CS6005
DEEP LEARNING
ASSIGNMENT – 3
Flower Recognition using CNN

Date: 19/04/2021

Sayf Hussain Z
2018103059
P - Batch

Problem statement:

The aim of this mini project is to use Convolutional Neural Networks to classify images of flowers into different categories. For this purpose, a suitable dataset containing images of different types of flowers of size 230 MB was identified and worked on.

Dataset details:

The dataset contains 4323 images of flowers of 5 different categories namely – Daisy, Dandelion, Rose, Sunflower and Tulips [1]. The dataset is available on Kaggle and is curated using scrapped data from flicr, google images and Yandex images. The images are not of very high resolution and are of varied size. The entire dataset is of size 230 MB.

Category	Number of images
Rose	784
Tulip	984
Dandelion	1055
Daisy	769
Sunflower	734

Modules:

Reading data:

The images are stored in different folders depending on the category that they belong to. The images are all first read from the different directories and are stored into a numpy array. A corresponding output array is also created which contains the category that the current flower belongs to. 0 for Daisy, 1 for Dandelion, 2 for Rose, 3 for Sunflower and 4 for Tulip.

Preprocessing the data:

The images are all resized to 128*128 dimensions. The output array (y) is converted to a class matrix with 5 columns, each column representing one category of flower. Additionally, image augmentation was performed by randomly rotating the images by 60 degree, zooming into the images by a factor of 0.1. The image pixels are rescaled by dividing each pixel by 255.

Creating the model:

A sequential model is created with the Keras Sequential API. 5 blocks of Convolutional layers followed by max pooling layers are used. The convolutional layers are padded so as to maintain the input and output size. The feature map changes size each time it passes through a Maxpooling layer. The 1st block contains 64 filters, the 2nd and 3rd block contain 128 filters each, 4th block contains 256 filters and the last block contains 512 filters. The convolutional layer in each block has a dropout of 0.2. The model is fattened and is then passed through a Fully Connected Layer with 1024 nodes with Relu activation function. The resulting mapping is passed through the output layer with 5 nodes with the softmax activation function giving the probability of the image belonging to a class at each node.

Training, testing and validation split:

Category	Train	Validation	Test
Daisy	549	95	125
Rose	571	110	103
Tulip	724	119	141
Dandelion	737	131	184
Sunflower	541	97	96
	3122	552	649

Training the Model:

Parameter	Value
Epochs	75
Optimizer	Adam
Learning rate	0.001
Loss function	Categorical crossentropy

CNN model Summary:

Types of layers used:

1. Convolutional layer
2. Maxpooling layer
3. Batch normalization layer
4. Fully Connected Dense layer

Activation Functions used:

1. Relu
2. Softmax

Other parameters:

1. Dropout =0.2 for Convolutional layers , 0.5 for Fully Connected layer
2. Stride for Convolutional layer = 1 ,for Maxpooling layer = 2
3. Kernel size for Convolutional layer = 3 * 3

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 64)      1792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
dropout_3 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 256)       295168
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
dropout_4 (Dropout)          (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 512)         1180160
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 4, 4, 512)         2048
_________________________________________________________________
dropout_5 (Dropout)          (None, 4, 4, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              8389632
_________________________________________________________________
dropout_6 (Dropout)          (None, 1024)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 5125
=================================================================
Total params: 10,101,765
Trainable params: 10,097,541
Non-trainable params: 4,224
_________________________________________________________________
```

Coding Snapshots:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)

import seaborn as sns
import cv2

import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D,MaxPool2D
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator

import os
```

Reading data:

