FOREST FIRE DETECTION FOR EMERGENCIES
Final Report

Hasan Malkoç 2200356826
Gizem Aleyna Tuzcu 2200356816

June 17, 2023

Abstract

Developing a computer vision system that can detect fires in real-time using cameras and sensors. The system can help firefighters respond quickly and efficiently to fires and minimize damage.

1 INTRODUCTION

1.1 Problem

The development of methods and strategies to automatically detect and identify instances of fire or flames in images or videos is the problem of fire detection in computer vision. To ensure prompt response and intervention in fire-related accidents and reduce the possibility of property damage, injuries, and fatalities.

1.2 Aim

The development of methods and strategies to automatically detect and identify instances of fire or flames in images or videos is the problem of fire detection in computer vision. To ensure prompt response and intervention in fire-related accidents and reduce the possibility of property damage, injuries, and fatalities.

1.3 Challenges

The complexity and dynamic nature of fire present a number of challenges for fire detection in computer vision. Among the principal difficulties are:

- **Fire Appearance Variation:** Fires can differ in terms of their size, shape, color, intensity, and occlusion, making it difficult to create algorithms that can properly handle these differences.
- **Background Clutter and Occlusion:** Fire frequently happens in crowded situations with a variety of objects, and occlusion can make the detection procedure even more challenging. Robust techniques are needed to handle occlusion and separate fire from other objects.
- **Limited training data:** Due to the rarity of fire events and potential safety issues, obtaining a big and diverse data set for training fire detection algorithms might be difficult. The models' capacity to generalize may be impacted by the lack of sufficient training data.

1.4 Motivation

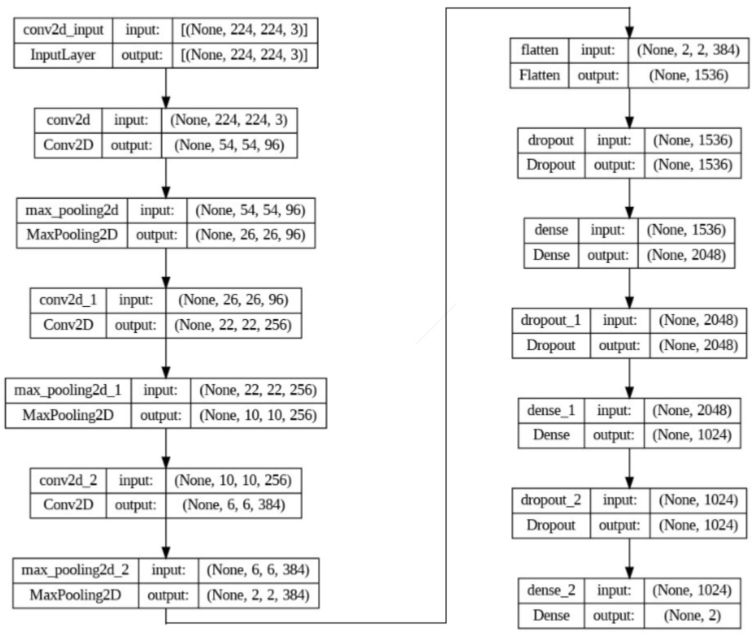
The necessity to improve public safety, safeguard property, and stop fire-related tragedies is what drives the development of fire detection in computer vision. It can activate quick-response systems that can inform authorities, sound alarms, start fire suppression systems, and evacuate individuals.

1.5 Relation to the Theme

For early warning systems, emergency response planning, and mitigation methods, fire detection in computer vision is crucial. Emergency management systems can swiftly identify fires, set off alarms, inform authorities, and coordinate actions by utilizing computer vision algorithms. This lessens the effects of fires, protects lives, and shields resources and infrastructure.

