

E MERGING METHODS FOR EARLY DETECTION OF FOREST FIRE

MODEL BUILDING

ADDING DENSE LAYERS

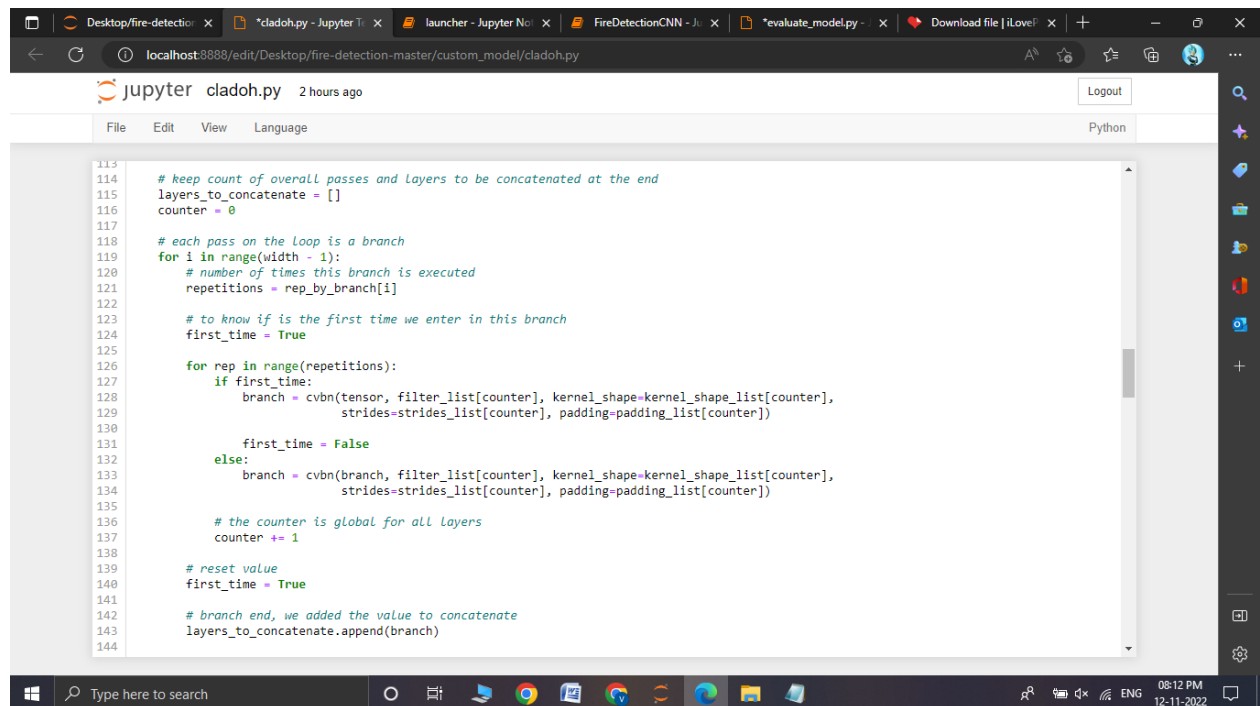
Team ID	PNT2022TMID21968
Project Name	Project-Emerging methods for early detection of forest fire

ADDING DENSE LAYERS:

The name suggests that layers are fully connected (dense) by the neurons in a network layer. Each neuron in a layer receives input from all the neurons present in the previous layer. Dense is used to add the layers.

Adding Hidden layers:

This step is to add a dense layer (hidden layer). We flatten the feature map and convert it into a vector or single dimensional array in the Flatten layer. This vector array is fed it as an input to the neural network and applies an activation function, such as sigmoid or other, and returns the output.



```
113 # keep count of overall passes and layers to be concatenated at the end
114 layers_to_concatenate = []
115 counter = 0
116
117 # each pass on the loop is a branch
118 for i in range(width - 1):
119     # number of times this branch is executed
120     repetitions = rep_by_branch[i]
121
122     # to know if is the first time we enter in this branch
123     first_time = True
124
125     for rep in range(repetitions):
126         if first_time:
127             branch = cvbn(tensor, filter_list[counter], kernel_shape=kernel_shape_list[counter],
128                           strides=strides_list[counter], padding=padding_list[counter])
129
130             first_time = False
131         else:
132             branch = cvbn(branch, filter_list[counter], kernel_shape=kernel_shape_list[counter],
133                           strides=strides_list[counter], padding=padding_list[counter])
134
135     # the counter is global for all layers
136     counter += 1
137
138     # reset value
139     first_time = True
140
141     # branch end, we added the value to concatenate
142     layers_to_concatenate.append(branch)
143
144
```

```

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)

```

Adding output layer:

This step is to add a dense layer (output layer) where you will be specifying the number of classes your dependent variable has, activation function and weight initializer as the arguments. We use add () method to add dense layers. In this layer, no need of mentioning input dimensions as we have mentions them in the above layer itself.

```

258     # from here is the final setup and return
259
260     # output layers
261     if include_top:
262         # Classification block
263         tensor = layers.GlobalAveragePooling2D(name='avg_pool')(tensor)
264         tensor = layers.Dense(classes, activation='softmax', name='predictions')(tensor)
265     else:
266         if pooling == 'avg':
267             tensor = layers.GlobalAveragePooling2D()(tensor)
268         elif pooling == 'max':
269             tensor = layers.GlobalMaxPooling2D()(tensor)
270
271     # print('*40, 'tensor_shape: ', tensor.shape)
272     # Ensure that the model takes into account
273     # any potential predecessors of 'input_tensor'.
274     if input_tensor is not None:
275         inputs = keras_utils.get_source_inputs(input_tensor)
276     else:
277         inputs = img_input
278
279     # model creation and return
280     model = models.Model(inputs, tensor, name='cladoh')
281     return model
282
283
284 # we use the same preprocessing as in inception
285 def preprocess_input_custom(x, **kwargs):
286     return imagenet_utils.preprocess_input(x, mode='tf', **kwargs)
287
288

```