**Assignment -3**

Python Programming

|  |  |
| --- | --- |
| Assignment Date | 30 September 2022 |
| Student Name | Shiny.R |
| Student Roll Number | 311419205036 |
| Maximum Marks | 2 Marks |

**Problem Statement :- Build CNN Model for Classification Of Flowers**

* Download the Dataset : Dataset
* Image Augmentation
* Create Model
* Add Layers (Convolution,MaxPooling,Flatten,Dense-(Hidden
* Layers),Output))
* Compile The Model
* Fit The Model
* Save The Model
* Test The Model

**Solution:**

# Used for manipulating directory paths

import os

import shutil

from os.path import isfile, join, abspath, exists, isdir, expanduser

from os import listdir, makedirs, getcwd, remove

from pathlib import Path

# Data visualisation

import pandas as pd

import seaborn as sns

from PIL import Image

from skimage.io import imread

import cv2

from tensorflow.keras.utils import to\_categorical

# Specifically for manipulating zipped images and getting numpy arrays of pixel values of images.

import matplotlib.pyplot as plt

import matplotlib.image as mimg

import numpy as np

# Plotting library

from mpl\_toolkits.mplot3d import Axes3D  # needed to plot 3-D surfaces

# dl libraries specifically for CNN

from keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.utils import load\_img

from tensorflow.keras.utils import img\_to\_array

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

from keras import optimizers

# Tells matplotlib to embed plots within the notebook

%matplotlib inline

import math

# Dataset folder

flowersPath = Path('C:/Users/sri nandhini/Downloads/Flowers-Dataset/flowers')

# Each species of flower is contained in a separate folder, & this is to get all the sub-directories

flowers = os.listdir(flowersPath)

print("Number of types of flowers: ", len(flowers))

print("Types of flowers: ", flowers)

# A list which contains tuples, the type of flower and the corresponding image path

flowersList = []

for species in flowers:

    # Get all the file names

    allFlowers = os.listdir(flowersPath / species)

    # Add them to the list

    for flower in allFlowers:

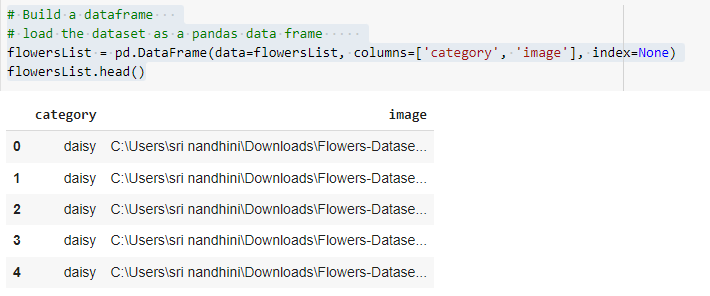
        flowersList.append((species, str(flowersPath /species) + '/' + flower))

# Build a dataframe

# load the dataset as a pandas data frame

flowersList = pd.DataFrame(data=flowersList, columns=['category', 'image'], index=None)

flowersList.head()



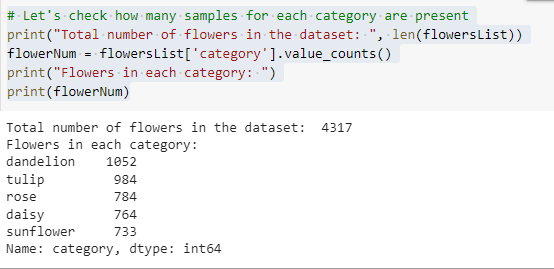
# Let's check how many samples for each category are present

print("Total number of flowers in the dataset: ", len(flowersList))

flowerNum = flowersList['category'].value\_counts()

print("Flowers in each category: ")

print(flowerNum)

****

# A list for storing names of some random samples from each category

RanSamples = []

# Get samples fom each category

for category in flowerNum.index:

    samples = flowersList['image'][flowersList['category'] == category].sample(4).values

    for sample in samples:

        RanSamples.append(sample)

