20th April 2021

Major Project Inmovidu

FLOWER CLASSIFICATION MODEL USING CNN

Sankalp Singh Bais

INTRODUCTION

AI algorithms will be incorporated into more and more everyday applications. For example, we might want to include an image classifier in a smart phone app. To do this, we'd use a deep learning model trained on hundreds of thousands of images as part of the overall application architecture. A large part of software development in the future will be using these types of models as common parts of applications.

The project which I chose is to create a model which when provide the image of a flower will tell its name. We can imagine like converting it in an app which will use the camera, and when camera is pointed at a flower the species name will popup.

Well, there are already so many apps for doing the same thing. Some models are trained so much on few species of plants so that they can even tell if that plant is healthy or have some disease. Such a model can be seen in the link below

Crop Disease Detection Using Machine Learning and Computer Vision - KDnuggets .

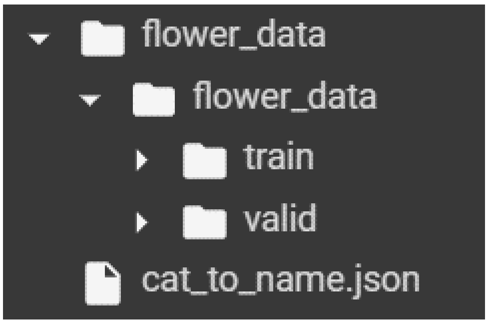
ABOUT MY MODEL AND DATA SET

My model takes in an image of flower from the user and then using CNN it tries to predict its species. This model can differentiate between 102 different species of flower.

Dataset

To train my model I used community provided dataset from the site kaggle.com. Link to my dataset is here. This data set contains two folder train and valid and a cat_to_name.json file.

The two folders are then sub divided into 102 folder labeled from 1 to 102 , each of the folder contains varying



number of images of the flower, but one folder contains only one species of flower.

The cat_to_name.json file contains texts which are like python dictionary that maps the folder number to the species type of flower.

cat_to_name.json

```
{
  "21": "fire lily",
  "3": "canterbury bells",
  "45": "bolero deep blue",
  "1": "pink primrose",
  "34": "mexican aster",
  "27": "prince of wales feathers",
  "7": "moon orchid",
  "16": "globe-flower",
  "25": "grape hyacinth",
  "26": "corn poppy",
  "79": "toad lily",
  "39": "siam tulip",
  "24": "red ginger",
  "67": "spring crocus",
  "35": "alpine sea holly",
```

WORKFLOW

This project has been broken into multiple steps

1. Load and preprocess the image data
2. Creating model
3. Training model
4. Finetuning pretrained resnet34 model

Although each and every step has already been explained in the notebook, so I will explain in short here.

Load and Preprocess image data

1. The first step is always to import the module you think will be needed the modules I imported are

```
# importing libraries which might be required
import torch as T     #for palying with tensors
import torch.nn as nn       #for using nn.Module while creating own model
import torchvision
import torch.nn.functional as F   #torch module containg activtion functions
import torch.optim as optim       #torch module containing optimizers

import numpy as np
import matplotlib       #for plotting graphs
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns      # for plotting bar graph
import os                  # for getting downloaded data dir
from tqdm.notebook import tqdm    # for showing progress bar while training
```

2. Then downloading data from Kaggle

```
#install opendatasets and importing in colab
!pip install opendatasets
import opendatasets as od
```

```
# Downloading dataset from kaggle.com
dataset_url = "https://www.kaggle.com/nunenuh/pytorch-
challange-flower-dataset"
od.download(dataset_url)
```

3. Then now we need to understand the data and create a dataset and dataloader

```
[4] #finding current working directory
    print(os.getcwd(), os.listdir())

    /content ['.config', 'pytorch-challange-flower-dataset', 'sample_data']
```

```
[5] #exploring directory and dataset
    data_dir = "/content/pytorch-challange-flower-dataset/flower_data"
    train_dir = data_dir + "/flower_data/train"
    test_dir = data_dir + "/flower_data/valid"      #using validation data as test data
    classes = os.listdir(train_dir)
    print(len(classes), classes)

    102 ['38', '22', '49', '70', '67', '32', '96', '13', '86', '62', '10', '37', '19', '8
```

In this dataset, we are provided with two folder namely train and valid so , I will be using valid folder as a test folder and will divide the train folder into two parts in the ratio 75:25 for training and validating.

Now Creating dataset

```
#Traning cant be done with real images. The images are needed to transformed into tenso
r for training on pytorch
#The data we're using has already been categorised into folder, so we can use ImageFold
er method to convert img to tensors
# also lets make the size of all images equal if they are not
img_size = 224
imagenet_stats = ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
transforms = transf.Compose([
                            transf.Resize(img_size),      #resizing all image to 224x22
4
                            transf.Pad(8, padding_mode='reflect'),  #adds space of a 4
units to both side of image
                            transf.RandomCrop(img_size),  #randomly selects the part of
 image
                            transf.ToTensor(),            #converting image to tensor
                            transf.Normalize(*imagenet_stats) #normalizes the image
                            ])

dataset = ImageFolder(train_dir, transform = transforms)   #creating the dataset
```

Everything has been explained in detail in notebook too.

Ok so now data has been transformed and I made sure that all images are of size 224 x 224

Now dividing dataset into two parts

```
from torch.utils.data import random_split
train_ds_size = 4914       #using 25 % images out of 6552 image to train
val_ds_size = len(dataset) - train_ds_size #rest as validation

train_ds ,val_ds = random_split(dataset,[train_ds_size, val_ds_size]) #randomly dis
tributing images
```

