* Attached is the Python code generated in Jupyter NB. This dataset is available on Kaggle and it can be downloaded using the below link.

<https://www.kaggle.com/puneet6060/intel-image-classification>

**What is Convolutional Neural Network:**

A convolutional neural network (CNN) is a class of deep learning neural networks. CNN is a breakthrough in image recognition. CNN is very fast and efficient in recognising and categorising the images.

CNN Image classification works by taking an input in the form of image (in our case the image is 3 dimensional (150\*150\*3), i.e., the first two dimensions represents pixels in length and width and the third dimension represents the 3 channels of a colour image RGB) and outputting a class with a probability.

But how do can CNN recognises images? The stack of modules of CNN which performs some operations to classify the images are

1. Convolution
2. ReLU (Activation function)
3. Pooling
4. Fully connected layers

CNN has an input layer, hidden layers and output layer. The hidden layers usually consist of convolutional layers, ReLU layers, pooling layers, and fully connected layers.

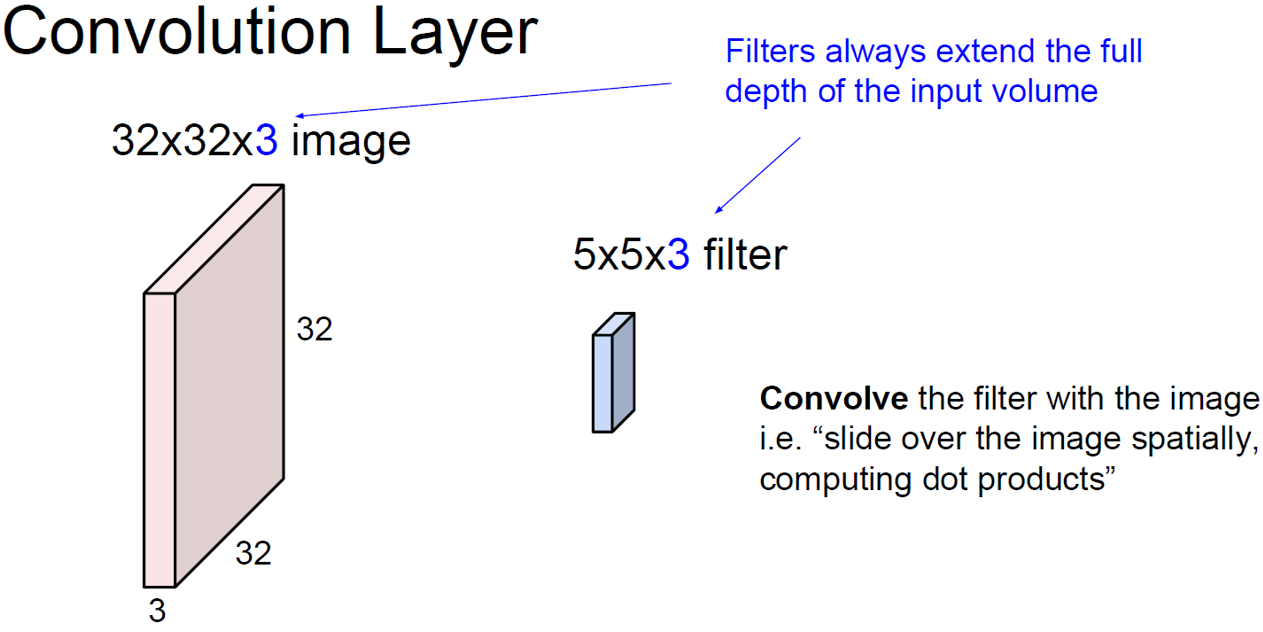
**Convolution:**

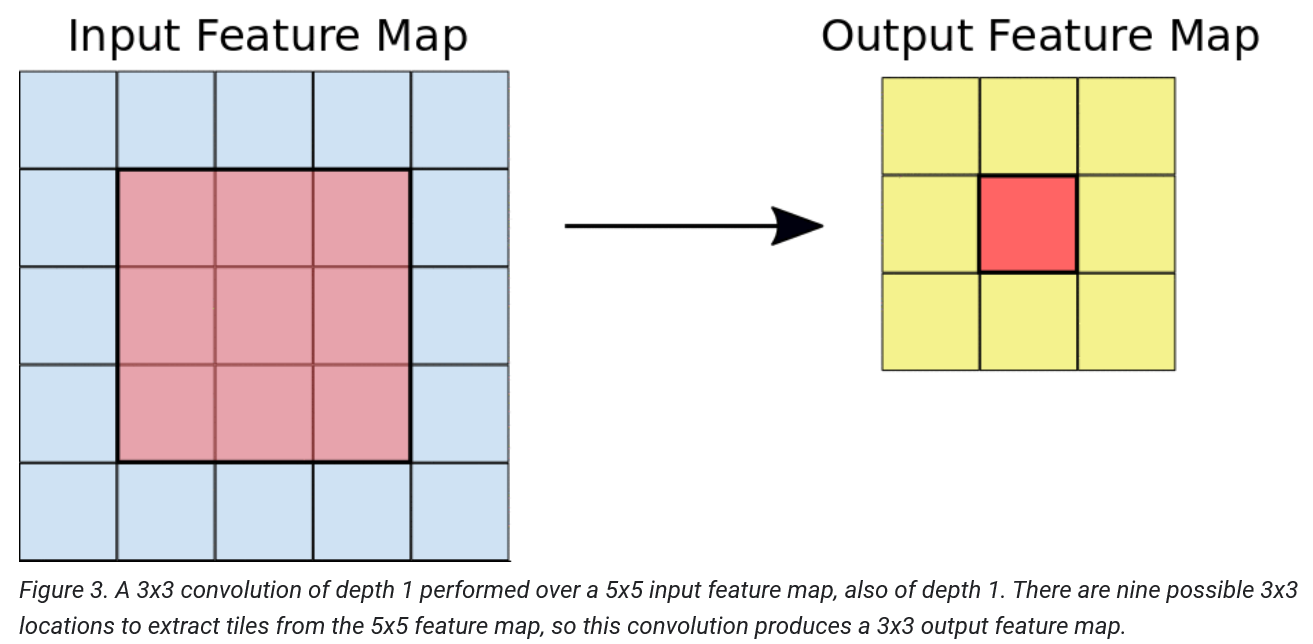
* This is the first step in CNN. The main purpose of the convolution step is to extract features from the input image.
* This layer extract the input feature image and applies filters to them to compute new features, producing an convolved feature which may have a different size and depth than the input feature image.
* I have used, **Convolution in 2D:** It means that the input of the convolution operation is three-dimensional, for example, a color image which has a value for each pixel across three layers: red, blue and green. However, it is called a “2D convolution” because the movement of the filter across the image happens in two dimensions with no padding.