# Plot the samples

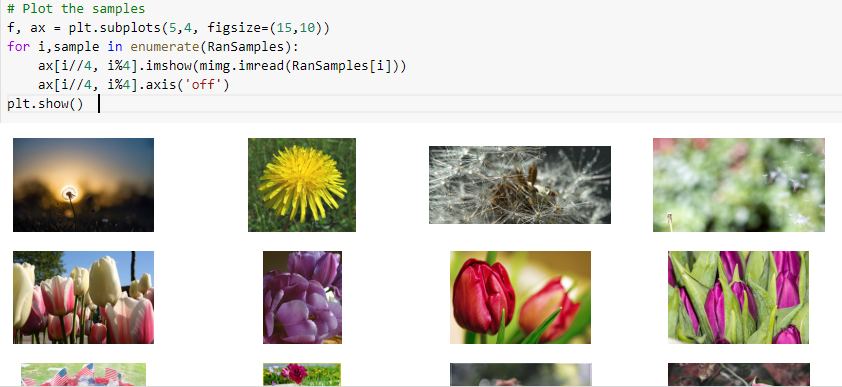
f, ax = plt.subplots(5,4, figsize=(15,10))

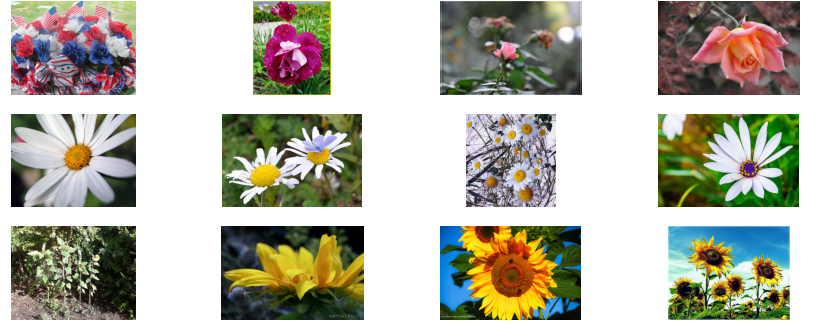
for i,sample in enumerate(RanSamples):

    ax[i//4, i%4].imshow(mimg.imread(RanSamples[i]))

    ax[i//4, i%4].axis('off')

plt.show()





# Let's do some visualization and see how many samples we have for each category

f, axe = plt.subplots(1,1,figsize=(14,6))

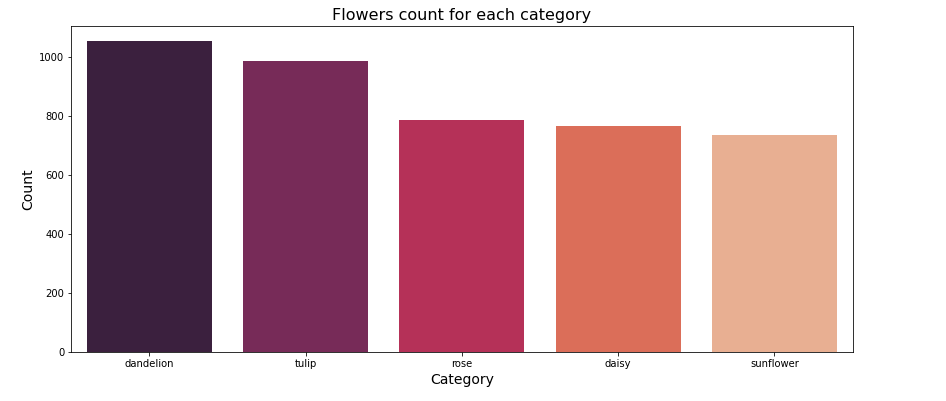
sns.barplot(x = flowerNum.index, y = flowerNum.values, ax = axe, palette="rocket")

axe.set\_title("Flowers count for each category", fontsize=16)

axe.set\_xlabel('Category', fontsize=14)

axe.set\_ylabel('Count', fontsize=14)

plt.show()



# Make directory 'test', with 2 sub directories, 'trainDir', & 'validDir'

trainDir = './test/trainDir'

valDir = './test/valDir'

# test\_dir = './test/test\_dir'

def create\_directory(dirName):

    if os.path.exists(dirName):

        shutil.rmtree(dirName)

    os.makedirs(dirName)

    # Inside the trainDir & valDir sub-directories, sub-directories for each flower is created

    for flower in flowers:

        os.makedirs(os.path.join(dirName, flower))

create\_directory(trainDir)

create\_directory(valDir)

# lists for training & validation image & label

trainImg = []

trainLabel = []

validImg = []

validLabel = []