In [2]:
```
x_ = list()
y = list()
IMG_SIZE = 128
for i in os.listdir("../input/flowers/flowers/daisy"):
    try:
        path = "../input/flowers/flowers/daisy/"+i
        img = plt.imread(path)
        img = cv2.resize(img,(IMG_SIZE,IMG_SIZE))
        x_.append(img)
        y.append(0)
    except:
        None
for i in os.listdir("../input/flowers/flowers/dandelion"):
    try:
        path = "../input/flowers/flowers/dandelion/"+i
        img = plt.imread(path)
        img = cv2.resize(img,(IMG_SIZE,IMG_SIZE))
        x_.append(img)
        y.append(1)
    except:
        None
for i in os.listdir("../input/flowers/flowers/rose"):
    try:
        path = "../input/flowers/flowers/rose/"+i
        img = plt.imread(path)
        img = cv2.resize(img,(IMG_SIZE,IMG_SIZE))
        x_.append(img)
        y.append(2)
    except:
        None
for i in os.listdir("../input/flowers/flowers/sunflower"):
    try:
        path = "../input/flowers/flowers/sunflower/"+i
        img = plt.imread(path)
        img = cv2.resize(img,(IMG_SIZE,IMG_SIZE))
        x_.append(img)
        y.append(3)
    except:
        None
for i in os.listdir("../input/flowers/flowers/tulip"):
    try:
        path = "../input/flowers/flowers/tulip/"+i
        img = plt.imread(path)
        img = cv2.resize(img,(IMG_SIZE,IMG_SIZE))
        x_.append(img)
        y.append(4)
    except:
        None
x_ = np.array(x_)
from keras.utils.np_utils import to_categorical
y = to_categorical(y,num_classes = 5)
```

Example images from each category:

In [3]:
```python
plt.figure(figsize = (20,20))
for i in range(5):
    img = x_[950*i]
    plt.subplot(1,5,i+1)
    plt.imshow(img)
    plt.axis("off")
    plt.title(y[950*i])
```

```
/opt/conda/lib/python3.6/site-packages/matplotlib/text.py:1191: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
```

Train, validation and test split:

In [4]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x_,y,test_size = 0.15,random_state = 42)
```

In [5]:
```python
x_train,x_val,y_train,y_val = train_test_split(x_train,y_train,test_size = 0.15,random_state = 42)
```

Creating the model:

In [7]:
```python
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(3,3),padding="Same",activation="relu" , input_shape = (IMG_SIZE,IMG_SIZE,3)))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=(3,3),padding="Same",activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=(3,3),padding="Same",activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(filters=256,kernel_size = (3,3),padding="Same",activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(filters=512,kernel_size = (3,3),padding="Same",activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Flatten())

model.add(Dense(1024,activation="relu"))
model.add(Dropout(0.5))
model.add(BatchNormalization())

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

model.summary()
```

Compiling the model, preprocessing the data and training the model:

```
In [8]: model.compile(loss='categorical_crossentropy',
                      optimizer=Adam(lr=0.001),
                      metrics=['accuracy'])
```

```
In [9]: epoch = 75
        batch_size = 64
```

```
In [10]: datagen = ImageDataGenerator(
             rescale=1./255
             featurewise_center=False,
             samplewise_center=False,
             featurewise_std_normalization=False,
             samplewise_std_normalization=False,
             rotation_range=60,
             zoom_range = 0.1,
             width_shift_range=0.1,
             height_shift_range=0.1,
             shear_range=0.1,
             fill_mode = "reflect"
             )
         datagen.fit(x_train)
```

```
In [11]: history = model.fit_generator(datagen.flow(x_train,y_train,batch_size=batch_size),
                                       epochs= epoch,validation_data=(x_val,y_val),
                                       steps_per_epoch=48, verbose =2,
                                       )
```

Test accuracy:

```
In [12]: print("Test Accuracy: {0:.2f}%".format(model.evaluate(x_test,y_test)[1]*100))
```

Visualizing:

```
In [14]: x_ = np.array(range(len(history.history['loss'])))
         trace1 = go.Scatter(
                 x = x_,
                 y = history.history['acc'],
                 mode = "lines",
                 marker = dict(color = "rgba(0,0,255,0.9)"),
                 text = "Accuracy"
         )
         data = [trace1]
         layout = dict(title = "Training Accuracy")
         fig = dict(data = data,layout=layout)
         iplot(fig)
```