**Filters:**

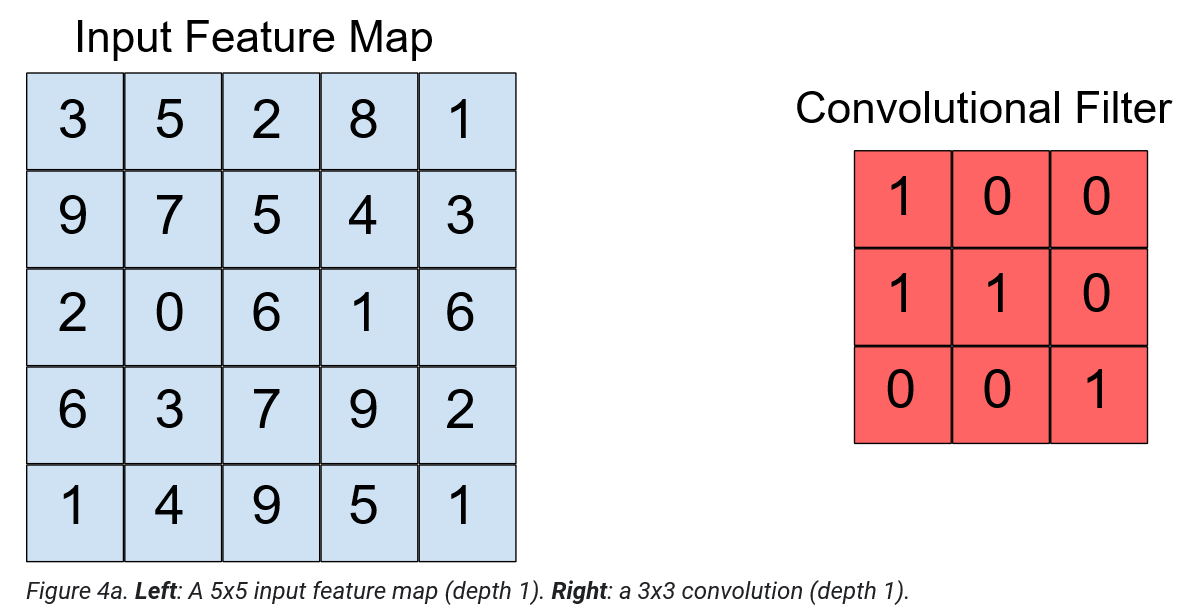
* Filters are used to extract information from multiple types of images
* in 2D convolutions, filters are 3D matrices (which is essentially a concatenation of 2D matrices i.e. the kernels). The CNN layer with kernel dimensions h\*w and input channels k, the filter dimensions are k\*h\*w. A common convolution layer actually consist of multiple such filters.
* Convolution defined by two parameters.

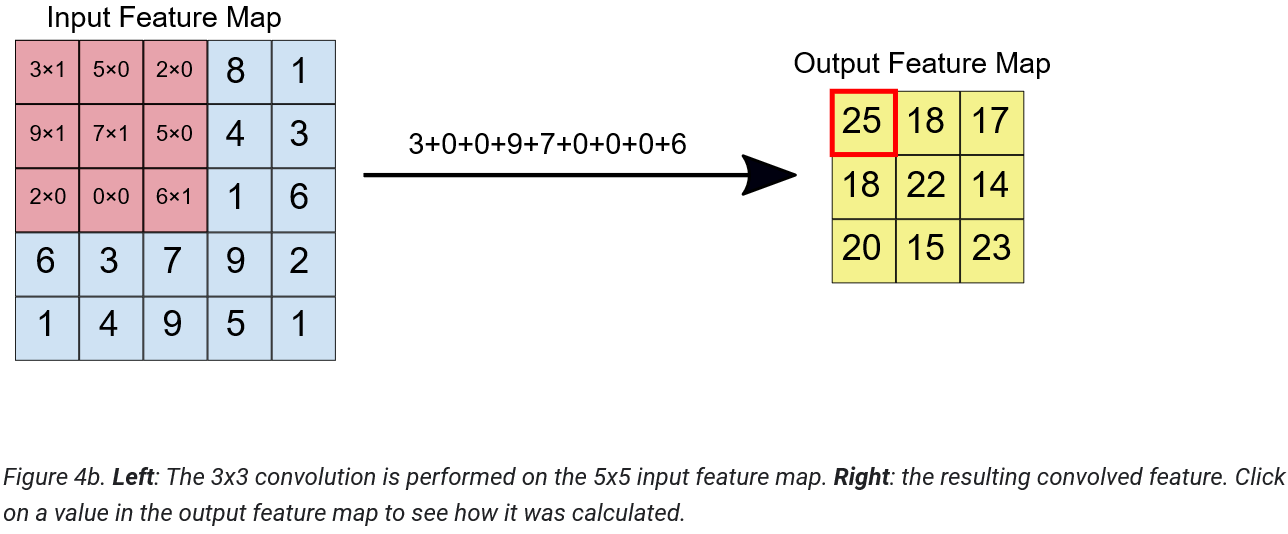
1. **Size of the tiles**: The size of the tiles of the images will be extracted to typically 3x3 or 5x5 pixels.





1. **The depth of the output feature Image:** For each tile pair, CNN performs element-wise multiplication of the filter matrix and the tile matrix, and then sums all the elements of the resulting matrix to get a single value



