Flower Recognition

Name - Pankaj Kumar

Section - RKM018

Roll Number - 46 (B)

Registration Number - 11910254

Course Name - INT-247 (Machine Learning Foundation)

Teacher Name - Respected Ankita Wadhawan

CA-1

Introduction

In this project, I will make a Flower Recognition Model. which help to identify the different types of flower. There are many species of flower in the world. Some species have many colors, such as roses. It is difficult to remember all the names of flower and their information. We train a model which help to recognize the different types of flower. The flower recognition system based on image processing. This system uses edge and colours characteristics of flower images to classify flowers.

To classify the different images, we use image classification. Image classification is the process of categorizing and labelling groups of pixels or vectors within an image based on specific rules. The categorization law can be devised using one or more spectral or textural characteristics. Two general methods of classification are 'supervised' and 'unsupervised'.

The dataset I am using here for the flower recognition from Kaggle Website.

Dataset Links -https://www.kaggle.com/datasets/alxmamaev/flowers-recognition

This dataset contains **4242** images of flower. The data collection is based on the data flicr, google images, Yandex images.

The pictures are divided into five classes: Tulip, Rose, Sunflower, Dandelion, Daisy. For each class there are about 800 photos. Photos are not high resolution, about 320 x 240 pixels. Photo are not reduced to a single size. they have different proportions.

Libraries

In this project there are many libraries used like –

i.) NumPy

ii.) TensorFlow

iii.) Keras

iv.) Pickle

v.) Pywin32

NumPy – NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

TensorFlow – TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

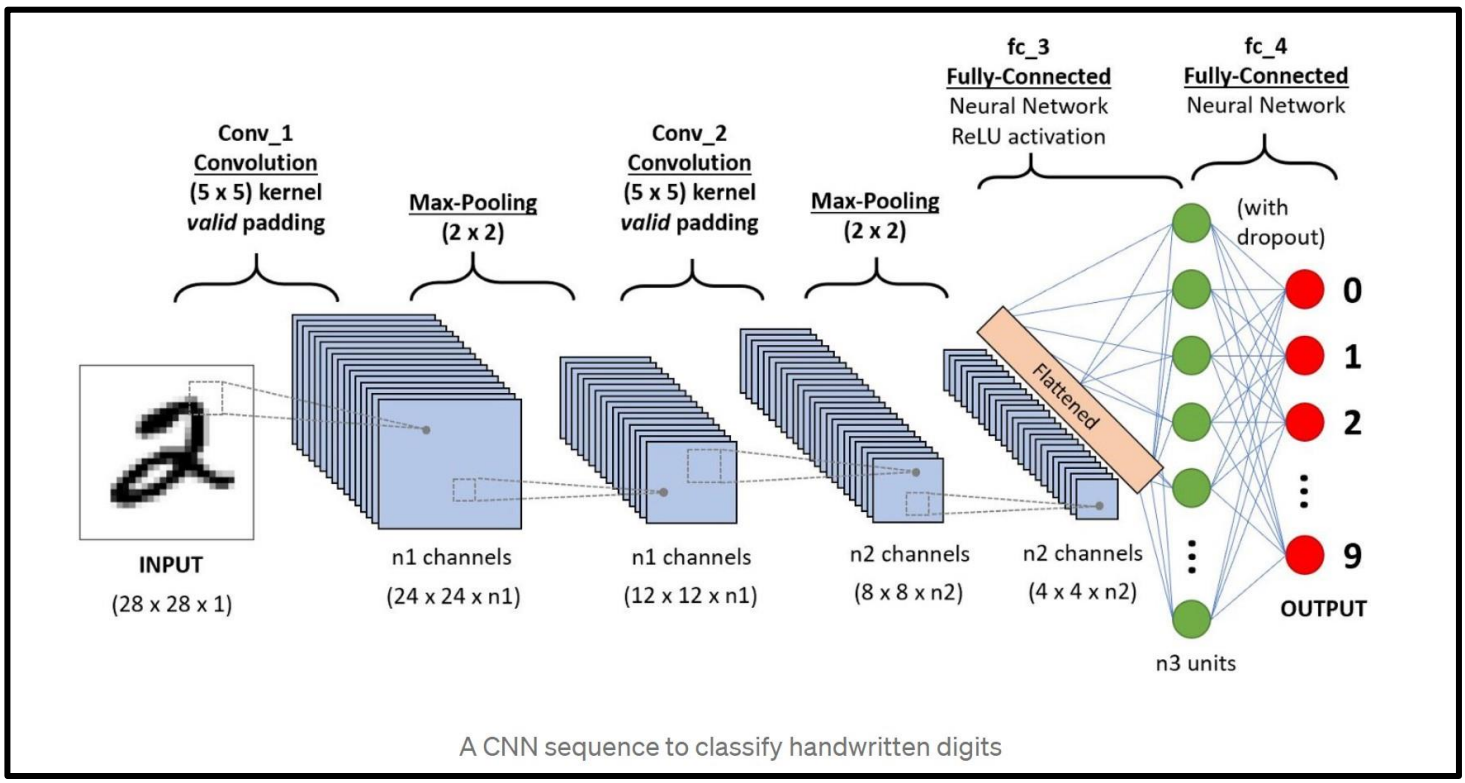
Keras – Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and Plaid ML.

