**Assignment -3**

Python Programming

# 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,
Ma xPooling2D
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
t o 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
correspondin g 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',
'imag e'], index=None)
flowersList.head()
```

```
# Build a dataframe
# load the dataset as a pandas data frame
flowersList = pd.DataFrame(data=flowersList, columns=['category', 'image'], index=None)
flowersList.head()
```

| | category | image |
|---|---|---|
| 0 | daisy | C:\Users\sri nandhini\Downloads\Flowers-Datase... |
| 1 | daisy | C:\Users\sri nandhini\Downloads\Flowers-Datase... |
| 2 | daisy | C:\Users\sri nandhini\Downloads\Flowers-Datase... |
| 3 | daisy | C:\Users\sri nandhini\Downloads\Flowers-Datase... |
| 4 | daisy | C:\Users\sri nandhini\Downloads\Flowers-Datase... |

```
# 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)
```

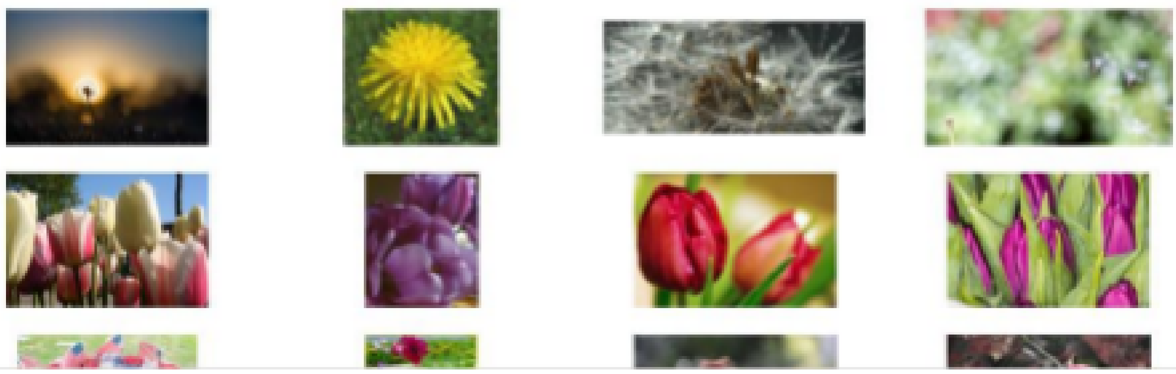
```
# Let's check how many samples for each category are present
print("Total number of flowers in the dataset: ", len(flowersList))
flowerNum = flowersList['category'].value_counts()
print("Flowers in each category: ")
print(flowerNum)
```

```
Total number of flowers in the dataset:  4317
Flowers in each category:
dandelion    1052
tulip         984
rose          784
daisy         764
sunflower     733
Name: category, dtype: int64
```

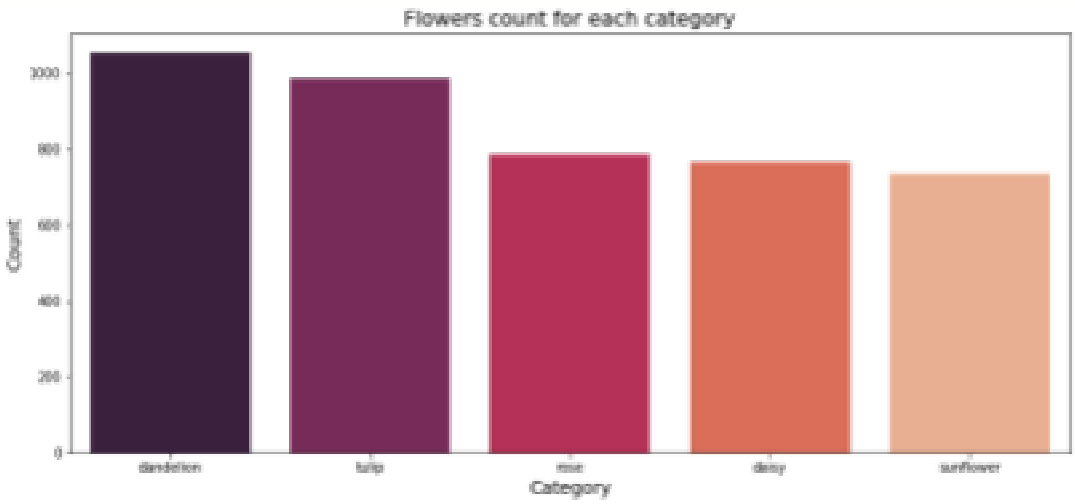
```
# A list for storing names of some random samples from each
category RanSamples = []
# Get samples fom each category
for category in flowerNum.index:
 samples = flowersList['image'][flowersList['category'] == category]
.sample(4).values
 for sample in samples:
 RanSamples.append(sample)