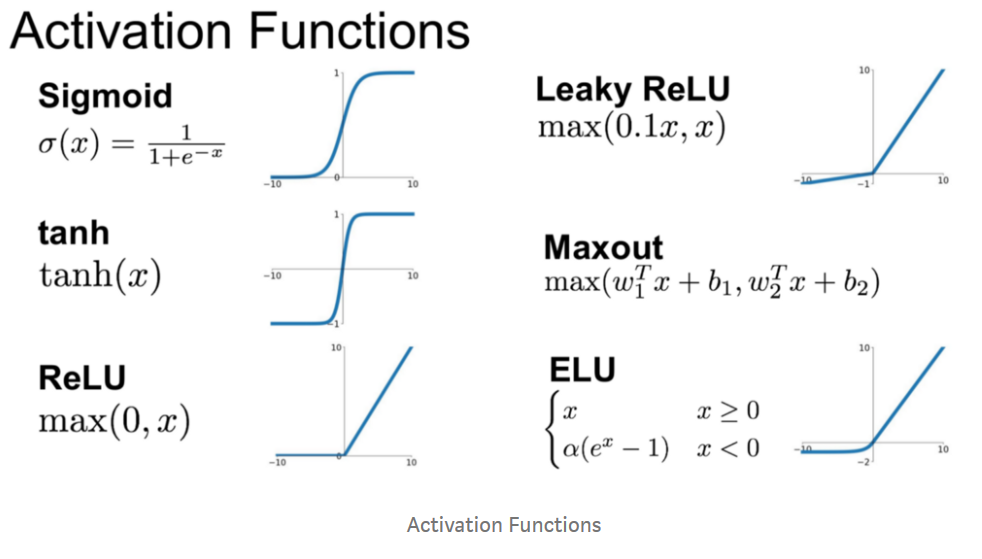


**ReLU:**

In this step, we are applying an **activation** **function** onto our feature images to increase non-linearity in the network. This is because images themselves are highly non-linear. It removes negative values from an activation map by setting them to zero.

The ReLU function, F(x)=max(0,x), returns x for all values of x > 0, and returns 0 for all values of x ≤ 0.

* There are many other activation functions that exist, but as out data has 6 categories, I used ReLU activation function as it is more appropriate than sigmoid (can be used binary classifications).



**Pooling:**

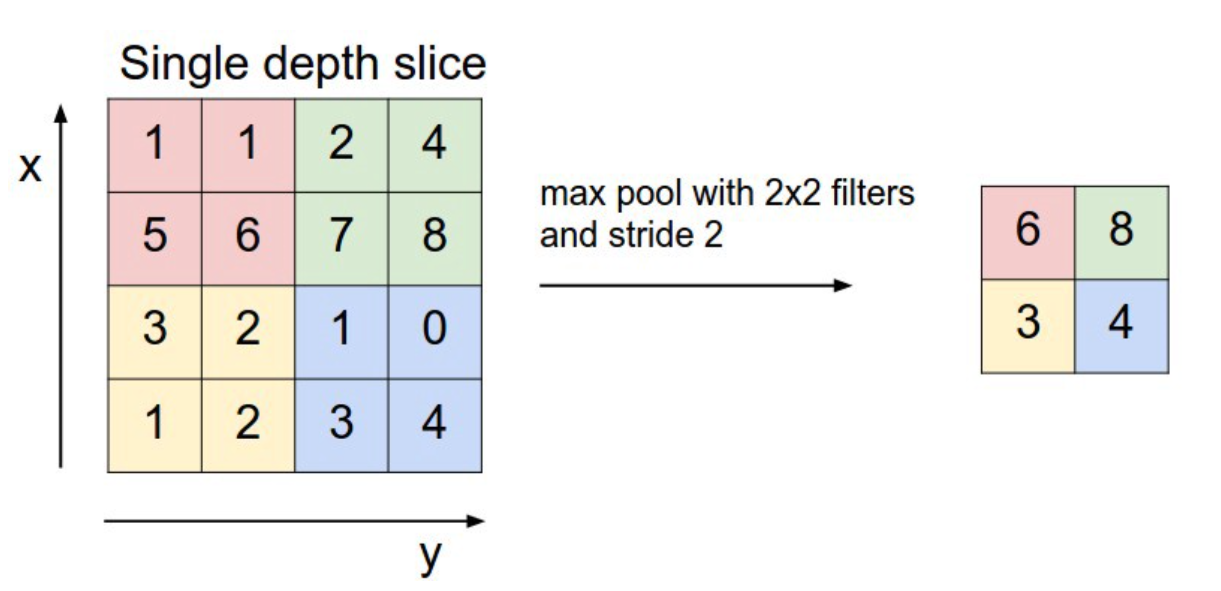
In this step, the CNN down samples the feature produced by the convolved layer to save the processing time, reducing the number of dimensions of the feature map, while still preserving the most critical feature information.

This layer basically reduces the number of parameters and computation in the network, controlling overfitting by progressively reducing the spatial size of the network.

[**Max pooling**](https://wikipedia.org/wiki/Convolutional_neural_network#Pooling_layer)**:**

* The algorithm used for our task is called [**max pooling**](https://wikipedia.org/wiki/Convolutional_neural_network#Pooling_layer) instead of **average pooling**.
* Max pooling, as name states, it will take out only the maximum from a pool. This is done with the use of filters sliding through the input; and at every stride, the maximum parameter is taken out and the rest is dropped. This is called down-sampling the network.
* Unlike the convolution layer, the pooling layer does not alter the depth of the network.
* Formula for output after max-pooling is: (N — F)/ S + 1;

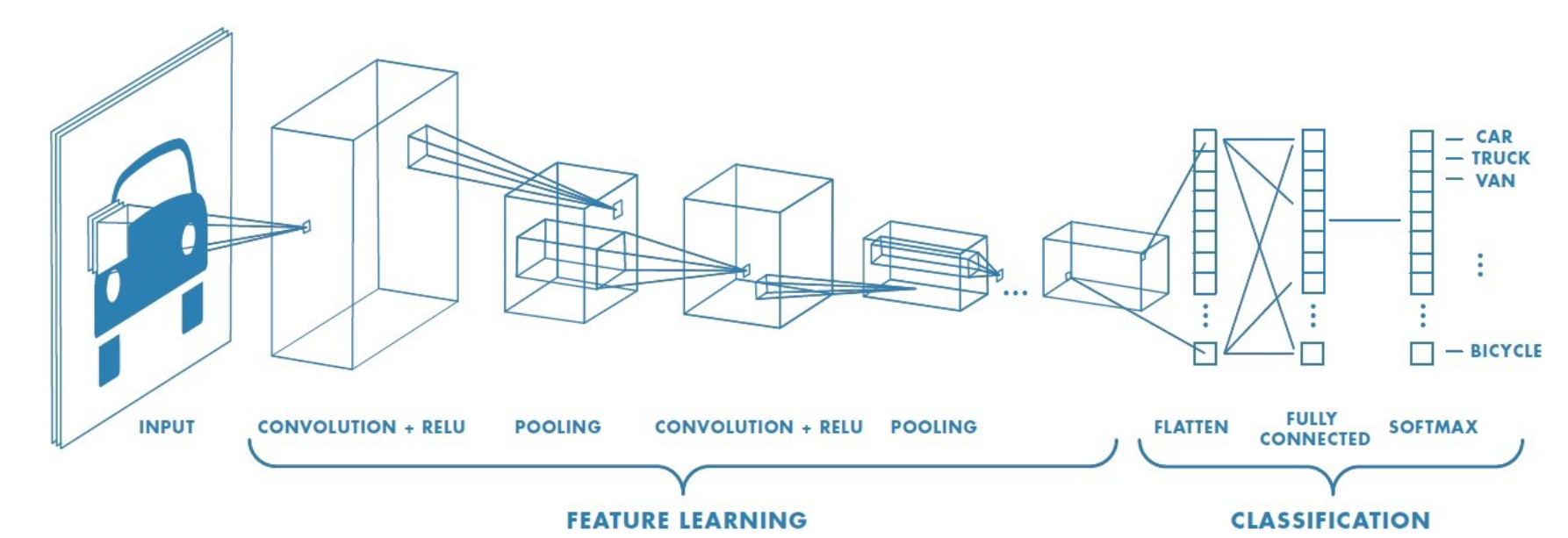
where N = Dimension of input to pooling layer, F=dimension of filter, S=stride.