Great!!, Now lets look at the some of the images

```
#in tensor form
img, label = train_ds[4300]
print(label,mapped_label(label), img)
img.shape
```

```
87 tree mallow tensor([[[-0.3198, -0.3027, -0.4226,  ..., -1.0733, -1.1418, -1.1075],
         [-0.3712, -0.4911, -0.6965,  ..., -1.0390, -1.0562, -0.9877],
         [-0.5082, -0.7822, -1.0219,  ..., -0.9705, -0.9020, -0.7822],
         ...,
         [-1.3644, -1.4672, -1.4500,  ...,  0.8447,  0.7248,  0.6563],
         [-1.3473, -1.3815, -1.4158,  ...,  1.1529,  1.1358,  1.2043],
         [-1.3987, -1.3987, -1.4329,  ...,  1.3584,  1.3413,  1.3413]],

        [[-0.1800, -0.2675, -0.2850,  ..., -0.4951, -0.5126, -0.4601],
         [-0.1975, -0.3375, -0.4426,  ..., -0.4601, -0.4951, -0.5126],
         [-0.3200, -0.4776, -0.6176,  ..., -0.4426, -0.4601, -0.5651],
         ...,
         [-0.6527, -0.7927, -0.7752,  ..., -0.2500, -0.3025, -0.1800],
         [-0.6001, -0.6352, -0.6702,  ..., -0.1275, -0.1099,  0.1001],
         [-0.6176, -0.5826, -0.6352,  ..., -0.0399, -0.0574,  0.0126]],

        [[-0.8458, -0.8981, -0.9678,  ..., -0.8458, -0.9330, -1.0550],
         [-0.9156, -1.0376, -1.2119,  ..., -0.8807, -0.9504, -0.9853],
         [-1.0550, -1.2816, -1.4907,  ..., -0.9156, -0.9156, -0.9156],
         ...,
         [-1.3164, -1.4733, -1.4559,  ...,  0.5136,  0.3916,  0.3393],
         [-1.2816, -1.3513, -1.3687,  ...,  0.7576,  0.7402,  0.8274],
         [-1.3164, -1.2990, -1.3339,  ...,  0.9319,  0.8971,  0.9145]]])
torch.Size([3, 224, 224])
```

Since our tensor is of the form (channel, height, width)

we need to convert it to the form (height, width, channel)

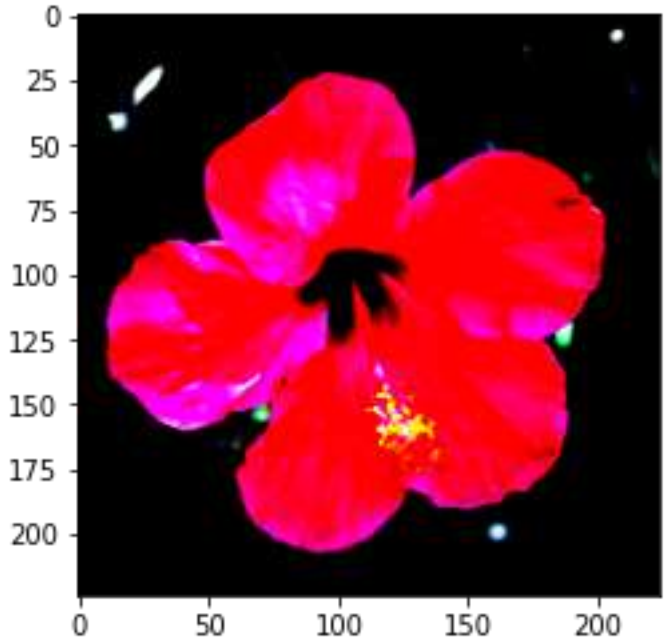
so that pyplot can display it

A user defined function we do this all

```
def show_img(ds, ind):      #takes dataset and image index number as input
  """
  Pass the dataset and the index of the image.
  """
  img, label = ds[ind]      #here image is storing the image but label is storing t
he index number of the category from the classes.
```

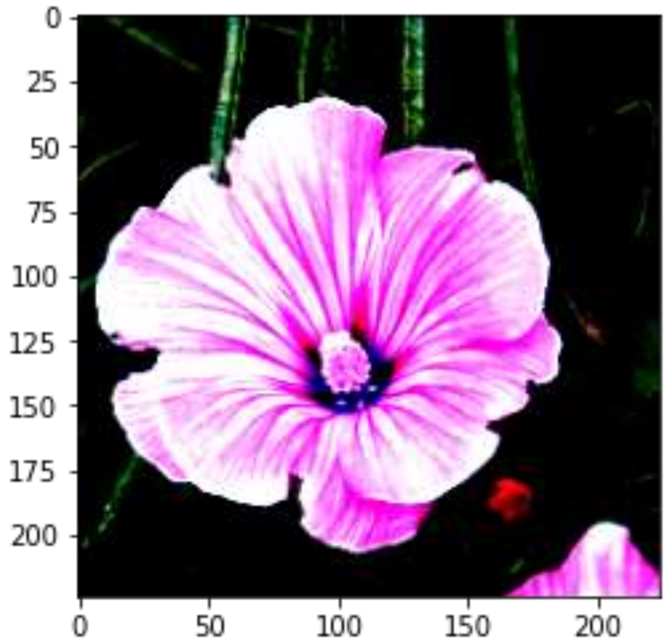
```
  cls_ind = dataset.classes[label]
  print(f"Label : {cls_ind}: {mapped_label(label)}")
  plt.imshow(img.permute(1, 2, 0))

show_img(train_ds,2)
```

```
Label : 83: hibiscus
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



Lets look one more image,

```
show_img(train_ds, 4300)
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Label : 86: tree mallow
```



Now its time to create DataLoaders

```
#defining data loader sizes
train_dl_size = len(train_ds)
val_dl_size = len(val_ds)
print(f"Train size : {train_dl_size}, Valid Size : {val_dl_size}")

Train size : 4914, Valid Size : 1638
```

```
[19] #creating dataloader
     from torch.utils.data.dataloader import DataLoader
     train_dl = DataLoader(train_ds, batch_size = 63 , shuffle = True,
                           num_workers = 2, pin_memory =True )
     val_dl = DataLoader(val_ds, batch_size = 63,num_workers = 2,
                         pin_memory =True )
```

Lets check how a batch from training data set looks,

```
#visualising dataloader batches
from torchvision.utils import make_grid
def disp_batches(dl):
  for img, label in dl:
    for i in label:
      print(dl.dataset.dataset.classes[i],cat_to_name_dict[dl.dataset.dataset.class
es[i]], end = ", ")
    fig, ax = plt.subplots(figsize = (15, 15))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(img, padding = 10).permute(1, 2, 0 ))
    break

disp_batches(train_dl)
```

```
29 artichoke, 31 carnation, 11 snapdragon, 57 gaura, 62 japanese anemone, 94 foxglove,
61 cautleya spicata, 12 colt's foot, 89 watercress, 31 carnation, 78 lotus lotus, 8
bird of paradise, 23 fritillary, 89 watercress, 44 poinsettia, 65 californian poppy, 3
canterbury bells, 39 siam tulip, 12 colt's foot, 41 barbeton daisy, 101 trumpet
creeper, 51 petunia, 21 fire lily, 70 tree poppy, 83 hibiscus, 45 bolero deep blue, 72
azalea, 74 rose, 85 desert-rose, 18 peruvian lily, 66 osteospermum, 77 passion flower,
3 canterbury bells, 99 bromelia, 70 tree poppy, 55 pelargonium, 39 siam tulip, 77
passion flower, 65 californian poppy, 99 bromelia, 92 bee balm, 43 sword lily, 86 tree
mallow, 97 mallow, 30 sweet william, 77 passion flower, 33 love in the mist, 29
artichoke, 18 peruvian lily, 76 morning glory, 73 water lily, 43 sword lily, 26 corn
poppy, 37 cape flower, 37 cape flower, 86 tree mallow, 79 toad lily, 77 passion flower,
5 english marigold, 89 watercress, 91 hippeastrum, 47 marigold, 77 passion flower,
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

The color of these images are due to the normalization applied above while creating a data set

Lets move on to the next step.

