and remove cells with certain number of black pixels.

Special emphasis was done in the development of this application, because it will make possible to built future datasets in a relatively short time. Data augmentation and resizing tools will make possible to try different Machine Learning algorithms that require specific size for image processing, like some neural-network-based algorithms.

After separating testing and training images, a Python scrip builds a testing and training CSV from the frames. I ended up having 1728 samples for testing and 5546 for training, with close to 50:50 positive and negative samples.

# Machine Learning algorithm

After building datasets, first algorithm I tried was Support Vector Machine. I used the same algorithm that I tested when working with MNIST during semester.

The first result I obtained was around 65% accurate. After the first test I decided to make some tweaks on the dataset building algorithm. At first, I created a pool of frames and then proceed to randomly separate them into training and testing frames. The problem with this approach is that it would be difficult to evaluate the misclassification frames in their original images. It was very difficult to determine which frame was part of a specific image.

I changed the separation algorithm to work with images rather than frames. I randomly select a group of images for testing and training and then cut the frames and build the datasets.

After changing the dataset building approach, I tested **Random Forest** algorithm and surprisingly I got ~97.5%. I also tested other algorithms like **Support Vector Machines** and **KNearestNeighbors**. The following table shows some of the obtained results:

The results of running a Random Forest classifier can be seen in the following image:

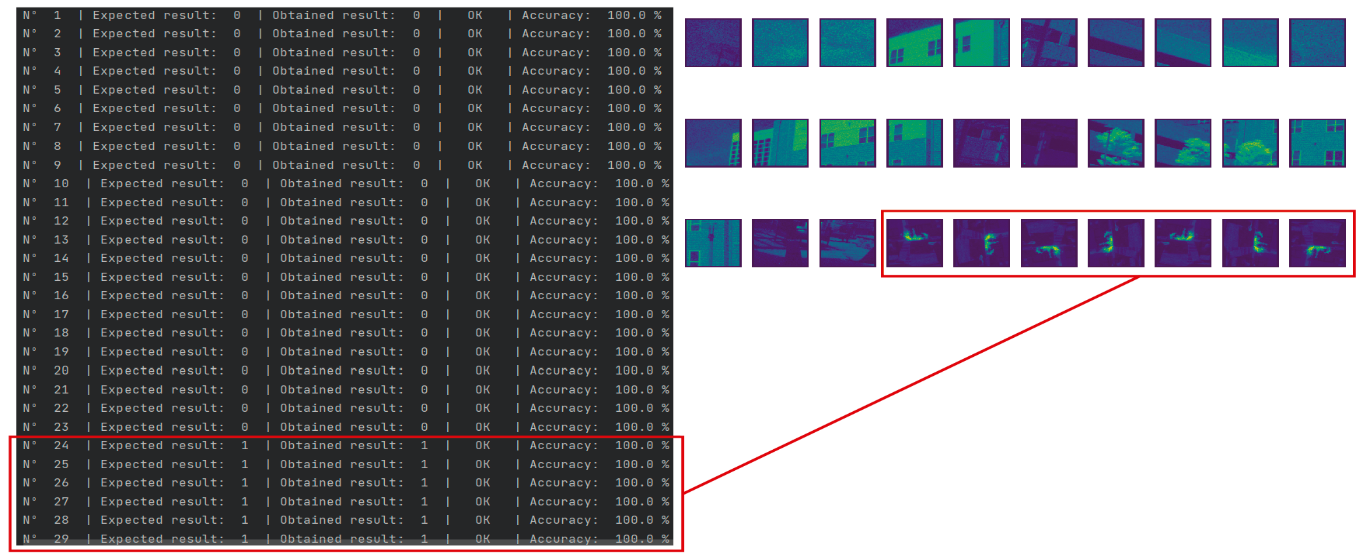
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Hyperparameters** | **Training samples** | **Testing samples** | **Accuracy** | **Execution time**  **in seconds (train / test)** |
| Random Forest Classifier | default | 5475 | 1728 | 97.68% | 113.53 / 0.43 |
| Support Vector Machine | default | 5475 | 1728 | 94.03% | 559.2 / 396.16 |
| KNEarestNeighbor | default | 5475 | 1728 | 78.07% | 8.96 / 18.88 |

These tests were performed in a computer with the following hardware characteristics:

* **Operative system:** Windows 10
* **RAM memory:** 32 GB
* **Processor:** Intel Core i7-7700HQ CPU @ 2.80 GHz. 4 physical cores and 8 logical cores

I was a little bit suspicious of this outcome, so I built a new dataset. The application I made for data separation creates a random dataset each time, so you will never get two equal datasets. The obtained results of the **Random Forest Classifier** were consistent with the previous outcome.

I also made some visual verification of the results by printing the outcome of each test and a grid of test images.



The best performance was with Random Forest Classifier, which has good accuracy and is fast to train and test.

Hyperparameter tuning can be used to further improve the accuracy of this algorithm, although it should be noted that the current accuracy measurement is for individual frames rather than entire images. When considering an image that consists of 48 frames, the probability of failure is reduced due to the numerous opportunities the algorithm has to correctly identify objects within the image captured by the camera.

# What’s Next?

Next step is to test the trained model with more images. We are planning to take more pictures of controller fires in a barbecue or something similar and test the model with images that it hasn’t seen before.

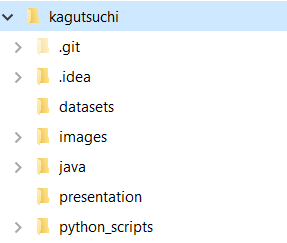
I spent a lot of time working on the dataset building tool, but now I can build training and testing dataset from images in a couple of minutes by just clicking at the frames with fire. I spent no more than 10 minutes selecting the 531 positive frames in the original dataset, which had 271 images.

After validation the model accuracy with further testing I am planning to run the algorithm from an embedded hardware device, like a Raspberry or a microcontroller. The idea is that for final implementation we can have a standalone prototype that does not depend on external sources to produce an accurate output.

# Results evaluation

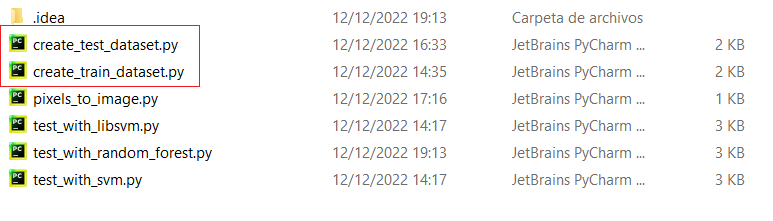
All files from my project can be found in the following Github repository: <https://github.com/AntonyGarcia/kagutsuchi>

This repository contains the following folder structure:



I did not include the datasets in this repository because it weights too much. The training CSV file is about 1.0 GB, and the testing file is around 300 MB. I included the testing and training images, thought.

To reproduce my results, you have to build CSV datasets from images, which can be done by running python scripts:



Each script will generate one dataset, which will be stored in the datasets folder.

After building datasets, the **test\_with\_random\_forest.py** (see image above) script can be run, and the results should be very similar to what I presented in this document.