Fire Image Detection Based on Convolutional Neural Network

Roshan Koirala
roshankoirala77@gmail.com

***Abstract*—In this work .....**
***Index Terms*—Deep Learning, Convolutional Neural Network, Image Processing, Fire Detection**

I. Introduction

Fire is very useful discovery of the humanity. Like any other human invention, mishandling of fire can cause huge damage to the humanity and nature. Every year, urban fire results huge life and property damage. Similarly the tragic loss of natural resources in uncontrolled wildfire is well known. The control of wildfire has becoming a huge challenge by using the traditional technologies. On the other hand, recently, the advancement in technology and especially the machine learning have benefited the society a lot in diverse aspects, ranging from self driving car to cancer research. Hence, it can surely contribute to the technology to early detect the fire so that we can react it earlier before getting it worse and making a lots of damage or being it out of control. We can train a deep neural network to distinguish fire and non-fire situation with very good accuracy. This technology aided with suitable hardware design can be vary useful to minimize the fire hazard. In this work we train a convolutional neural network which can achieve out of sample accuracy of 97% classifying the fire and non-fire images.

There are previous works in this direction

The organization of this work is the following:

II. Neural Networks

A. Artificial Neural Networks

Let $X = x_j^{(i)}$, $i = 1, 2, 3, \cdots, m$ and $j = 1, 2, 3, \cdots, n_X$ be the input matrix for an i^{th} observation where m is the total number of data in the sample and n_X is the size of each data. For example if we data 2000 images of 36×36 pixels then $m = 2000$ and $n_X = 36 \times 36 = 1296$. A neural network contains an input layer and a few hidden layer which takes input from output of the previous layers and finally there is a output layers. See FIG. 1. To find the output at the first layer we need to perform the calculation in two stages. The first one is the linear transformation.

$$Z^{[1]} = \Theta^{[1]} X + b^{[1]} \tag{1}$$

where Θ is the weight matrix to be learned through the training by the data. And the second one is the non-linear part

$$A^{[1]} = \sigma(Z^{[1]}) \tag{2}$$

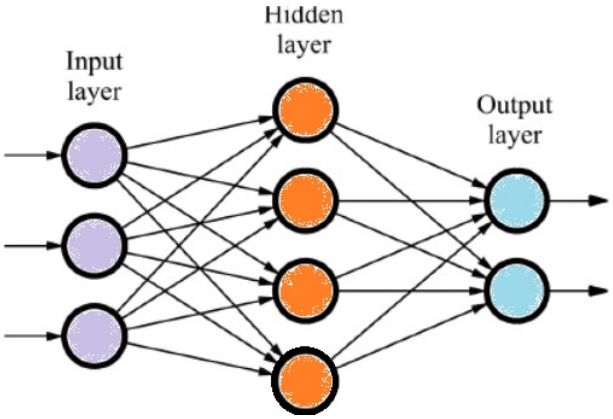


Fig. 1. An illustration of artificial neural network

where $\sigma(z)$ is called an activation function. Then the input at the second hidden layer is given by

$$Z^{[2]} = \Theta^{[2]} A^{[1]} + b^{[2]} \tag{3}$$

And the same process repeats in all the layers of the deep neural network. So in general we can write

$$Z^{[k]} = \Theta^{[k]} A^{[k-1]} + b[k] \tag{4}$$

$$A^{[k]} = \sigma(Z^{[k]}), \qquad A^{[0]} = X \tag{5}$$

where $k = 1, 2, \cdots n_L$ and n_L is the number of layers in the network. Usually the non-linear part is the Sigmoid function

$$g(z) = \frac{1}{1 + \exp(-z)} \tag{6}$$

the derivative of the Sigmoid function is

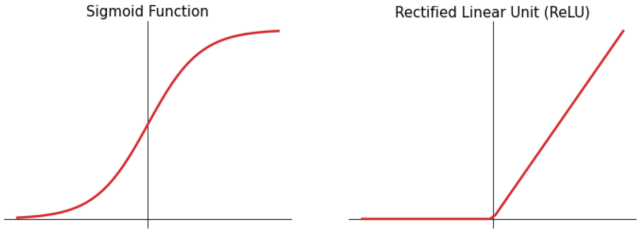
$$g'(z) = g(z)(1 - g(z)) \tag{7}$$

Another non-linear function is Rectified Linear Unit (ReLU)

$$f(z) = \max(0, z) \tag{8}$$

and the derivative of the ReLU is

$$f'(z) = \begin{cases} 0 & \text{for} \quad z \leqslant 0 \\ 1 & \text{for} \quad z > 0 \end{cases} \tag{9}$$