# 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()
```


```
# 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, palett e="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', & 'validDi
r'
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
d ir
for flower in flowerNum.index:
 samples = flowersList['image'][flowersList['category'] == flower].v
alues
 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) + '/' + n ame)
 #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.sha
pe(trainLabel))
```

```
[[1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 ...
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]]
[[1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 ...
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]]

Lengths of the corresponding array dimensions:

(500, 150, 150, 3) (500, 5) (3812, 150, 150, 3) (3812, 5)
```

---

```
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
 '''dfwsssssssssssssssssssssssssssssss
 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()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      896

 max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 75, 75, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 37, 37, 96)        55392

 max_pooling2d_2 (MaxPooling  (None, 18, 18, 96)       0
 2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 128)       110720

 max_pooling2d_3 (MaxPooling  (None, 9, 9, 128)        0
 2D)
```

```
 max_pooling2d_3 (MaxPooling  (None, 9, 9, 128)        0
 2D)

 flatten (Flatten)           (None, 10368)             0

 dense (Dense)               (None, 256)               2654464

 dense_1 (Dense)             (None, 5)                 1285

=================================================================
Total params: 2,841,253
Trainable params: 2,841,253
Non-trainable params: 0
_________________________________________________________________
```

```
# Create data argument to prevent overfitting
datagen = ImageDataGenerator(
 featurewise_center=False, # set input mean to 0 over the datas et
 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 (f raction
of total width)
 height_shift_range=0.1, # randomly shift images vertically (fr action
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
mentio n 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)
```

