**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.

For the design of the Machine Learning algorithm, there are some images with flames that were captured by some students as part of a final project of a professor who works in fire research here at WPI. These images include photos in RGB color scale contrasted with photos captured with an infrared filter on the camera lens.

The images look like this:

Imagen que contiene edificio, verde, ladrillo, cuarto

Descripción generada automáticamente

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.

# Building datasets

Original images are 1920x1440 pixels. 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, I can convert all images to pixels in a CSV file with a column with “ones” or “zeros” as labels for supervised training.

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.

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%.

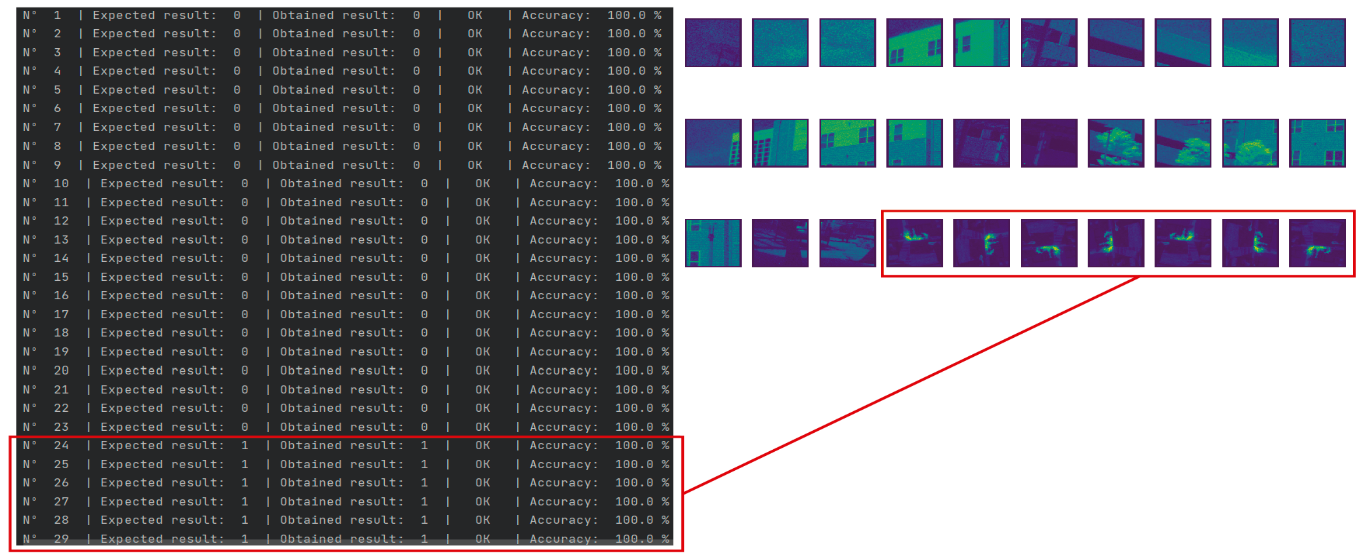
The results of running a Random Forest classifier can be seen in the following image:

Imagen que contiene computadora, teclado

Descripción generada automáticamente

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.



I also tested other algorithms like LibSVM, Support Vector Machines and Gaussian Process Classifier. The best performance was obtained with Support Vector Machines, but training and testing time was much higher than Random Forest.

# 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.

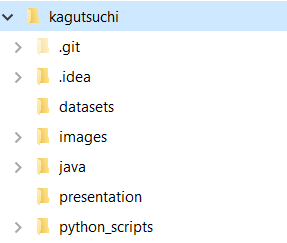
I spent a lot of time working on the dataset building tool, but now I can build training and testing dataset from images in a couple of minutes by just clicking at the frames with fire. I spent no more than 10 minutes selecting the 531 positive frames in the original dataset, which had 270 images.

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

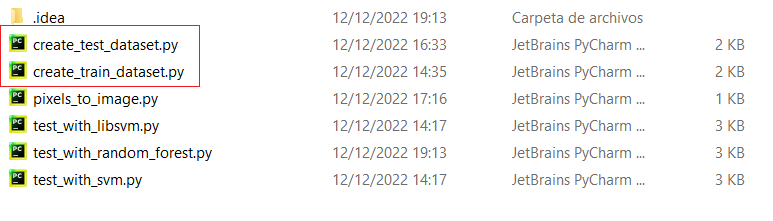
All files from my project can be found in the following Github repository: <https://github.com/AntonyGarcia/kagutsuchi>

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.