**Multi-Class Classification using CNN for custom Dataset.**

Using Multi-class Classification is similar to binary-class classification, with has some changes in the code.

Binary-class Classification:-

Binary-class CNN model contains classification of **2 classes**, Example cat or dog. Provided with set of **images(at least 100 for each class)** of both classes divided into **train and validation folders with classes folders inside each** which are used as input to the CNN model. The last **Dense layer of CNN** model uses “**sigmoid”** activation for processing the output and only one neuron for final output layer, Sigmoid activation classifies image into either **0 or 1** which is either cat or dog.

**tf.keras.layers.Dense(1, activation=’sigmoid’)**

The Binary Class uses **“binary\_crossentropy” loss function**forcalculation of loss value.

model.compile(loss=’binary\_crossentropy’,  
 optimizer=RMSprop(lr=0.001),  
 metrics=[‘acc’])

Multi-Class Classification:-

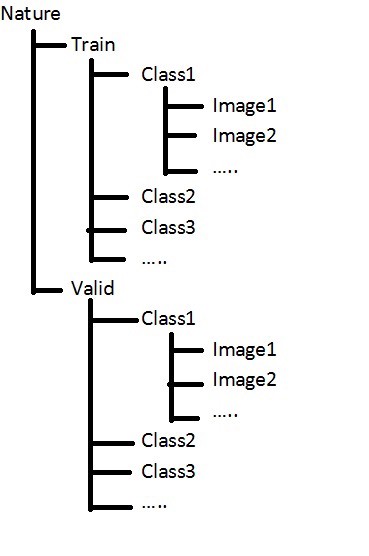
Similar to Binary-class classification Multi-class CNN model has **multiple classes** lets say 6 considering below example. Provided with set of **images(at least 100 for each class)** of both classes divided into **train and validation folders** which are used as input to the CNN model. The last **Dense layer of CNN** model uses **“softmax”** activation for processing the output with **number of classes = number of neurons** for final output layer.

**tf.keras.layers.Dense(6, activation=’softmax’)**

The Multi Class uses **“categorical\_crossentropy” loss function**forcalculation of loss value.

model.compile(loss=’categorical\_crossentropy’,  
 optimizer=RMSprop(lr=0.001),  
 metrics=[‘acc’])

The **distribution of train and validation images** are determined by the number of images for both types can vary form project to project. Generally it done as 80/20 that is 80% images in train folder and 20% in valid folder.

**Structure of Directories/folders**

https://cdn-images-1.medium.com/max/750/1*bzsHaWiWGc5VsTrb9brS9w.png

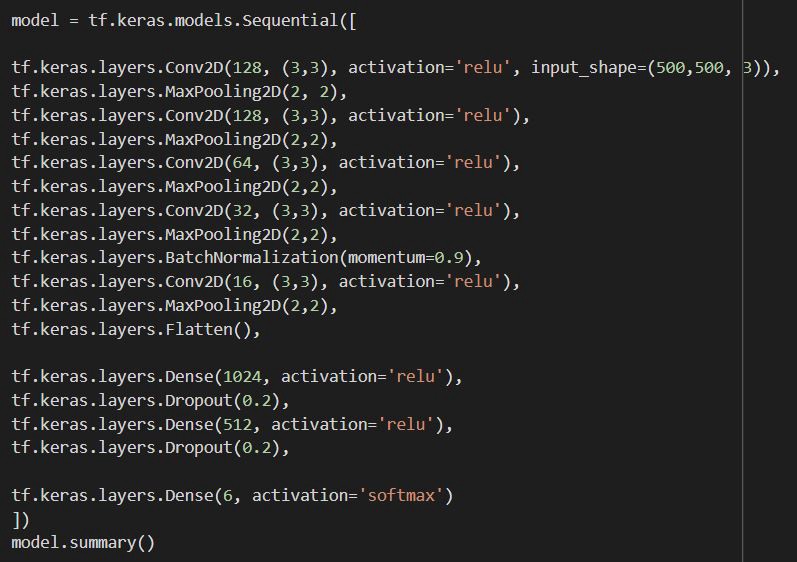
Accessing the directories created, **Only access till train and valid folder.**