```
Epoch 1/50
30/30 [==============================] - 81s 3s/step - loss: 1.6917 - accuracy: 0.2922 - val_loss: 1.8472 - val_accuracy: 0.2320
Epoch 2/50
30/30 [==============================] - 76s 3s/step - loss: 1.3195 - accuracy: 0.4557 - val_loss: 1.1729 - val_accuracy: 0.5260
Epoch 3/50
30/30 [==============================] - 76s 3s/step - loss: 1.1525 - accuracy: 0.5391 - val_loss: 1.0298 - val_accuracy: 0.6220
Epoch 4/50
30/30 [==============================] - 76s 3s/step - loss: 1.0546 - accuracy: 0.5847 - val_loss: 1.2276 - val_accuracy: 0.5220
Epoch 5/50
30/30 [==============================] - 76s 3s/step - loss: 0.9765 - accuracy: 0.6267 - val_loss: 0.9820 - val_accuracy: 0.6060
Epoch 6/50
30/30 [==============================] - 75s 3s/step - loss: 0.8994 - accuracy: 0.6388 - val_loss: 1.0915 - val_accuracy: 0.6040
Epoch 7/50
30/30 [==============================] - 75s 3s/step - loss: 0.8571 - accuracy: 0.6700 - val_loss: 0.9734 - val_accuracy: 0.6520
Epoch 8/50
30/30 [==============================] - 76s 3s/step - loss: 0.7530 - accuracy: 0.7122 - val_loss: 0.9513 - val_accuracy: 0.6440
Epoch 9/50
30/30 [==============================] - 76s 3s/step - loss: 0.7285 - accuracy: 0.7251 - val_loss: 0.8217 - val_accuracy: 0.6660
Epoch 10/50
30/30 [==============================] - 76s 3s/step - loss: 0.6202 - accuracy: 0.7636 - val_loss: 1.0112 - val_accuracy: 0.6020
Epoch 11/50
30/30 [==============================] - 76s 3s/step - loss: 0.5634 - accuracy: 0.7946 - val_loss: 0.9757 - val_accuracy: 0.6740
Epoch 12/50
30/30 [==============================] - 76s 3s/step - loss: 0.4831 - accuracy: 0.8200 - val_loss: 0.9257 - val_accuracy: 0.6700
Epoch 13/50
30/30 [==============================] - 76s 3s/step - loss: 0.4029 - accuracy: 0.8507 - val_loss: 0.9934 - val_accuracy: 0.6620
Epoch 14/50
30/30 [==============================] - 241s 8s/step - loss: 0.3064 - accuracy: 0.8901 - val_loss: 1.1231 - val_accuracy: 0.6840
Epoch 15/50
30/30 [==============================] - 76s 3s/step - loss: 0.2870 - accuracy: 0.8993 - val_loss: 1.3973 - val_accuracy: 0.6320
Epoch 16/50
30/30 [==============================] - 76s 3s/step - loss: 0.1974 - accuracy: 0.9370 - val_loss: 1.3903 - val_accuracy: 0.6720
Epoch 17/50
30/30 [==============================] - 76s 3s/step - loss: 0.2098 - accuracy: 0.9334 - val_loss: 1.2315 - val_accuracy: 0.6760
Epoch 18/50
30/30 [==============================] - 268s 9s/step - loss: 0.2109 - accuracy: 0.9465 - val_loss: 1.3186 - val_accuracy: 0.6820
Epoch 19/50
30/30 [==============================] - 76s 3s/step - loss: 0.1306 - accuracy: 0.9586 - val_loss: 1.3738 - val_accuracy: 0.7000
Epoch 20/50
30/30 [==============================] - 75s 2s/step - loss: 0.1326 - accuracy: 0.9633 - val_loss: 1.3699 - val_accuracy: 0.7120
Epoch 21/50
30/30 [==============================] - 75s 2s/step - loss: 0.0931 - accuracy: 0.9732 - val_loss: 1.4408 - val_accuracy: 0.7000
Epoch 22/50
30/30 [==============================] - 75s 3s/step - loss: 0.1330 - accuracy: 0.9675 - val_loss: 1.4551 - val_accuracy: 0.7140
Epoch 23/50
30/30 [==============================] - 76s 3s/step - loss: 0.1027 - accuracy: 0.9698 - val_loss: 1.4749 - val_accuracy: 0.6880
30/30 [==============================] - 76s 3s/step - loss: 0.0735 - accuracy: 0.9811 - val_loss: 1.6582 - val_accuracy: 0.6800
Epoch 25/50
30/30 [==============================] - 76s 3s/step - loss: 0.0967 - accuracy: 0.9740 - val_loss: 1.6456 - val_accuracy: 0.6820
Epoch 26/50