# for copying 100 samples to the validation dir & others to the train dir

for flower in flowerNum.index:

    samples = flowersList['image'][flowersList['category'] == flower].values

    diffPics = np.random.permutation(samples)

    for i in range(100):

        name = diffPics[i].split('/')[-1]

        shutil.copyfile(diffPics[i],'./test/valDir/' + str(flower) + '/'+ name)

        try:

            # add image to list

            img = plt.imread('./test/valDir/' + str(flower) + '/'+ name)

            #resize all of the image to 150\*150

            img = cv2.resize(img,(150,150))

            validImg.append(np.array(img))

            # add label to list

            if (str(flower)=="dandelion"):

                validLabel.append(0)

            elif (str(flower)=="tulip"):

                validLabel.append(1)

            elif (str(flower)=="rose"):

                validLabel.append(2)

            elif (str(flower)=="daisy"):

                validLabel.append(3)

            elif (str(flower)=="sunflower"):

                validLabel.append(4)

        except Exception as e:

            None

    for i in range(101,len(diffPics)):

        name = diffPics[i].split('/')[-1]

        shutil.copyfile(diffPics[i],'./test/trainDir/' + str(flower) + '/' + name)

        try:

            # add image to list

            img = plt.imread('./test/trainDir/' + str(flower) + '/' + name)

            #resize all of the image to 150\*150

            img = cv2.resize(img,(150,150))

            trainImg.append(np.array(img))

            # add label to list

            if (str(flower)=="dandelion"):

                trainLabel.append(0)

            elif (str(flower)=="tulip"):

                trainLabel.append(1)

            elif (str(flower)=="rose"):

                trainLabel.append(2)

            elif (str(flower)=="daisy"):

                trainLabel.append(3)

            elif (str(flower)=="sunflower"):

                trainLabel.append(4)

        except Exception as e:

            None

# Let computer read the 5 category

validLabel = to\_categorical(validLabel,num\_classes = 5)

trainLabel = to\_categorical(trainLabel,num\_classes = 5)

print(validLabel)

print(trainLabel)

# Make new test and validation images as pixcel

validImg=np.array(validImg)

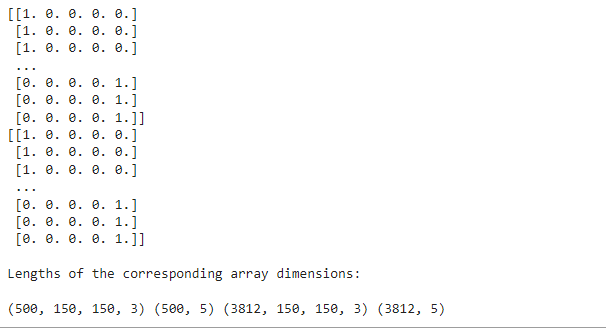
validImg=validImg/255

trainImg=np.array(trainImg)

trainImg=trainImg/255

print("\nLengths of the corresponding array dimensions: \n")

print(np.shape(validImg),np.shape(validLabel),np.shape(trainImg),np.shape(trainLabel))



def createModel():

    model = Sequential()

    #  learn a total of 32 filters, kernel size 3x3

    model.add(Conv2D(32, (3, 3), input\_shape=(150,150,3), padding="Same", activation='relu'))

    model.add(MaxPooling2D((2, 2)))

    #  learn a total of 64 filters, kernel size 3x3

    model.add(Conv2D(64, (3, 3), padding="Same", activation='relu'))

    model.add(MaxPooling2D((2, 2)))

    #  learn a total of 96 filters, kernel size 3x3

    model.add(Conv2D(96, (3, 3), padding="Same", activation='relu'))

    model.add(MaxPooling2D((2, 2)))

    #  learn a total of 128 filters, kernel size 3x3

    model.add(Conv2D(128, (3, 3), padding="Same", activation='relu'))

    model.add(MaxPooling2D((2, 2)))

    # Add Dense layers on top

    '''