**Fully-connected Layer:**

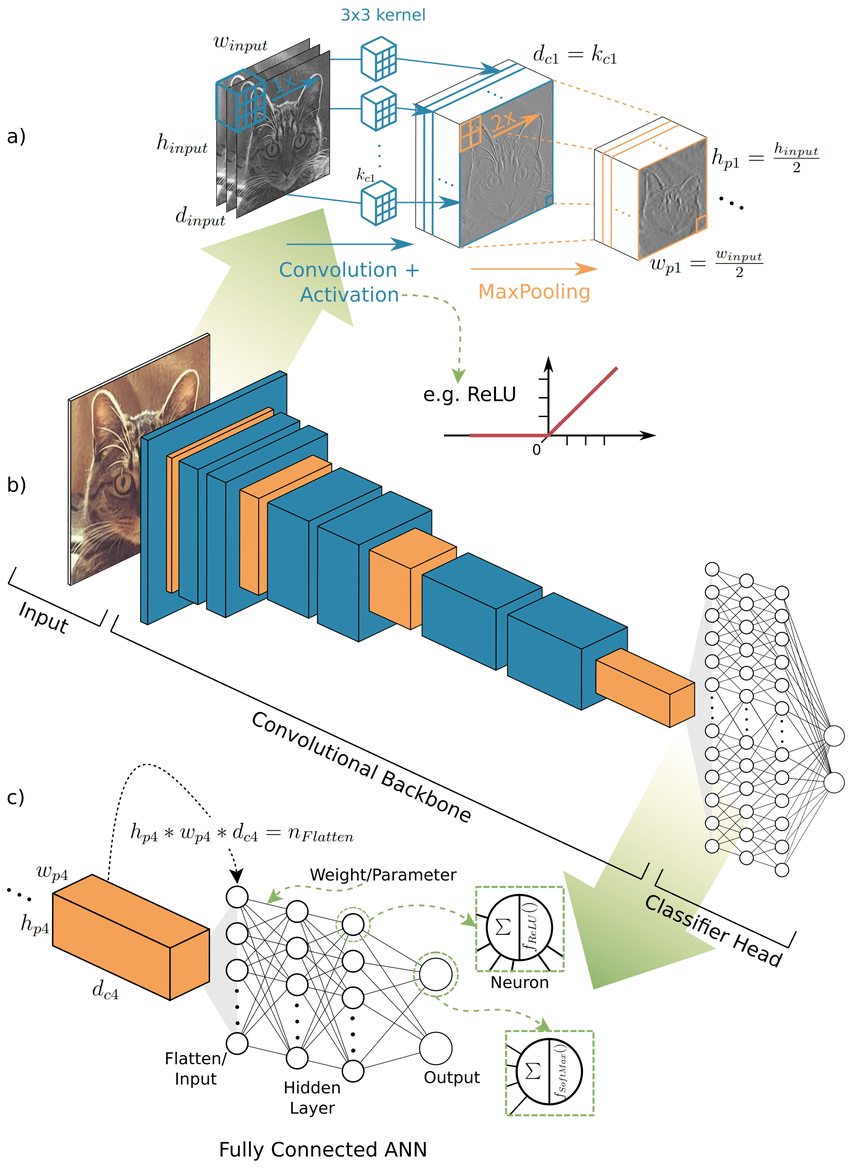
In this layer, the neurons have a complete connection to all the activations from the previous layers. Their activations can hence be computed with a matrix multiplication followed by a bias offset. This is the **last phase** for a CNN network.

The Convolutional Neural Network is actually made up of hidden layers and fully-connected layer(s).



**CNN architechture would looks like as follows:**

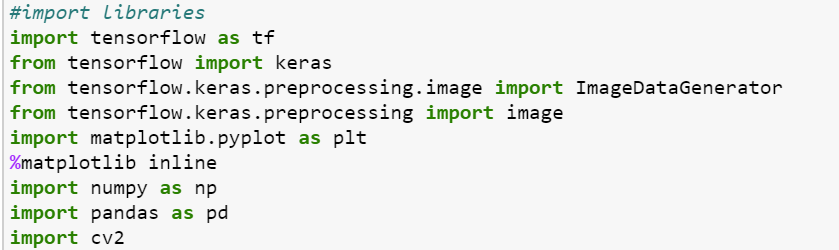
Input ->Convolution ->ReLU ->Convolution ->ReLU ->Pooling ->ReLU ->Convolution ->ReLU ->Pooling ->Fully Connected



**CODING:**

**IMPORT OF PACKAGES:**

Required packages are imported as follow.



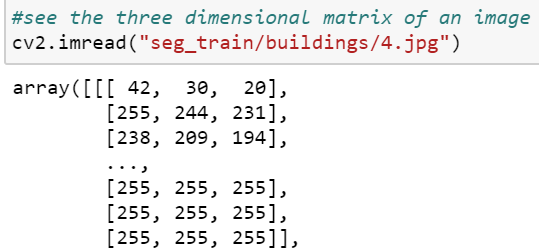
**Familiar with the data, image size and dimensions:**

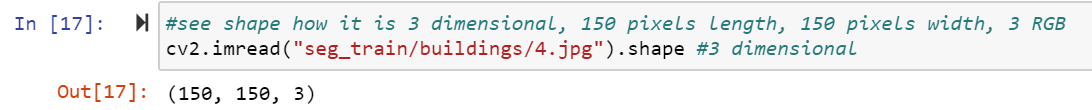
* Before, start of programming, it is really important to check the dimensions and size of an image.
* Based on its size and image, we can select the best model with specific layers.



