

# E MERGING METHODS FOR EARLY DETECTION OF FOREST FIRE

## MODEL BUILDING

### ADDING CNN LAYERS

Team ID	PNT2022TMID21968
Project Name	Project-Emerging methods for early detection of forest fire

### ADDING CNN LAYERS:

We will be adding three layers for CNN

- Convolution layer
- Pooling layer
- Flattening layer

#### 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.

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.

#### Adding Pooling Layer

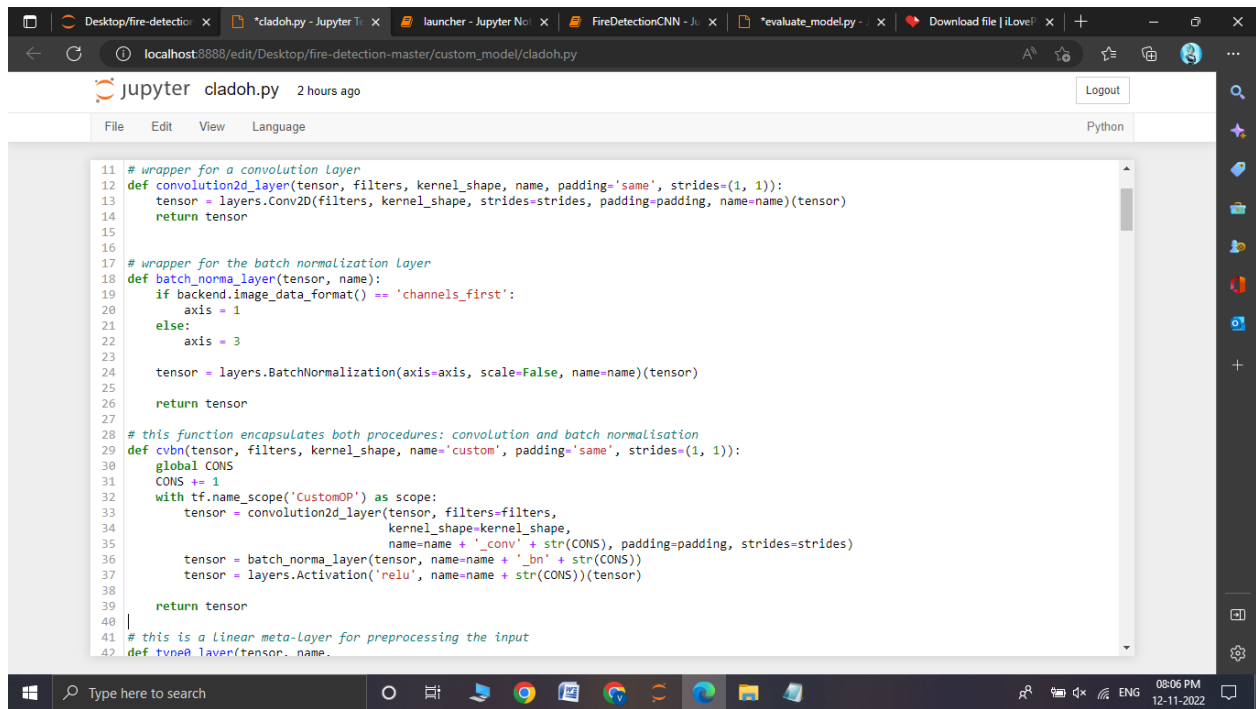
**Max Pooling** selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

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)