We start by assigning random weights and the correct weights will be gradually learned through the data using technique that is basically the gradient descent or its advanced variants. It is import to note that the weights should be initialized randomly but not all zero at the beginning otherwise the values cannot update. The objective is to minimize a cost function whose value is larger for miss classifying the data and lower gradually as we improve the model with classifying more and more data correctly. The cost function for a simple neural network is

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{nX}\Big[-y_k^{(i)}\log(h_\Theta(x^{(i)})_k) - (1-y_k^{(i)})\log(1-h_\Theta(x^{(i)})_k)\Big] \tag{10}$$

We need regularization to prevent the over fitting. One way to regularize a neural network is by adding penalty term in the cost function as given by

$$J_{\text{reg}}(\Theta) = \frac{\lambda}{2m}\left[\sum_{j=1}^{HL}\sum_{k=1}^{IL}\left(\Theta_{j,k}^{(1)}\right)^2 + \sum_{j=1}^{OL}\sum_{k=1}^{HL}\left(\Theta_{j,k}^{(2)}\right)^2\right] \tag{11}$$

So the total cost function is given by

$$J_{\text{tot}}(\Theta) = J(\Theta) + J_{\text{reg}}(\Theta) \tag{12}$$

Another method of regularization is by using drop out nodes. In this method we add a layer which randomly drops certain fraction of nodes from the layer in each iteration of the training. This prevents the network to memorize the data through the nodes.

B. Optimization & Back Propagation

Gradient descent is the key technique to minimize the cost function during the training process. In calculus gradient of a scalar function at a given point gives the direction of the maximum change of a scalar function. As our aim is to reach the lowest point of the cost function the gradient provides the right direction to the down hill by providing the incremental update to the weights and bias.

A neural network is a complicated one having to update weights and bias in each layers. The signal for the goodness of the model is achieved through the output layer where, for example, the difference of the predicted and actual output can be such signal. Then this information should pass back to through the input layer by updating the weights and bias in each successive layers. In literature this process is called back propagation. Mathematically, we do it by taking the chain rule of the functional in the following way

$$dZ^{[k-1]} = \begin{cases} A^{[k]} - y & \text{for} \quad k = n_L \\ W^{[k]\,T} dZ^{[k]} * \sigma'(Z^{[k-1]}) & \text{for} \quad k < n_L \end{cases} \tag{13}$$

$$\Theta^{[k]} \mapsto \Theta^{[k]} + d\Theta^{[k]} = \Theta^{[k]} + dZ^{[k]} A^{[k-1]\,T} \tag{14}$$

$$b^{[k]} \mapsto b^{[k]} + db^{[k]} = b^{[k]} + dZ^{[k]} \tag{15}$$

where the first equation is the gradient of the output of each layers. This is the step where chain rule have been implemented especially for $k < n_L$. The last two equations are the update on the weights and bias in each layers. The updated value is the original value plus a gradient.

C. Convolutional Neural Network

In mathematics the convolution of a function $f(x)$ on $g(x)$ is defined as

$$(f * g)(x) = \int f(\xi)g(x-\xi)d\xi \tag{16}$$

Naively the convolution gives nonzero value when the there is an overlap between two functions. We can use this technique to identify the pattern in the images. In CNN we use a layer (effectively a function) that takes the convolution of output from the previous layer to that layer. During the training process the model learns to make an effective convolutional layer that detect a specific pattern in each node. Fee FIG. **??** for an illustration.

A CNN is not totally convolutional layer but it is a mix of fully connected layer and convolutional layer. Often the model includes another type of model called a pooling layer. A pooling layer is used to reduce the size of the resolution in the data thus reducing the computational burden and still keeping the necessary information. A common type of pooling is max pooling where we take maximum from the group of pixels to be averaged. Another kind of pooling is average pooling where we take average of the group of the pixels.

D. Transfer Learning

Transfer learning is a technique that enables us to use knowledge gained from one context in order to solve new problems in other context. In order to successfully apply the transfer learning the new problem must be somewhat similar to the older one. The main advantage of the transfer learning is that we can learn the new problem quickly and effectively which would tackle much effort otherwise. The degree of similarity can determine how much of knowledge we can transfer in one another. It is a general concept and not restricted to machine learning.