Pickle – Pickling is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

Pywin32 – PyWin32 is a library of Python extensions for Windows that enables you to use the features of the Win32 application programming interface (API) on Python.

Convolutional Neural Networks

Artificial Intelligence has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Researchers and enthusiasts alike, work on numerous aspects of the field to make amazing things happen. One of many such areas is the domain of Computer Vision. The agenda for this field is to enable machines to view the world as humans do, perceive it in a similar manner and even use the knowledge for a multitude of tasks such as Image & Video recognition, Image Analysis & Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc. The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm - a **Convolutional Neural Network**.



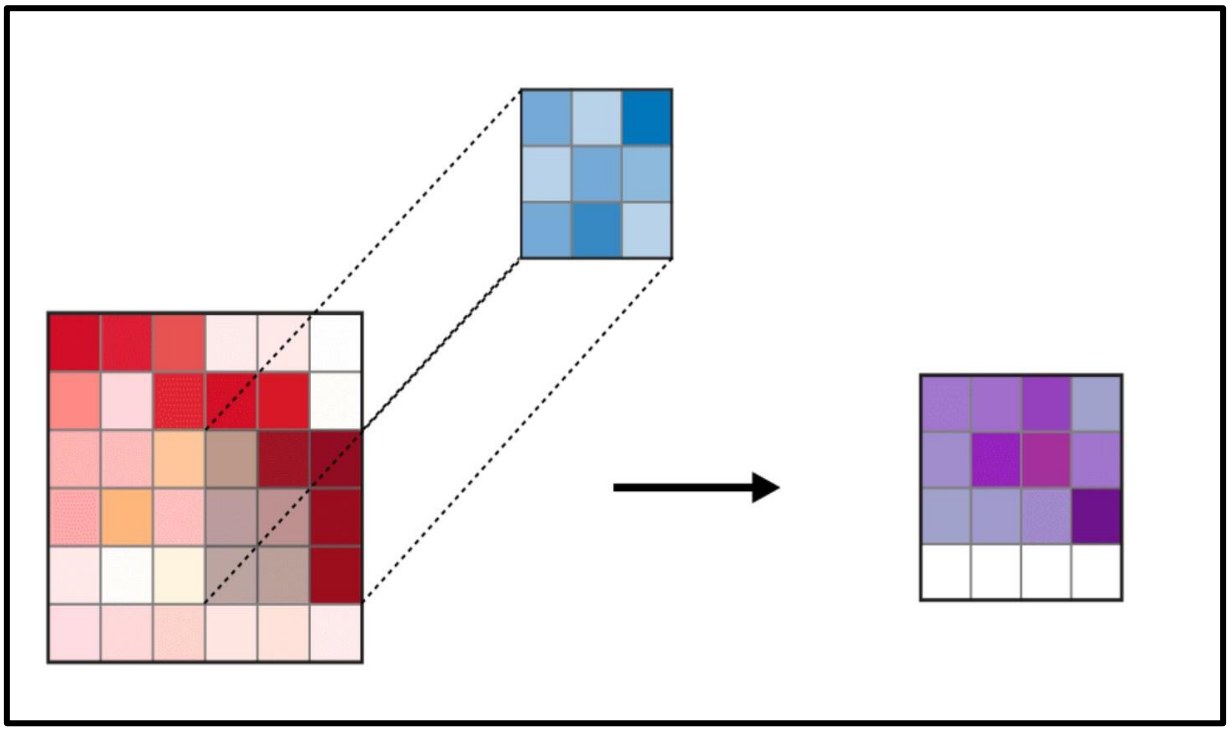
A CNN sequence to classify handwritten digits

A **Convolutional Neural Network (ConvNet / CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

Types of Layer –

(i) Convolution layer (CONV) -

The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input I*I* with respect to its dimensions. Its hyperparameters include the filter size F*F* and stride S*S*. The resulting output O*O* is called *feature map* or *activation map*.