Here is a simple **Convolution Neural Network (CNN)** for multi class classification. I developed this **Model** for implementing multi-class classification for Nature images (Landscapes, Ice Landscapes, Sunset, Waterfalls, Forests/ Woods and Beaches). It is a **Simple CNN with some Hyper-Parameters** tuned for better results. I took over 200+ images in total for all **6 Classes(Landscapes, Ice Landscapes, Sunset, Waterfalls, Forests/ Woods and Beaches).**

Image of Class (Beach)Image of Class (Sunset)

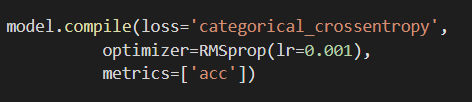
Trained them on local machine as well as on Google colab, used google colab for better performance and more Hyper-Parameter tuning.

Code snippet of CNN model I used:-



This is simple CNN model, you can use **Transfer Learning** and use pre-trained model like **inception model** that has been trained on over 10000 classes and has weights which can used to train your custom model.

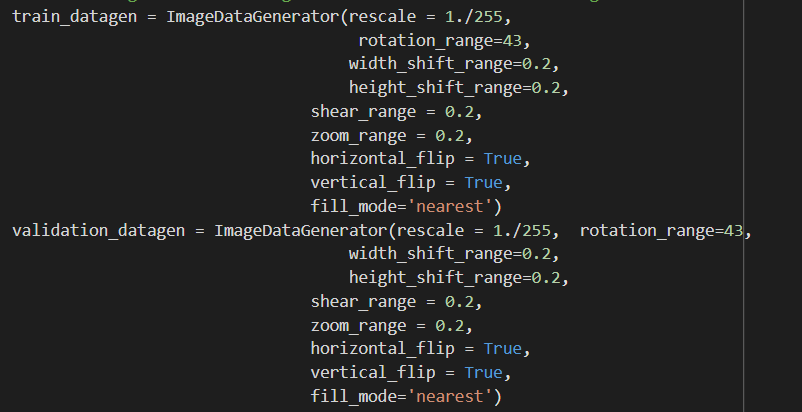
Snippet of compilation of the model:-



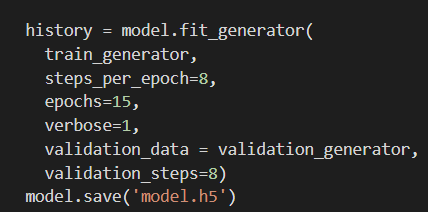
Here the loss Function **“categorical\_crossentropy”** is the major change for classification in multi-class CNN.

Optimizer used with hyper-parameter tuned for custom learning rate.

For better performance you can use Data Augmentation to transform images in code into various transformations (**Rotate, Shear, Zoom, Color change, …**)

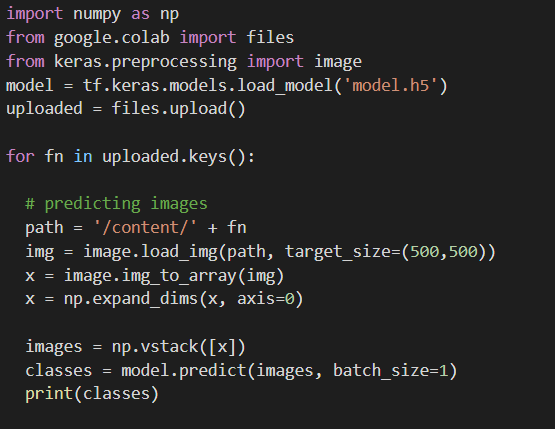


After all the above steps finally we fit the model and start the training.



Here, train the model for all the data processed above.

if you want you can save the **model weights** into a file, so you can use it for predicting your classes later.



Predicting classes is done by loading the model into the python file and then **input image**(**it should not be in train or valid folders**) for the model then predict the image and print classes generated, here after printing only those classes that are present in image will have **value which is closer to 1 or 1** depending on the model’s **Accuracy and loss** on the input image.

Example:class predicted= [0,0,1,0,0,0] here as per my model it will predict that the input image is a landscape image.

That’s all on simple multi-class classification hope this will help you guide through.