    1. flatten the 3D output to 1D

    2. add dense layer to top

    '''dfwsssssssssssssssssssssssssssssssss

    model.add(Flatten())

    model.add(Dense(256, activation='relu'))

    model.add(Dense(5, activation='softmax'))

    return model

# Compile

model = createModel()

batch\_size = 128

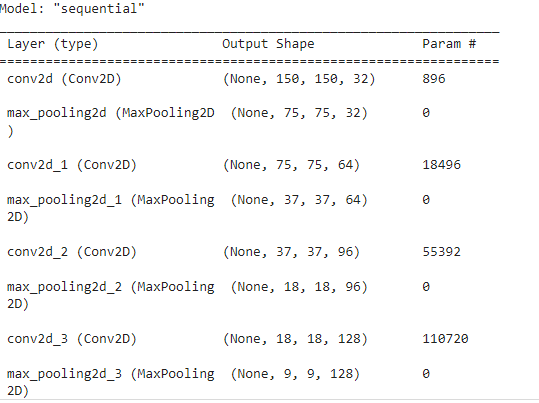
epochs = 50

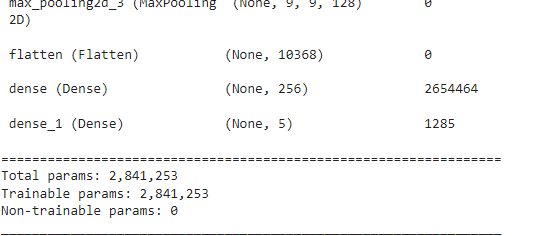
model.compile(loss='categorical\_crossentropy',

             optimizer='RMSProp',

             metrics=['accuracy'])

model.summary()





# Create data argument to prevent overfitting

datagen = ImageDataGenerator(

        featurewise\_center=False,  # set input mean to 0 over the dataset

        samplewise\_center=False,  # set each sample mean to 0

        featurewise\_std\_normalization=False,  # divide inputs by std of the dataset

        samplewise\_std\_normalization=False,  # divide each input by its std

        zca\_whitening=False,  # apply ZCA whitening

        rotation\_range=90,  # randomly rotate images in the range (90, 0 to 180)

        zoom\_range = 0.1, # Randomly zoom image

        width\_shift\_range=0.1,  # randomly shift images horizontally (fraction of total width)

        height\_shift\_range=0.1,  # randomly shift images vertically (fraction of total height)

        shear\_range=0.1,

        horizontal\_flip=True,  # randomly flip images

        vertical\_flip=False  # randomly flip images

)

datagen.fit(trainImg)

# start training

'''

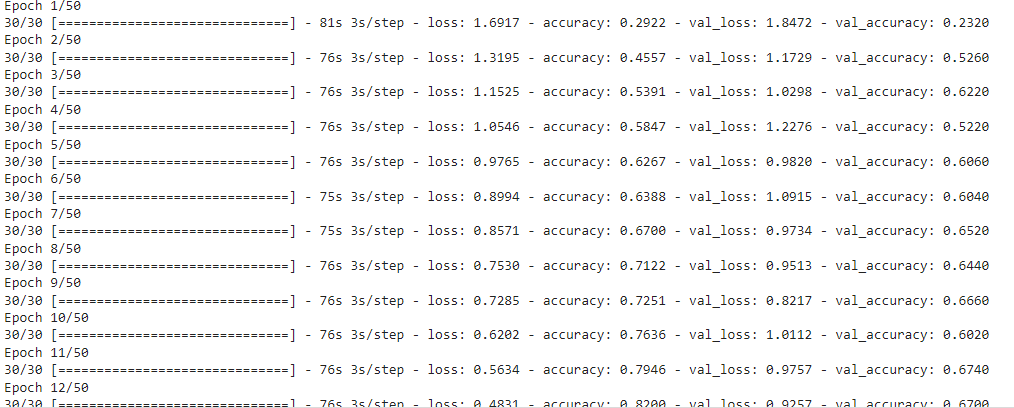
verbose - 0 shows nothing; 1 will show animated progress bar; 2 will only mention the number of epoch.

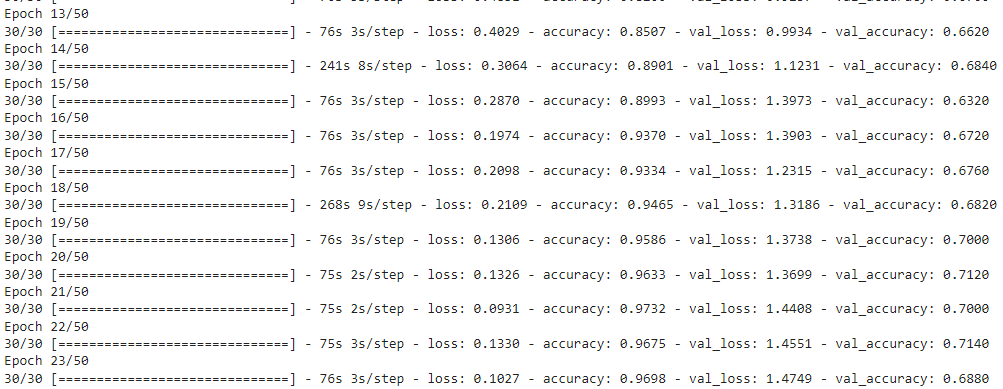
batch\_size - the number of samples that will be propagated through the network.