(ii)Pooling (POOL) –

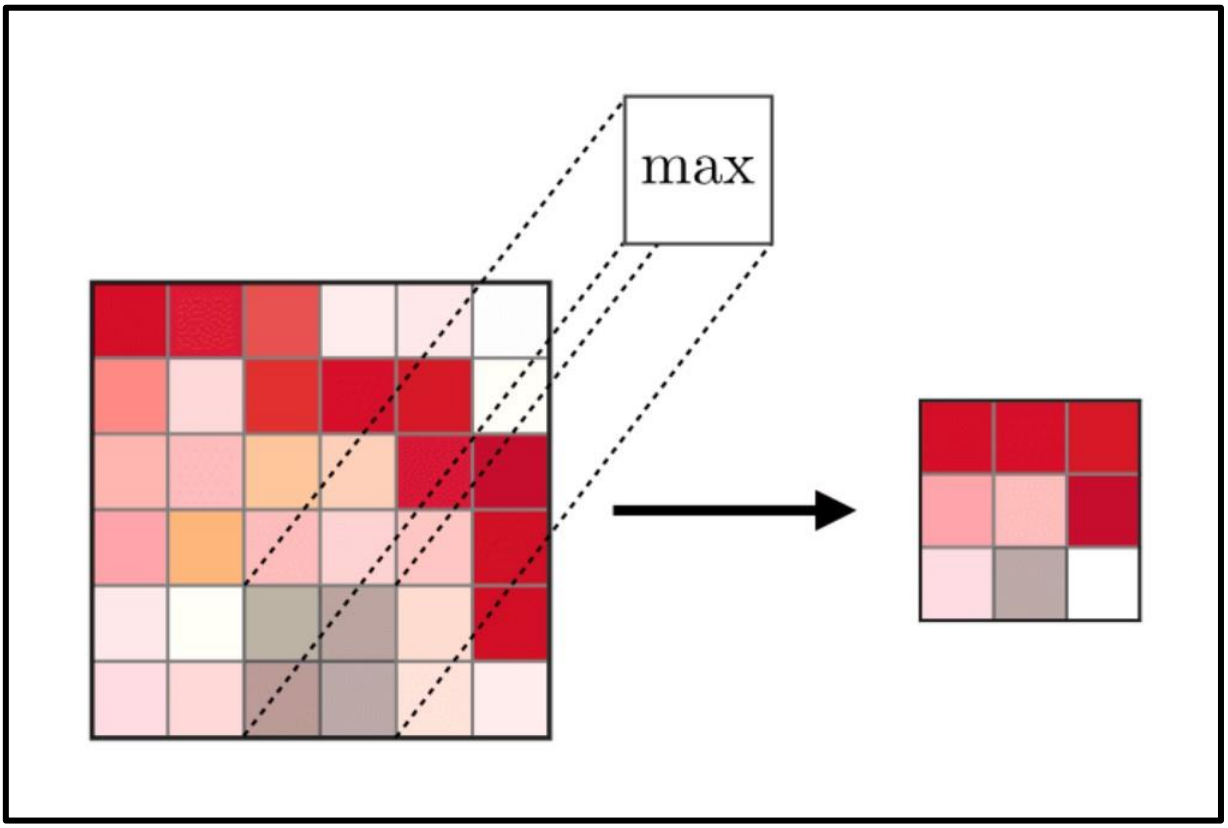
The pooling layer (POOL) is a down sampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

Max Pooling-

Each pooling operation selects the maximum value of the current view.

➔ Preserves detected features.

➔ Most commonly used.

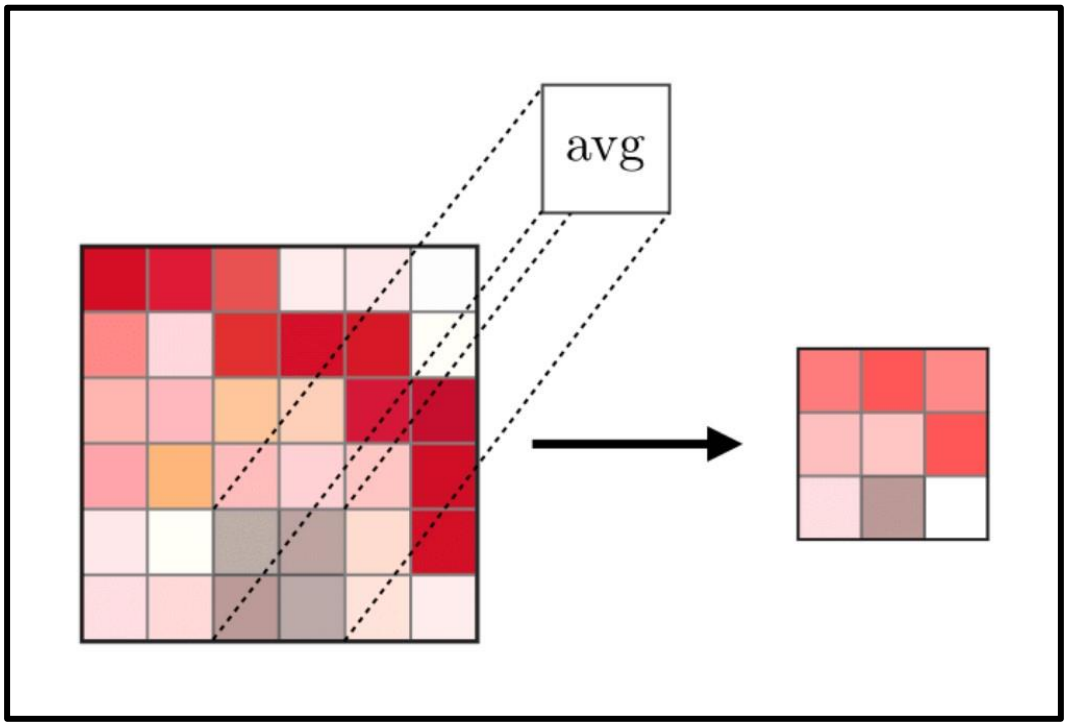


Average Pooling-

Each pooling operation averages the values of the current view.