Creating Own Model

Here I tried to create my own model.

First I defined the , functions which might be required while training a model

```
class Model_Base(nn.Module):
  def training_step(self, batch):
    """
    Passes the training dataloader's batch through Network once and then calculates
    and  returns the loss
    """
    images, labels = batch
    out = self(images)                  # Outputs/Predictions returned by network
    loss = F.cross_entropy(out, labels) # Calculate loss
    return loss

  def validation_step(self, batch):
    """
    Calculates and returns the loss and validation accuracy
    """
    tot_correct = 0
    images, labels = batch
    out = self(images)                     # Generate predictions
    loss = F.cross_entropy(out, labels)    # Calculate loss
    acc = accuracy(out, labels)            # Calculate accuracy
    tot_correct = get_num_correct(out, labels)    #get total number of corect
    return {"val_loss": loss.detach(), "val_acc": acc, "tot_correct": tot_correct}

  def validation_epoch_end(self, outputs):
    """
    Calculate and return validation loss and accuracy after end of each epoch
    """
    tot_correct = 0
    batch_losses = [x['val_loss'] for x in outputs]
    epoch_loss = T.stack(batch_losses).mean()    # Combine losses
    batch_accs = [x['val_acc'] for x in outputs]
    epoch_acc = T.stack(batch_accs).mean()       # Combine accuracies
    tot_correct = sum([x['tot_correct'] for x in outputs])
    return {"val_loss": epoch_loss.item(), "val_acc": epoch_acc.item(), "tot_correc
t": tot_correct}

  def epoch_end(self, epoch, result):
    """
    Calculate and return Training loss after end of each epoch
    """
```

```
    print(f"Epoch {epoch+1}, Train loss : {result['train_loss']}, Val. Loss : {resu
lt['val_loss']}, val_acc : {result['val_acc']}, Total_Correct : {result['tot_correc
t']}")
```

Then Defining the Neural Network

```
#defining neural network model
class Flower_Classification_Model(Model_Base):
  """
  Pass colour channels and total number of classes as an argument.
  The Network consists of 6 Conv2d layers and  4 linear layers
  """
  def __init__(self,input_channels,output_classes):
    self.input_channels = input_channels
    self.output_classes = output_classes

    super().__init__()                              #defining Network layers
    self.network= nn.Sequential(

            nn.Conv2d(self.input_channels,64,kernel_size = 3, padding=1),   #conv l
ayer 1
            nn.ReLU(),         #activation function
            nn.Conv2d(64,72,kernel_size = 3, padding=1),#conv layer 2
            nn.ReLU(),
            nn.MaxPool2d(2,2),  # 72x 112 x 112          #Exatracting features usin
g max pool

            nn.Conv2d(72, 80, kernel_size=3, stride=1, padding=1),    #conv layer 3
            nn.ReLU(),
            nn.Conv2d(80, 88, kernel_size=3, stride=1, padding=1),    #conv layer 4
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 88 x 56 x 56

            nn.Conv2d(88, 96, kernel_size=3, stride=1, padding=1),      #conv layer
 5
            nn.ReLU(),
            nn.Conv2d(96, 104, kernel_size=3, stride=1, padding=1,),    #conv layer
 6
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 104 x 28 x 28

            nn.Conv2d(104,112, kernel_size=3, stride=1, padding=1),      #conv laye
r 7
            nn.ReLU(),
            nn.Conv2d(112, 120, kernel_size=3, stride=1, padding=1,),    #conv laye
r 8
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 120 x 14 x 14
```

```
            nn.Flatten(),           #Flatting the image tensor to pass it to linear
 layer
            nn.Linear(in_features = 120*14*14, out_features = output_classes),
    )

  def forward(self, t):             #this method is required while creating a model
    return self.network(t)
```

Now defining a function to train and validate a model

```
@T.no_grad() #turning off pytorch's gradient feature

def evaluate(model, dloader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in dloader]
    return model.validation_epoch_end(outputs)

def fit(model, train_dloader, val_dloader, epochs = 1, lr = 0.01):
  """
  Takes in model , train dataloader, validation dataloader, epochs, learning rate
  and the trains the model for epoch times with provided learning rate
  """
  history = []    #list to store the result after each epoch
  optimizer = optim.Adam(model.parameters(), lr)

  for epoch in range(epochs):
    print(f"Epoch : {epoch+1} of {epochs}:\nTraining ")
    #Training Model
    model.train()
    train_losses = []
    for batch in tqdm(train_dloader):
      loss = model.training_step(batch)
      train_losses.append(loss)
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
    #Validating after each epoch
    print("Validating ")
    result = evaluate(model, val_dloader)
    print("\n\t")
    result["train_loss"] = T.stack(train_losses).mean().item()
    model.epoch_end(epoch, result)
    history.append(result)
  return history
```

The codes above are copied from note book, so check the notebook for more details

Training the model

Now here comes the training part,

First , an instance of the model is created then its accuracy is checked before training and then it is trained and validated using test data .

```
input_channel = 3
output_classes = len(classes)
fcmodel =Flower_Classification_Model(input_channel,output_classes) #creating an
instance of the model
fcmodel =to_device(fcmodel, device)
print(fcmodel)
```

```
Flower_Classification_Model(
  (network): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(64, 72, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(72, 80, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU()
    (7): Conv2d(80, 88, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU()
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(88, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU()
    (12): Conv2d(96, 104, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU()
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (15): Conv2d(104, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (16): ReLU()
    (17): Conv2d(112, 120, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU()
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (20): Flatten(start_dim=1, end_dim=-1)
    (21): Linear(in_features=23520, out_features=102, bias=True)
  )
)
```

The above code's output is what we call a model architecture.

My model has 8 convolutional layers and 1 linear network , i used Adam optimizer and for calculating loss , its Cross Entropy.