2 METHOD

1. **Prepocessing:** The preprocessing techniques specified in the code:
   - **Normalizing:** This technique rescales the pixel values of the images. Dividing by 255 is a common approach to normalize the pixel values, as it scales them to the range of 0-1. This normalization helps in stabilizing the training process and improves convergence.
   - **Rescaling:** The input images will be resized to a square shape with a height and width of 224 pixels. By resizing the images to a specific target size, we ensure that all images in the dataset have the same dimensions, which is necessary for training deep learning models.
   - **Horizontal Flip:** This technique horizontally flips the images randomly. It is useful for scenarios where horizontal symmetry is present, as it provides additional training data by creating mirror images of the original ones.
   - **Rotation Range:** This technique applies random rotations to the images within the specified range. In this case, the images can be rotated up to 30 degrees in either clockwise or counter-clockwise direction. Rotating the images helps in making the model more robust to variations in object orientations.
   - **Height Shift Range:** This technique randomly shifts the height (vertical position) of the images by a fraction of their total height. In this case, the height can be shifted up to 20
   - **Fill Mode:** This technique determines the strategy used to fill in the pixels that may be created due to transformations like rotation or shifting. In this case, the 'nearest' mode is used, which fills the new pixels with the value of the nearest existing pixel. This mode helps in preserving the structure and details of the original image.
2. **CNN Architecture:** A CNN model used for detecting fires. This model follows a typical convolutional neural network (CNN) architecture commonly used for image classification tasks. It consists of convolutional layers for feature extraction, pooling layers for down-sampling, and dense layers for classification. Dropout layers are included to reduce over-fitting, and ReLU activations introduce non-linearity to the network. The model aims to classify inputs into one of the two classes represented by the soft-max output layer.


conv2d_input | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
conv2d | input: | (None, 224, 224, 3)
Conv2D | output: | (None, 54, 54, 96)
max_pooling2d | input: | (None, 54, 54, 96)
MaxPooling2D | output: | (None, 26, 26, 96)
conv2d_1 | input: | (None, 26, 26, 96)
Conv2D | output: | (None, 22, 22, 256)
max_pooling2d_1 | input: | (None, 22, 22, 256)
MaxPooling2D | output: | (None, 10, 10, 256)
conv2d_2 | input: | (None, 10, 10, 256)
Conv2D | output: | (None, 6, 6, 384)
max_pooling2d_2 | input: | (None, 6, 6, 384)
MaxPooling2D | output: | (None, 2, 2, 384)
flatten | input: | (None, 2, 2, 384)
Flatten | output: | (None, 1536)
dropout | input: | (None, 1536)
Dropout | output: | (None, 1536)
dense | input: | (None, 1536)
Dense | output: | (None, 2048)
dropout_1 | input: | (None, 2048)
Dropout | output: | (None, 2048)
dense_1 | input: | (None, 2048)
Dense | output: | (None, 1024)
dropout_2 | input: | (None, 1024)
Dropout | output: | (None, 1024)
dense_2 | input: | (None, 1024)
Dense | output: | (None, 2)

3. **Training Data:** The provided code snippet uses the model.fit() function to train a model with data augmentation using a generator. The 'train-generator' provides batches of augmented training images, while the 'validation-generator' supplies batches of validation data. The model is trained for 50 epochs, with each epoch processing 15 steps (batches) of training data. After each epoch, the model's performance is evaluated on the validation data using 15 validation steps. The training history, including metrics such as loss and accuracy, is stored in the history object for further analysis and visualization. The result of the training is a boolean value to indicate wheter the image consists fire or not.

4. **Detecting Fire:** After training the data according to the CNN model, paths of images that consists fire are separated to fire detection. Then some steps are performed on these images. Steps that applied on the images consists:

   - **Gaussian Blur:** To reduce noise and smooth the image, a Gaussian blur is applied using the "cv2.GaussianBlur()" function. This helps in improving the accuracy of subsequent color-based operations.

   - **Color Space Conversion:** The image is converted from the BGR color space (the default color space for OpenCV) to the HSV (Hue, Saturation, Value) color space using "cv2.cvtColor()". HSV color space separates the color information from the image, making it easier to isolate specific colors, such as those associated with fire.