```
In [15]: x_ = np.array(range(len(history.history['loss'])))
         trace2 = go.Scatter(
                 x = x_,
                 y = history.history['loss'],
                 mode = "lines",
                 marker = dict(color = "rgba(0,0,255,0.9)"),
                 text = "training loss"
         )
         data = [trace2]
         layout = dict(title = "Training loss")
         fig = dict(data = data,layout=layout)
         iplot(fig)
```

```
In [17]: from sklearn.metrics import classification_report, confusion_matrix
         print('Confusion Matrix')
         cm = confusion_matrix(Y_true,Y_pred_classes)
         print(sns.heatmap(confusion_matrix(Y_true,Y_pred_classes),annot=True,fmt= "d"))
         print(classification_report(Y_true,Y_pred_classes, digits=5))
```

Results:

```
In [11]: history = model.fit_generator(datagen.flow(x_train,y_train,batch_size=batch_size),
                                       epochs= epoch,validation_data=(x_val,y_val),
                                       steps_per_epoch=48, verbose =2,
                                       )
```

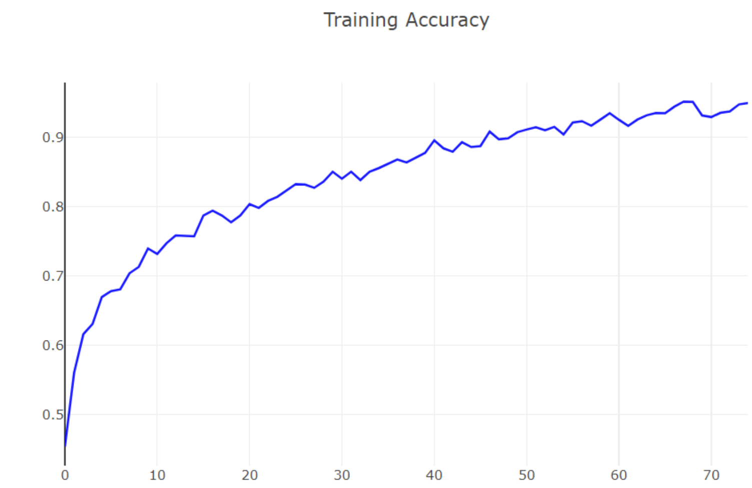
```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflo
w.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/75
 - 19s - loss: 1.6486 - acc: 0.4479 - val_loss: 1.2444 - val_acc: 0.6014
Epoch 2/75
 - 14s - loss: 1.2342 - acc: 0.5313 - val_loss: 1.1737 - val_acc: 0.5996
Epoch 3/75
 - 13s - loss: 1.1033 - acc: 0.5883 - val_loss: 1.7458 - val_acc: 0.4620
Epoch 4/75
 - 13s - loss: 1.0046 - acc: 0.6247 - val_loss: 0.9485 - val_acc: 0.6649
Epoch 5/75
 - 14s - loss: 0.9323 - acc: 0.6604 - val_loss: 1.2421 - val_acc: 0.5652
Epoch 6/75
 - 13s - loss: 0.8836 - acc: 0.6678 - val_loss: 0.9709 - val_acc: 0.6522
Epoch 7/75
 - 14s - loss: 0.8238 - acc: 0.6941 - val_loss: 0.8184 - val_acc: 0.6902
Epoch 8/75
 - 13s - loss: 0.8290 - acc: 0.6949 - val_loss: 1.6332 - val_acc: 0.4909
Epoch 9/75
 - 14s - loss: 0.8348 - acc: 0.6863 - val_loss: 0.9770 - val_acc: 0.6522
Epoch 10/75
 - 13s - loss: 0.7341 - acc: 0.7237 - val_loss: 0.7863 - val_acc: 0.7011
Epoch 11/75