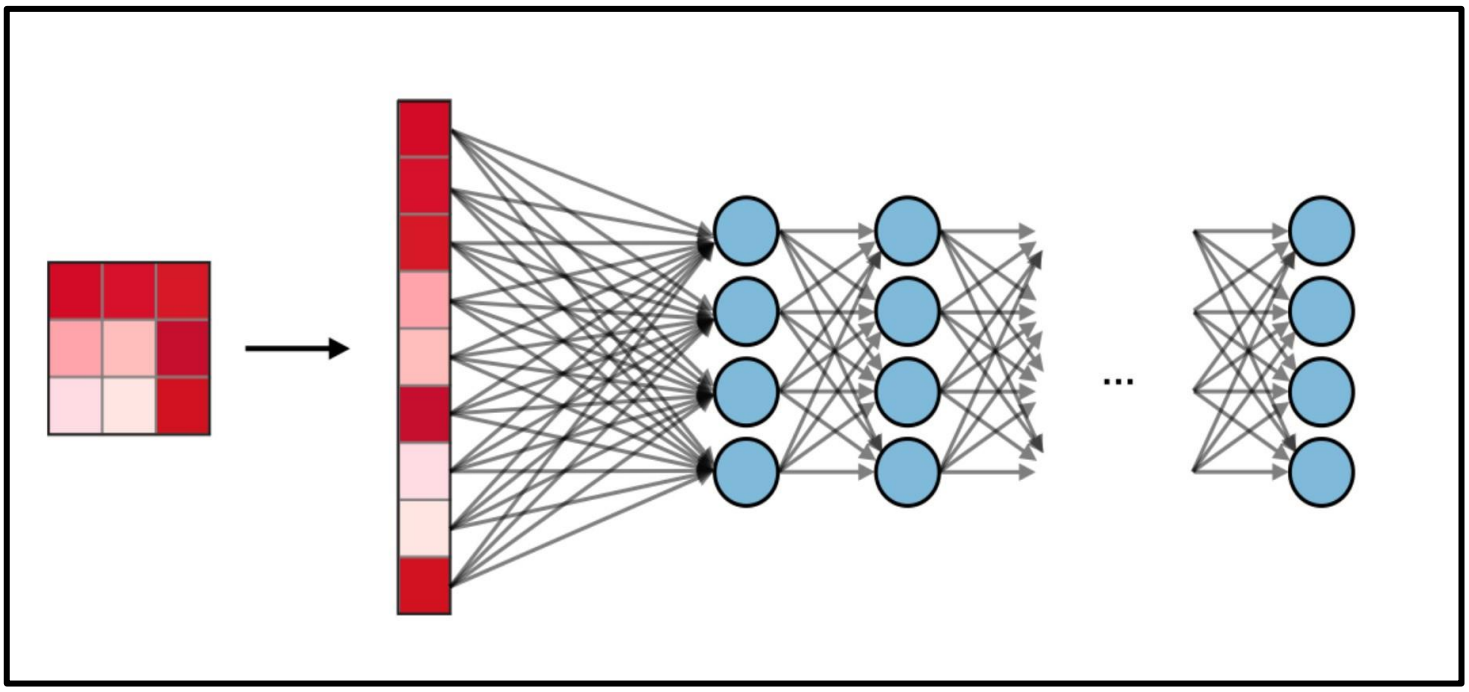
➔ Down samples feature map.

➔ Used in LeNet.



(iii) Fully Connected (FC) –

The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.



Filter Hyperparameters

i) Number of Filters-

The number of filters affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.

ii) Stride-

Stride is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.

i) Zero Padding-

Zero-padding is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output. There are three types of padding:

- **Valid padding:** This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.
- **Same padding:** This padding ensures that the output layer has the same size as the input layer.
- **Full padding:** This type of padding increases the size of the output by adding zeros to the border of the input.

