Emerging Methods for Early Detection of Forest Fires

Model Building

Adding CNN Layers

We will be adding three layers for CNN

- Convolution layer
- Pooling layer
- Flattening layer

Task 1: Adding Convolutional Layer

The convolutional layer is the first and core layer of CNN. It is one of the building blocks of a CNN and is used for extracting important features from the image.

In the Convolution operation, the input image will be convolved with the feature detector/filters to get a feature map. The important role of the feature detector is to extract the features from the image. The group of feature maps is called a feature layer.

```
#add convolutional layer
model.add(Convolution2D(32,(3,3),input_shape=(128,128,3),activation='relu'))
```

In the convolution2D function, we gave arguments that include 32,(3,3), that refers to we are applying 32 filters of 3x3 matrix filter, and input_shape is the input image shape with RGB, here 64x64 is the size and 3 represent the channel, RGB colour images.

Activation Function: These are the functions that help us to decide if we need to activate the node or not. These functions introduce non-linearity in the networks.

Task 2: Adding Pooling Layer

Max Pooling selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

After the convolution layer, a pooling layer is added. Max pooling layer can be added using MaxPooling2D class. It takes the pool size as a parameter. Efficient size of the pooling matrix is (2,2). It returns the pooled feature maps. (Note: Any number of convolution layers, pooling and dropout layers can be added)

```
#add maxpooling layer
model.add(MaxPooling2D(pool_size=(2,2)))
```

In the above code, pool_size refers to pooling filter or kernel size.

Task 3: Adding Flatten Layer

Now the pooled feature map from the pooling layer will be converted into one single dimension matrix or map, where each pixel in one single column, nothing but flattening. The flattening layer converts the multi-dimension matrix to one single dimensionlayer.

```
In [ ]: #add flatten layer
        model.add(Flatten())
```