**Dimension and Size:**

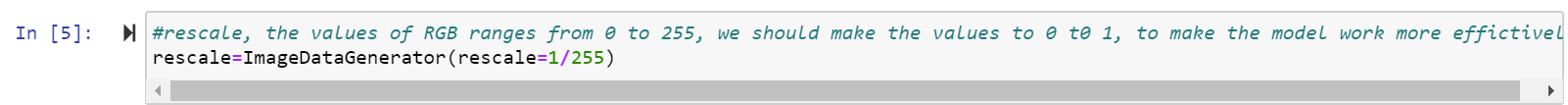
* I have used “imread” and “shape” functions to know the dimensions of the image.
* Based on below outputs of array, I can say that it is a three-dimensional array, with size of 150\*150 and RGB image (3 channels).





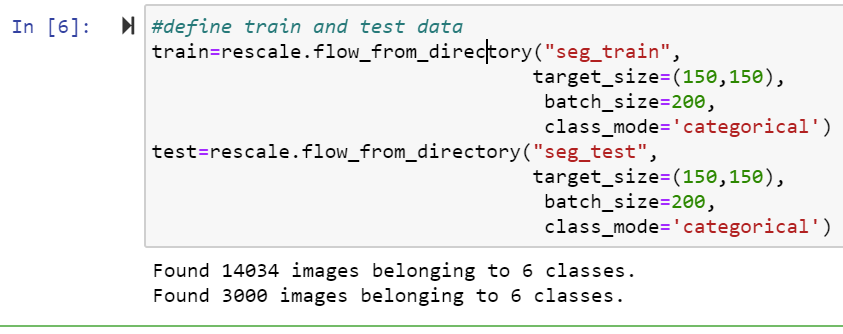
**Rescale the data:**

* As the RGB images are with the matrix values of ranging from 0 to 255, to get the efficient model with best accuracy, I wanted to rescale the data by dividing it with 255 to make the data values ranges from 0 to 1.



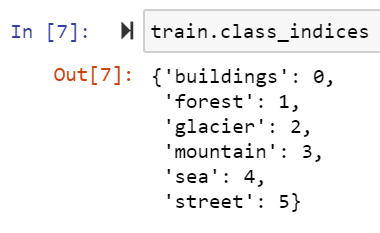
**Define Train and Test data:**

* After rescaling, the test and train images will be defined using “flow\_from\_directory” function by specifying its image target size and batch size.
* Based on the output, I have come to know that the Train folder has 6 sub folders and has overall of 14034 images.
* Whereas the Test folder has also 6 sub folders but with overall of 3000 images.

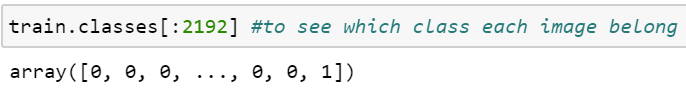


**Identify the list of categories:**

* Through the above step, as I have come to know that there the 6 categories, I wrote a sample program to list the categories.

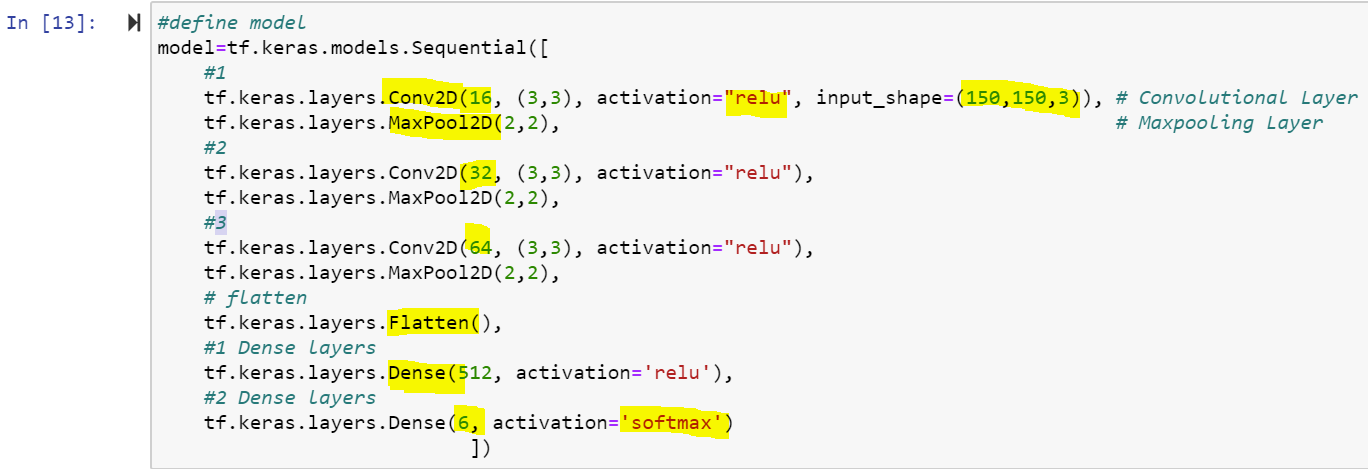


* Below is the sample program to check which class each image belongs upto 2192 images, if you see the output the last image classification is 1 and remaining ones are 0. So, it says that the images from 1 to 2191 are related to building (building folder has 2191 images) where as 2192 belongs to forest.