Code Implementations

```
#importing libraries

import numpy as np

import tensorflow as tf

from keras.preprocessing.image import ImageDataGenerator

#Data Preprocessing

#Training Image Preprocessing

train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)

 training_set = train_datagen.flow_from_directory('D:/Software/Project/training_set',

 target_size=(64, 64),

 batch_size=32,

 class_mode='categorical')

#Test Image Preprocessing

test_datagen = ImageDataGenerator(rescale=1./255)

test_set = test_datagen.flow_from_directory('D:/Software/Project/test_set',

 target_size=(64, 64), batch_size=32, class_mode='categorical')

#Class Mode have two Category Then Used 'Binary' If More Than Two Then used 'Categorical'
```

```
#Building Model
cnn = tf.keras.models.Sequential()
#Building Convolution
cnn.add(tf.keras.layers.Conv2D(filters=64 , kernel_size=3 , activation='relu' , input_shape=[64,64,3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
#<------:Activation Function:------>
        #relu function
        #sigmoid function
        #softmax function
        #softplus function
        #softsign function
        #tanh function
        #selu function
        #elu function
        #exponential function
cnn.add(tf.keras.layers.Conv2D(filters=64 , kernel_size=3 , activation='relu' ))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2 , strides=2))
#pool_size = 2 means Matrix Size is 2
#strides->jump

#randomly sets input units to 0 with a frequency of rate at each step during training time,which helps prevent overfitting
cnn.add(tf.keras.layers.Dropout(0.5))
#flattens the multi-dimensional input tensors into a single dimension
cnn.add(tf.keras.layers.Flatten())
#Adding Hidden Layer
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
#Output Layer
cnn.add(tf.keras.layers.Dense(units=5 , activation='softmax'))
#Units = 5, Because Their Five Type Of Flower
#Output Like- > [[0. 0. 0. 1. 0.]]
```

```
#The purpose of loss functions is to compute the quantity that a model should seek to minimize during training
cnn.compile(optimizer = 'rmsprop' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
#<------:optimizers------>
        #SGD
        #RMSprop
        #Adam
        #Adadelta
        #Adagrad
        #Adamax
        #Nadam
        #Ftrl
#Training The Model
cnn.fit(x = training_set , validation_data = test_set , epochs = 43)
#Show Sequence of flower
training_set.class_indices
#Preprocessing the New Images For Testing
from keras.preprocessing import image
test_image = image.load_img('D:/Software/Project/Prediction/rose.jpg',target_size=(64,64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image,axis=0)
result = cnn.predict(test_image)
#Finding the answer
if result[0][0]==1:
   ans = "Daisy"
elif result[0][1]==1:
   ans = "Dandelion"
elif result[0][2]==1:
   ans = "Rose"
elif result[0][3]==1:
   ans = "Sunflower"
```

```
elif result[0][4]==1:
    ans = "Tultip"
#Speak About Flower And Result
import win32com.client
speaker = win32com.client.Dispatch("SAPI.SpVoice")
if ans == "Daisy":
     speaker.Speak(ans)
     speaker.Speak("About Daisy")
    speaker.Speak("Bellis perennis, the daisy, is a European species of the family Asteraceae, often considered the archetypal species of that name. To distinguish this species from other plants known as daisies, it is sometimes qualified as common daisy, lawn daisy or English daisy.")
elif ans == "Dandelion":
    speaker.Speak(ans)
    speaker.Speak("About Dandelion")
    speaker.Speak("Taraxacum is a large genus of flowering plants in the family Asteraceae, which consists of species commonly known as dandelions. The scientific and hobby study of the genus is known as taraxacology.")
elif ans == "Rose":
    speaker.Speak(ans)
    speaker.Speak("About Rose")
    speaker.Speak("A rose is a woody perennial flowering plant of the genus Rosa, in the family Rosaceae, or the flower it bears. There are over three hundred species and tens of thousands of cultivars. They form a group of plants that can be erect shrubs, climbing, or trailing, with stems that are often armed with sharp prickles.")
elif ans == "Sunflower":
    speaker.Speak(ans)
    speaker.Speak("About Sunflower")
    speaker.Speak("Helianthus is a genus comprising about 70 species of annual and perennial flowering plants in the daisy family Asteraceae commonly known as sunflowers. Except for three South American species, the species of Helianthus are native to North America and Central America.")


#Pickling” is the process whereby a Python object hierarchy is converted into a byte stream,
```

```
#and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object)

#is converted back into an object hierarchy.

import pickle

with open('D:/Software/Project/cnn_pickle','wb') as f:

    pickle.dump(cnn,f)

#Here We Save The Model using Keras And Predict

filename = "D:/Software/Project/cnn_fl.h5"

cnn.save(filename)


#Here we load the data and predict the answer

from tensorflow.keras.models import load_model

from keras.preprocessing import image


test_image = image.load_img('D:/Software/Project/Prediction/Sunflowers.jpg',target_size=(64,64))

test_image = image.img_to_array(test_image)

test_image = np.expand_dims(test_image,axis=0)


load_cnn = load_model(filename)

cnn_pred = load_cnn.predict(test_image)
```

Working

Step-:1-> We download the dataset from Kaggle Website. Then We make the 3 folders, first one for training_data, second one for testing_data, third one for prediction (in which many images download from internet perform, to see that model recognize or not. To avoid the overfitting, we keep different images in training_data and testing_data.

Step-:2-> In second step we import all import Libraries which are used for training the model like Numpy, tensorflow, keras etc.

Step-:3-> We train the training data using ImageDataGenerator. In target_size we choose (64,64) because after Preprocessing we want our result in (64,64) pixels. In class mode we select the categorical because our training data set have more than two types of classification.

Step-:4-> After, Same things do for testing_data used ImageDataGenerator, target_size, batch_size, class_mode . here we not define shear_range, zoom_range because we already define in training_data .

Step-:5-> After we Build the Convolution Neural Network (CNN).

Step-:6-> Then Add hidden layer called Relu. There are many hidden layers like sigmoid, tanh, elu etc.

But we used relu because they give better result. The **rectified linear activation function** or **ReLU** for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

Step-:7-> Then We used MaxPool, Strides = 2. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time.

Step-:8-> Then we apply randomly sets input to o with a frequency of rate at each step during training time, which helps prevent overfitting.

Step-:9-> After we flattens data. {the multi – dimensional input tensors into a single dimension.}

Step-:10-> For Hidden Layer we used relu and for Output Layer we add softmax. Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. In Units = 5 because we have five different categories of flower. Which print out like [[0. 0. 0. 1. 0.]] .