In the context of the convolutional deep neural network we can use the model trained to classify some objects, say cats and dogs, now to train a model that classify different objects, like cars and trucks. Generally we remove few layers close to output side keeping most of the layers from the input side from pre-trained models. The reason why transfer learning works in CNN is the following. In CNN the first few layers

learns very generic features about the patterns, for example, vertical, horizontal, diagonal edges or lines which is same for all image classification tasks. However, in subsequent layers the model learns more specific features about the objects we are classifying.

There are many pre-trained CNN available which we can use for the transfer learning. In this work we use VGG16 removing the top layer first and in next round unlocking a layer below top layer for the fine tuning. The main advantage of VGG16 is that it is accurate and simple to understand. The main drawback is that this model is very big and can be slow.

III. Fire Detection Model

A. Data Collection

1) Collecting Fire Images: The secondary source fire image data set used in this project are primarily from the following sources [1]–[3]. The first data set [1] contains 1405 images of fires. There are mostly urban fire and few wildfires. But the data is not cleaned. It contains many images without fire: site before and after the fire, firefighters and their equipment only, map of the area of the fire, a cartoon of fire, etc. So the data cleaning has been done. All black and white images are removed. All images without fire or smoke are removed. All the repeated images had been removed. It was a challenge whether or not to keep the picture involving fire damage and lots of ash. Ash might be the indication that there was a fire. But as a fire detection tool there might be little interest in the post damage picture. So I decided to remove those images as well. After removing all the irrelevant images we retain 768 images in our data set. In total, I removed 637 images from the data set. Which is all human labor.

The second data set [2] contains 755 images containing fire and 244 images without fire. Clearly, there is a disparity in the number of images in two categories. Also, the fire images mostly contain wildfire and few urban fires. However, the non-fire images are mostly forest images. A significant amount of fire images are taken at night while the non-fire images are taken on the day. So, the non-fire data set is not representative. So, I use a non-fire data set from other sources to make it more representative. Throughout this project JEPG image with RGB, the format is chosen as default. The fire images in this data set are in .png format. So conversion was necessary to merge into the final data set. The labels were all correct in this set but there were few duplicate images. After removing all the duplicates there are 717 fire images. The third data set [3] contains 110 fire images. No cleaning is done in this data set other than removing one duplicate. This data set contains non-fire images as well. But they are mostly indoor images. Almost all images are taken at day. So we have 1595 fire images in total. However, the data set is not representative of all fire cases.

Following are the limitations of the data set. There are no indoor fire images. There are no road/highway/vehicle fire images. Therefore, I use web scraping techniques to collect more fire images, which I will discuss in next section. So in total, we have 2000 fire images.

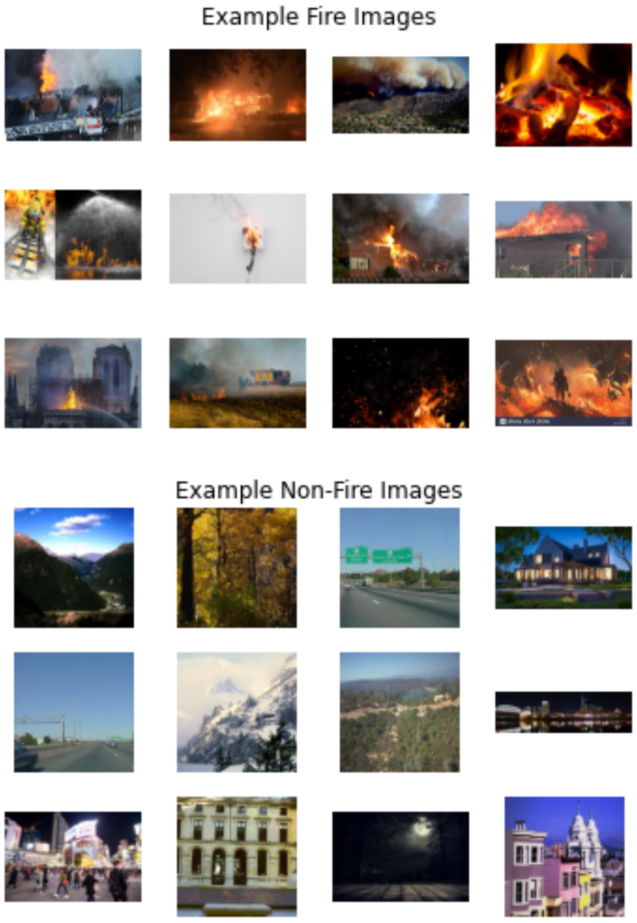


Fig. 2. A sample of fire and non-fire images from the data set used to train neural network.