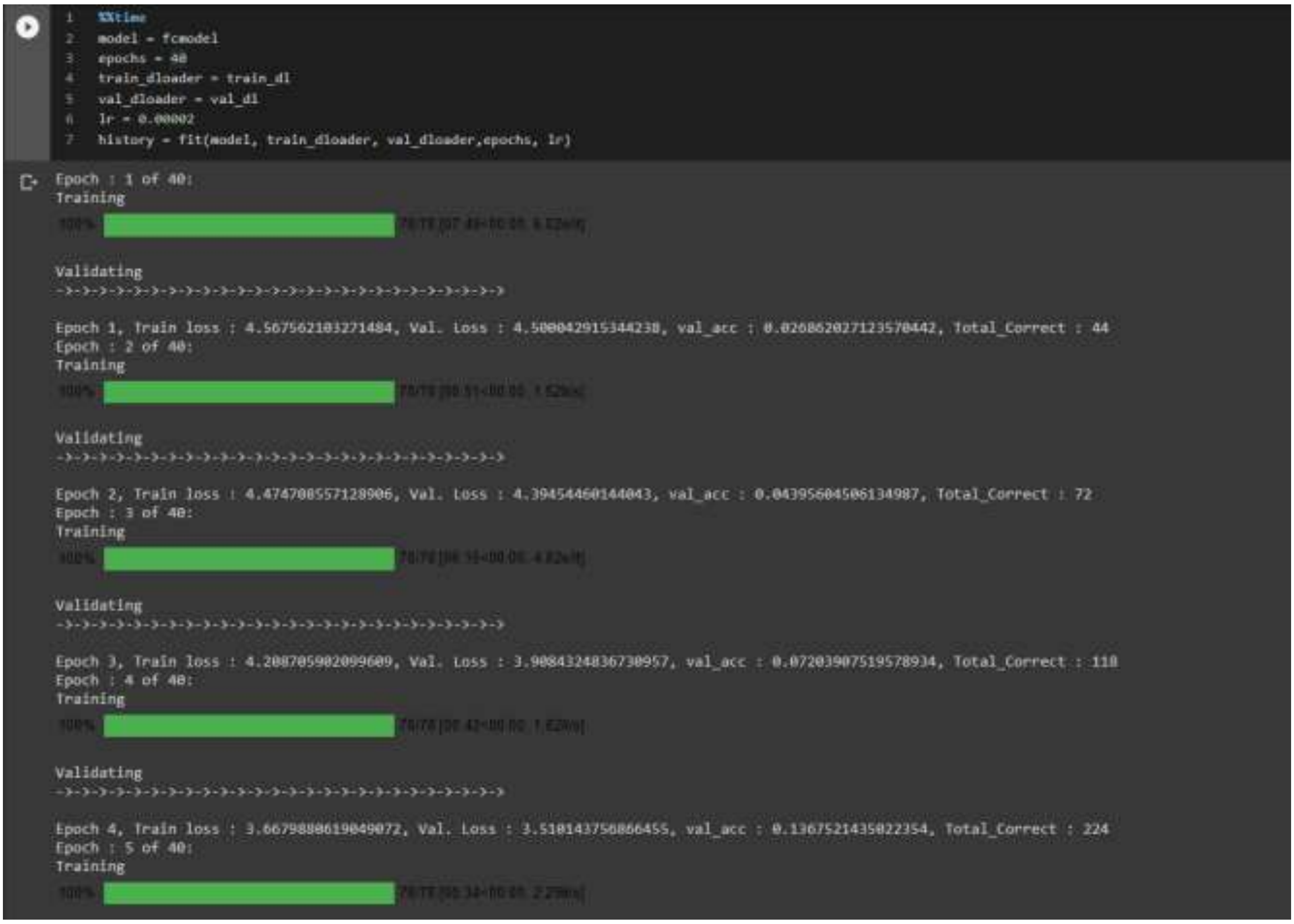
Before training lets check the accuracy of the model.

```
%%time
evaluate(fcmodel, val_dl)
->->->->->->->->->->->->->->->->->->->->->->->->->->->->->CPU times: user 1.25 s, sys: 907
ms, total: 2.16 s
```

```
Wall time: 12.1 s
{'tot_correct': 35,
 'val_acc': 0.021367521956562996,
 'val_loss': 4.624234676361084}
```

As expected the accuracy is very bad its only 2 percent.

Now lets train the model



```
1 %%time
2 model = fcmodel
3 epochs = 40
4 train_dloader = train_dl
5 val_dloader = val_dl
6 lr = 0.00002
7 history = fit(model, train_dloader, val_dloader,epochs, lr)
```

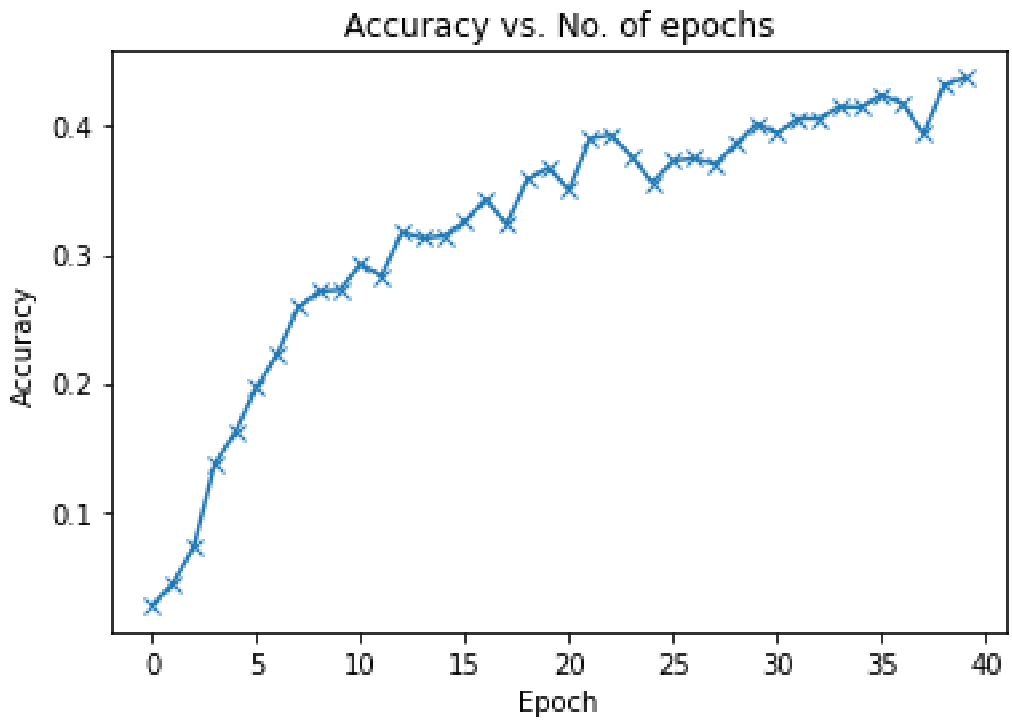
```
Epoch : 1 of 40:
Training

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 1, Train loss : 4.567562103271484, Val. Loss : 4.500042915344238, val_acc : 0.026862027123570442, Total_Correct : 44
Epoch : 2 of 40:
Training

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 2, Train loss : 4.474708557128906, Val. Loss : 4.39454460144043, val_acc : 0.04395604506134987, Total_Correct : 72
Epoch : 3 of 40:
Training

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 3, Train loss : 4.208705902099609, Val. Loss : 3.9084324836730957, val_acc : 0.07203907519578934, Total_Correct : 118
Epoch : 4 of 40:
Training

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 4, Train loss : 3.6679880619049072, Val. Loss : 3.510143756866455, val_acc : 0.1367521435022354, Total_Correct : 224
Epoch : 5 of 40:
Training
```