30/30 [==============================] - 289s 10s/step - loss: 0.1099 - accuracy: 0.9756 - val_loss: 1.4278 - val_accuracy: 0.6820
Epoch 27/50
30/30 [==============================] - 76s 3s/step - loss: 0.0334 - accuracy: 0.9945 - val_loss: 3.0830 - val_accuracy: 0.5840
Epoch 28/50
30/30 [==============================] - 76s 3s/step - loss: 0.0711 - accuracy: 0.9814 - val_loss: 1.8990 - val_accuracy: 0.7040
Epoch 29/50
30/30 [==============================] - 76s 3s/step - loss: 0.1002 - accuracy: 0.9780 - val_loss: 1.7395 - val_accuracy: 0.7000
Epoch 30/50
30/30 [==============================] - 76s 3s/step - loss: 0.0648 - accuracy: 0.9850 - val_loss: 2.1520 - val_accuracy: 0.6580
Epoch 31/50
30/30 [==============================] - 76s 3s/step - loss: 0.0446 - accuracy: 0.9908 - val_loss: 2.0066 - val_accuracy: 0.6840
Epoch 32/50
30/30 [==============================] - 211s 7s/step - loss: 0.0675 - accuracy: 0.9808 - val_loss: 1.8640 - val_accuracy: 0.7100
Epoch 33/50
30/30 [==============================] - 76s 3s/step - loss: 0.0981 - accuracy: 0.9795 - val_loss: 1.6887 - val_accuracy: 0.7100
Epoch 34/50
30/30 [==============================] - 76s 3s/step - loss: 0.0327 - accuracy: 0.9903 - val_loss: 1.9431 - val_accuracy: 0.6700
Epoch 35/50
30/30 [==============================] - 76s 3s/step - loss: 0.0128 - accuracy: 0.9971 - val_loss: 1.9119 - val_accuracy: 0.7140
Epoch 36/50
30/30 [==============================] - 76s 3s/step - loss: 0.1129 - accuracy: 0.9769 - val_loss: 1.6934 - val_accuracy: 0.6900
30/30 [==============================] - 237s 8s/step - loss: 0.0086 - accuracy: 0.9984 - val_loss: 1.7980 - val_accuracy: 0.6980
Epoch 38/50
30/30 [==============================] - 75s 2s/step - loss: 0.1073 - accuracy: 0.9801 - val_loss: 1.8164 - val_accuracy: 0.6580
Epoch 39/50
30/30 [==============================] - 54s 2s/step - loss: 0.0138 - accuracy: 0.9963 - val_loss: 2.1048 - val_accuracy: 0.6700
Epoch 40/50
30/30 [==============================] - 55s 2s/step - loss: 0.0083 - accuracy: 0.9982 - val_loss: 1.9964 - val_accuracy: 0.6960
Epoch 41/50
30/30 [==============================] - 55s 2s/step - loss: 0.0568 - accuracy: 0.9811 - val_loss: 2.0973 - val_accuracy: 0.6820
Epoch 42/50
30/30 [==============================] - 55s 2s/step - loss: 0.0049 - accuracy: 0.9984 - val_loss: 2.2556 - val_accuracy: 0.6940
Epoch 43/50
30/30 [==============================] - 56s 2s/step - loss: 0.1110 - accuracy: 0.9785 - val_loss: 2.1043 - val_accuracy: 0.7000
Epoch 44/50
30/30 [==============================] - 56s 2s/step - loss: 0.0073 - accuracy: 0.9974 - val_loss: 2.1646 - val_accuracy: 0.7020
Epoch 45/50
30/30 [==============================] - 57s 2s/step - loss: 0.1225 - accuracy: 0.9808 - val_loss: 2.0253 - val_accuracy: 0.6840
Epoch 46/50
30/30 [==============================] - 56s 2s/step - loss: 0.0061 - accuracy: 0.9982 - val_loss: 2.3290 - val_accuracy: 0.6800
Epoch 47/50
30/30 [==============================] - 57s 2s/step - loss: 0.0643 - accuracy: 0.9819 - val_loss: 2.2455 - val_accuracy: 0.6700
Epoch 48/50
```

```
Epoch 49/50
30/30 [==============================] - 58s 2s/step - loss: 0.0523 - accuracy: 0.9881 - val_loss: 2.1841 - val_accuracy: 0.6740
Epoch 50/50
30/30 [==============================] - 57s 2s/step - loss: 0.0061 - accuracy: 0.9987 - val_loss: 2.0135 - val_accuracy: 0.6800
```

```
# start training
 '''
verbose -
 0 shows nothing; 1 will show animated progress bar; 2 will only
mentio n 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)
```