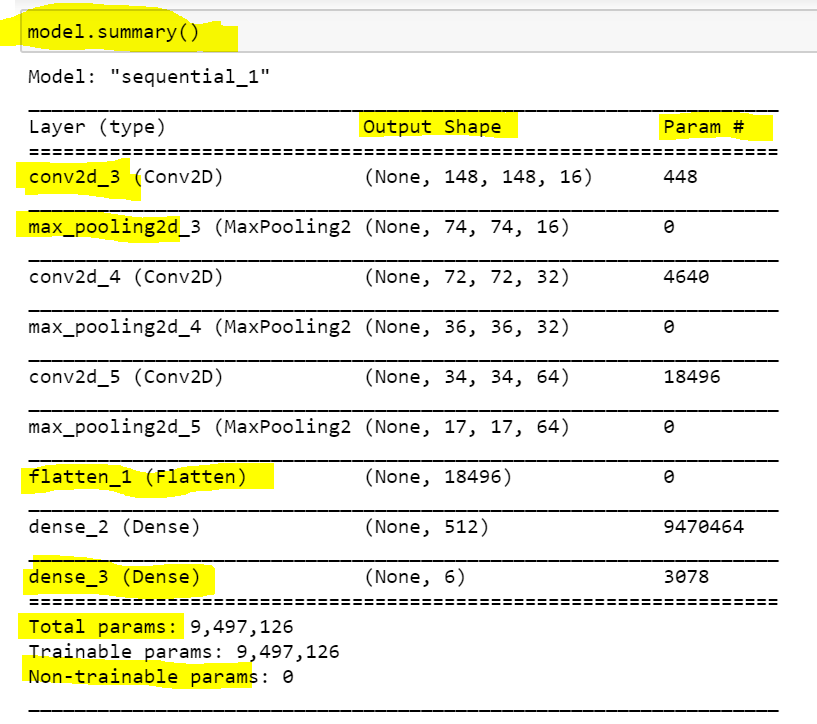
**Define a model:**

* I have created a model, using a tensor flow, keras libraries.
* I have used 3 **convolutional** layers which are **2d** with **filters** of 16, 32 and 64 respectively.
* The input is only required to the first layer, for the input we should specify the size and dimensions of the image.
* I have already mentioned the use of filters and Convolutional 2D in the theoretical part.
* The activation layer that I have used is “**ReLU**” as it suits our data as we have multiple categories, with a Maxpool of 2D.
* Once I have applied the convolutional, activation layers, I have **flatten** the data and then applied the filly connected dense layers with **softmax**.
* Here, the output should be **6** as we have 6 classes.



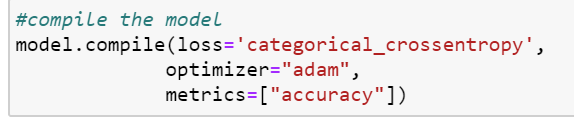
**Visualisation of model:**

* The model given by us can be visualised by using “summary” function.
* In the output, you can find the no.of layers we have used and the list of params selected for the respective layers with the output shape.



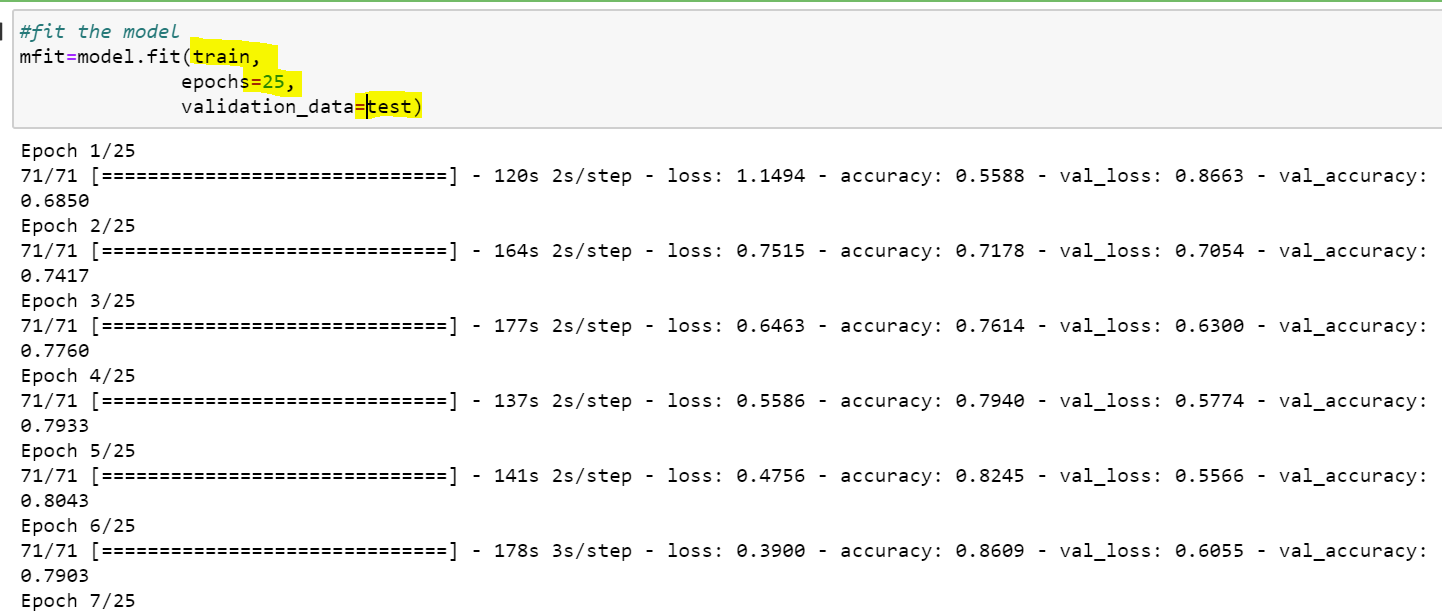
**Compile the model:**

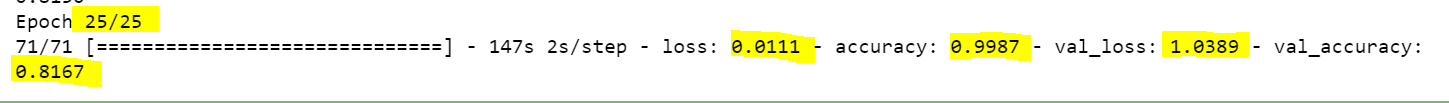
* Before we fit the model, we must compile it, to mention the parameters we should use to calculate the loss and accuracy.
* I have used the “categorical\_crossentropy” parameter to calculate the loss as we have more than 2 categories and used “accuracy” to calculate the efficiency of the model.



**Fit the model:**

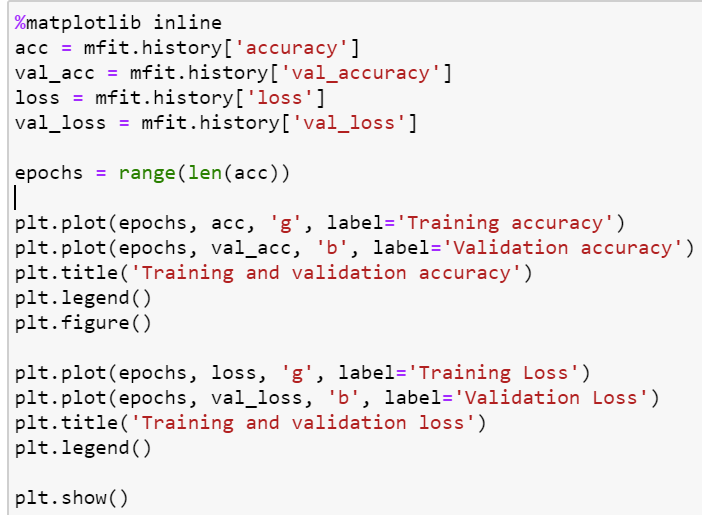
* Once compilation is done, the next step is to fit the model.
* I have used epochs=25 to get the good accuracy/efficiency for the model.
* For the 25th epoch, I have got the **99.8%** accuracy for the train data and **81.6%** accuracy for the test data.
* The losses for the train and test are 0.01 and 1.03 respectively.