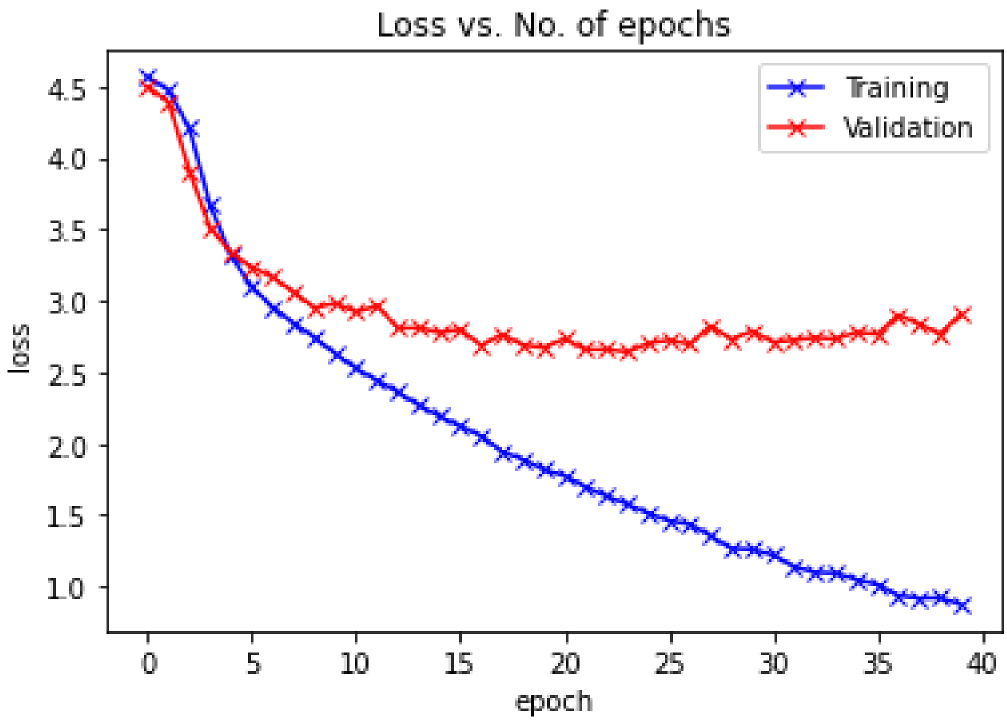
-
-
-

```
Epoch 36, Train loss : 1.0130548477172852, Val. Loss : 2.766054153442383, val_acc : 0.42429792881011963, Total_Correct : 695
Epoch : 37 of 40:
Training
100% 78/78 [00:59<00:00, 1.32it/s]

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 37, Train loss : 0.9384902715682983, Val. Loss : 2.9000091552734375, val_acc : 0.4175823926925659, Total_Correct : 684
Epoch : 38 of 40:
Training
100% 78/78 [00:33<00:00, 2.30it/s]

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 38, Train loss : 0.9162383675575256, Val. Loss : 2.8385674953460693, val_acc : 0.3937729299068451, Total_Correct : 645
Epoch : 39 of 40:
Training
100% 78/78 [00:33<00:00, 2.32it/s]

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 39, Train loss : 0.9208137392997742, Val. Loss : 2.7672595977783203, val_acc : 0.43223443627357483, Total_Correct : 708
Epoch : 40 of 40:
Training
100% 78/78 [00:33<00:00, 2.30it/s]

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 40, Train loss : 0.8747316002845764, Val. Loss : 2.906343460083008, val_acc : 0.4377289414405823, Total_Correct : 717
CPU times: user 1min 43s, sys: 1min 2s, total: 2min 46s
Wall time: 30min 38s
```

Now after 40 epochs the accuracy of my model is only 43.77% which is very bad because it will do a lot of mis-predictions

Graphically validation accuracy per epochs has been represented below





Now, let's see what's the accuracy of my model on test dataset and then after uploading an image what predictions it will be making.