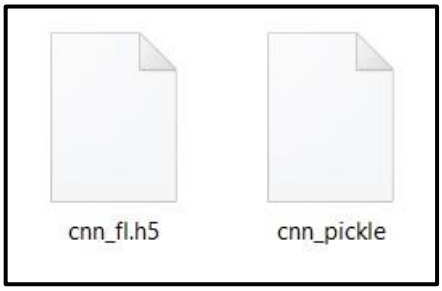
Step-:11-> Then we add optimizer as 'rmsprop' and loss 'categorical_crossentropy'. There are many types of optimizer like SGD, Adam, Ftrl, Adagrad etc. An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate. Thus, it helps in reducing the overall loss and improve the accuracy. There are many losses like – BinaryCrossentropy, Poisson Class, sparse_categorical_crossentropy function etc. The purpose of loss functions is to compute the quantity that a model should seek to minimize during training.

Step-:12-> We train the model. in first Epoch I got the accuracy: 0.4312 and last Iteration I got the accuracy: 0.8803.

Step-:13-> Then After we Preprocessing the new Images For Testing {Means here we download any flower Image form Internet then with help of model we recognize the images.}

Step-:14-> After I implement voice System. For example, if model recognize the images then system speak about the flower.

Step-:15-> Then To save the model we used two different technique One Pickling and Keras Save Model. After Saving the model. Using Keras load model we predict the result and we can also share the model, from model we can directly predict the result without training the data.



Result & Analysis

When I start train my model in starting (first epoch) I got the accuracy 0.4312 and last Epoch = 0.8803. When number of Iteration Increased its Accuracy Increases. And Finally, when I test many

Images, from internet then model predict very accurate result. I predict more than 30 images it predicts accurate. To avoid overfitting, I train the model with different data and test the model with different test data.