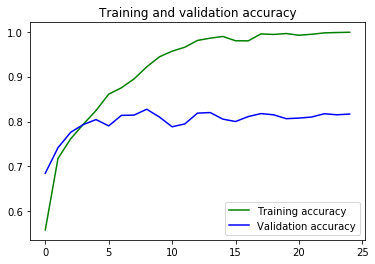




**Visualisation of Accuracy and Loss functions for our model:**

* Based on these line graphs, we can visualise how the accuracy and loss are changing for TRAIN and TEST data by increasing the no.of EPOCHs.
* For both the line plots, the accuracy for training data is high and loss is low.





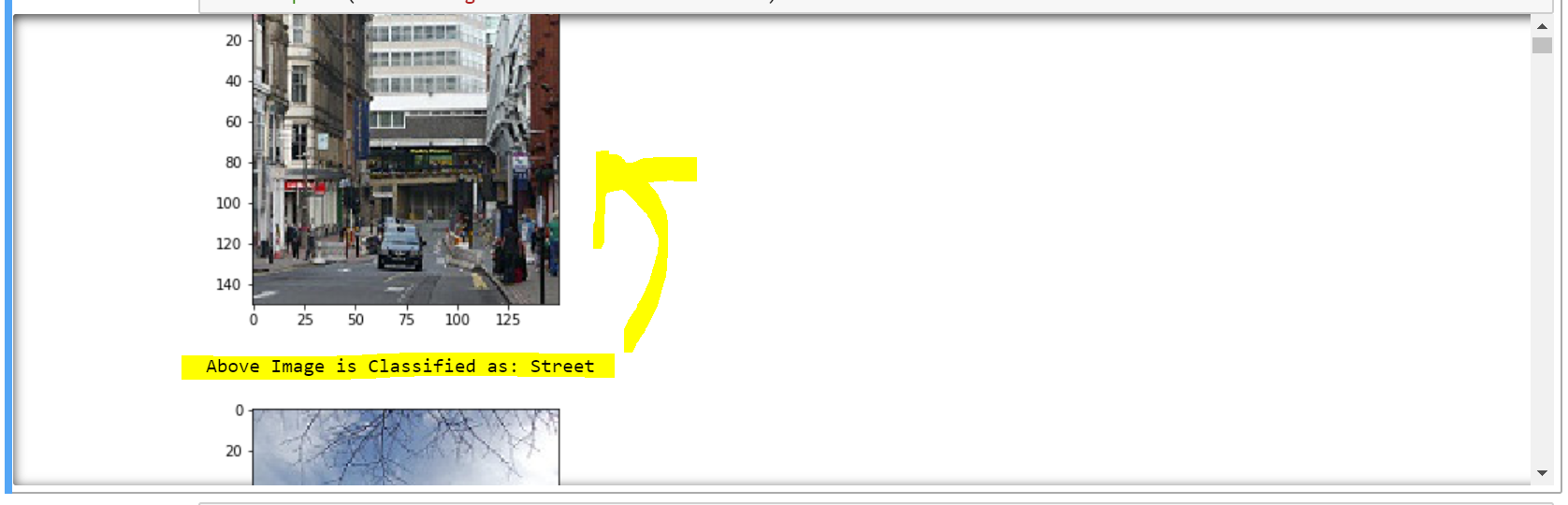


**Prediction of images located in PRED folder:**

* I have predicted all the images located the PRED folder using the PREDICT function.
* I had to use lot other functions which are highlighted in yellow in below screenshot to read the image format.

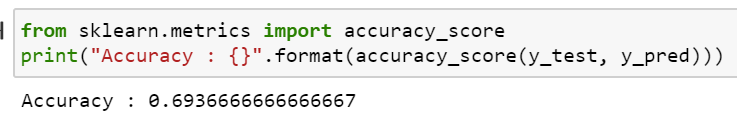


* Below is the sample output of my predictions.