 - 13s - loss: 0.7100 - acc: 0.7293 - val_loss: 0.8731 - val_acc: 0.6757
Epoch 12/75
 - 14s - loss: 0.6790 - acc: 0.7390 - val_loss: 0.9093 - val_acc: 0.6721
Epoch 13/75
 - 13s - loss: 0.6589 - acc: 0.7464 - val_loss: 0.8356 - val_acc: 0.6975
Epoch 14/75
 - 14s - loss: 0.6641 - acc: 0.7582 - val_loss: 1.3187 - val_acc: 0.5471
Epoch 65/75
 - 13s - loss: 0.1850 - acc: 0.9325 - val_loss: 0.7546 - val_acc: 0.7862
Epoch 66/75
 - 13s - loss: 0.1772 - acc: 0.9385 - val_loss: 0.8893 - val_acc: 0.7681
Epoch 67/75
 - 13s - loss: 0.1666 - acc: 0.9400 - val_loss: 0.9368 - val_acc: 0.7754
Epoch 68/75
 - 13s - loss: 0.1530 - acc: 0.9420 - val_loss: 0.7704 - val_acc: 0.7899
Epoch 69/75
 - 13s - loss: 0.1378 - acc: 0.9495 - val_loss: 0.8711 - val_acc: 0.7681
Epoch 70/75
 - 13s - loss: 0.1508 - acc: 0.9436 - val_loss: 1.0406 - val_acc: 0.7428
Epoch 71/75
 - 13s - loss: 0.1624 - acc: 0.9441 - val_loss: 1.1076 - val_acc: 0.7210
Epoch 72/75
 - 13s - loss: 0.1528 - acc: 0.9437 - val_loss: 0.8392 - val_acc: 0.7880
Epoch 73/75
 - 13s - loss: 0.1421 - acc: 0.9491 - val_loss: 0.9106 - val_acc: 0.7899
Epoch 74/75
 - 13s - loss: 0.1446 - acc: 0.9451 - val_loss: 0.7797 - val_acc: 0.8062
Epoch 75/75
 - 13s - loss: 0.1398 - acc: 0.9507 - val_loss: 0.8618 - val_acc: 0.8062
```

Test set accuracy:

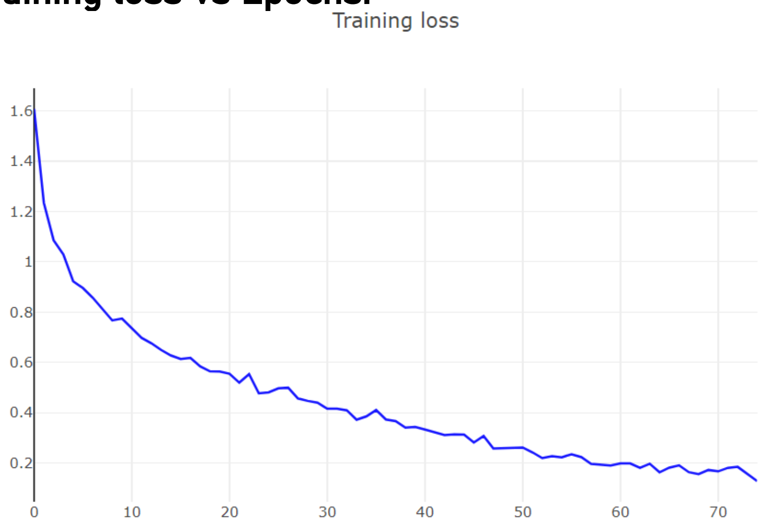
```
In [12]: print("Test Accuracy: {0:.2f}%".format(model.evaluate(x_test,y_test)[1]*100))
```

```
649/649 [==============================] - 0s 590us/step
Test Accuracy: 81.82%
```

Training accuracy vs Epochs:



Training loss vs Epochs:



Confusion Matrix:

Class wise performance metrics:

	precision	recall	f1-score	support
0	0.92453	0.78400	0.84848	125
1	0.87079	0.84239	0.85635	184
2	0.70248	0.82524	0.75893	103
3	0.76271	0.93750	0.84112	96
4	0.80159	0.71631	0.75655	141
micro avg	0.81510	0.81510	0.81510	649
macro avg	0.81242	0.82109	0.81229	649
weighted avg	0.82341	0.81510	0.81544	649

Conclusion:

A Custom CNN model has been built to recognize and categorize 5 different categories of flowers. The model achieves a test set accuracy of 81.82 %.

References:

1. https://www.kaggle.com/alxmamaev/flowers-recognition