   - **Thresholding:** Lower and upper threshold values are defined in the HSV color space to determine the desired color range. This range represents the colors associated with fire. The lower and upper thresholds are specified using arrays, indicating the minimum and maximum values for hue, saturation, and value. These thresholds define the color range to be detected.

   - **Mask Application:** A mask is created by applying the defined lower and upper thresholds to the HSV image using "cv2.inRange()". The resulting mask contains white pixels for the colors falling within the specified range, representing potential fire regions. Pixels outside the range are set to black.

   - **Contour Detection:** Contours are extracted from the mask using "cv2.findContours()". Contours are continuous curves that enclose regions with similar colors. In this case, contours help identify the boundaries of potential fire regions in the image.

   - **Drawing Rectangles:** For each contour detected, a bounding rectangle is calculated using

”cv2.boundingRect()”. The bounding rectangle specifies the coordinates, width, and height of the detected region. These rectangles are drawn on the original image using ”cv2.rectangle()”, with a specified color (typically green) and thickness (2 pixels). This visualizes the detected fire regions by outlining them with rectangles.

By performing these steps, the code enhances the fire regions in the images, making them more distinguishable. This aids in fire detection and allows for further analysis or action to be taken based on the identified regions.

3 EXPERIMENTAL SETTINGS

3.1 Details of Dataset

In this study, 1,400 photos of flames were utilized as the dataset. The dataset was divided into three subgroups with percentages of 70

Environment used Google Colab Jupyter Notebook.

Libraries used are TensorFlow, OpenCV, Matplotlib, Numpy...

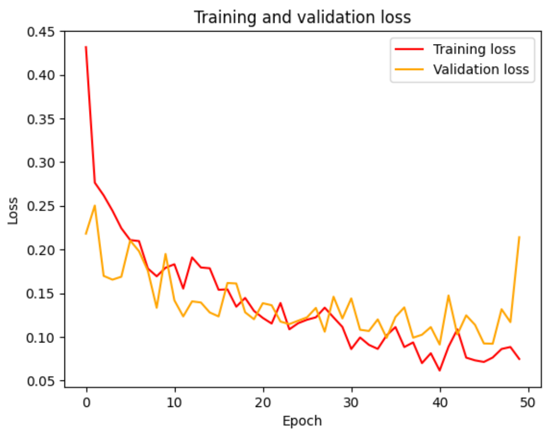
4 EXPERIMENTAL RESULTS

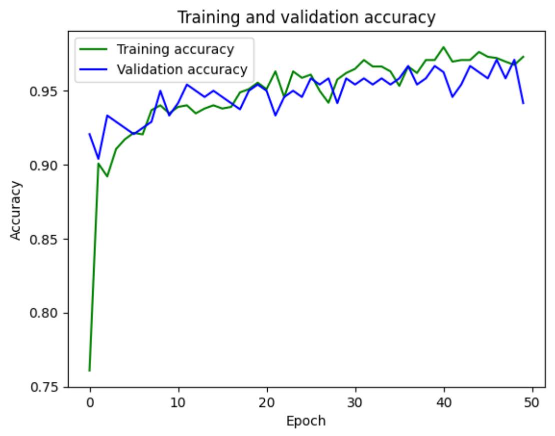
In this section, we present the experimental results of our CNN model trained to detect the presence of fire in images.

The model was trained using a dataset consisting of images labeled as either containing fire or not. We evaluated the model's performance using various metrics, including accuracy, loss, precision, recall, and F1-score.