2) Collecting Non-Fire Images: There is a huge collection of images in 8 scene categories data set which we may use for non-fire images. This can be found in the following link [5]. There are 2688 images in various categories: coast, forest, highway, city, mountain, land, street, tall building, etc, and their subcategories. But almost all images are taken at day. This produces the risk, especially for night images with urban artificial light to miss classify as fire images. So, I need another data set containing night images without fire. So I used night images from the following source [6] These are all urban night images. There are 93 such images. So far we have sufficient non-fire images to match the number of fire images. But we have the following limitations. There are not sufficient night images. Night images are not diverse: forest, highway, etc. Therefore, I collected more night images without fire. So in total, we have used 698 night-images in this model. This is little over one-third of total non-fire images.

3) Downloading Images from Website: Because of the limitations above we decided to collect more data from the website to make the dataset more representative. I used the adobe stock website to collect images [4].

I used the BeautifulSoup package to extract images from the website. I used 43 different keywords to search for fire images

and 35 different keywords to search the night images without fire. I cleaned the data removing all the images not containing fire and smoke from the first set. Similarly, I cleaned the second dataset for the non-fire night images. In this way, I added 407 new fire images and 605 new night images.

Here are few example of fire images and non-fire images used in this model.

B. Training a Deep Learning Model

1) Model Architecture: We have 4000 total labeled sample images in total to work with. That means 2000 fire images and 2000 non-fire images. Among them I have used 3000 images for training and 1000 for testing equally splitting among both the labels. Although this data set is of decent size, it is not enough to train the model from scratch using Keras model. And as we see later it is not necessary as well. Instead we can use some of the pre-trained model.

I use VGG16 pre-trained model. VGG-16 is a trained Convolutional Neural Network (CNN), from Visual Geometry Group (VGG), Department of Engineering Science, University of Oxford. The number 16 means the number of layers with trainable weights. The reference paper about the VGG16 is here [8] and a review article is here [9].

Pre-trained model are trained in different data, not necessarily similar to the data we are training in this model. But we can still use it because of the following reason: The CNN has series of layers. Each layer learns a set of features from the image data. The lower layers learn fundamental patterns like edges, lines, curves etc. The higher layers on the other hand are specific to the images on the model. Hence, the featured learned by the lower level can be general to the large class of images, even the images which model did not see during its training. Because of this reason we only use the base of the pre-trained model removing the top. We do this here in two steps.

First, we retain all the base and remove only the dense top layer and train the model. Which gives us validation accuracy close to 90%. And in second step, we unlock the top convolutional model on the base and further train the model. Since we already have a decent accuracy we can imagine that the model is already close to the optimum model. So we only need to fine tune. For this reason we drop the learning rate to 10% of the previous case and train for larger number of iterations. Doing so we achieve a validation accuracy close to 97%. Which is pretty decent result.

Setting up model is the following: (1) Loss: Since this is classification problem and there are two classes, we use the binary cross-entropy as the loss function. (2) Optimizer: We use RMSprop optimizer with customized learning rate. (3) Metrics: In addition to the loss we want to observe the accuracy. We optimize our model based on this metric.

2) Data Generation & Data Augmentation: For the large data set it is not convenient to load all the data into memory. So we use image data generator to load the data from hard disc to memory in small batch. We do the same of the training and test set.