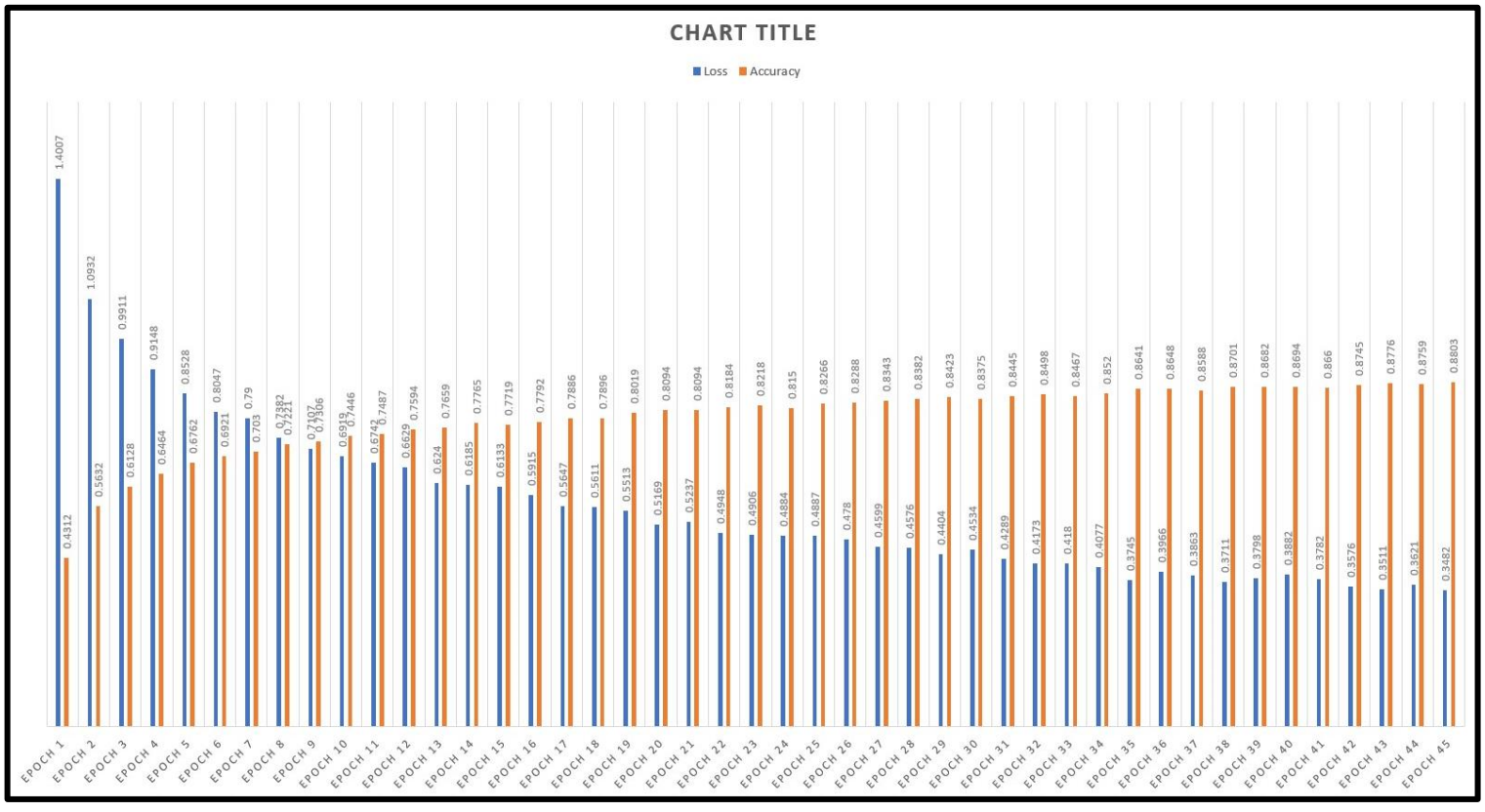
```
Epoch 1/45
130/130 [==============================] - 132s 1s/step - loss: 1.4007 - accuracy: 0.4312 - val_loss: 1.1148 - val_accuracy: 0.
5508
Epoch 2/45
130/130 [==============================] - 25s 196ms/step - loss: 1.0932 - accuracy: 0.5632 - val_loss: 0.9808 - val_accuracy:
0.6294
Epoch 3/45
130/130 [==============================] - 26s 202ms/step - loss: 0.9911 - accuracy: 0.6128 - val_loss: 0.9260 - val_accuracy:
0.6377
Epoch 4/45
130/130 [==============================] - 27s 205ms/step - loss: 0.9148 - accuracy: 0.6464 - val_loss: 0.9057 - val_accuracy:
0.6625
Epoch 5/45
130/130 [==============================] - 27s 207ms/step - loss: 0.8528 - accuracy: 0.6762 - val_loss: 0.9210 - val_accuracy:
0.6516
Epoch 6/45
130/130 [==============================] - 30s 227ms/step - loss: 0.8047 - accuracy: 0.6921 - val_loss: 0.9553 - val_accuracy:
0.6315
Epoch 7/45
130/130 [==============================] - 27s 210ms/step - loss: 0.7900 - accuracy: 0.7030 - val_loss: 0.6883 - val_accuracy:
0.7364
Epoch 8/45
130/130 [==============================] - 27s 207ms/step - loss: 0.7382 - accuracy: 0.7221 - val_loss: 0.6265 - val_accuracy:
0.7635
Epoch 9/45
130/130 [==============================] - 27s 205ms/step - loss: 0.7107 - accuracy: 0.7306 - val_loss: 0.6126 - val_accuracy:
0.7755
Epoch 10/45
130/130 [==============================] - 27s 205ms/step - loss: 0.6919 - accuracy: 0.7446 - val_loss: 0.7259 - val_accuracy:
0.7359
Epoch 11/45
130/130 [==============================] - 27s 205ms/step - loss: 0.6742 - accuracy: 0.7487 - val_loss: 0.5635 - val_accuracy:
0.7864
```

```
Epoch 12/45
130/130 [==============================] - 27s 206ms/step - loss: 0.6629 - accuracy: 0.7594 - val_loss: 0.5184 - val_accuracy:
0.8277
Epoch 13/45
130/130 [==============================] - 27s 206ms/step - loss: 0.6240 - accuracy: 0.7659 - val_loss: 0.5101 - val_accuracy:
0.8114
Epoch 14/45
130/130 [==============================] - 27s 205ms/step - loss: 0.6185 - accuracy: 0.7765 - val_loss: 0.5977 - val_accuracy:
0.7892
Epoch 15/45
130/130 [==============================] - 27s 205ms/step - loss: 0.6133 - accuracy: 0.7719 - val_loss: 0.4846 - val_accuracy:
0.8251
Epoch 16/45
130/130 [==============================] - 27s 206ms/step - loss: 0.5915 - accuracy: 0.7792 - val_loss: 0.4990 - val_accuracy:
0.8184
Epoch 17/45
130/130 [==============================] - 27s 208ms/step - loss: 0.5647 - accuracy: 0.7886 - val_loss: 0.6201 - val_accuracy:
0.7774
Epoch 18/45
130/130 [==============================] - 28s 218ms/step - loss: 0.5611 - accuracy: 0.7896 - val_loss: 0.5209 - val_accuracy:
0.7980
Epoch 19/45
130/130 [==============================] - 27s 207ms/step - loss: 0.5513 - accuracy: 0.8019 - val_loss: 0.3916 - val_accuracy:
0.8728
Epoch 20/45
130/130 [==============================] - 27s 209ms/step - loss: 0.5169 - accuracy: 0.8094 - val_loss: 0.4399 - val_accuracy:
0.8411
Epoch 21/45
130/130 [==============================] - 27s 209ms/step - loss: 0.5237 - accuracy: 0.8094 - val_loss: 0.5119 - val_accuracy:
0.8126
Epoch 22/45
130/130 [==============================] - 28s 217ms/step - loss: 0.4948 - accuracy: 0.8184 - val_loss: 0.3613 - val_accuracy:
0.8754
Epoch 23/45
130/130 [==============================] - 27s 206ms/step - loss: 0.4906 - accuracy: 0.8218 - val_loss: 0.4457 - val_accuracy:
0.8413
```

```
Epoch 24/45
130/130 [==============================] - 29s 220ms/step - loss: 0.4884 - accuracy: 0.8150 - val_loss: 0.3120 - val_accuracy:
0.8918
Epoch 25/45
130/130 [==============================] - 27s 207ms/step - loss: 0.4887 - accuracy: 0.8266 - val_loss: 0.3335 - val_accuracy:
0.8870
Epoch 26/45
130/130 [==============================] - 27s 207ms/step - loss: 0.4780 - accuracy: 0.8288 - val_loss: 0.3389 - val_accuracy:
0.8856
Epoch 27/45
130/130 [==============================] - 28s 217ms/step - loss: 0.4599 - accuracy: 0.8343 - val_loss: 0.3990 - val_accuracy:
0.8587
Epoch 28/45
130/130 [==============================] - 27s 211ms/step - loss: 0.4576 - accuracy: 0.8382 - val_loss: 0.2890 - val_accuracy:
0.9069
Epoch 29/45
130/130 [==============================] - 28s 215ms/step - loss: 0.4404 - accuracy: 0.8423 - val_loss: 0.4687 - val_accuracy:
0.8337
Epoch 30/45
130/130 [==============================] - 27s 211ms/step - loss: 0.4534 - accuracy: 0.8375 - val_loss: 0.3672 - val_accuracy:
0.8691
Epoch 31/45
130/130 [==============================] - 28s 212ms/step - loss: 0.4289 - accuracy: 0.8445 - val_loss: 0.2813 - val_accuracy:
0.9048
Epoch 32/45
130/130 [==============================] - 27s 211ms/step - loss: 0.4173 - accuracy: 0.8498 - val_loss: 0.2260 - val_accuracy:
0.9245
Epoch 33/45
130/130 [==============================] - 28s 212ms/step - loss: 0.4180 - accuracy: 0.8467 - val_loss: 0.2543 - val_accuracy:
0.9157
Epoch 34/45
130/130 [==============================] - 28s 212ms/step - loss: 0.4077 - accuracy: 0.8520 - val_loss: 0.2385 - val_accuracy:
0.9282
Epoch 35/45
130/130 [==============================] - 27s 211ms/step - loss: 0.3745 - accuracy: 0.8641 - val_loss: 0.2401 - val_accuracy:
0.9261
Epoch 36/45
130/130 [==============================] - 29s 219ms/step - loss: 0.3966 - accuracy: 0.8648 - val_loss: 0.2714 - val_accuracy:
```

```
Epoch 37/45
130/130 [==============================] - 28s 213ms/step - loss: 0.3863 - accuracy: 0.8588 - val_loss: 0.2466 - val_accuracy:
0.9233
Epoch 38/45
130/130 [==============================] - 30s 229ms/step - loss: 0.3711 - accuracy: 0.8701 - val_loss: 0.2690 - val_accuracy:
0.9110
Epoch 39/45
130/130 [==============================] - 28s 215ms/step - loss: 0.3798 - accuracy: 0.8682 - val_loss: 0.2559 - val_accuracy:
0.9152
Epoch 40/45
130/130 [==============================] - 28s 213ms/step - loss: 0.3882 - accuracy: 0.8694 - val_loss: 0.1805 - val_accuracy:
0.9456
Epoch 41/45
130/130 [==============================] - 28s 213ms/step - loss: 0.3782 - accuracy: 0.8660 - val_loss: 0.2239 - val_accuracy:
0.9266
Epoch 42/45
130/130 [==============================] - 28s 214ms/step - loss: 0.3576 - accuracy: 0.8745 - val_loss: 0.1762 - val_accuracy:
0.9486
Epoch 43/45
130/130 [==============================] - 28s 218ms/step - loss: 0.3511 - accuracy: 0.8776 - val_loss: 0.2399 - val_accuracy:
0.9240
Epoch 44/45
130/130 [==============================] - 28s 215ms/step - loss: 0.3621 - accuracy: 0.8759 - val_loss: 0.3318 - val_accuracy:
0.8914
Epoch 45/45
130/130 [==============================] - 27s 210ms/step - loss: 0.3482 - accuracy: 0.8803 - val_loss: 0.2299 - val_accuracy:
0.9331
```