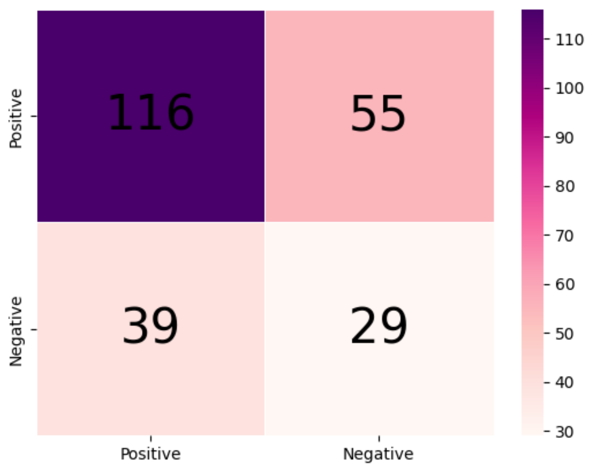
Additionally, we generated a confusion matrix to gain insights into the model's performance on different classes.

- **Training and Validation Metrics:** During the training process, we monitored the following metrics to assess the model's performance:





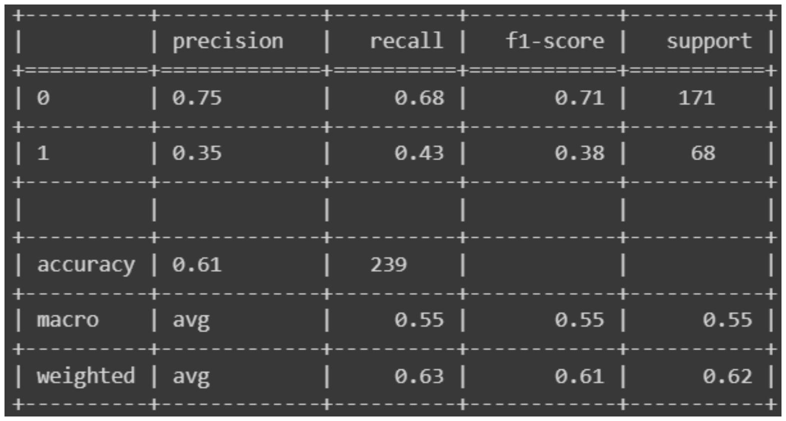
- **Confusion matrix:** The confusion matrix provides a detailed breakdown of the model's performance on the test set:



The model has a relatively higher number of false positives (55) compared to false negatives (39), indicating a tendency to incorrectly classify instances as positive ("Fire") when they are actually negative ("No Fire").

The true positive count (116) is higher than the true negative count (29), suggesting a higher sensitivity or ability to correctly identify positive instances ("Fire") compared to negative instances ("No Fire"). This may be the result of "No Fire" images are less numerous than "Fire" images.

- **Classification Report:** We computed precision, recall, F1-score, and support for both classes (fire and non-fire) to further evaluate the model's performance:



```
+----------+-------------+---------+-----------+----------+
|          | precision   |  recall |  f1-score |  support |
+==========+=============+=========+===========+==========+
| 0        | 0.75        |    0.68 |      0.71 |   171    |
+----------+-------------+---------+-----------+----------+
| 1        | 0.35        |    0.43 |      0.38 |    68    |
+----------+-------------+---------+-----------+----------+
|          |             |         |           |          |
+----------+-------------+---------+-----------+----------+
| accuracy | 0.61        |  239    |           |          |
+----------+-------------+---------+-----------+----------+
| macro    | avg         |    0.55 |      0.55 |     0.55 |
+----------+-------------+---------+-----------+----------+
| weighted | avg         |    0.63 |      0.61 |     0.62 |
+----------+-------------+---------+-----------+----------+
```

- Precision is a measure of how many of the predicted positive instances are actually positive, the precision values suggest that the model has a higher tendency to correctly identify "Fire" instances compared to "No Fire" instances.