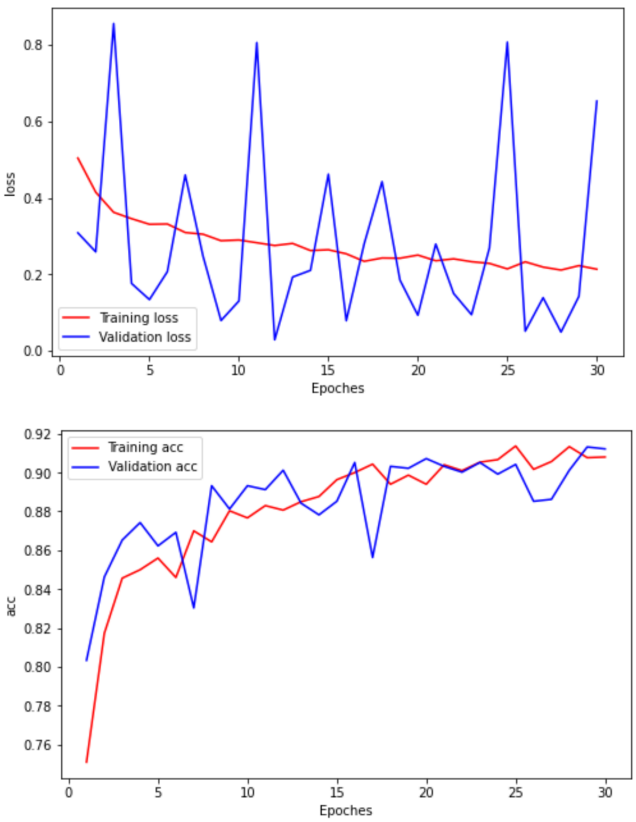


Fig. 3. (Upper) Training and validation loss as a function of the number of epochs. (Lower) Training and validation accuracy as a function of the number of epochs. Trained by only removing the top dense layer.

Further, when initiating the image data generator we can do the data augmentation. This is the step to create more data from existing data by transforming the image. This artificially provides more data to train. Here we use rotation, translation, shear, zooming and horizontal flip for data augmentation. Other transformations like vertical flip is not suitable. We only do the data augmentation in the training set and not on the validation and test set.

We pass the training data from the train generator. We train for 30 epochs. We pass the validation data from the validation generator. We get validation accuracy above to 90% from this.

Here is the graph of training and validation loss and training and validation accuracy.

3) Fine Tuning the Model: We trained previously with only top layer removed from VGG16. Here we unlock top base layer from VGG16 and fine tune the model. Doing so we reduce the learning rate by an order from 10^{-4} to 10^{-5}. We train for the 50 epochs. The model surpass the validation accuracy of 97% shortly after 30 epochs. It is not unlikely to improve the model after 50 epochs. But I am happy with this for now. The future plan is to check with other pre-trained model rather.

The training and validation graphs are here

4) Error Analysis: In this section we analyze the error of the model, i.e. miss classified images. We first see few

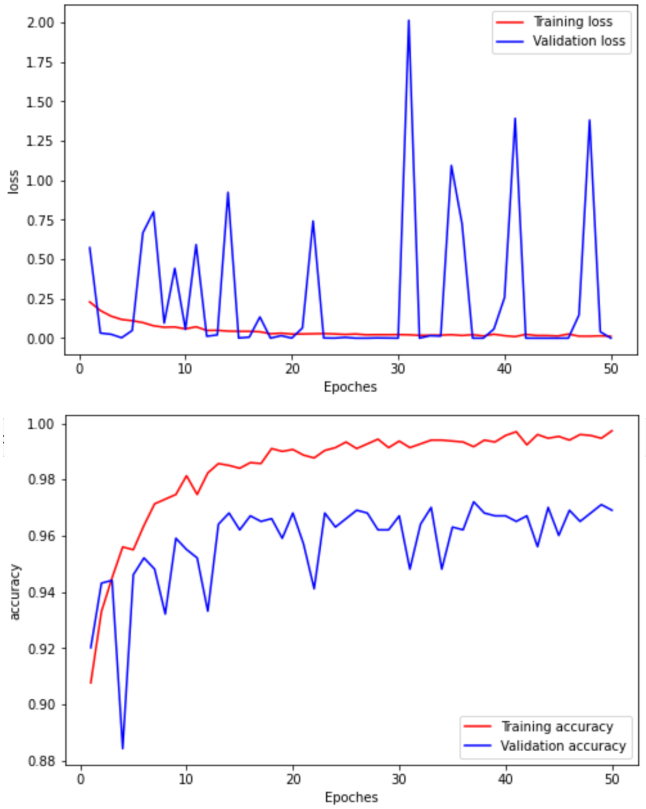


Fig. 4. Performance of the fine tuned model: (Upper) Training and validation loss as a function of the number of epochs. (Lower) Training and validation accuracy as a function of the number of epochs.

examples of the correctly classified images in FIG. 5. Then we visualize the confusion matrix and ROC curve in FIG. **??**. And finally, we see separately fire images classified as non-fire and non-fire images classified as fire in FIG. 6.

Some of the miss classified figure have fire but that is too small. So even human observer is easy to confuse with them. Though some of the big explicit fire images are miss classified too. May be that is painting of fire but not the picture. Miss classified fire images are mostly bonfire, stove fire, fire torch, kitchen fire etc. This is not big surprise because there were not enough fire sample in training set in that categories.