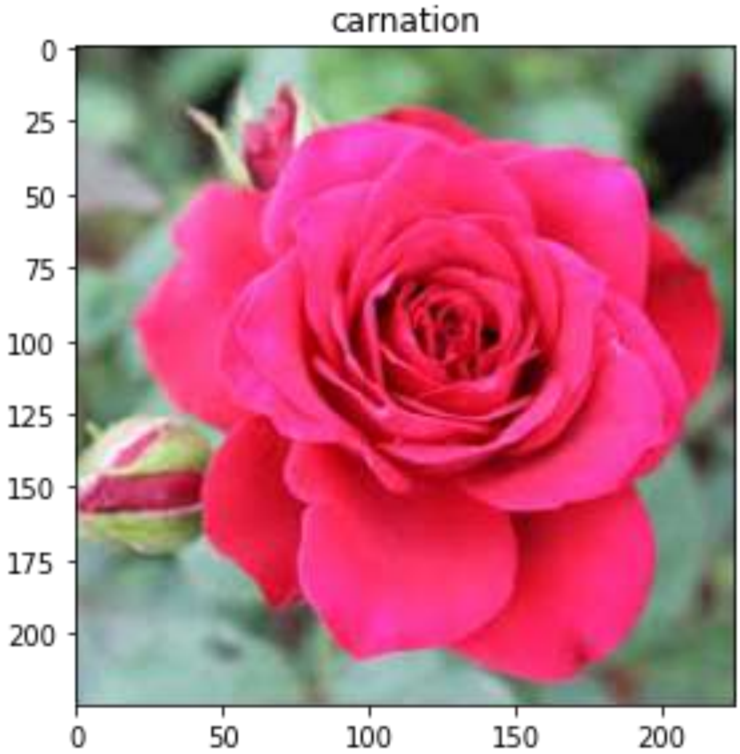
```
evaluate(fcmodel, test_dl)
```

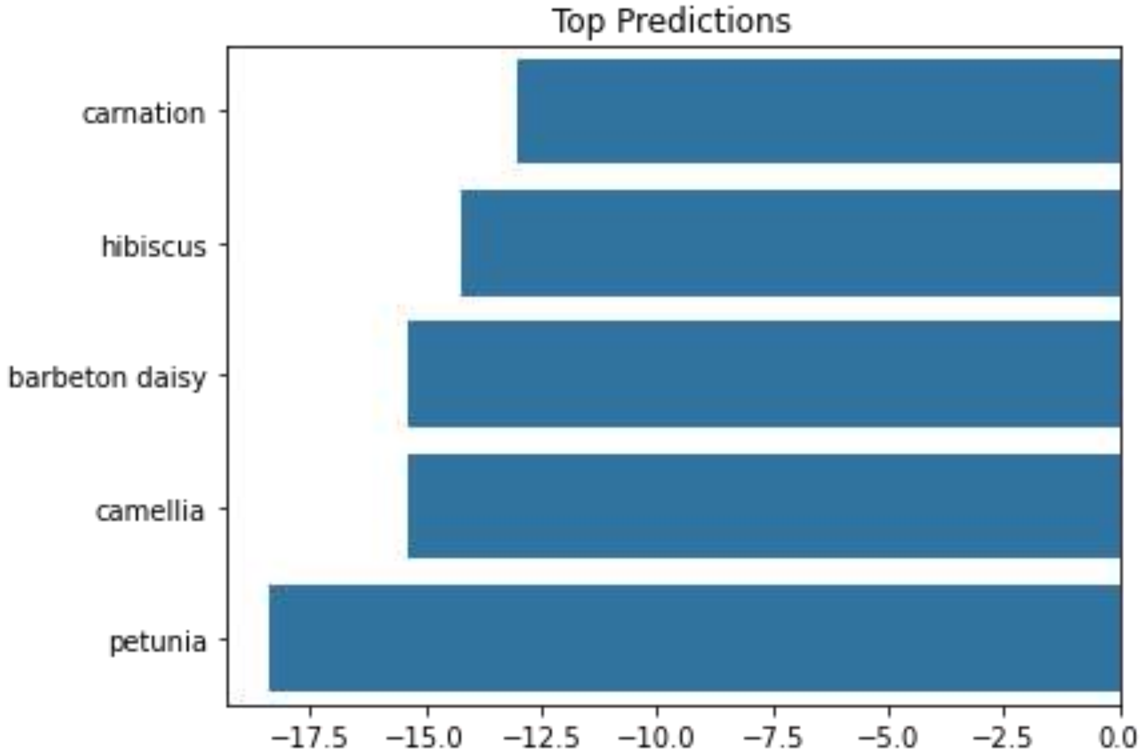
```
->->->->->->->->->->{'tot_correct': 352,
 'val_acc': 0.3834422826766968,
 'val_loss': 2.8641252517700195}
```

So finally the accuracy of custom model is 38.34%

Now I uploaded and image of a Rose and the top 5 predictions made by my model are given below

(code is in the notebook)





So due to bad training it predicted the wrong label.

Using resnet34 model

Since it was my first time creating a model even if the accuracy of the above model was not that much good, for me it was exciting ,so for the sake of this project I used resnet34 model too

Which in the end gave good results

About resnet34 model

Resnet34 is a 34 layer convolutional neural network that can be utilized as a state-of-the-art image classification model. This is a model that has been pre-trained on the ImageNet dataset--a dataset that has 100,000+ images across 200 different classes. However, it is different from traditional neural networks in the sense that it takes residuals from each layer and uses them in the subsequent connected layers (similar to residual neural networks used for text prediction).

Architecture of resnet34 model used in notebook

```
PreTrainedClassificationModel(
  (network): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
      (3): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (3): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
      (4): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (5): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
        )
      )
      (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
      (fc): Linear(in_features=512, out_features=102, bias=True)
    )
)
```

For tuning the resnet34 model as per my need, I had to define some functions to fit it., the complete code is in the notebook, the fit function is given below

```
def fit_one_cycle(epochs, max_lr, model, train_loader, val_loader,
                  weight_decay=0, grad_clip=None, opt_func=optim.SGD):
    if T.cuda.is_available():
      T.cuda.empty_cache()

    history = []
    optimizer = opt_func(model.parameters(), max_lr, weight_decay=weight_decay)# Set up
 custom optimizer with weight decay

    sched = T.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epochs,
                                                steps_per_epoch=len(train_loader))     #
 Set up one-cycle learning rate scheduler

    for epoch in range(epochs):
        print(f"Epoch : {epoch+1} of {epochs}:\nTraining")
        # Training Phase
        model.train()
        train_losses = []
        lrs = []
        for batch in tqdm(train_loader):
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

            # Gradient clipping
            if grad_clip:
                nn.utils.clip_grad_value_(model.parameters(), grad_clip)

            optimizer.step()
            optimizer.zero_grad()

            # Record & update learning rate
            lrs.append(get_lr(optimizer))
            sched.step()
        # Validation phase
        print("\nValidating ")
        result = evaluate(model, val_loader)
        print("\n\t")
```

```
        result['train_loss'] = T.stack(train_losses).mean().item()
        result['lrs'] = lrs
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

Creating an instance of the resnet34 model

```
output_classes = len(dataset.classes)
print(output_classes)
fcmodel2 = PreTrainedClassificationModel(output_classes,pretrained = True)
fcmodel2 = to_device(fcmodel2, device)
```

Checking accuracy before tuning.

```
history2 = [evaluate(fcmodel2, val_dl)]
print(history2)
```

```
->->->->->->->->->->->->->->->->->->->->->->->->->->->[{'val_loss':
4.918607234954834, 'val_acc': 0.007326007820665836, 'tot_correct': 12}]
```

The accuracy is low only 0.7 %.

Let's train this model.