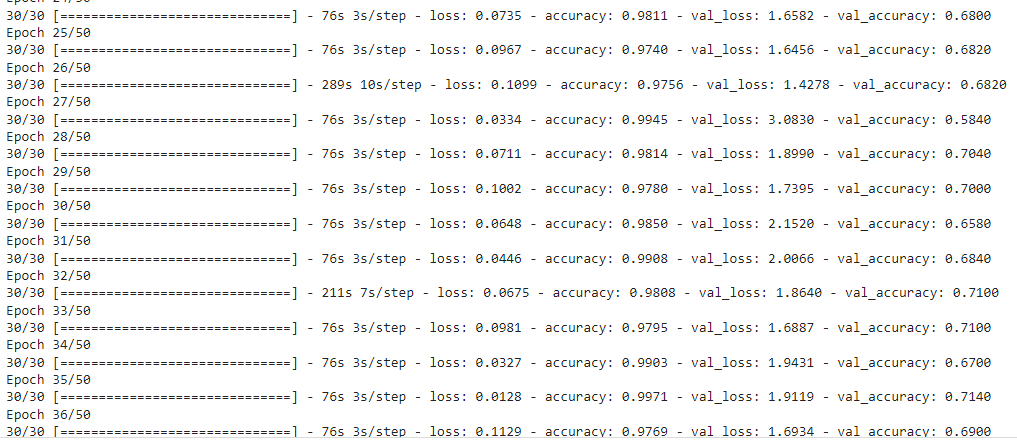
epochs - an arbitrary cutoff, use to separate training into distinct phases.

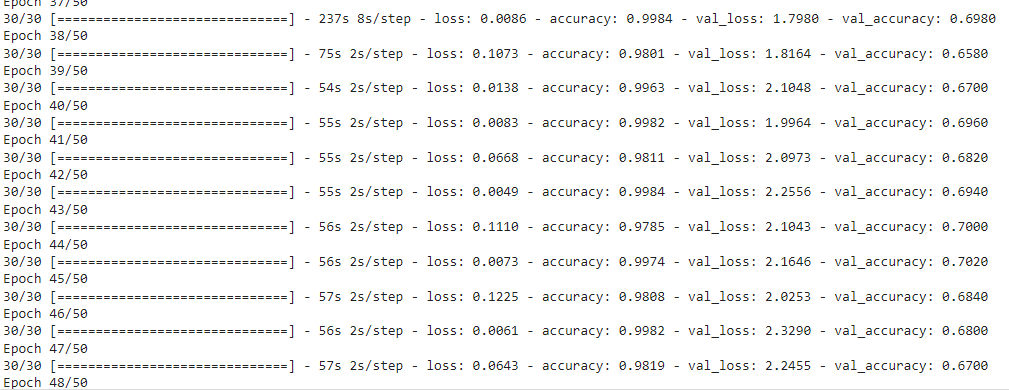
'''

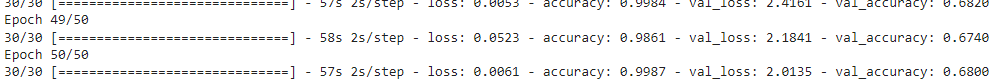
History = model.fit(trainImg, trainLabel, batch\_size=batch\_size, epochs = epochs, validation\_data = (validImg, validLabel),verbose=1)











# start training

'''

verbose - 0 shows nothing; 1 will show animated progress bar; 2 will only mention the number of epoch.

batch\_size - the number of samples that will be propagated through the network.

epochs - an arbitrary cutoff, use to separate training into distinct phases.

'''

History = model.fit(trainImg, trainLabel, batch\_size=batch\_size, epochs = epochs, validation\_data = (validImg, validLabel),verbose=1)