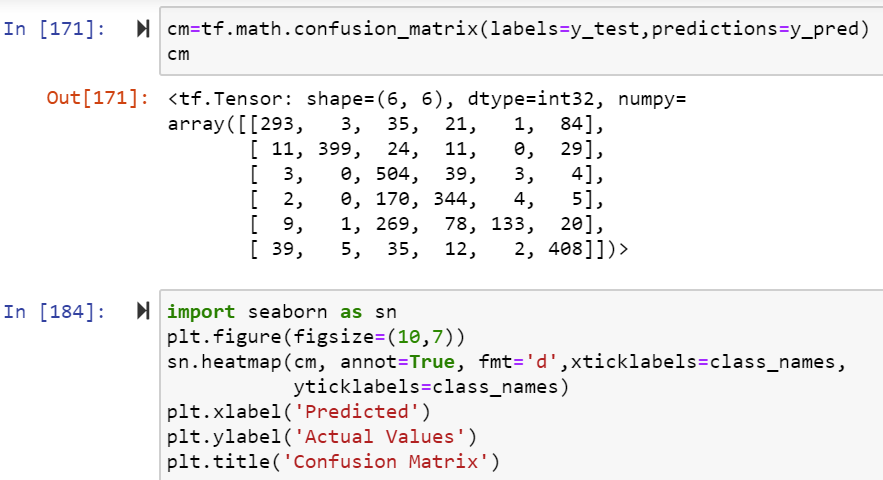
**Accuracy of the Model:**

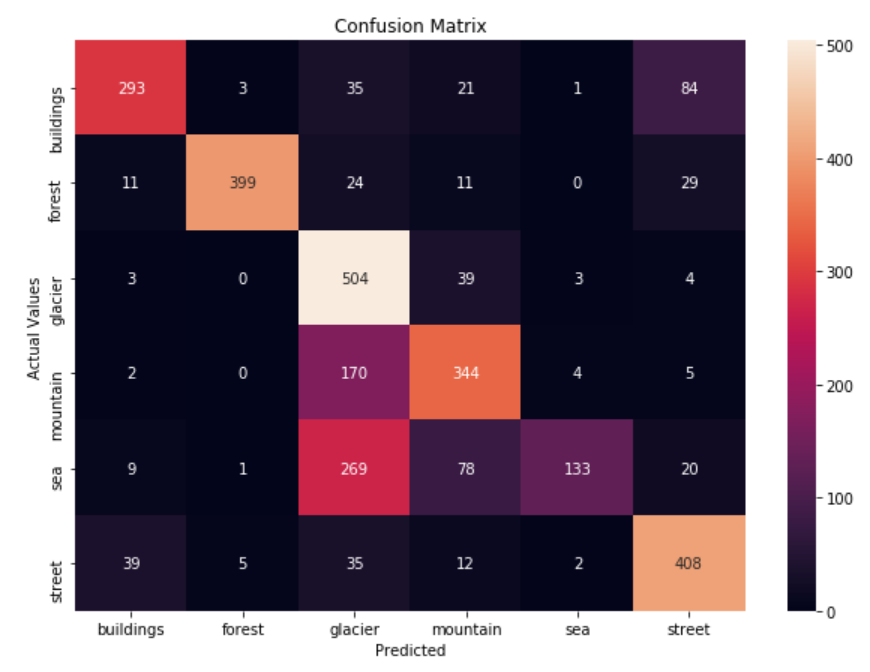
* The accuracy of the model is about 70%, we can increase the accuracy by increasing the no.of epochs and also increasing the hidden layers.
* However, as it is taking much time, I have left as is.



**Confusion Matrix:**

* Confusion matrix is the one which is the visual representation of how the errors and the correct predictions.

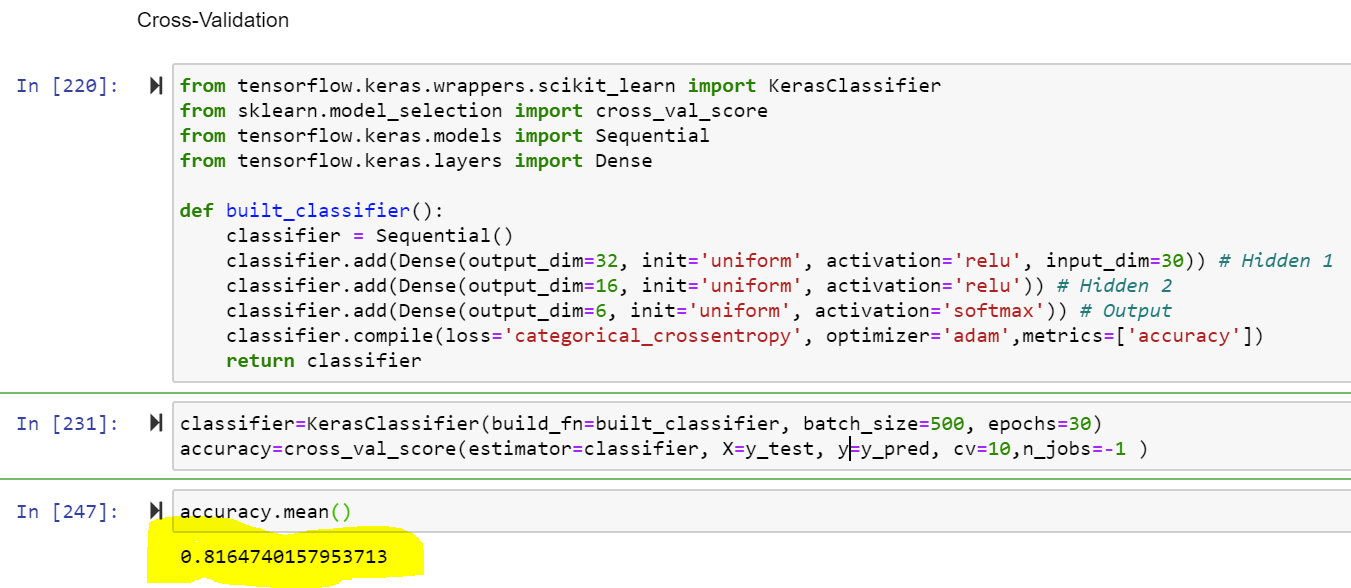




* Based on the above image, we can say that there is a confusion in classification of GLACIER images. The error rate is more when compared to others as it is getting confused among SEA, MONTAIN and GLACIERS.
* It is common to get confused as in many pictures, the mountains, sea and glaciers are together.
* In the same wise, to classify mountain, it is getting confused with the sea images.
* All other categories are okay with minor miss classifications, for example if you take buildings, 293 are classified correctly and others are minor errors.

**10 fold Cross-Validation:**

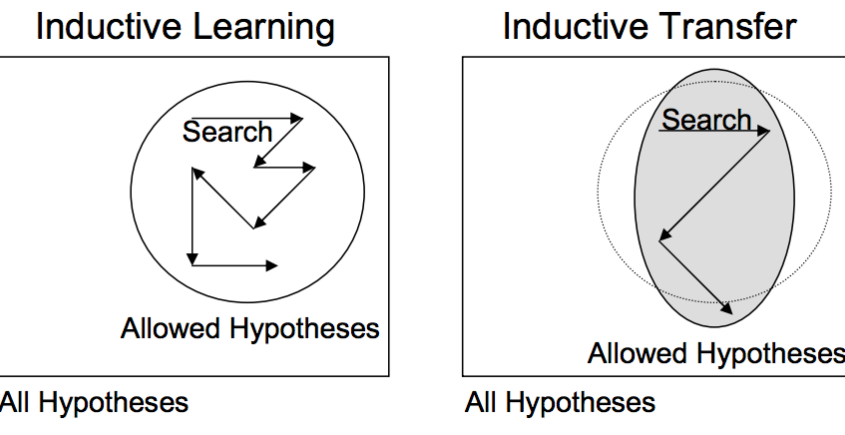
* Using the tensorflow, I have created input, hidden and output layers, followed by compilation and fit the model to calculate the 10-fold accuracy.
* N-fold cross validation involves randomly dividing the training set into N different folds/ groups of approximately equal size. The first fold is treated as a validation set and the method is fit on the remaining N-1 folds.
* I have used a fold of 10 (N=10). Hence, 10 folds will be created and for each fold, one accuracy value will be generated, however we should take the mean accuracy of 10 folds.
* The mean accuracy for the 10 folds is 81.64%.



**Transfer Learning:**

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point. Transfer learning is an optimization that allows rapid progress or improved performance when modelling the second task.

This form of transfer learning used in deep learning is called inductive transfer. This is where the scope of possible models (model bias) is narrowed in a beneficial way by using a model fit on a different but related task.



* I have used VGG16 pretrained model to compare the classification performance.
* With the output accuracy of this model, I have compared the accuracy of the previous model and found that the accuracy and efficiency of the model has been improvised a little.
* However, if we increase the epochs, it will be much better.
* Please find below program and outputs.