Graph – Loss / Accuracy



Challenges

When I Start working on my project in starting I unable to implement the image classifications on my data set. But learn from scratch and able to understand how CNN working. In starting I got many Error but resolve error from Stackoverflow then learn from keras And Tensorflow documentation. When I train my model many times, model overfit. Model working on training data perfectly but when I test the model on testing data then I got wrong result. Model unable to predict the result. Later I used different activation function like RELU which help me to get more accurate result. Then I try different – different activation function in hidden layer check which activation function work good in image classification. And I also used different type of optimizer and loss function to optimize my model. After many steps, may changes I done in my project. Then I got the good result. After I start working how can implement speech system in my model. Installing new libraries pywin32 to work on speech system. And also implement pickling in my model.

Learning Outcome

In this project I learn many new things in machine learning like learn more about TensorFlow, Keras, Speech System. Like I able implement speech system in my project. I implement how can save the model and reused the model without training. Using different technique like pickling and keras save model. After training the model I able to share the model any predict the image from this model without training. Able To understand how model able identify the image. From many articles learn a how CNN working. Learn about how can train the data. How can avoid the overfitting.

My GitHub - https://github.com/pkp245464/INT-247-CA-1-

Data Set - https://www.kaggle.com/datasets/alxmamaev/flowers-recognition

References

-> https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

-> https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

-> https://www.tensorflow.org/guide

-> https://www.ibm.com/cloud/learn/convolutional-neural-networks

->https://keras.io/api/