```
[77]
epochs = 10
max_lr = 0.01
grad_clip = 0.1
weight_decay = 1e-4
opt_func = optim.Adam
```

```
%%time
history2 += fit_one_cycle(epochs, max_lr, fcmodel2, train_dl, val_dl,
                          grad_clip=grad_clip, weight_decay=weight_decay,
                          opt_func = opt_func)
```

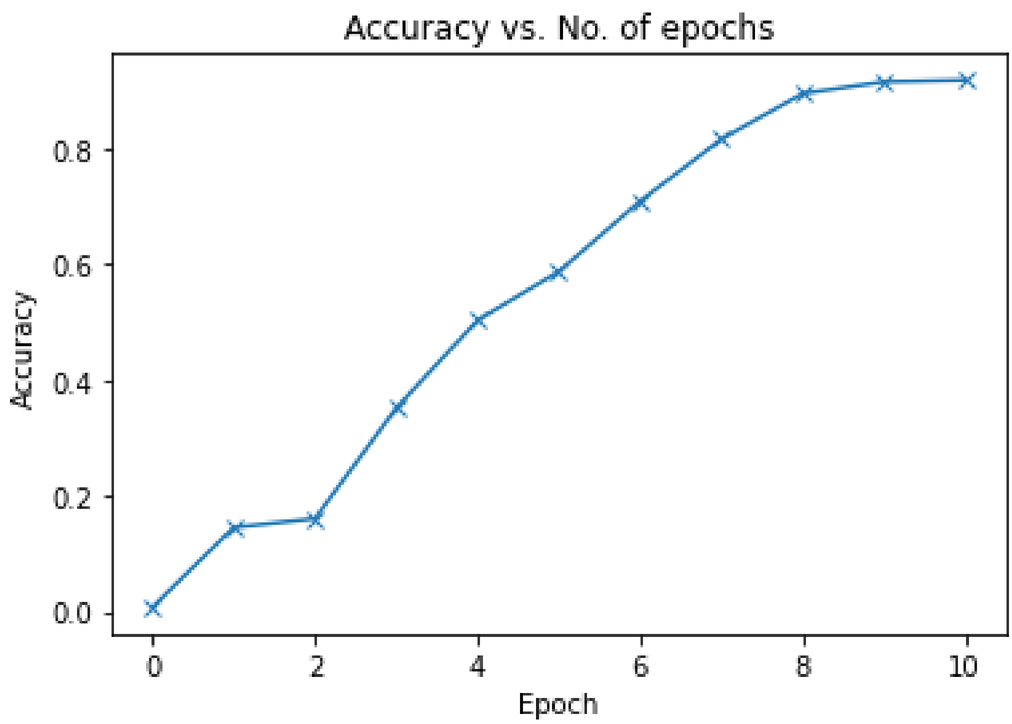
```
Epoch : 1 of 10:
Training
100% 78/78 [04:02<00:00, 3.11s/it]

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 1, Train loss : 1.8736412525177002, Val. Loss : 6.946482181549072, val_acc : 0.14529913663864136, Total_Correct : 238
Epoch : 2 of 10:
Training
100% 78/78 [03:13<00:00, 2.48s/it]

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 2, Train loss : 2.383558750152588, Val. Loss : 7.000977516174316, val_acc : 0.160561665892601, Total_Correct : 263
Epoch : 3 of 10:
Training
100% 78/78 [02:23<00:00, 1.84s/it]

Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 3, Train loss : 1.9479068517684937, Val. Loss : 3.0207314491271973, val_acc : 0.35164836049079895, Total_Correct : 576
Epoch : 4 of 10:
Training
100% 78/78 [01:34<00:00, 1.21s/it]
```

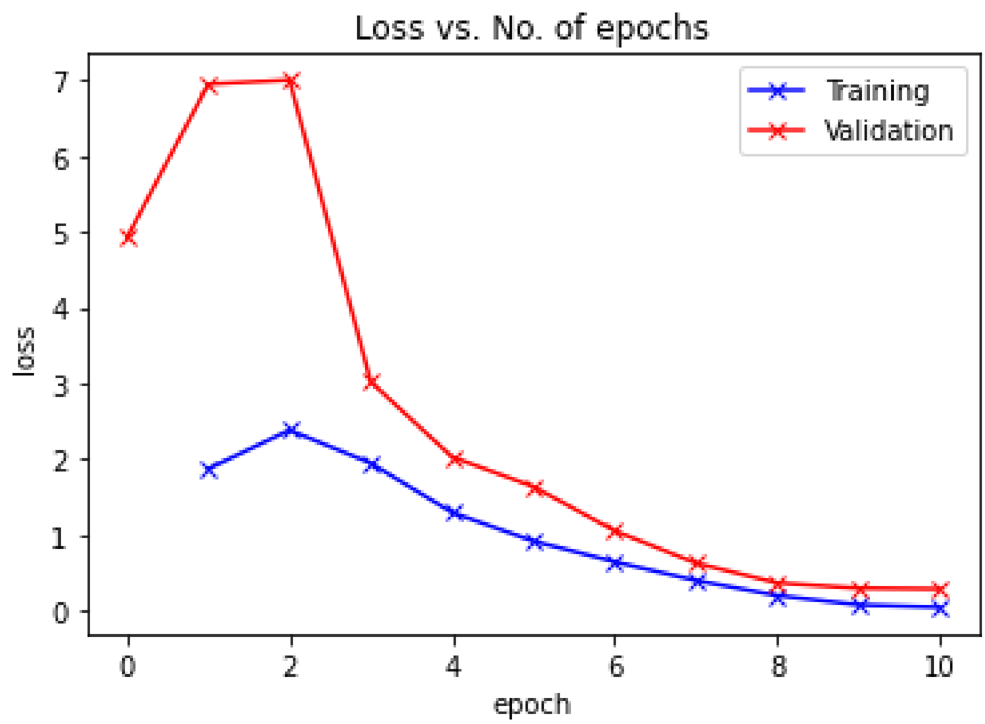
.