Looking at this miss classified set some of the picture actually seem to have fire. So, the problem is about the mis-labeling. Others don't have fire but have artificial red light or are picture with hue of dawn and dusk almost appearing as fire.

Overall the model has done very good job separating those images with solid 97% accuracy in out of sample images.

IV. Discussion & Conclusion

In this work we build a model that can identify the presence of fire in a image data. We trained a deep convolutional neural network for the image classification task. Instead of training the model from scratch we use an advanced method of transfer learning. We achieved very good accuracy of 97% on the

Fig. 5. Illustration of the classification made by the model.

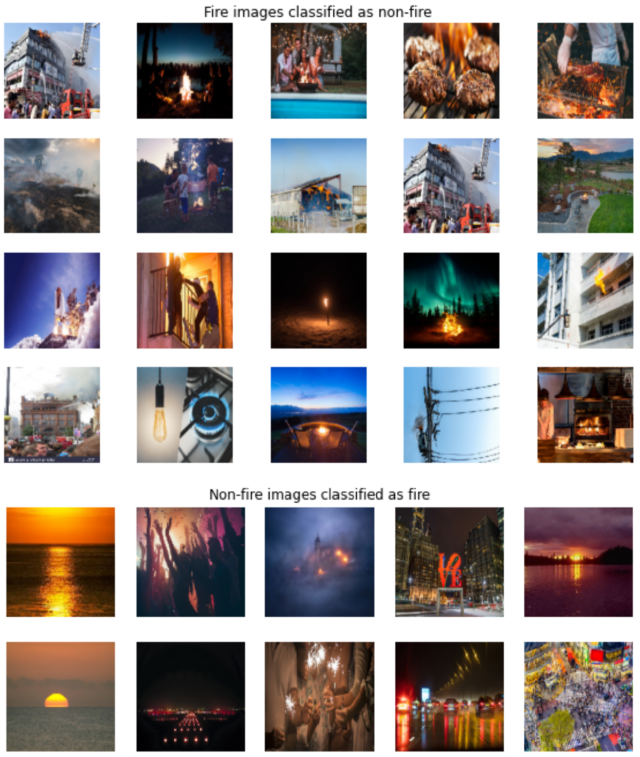


Fig. 6. Images miss classified by the model.

validation data thanks to the transfer learning although our data set was of moderate size. Also the training was done in reasonable time in a personal computer thanks to the transfer learning. We used Keras/TensorFlow frame work for training the model. The GPU implementation was very helpful to accelerate the training process especially during the fine tuning process.

An special effort had been put forth to prepare a data set

that contains the fire image and non-fire image with many possible representative cases in real life. We use secondary data sources wherever available and also supplemented them with sufficient amount of data collected from the web scraping technique. Beside the accuracy our model can be summarized in terms of the following metrics.

$Precision = 98.99\%, Recall = 98.00\%, Accuracy = 98.50\%$

Confusion matrix, ROC and AUC: TBD

There are few room for improvement in this work and there are few places where we can extend further. Although the accuracy is good VGG16 is large and slow to train. Try other pre-trained networks: Xception (smaller size higher accuracy), MobileNet (much smaller in size with comparable accuracy). Look here: https://keras.io/api/applications/ Have test, train and validation split: So far there is only test and validation set. The data scraping code is not very general. It collects only 25 images per keyword. It is not generalized for all kinds of website. Collecting even more images.

ACKNOWLEDGMENT

I am thankful to the creator of all the secondary data sources I used.

REFERENCES

[1] https://drive.google.com/file/d/11KBgD_W2yOxhJnUMiyBkBzXDPXhVmvCt/view
[2] https://www.kaggle.com/phylake1337/fire-dataset
[3] https://github.com/cair/Fire-Detection-Image-Dataset/find/master
[4] https://stock.adobe.com/
[5] https://people.csail.mit.edu/torralba/code/spatialenvelope/
[6] https://www.visuallocalization.net/datasets/
[7] https://github.com/roshankoirala/Fire_Detection$_Model$.
[8] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
[9] https://neurohive.io/en/popular-networks/vgg16/
[10] Chamlin, M.B. Crime and arrests: An autoregressive integrated moving average (ARIMA) approach. J Quant Criminol 4, 247–258 (1988). https://doi.org/10.1007/BF01072452
[11] R. H. Shumway, D. S. Stoffer, "Time Series Analysis and Its Applications, Fourth Edition," Springer, 2016.