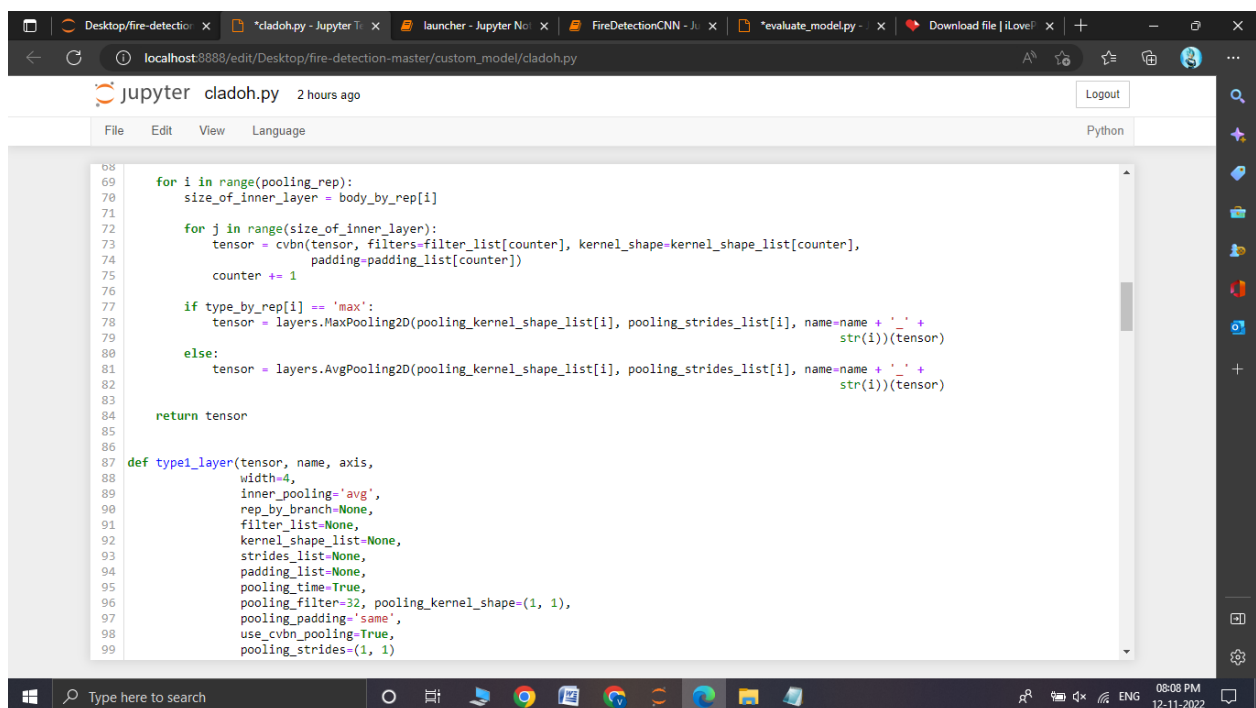
In the above code, pool\_size refers to pooling filter or kernel size.

#### 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 dimension layer.



```
11 # wrapper for a convolution layer
12 def convolution2d_layer(tensor, filters, kernel_shape, name, padding='same', strides=(1, 1)):
13     tensor = layers.Conv2D(filters, kernel_shape, strides=strides, padding=padding, name=name)(tensor)
14     return tensor
15
16
17 # wrapper for the batch normalization layer
18 def batch_norma_layer(tensor, name):
19     if backend.image_data_format() == 'channels_first':
20         axis = 1
21     else:
22         axis = 3
23
24     tensor = layers.BatchNormalization(axis=axis, scale=False, name=name)(tensor)
25
26     return tensor
27
28 # this function encapsulates both procedures: convolution and batch normalisation
29 def cvbn(tensor, filters, kernel_shape, name='custom', padding='same', strides=(1, 1)):
30     global CONS
31     CONS += 1
32     with tf.name_scope('CustomOP') as scope:
33         tensor = convolution2d_layer(tensor, filters=filters,
34                                     kernel_shape=kernel_shape,
35                                     name=name + '_conv' + str(CONS), padding=padding, strides=strides)
36         tensor = batch_norma_layer(tensor, name=name + '_bn' + str(CONS))
37         tensor = layers.Activation('relu', name=name + str(CONS))(tensor)
38
39     return tensor
40
41 # this is a Linear meta-Layer for preprocessing the input
42 def tvne0_layer(tensor, name,
```



```
68
69     for i in range(pooling_rep):
70         size_of_inner_layer = body_by_rep[i]
71
72         for j in range(size_of_inner_layer):
73             tensor = cvbn(tensor, filters=filter_list[counter], kernel_shape=kernel_shape_list[counter],
74                           padding=padding_list[counter])
75             counter += 1
76
77         if type_by_rep[i] == 'max':
78             tensor = layers.MaxPooling2D(pooling_kernel_shape_list[i], pooling_strides_list[i], name=name + '_' +
79                                           str(i))(tensor)
80         else:
81             tensor = layers.AvgPooling2D(pooling_kernel_shape_list[i], pooling_strides_list[i], name=name + '_' +
82                                           str(i))(tensor)
83
84     return tensor
85
86
87 def type1_layer(tensor, name, axis,
88                 width=4,
89                 inner_pooling='avg',
90                 rep_by_branch=None,
91                 filter_list=None,
92                 kernel_shape_list=None,
93                 strides_list=None,
94                 padding_list=None,
95                 pooling_time=True,
96                 pooling_filter=32, pooling_kernel_shape=(1, 1),
97                 pooling_padding='same',
98                 use_cvbn_pooling=True,
99                 pooling_strides=(1, 1)
```