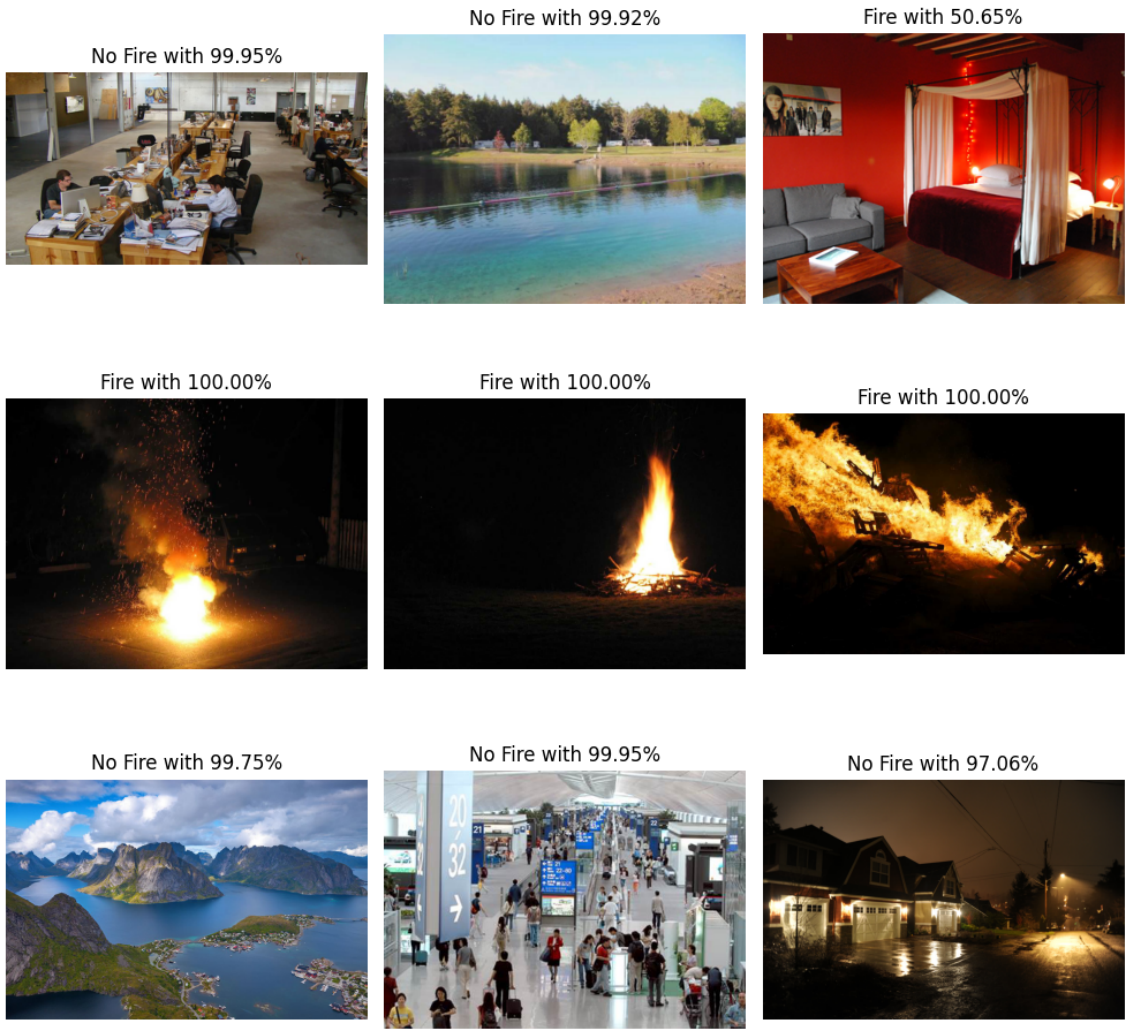
- Recall (also known as sensitivity or true positive rate) is a measure of how many of the actual positive instances are correctly identified, the recall values suggest that the model performs relatively better in identifying "Fire" instances compared to "No Fire" instances.

- The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both measures, the F1-scores indicate that the model's performance is relatively better for "Fire" compared to "No Fire".