```
Epoch 6, Train loss : 0.6462797522544861, Val. Loss : 1.055501937866211, val_acc : 0.708791196346283, Total_Correct : 1161
Epoch : 7 of 10:
Training
100% 78/78 [00:38<00:00, 2.05it/s]


Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 7, Train loss : 0.4004979133605957, Val. Loss : 0.6292619705200195, val_acc : 0.8162393569946289, Total_Correct : 1337
Epoch : 8 of 10:
Training
100% 78/78 [00:37<00:00, 2.06it/s]


Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 8, Train loss : 0.20121198892593384, Val. Loss : 0.36766088008880615, val_acc : 0.8949938416481018, Total_Correct : 1466
Epoch : 9 of 10:
Training
100% 78/78 [00:38<00:00, 2.04it/s]


Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 9, Train loss : 0.07862719893455505, Val. Loss : 0.29787442088127136, val_acc : 0.9151403903961182, Total_Correct : 1499
Epoch : 10 of 10:
Training
100% 78/78 [00:37<00:00, 2.07it/s]


Validating
->->->->->->->->->->->->->->->->->->->->->->->->->->->

Epoch 10, Train loss : 0.04621998965740204, Val. Loss : 0.2880733013153076, val_acc : 0.9181929230690002, Total_Correct : 1504
CPU times: user 1min 10s, sys: 38.5 s, total: 1min 49s
Wall time: 8min 16s
```

Wow!!! The accuracy of the resnet34 model is 91.81%

Accuracy and loss graphs per epochs.





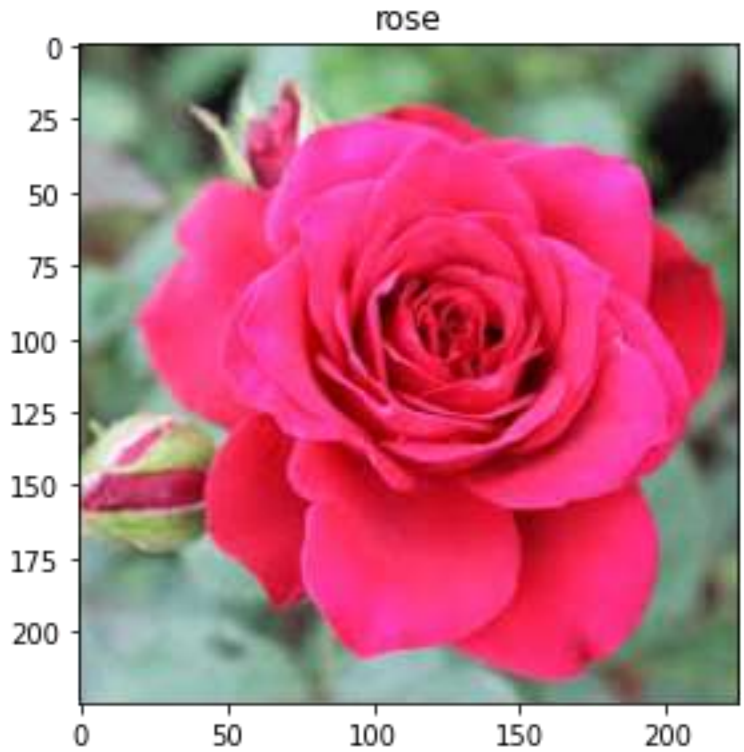
Checking accuracy on test dataset

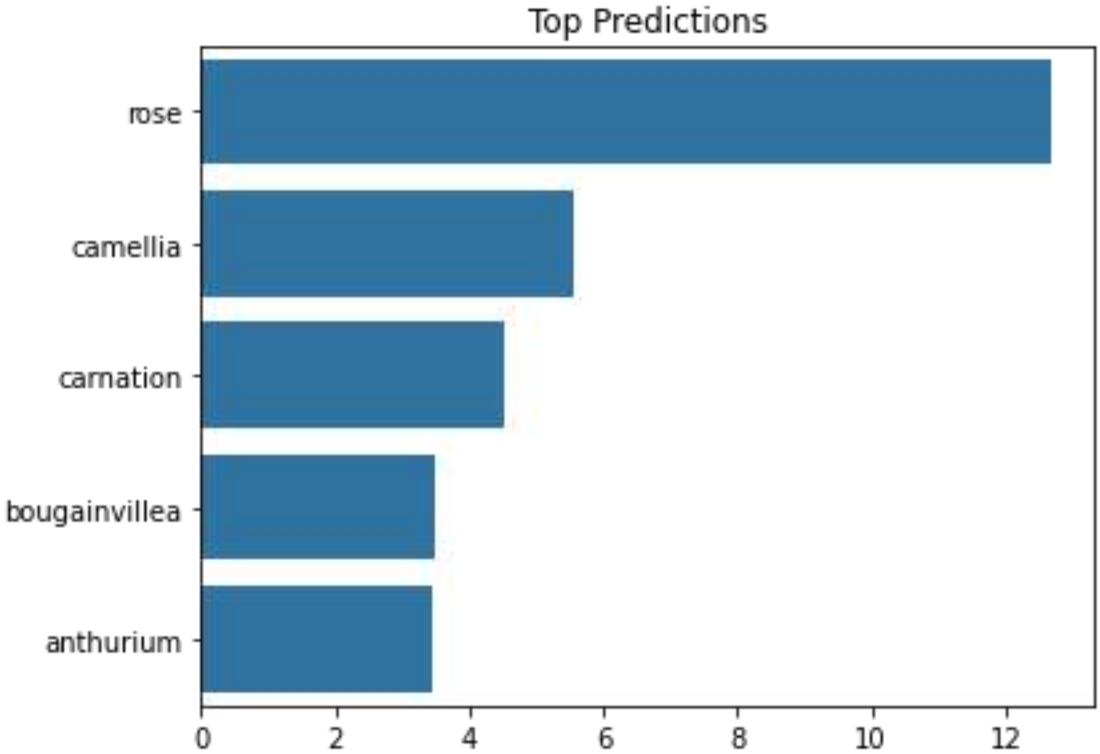
```
evaluate(fcmodel2,test_dl)
```

```
->->->->->->->->->->{'tot_correct': 752,
'val_acc': 0.9281045198440552,
'val_loss': 0.2897782623767853}
```

On test data set it gave the accuracy of 92.81 %

And the result of the image I uploaded is below”





Finally, while custom model took 30 mins to train for 40 epochs and gave only 38.34% accuracy on test data , the resnet34 model gave 92% accuracy on same test data , and it just took 10 epochs and less than 10 minutes

Why is this so? What could be the reason ?

Well, what I think is this model is denser than my model, and has already been trained over a very large data set when it was created, so its weight are way more flexible then my model, with a little bit of adjustments it can give really high accuracy about 97%, while when creating a model from the scratch we have to play a lot with hyper parameters.

References:

1. Kaggle.com
2. Towards Data science
3. Analytics Vidhya
4. Some random github repositories
5. Deeplizard.com
6. And some random youtube videos