- Accuracy measures the overall correctness of the model's predictions across all classes. The accuracy is 0.61, suggesting that the model correctly classifies approximately 61

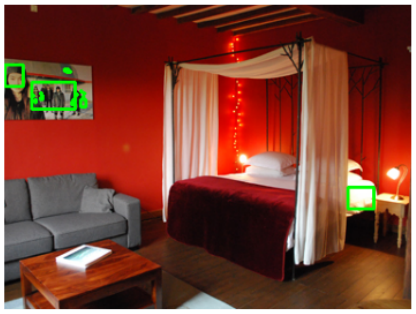
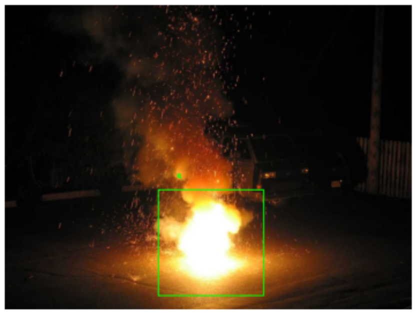
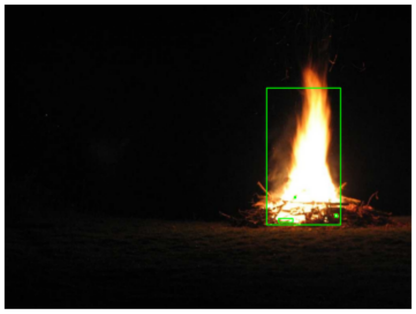
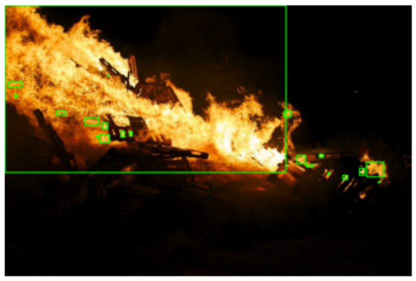
Overall, the model seems to perform better in terms of precision, recall, and F1-score for ”Fire” compared to ”No Fire”. The reason for this result can be the inequal distrubition of the images in the classes. (”No Fire” class has significantly less images than the ”Fire” class, almost the third of it.)

- **Model Persistence:** After training the CNN model, we saved it as a .h5 file for future use. This allows us to load the trained model and make predictions on new images efficiently.
- **Detection of Fire in Test Images** To evaluate the model’s ability to detect fire in real-world scenarios, we loaded the trained model and performed inference on a set of example images. For each image, the model predicted whether it contained fire or not. If the output was true (indicating the presence of fire), we saved the path of the image for later analysis. Some of the images that the model tried are below:



The misprediction of an image (third image above) labeled as fire could be attributed to its predominant red and orange colors, which may resemble the visual characteristics typically associated with fire in the model’s training data.

To visualize the detected fire regions, we applied image processing techniques to highlight the areas of interest. We then displayed the processed image with the bounding boxes and fire regions highlighted using Matplotlib.

5 DISCUSSION AND CONCLUSION

We successfully implementing a model architecture that can effectively detect fire in images.

Strongest part of the project can be the use of convolutional neural networks (CNNs) in the model architecture is a strong aspect. CNNs are particularly effective for image classification tasks due to their ability to capture spatial information and extract relevant features.

Weakest part of the project the performance of the model on class ”No Fire” (evident from the provided confusion matrix and metrics) appears to be relatively lower compared to class ”Fire”. This imbalance in performance indicates a weakness in correctly identifying instances from class ”No Fire”.

The overall accuracy of the model is at 61

Improvements for the result:

- Balancing the dataset or implementing techniques like oversampling or under-sampling for the minority class ”No Fire” could help improve the model’s performance on ”No Fire” instances.
- Fine-tuning the model architecture, hyperparameter tuning, or exploring alternative architectures might lead to improved results.
- Understanding the reasons behind misclassifications can provide insights for further improvements, such as collecting more diverse training data, enhancing data preprocessing, or adjusting the model architecture and parameters.

Sources:

- Dataset for images with fire: https://archive.ics.uci.edu/dataset/162/forest+fires
- Dataset for images without fire: https://www.kaggle.com/datasets/balraj98/stanford-background-dataset