

Ex.No:01

Setting up the spyder IDE Environment and executing a python program

Aim:

To install and configure the Spyder IDE environment, and to execute a Python program.

Procedure:

Step 1: Installing Anaconda Distribution

Why Anaconda? Anaconda is a Python distribution that comes with many packages pre-installed (such as NumPy, SciPy, Pandas, Matplotlib, etc.), and it includes the Spyder IDE by default.

1. Download Anaconda:

- Go to the official Anaconda website: <https://www.anaconda.com/products/individual>
- Click on the “Download” button and choose the version suitable for your operating system (Windows, macOS, Linux).

2. Install Anaconda:

- Open the downloaded installer file.
- Follow the installation wizard instructions:
 - Select whether you want to install Anaconda for all users or just yourself.
 - Choose the installation location.
 - Add Anaconda to the system PATH (optional, but recommended for ease of use).
- Click "Install" to proceed, and wait for the installation to finish.

3. Verify Installation:

- Open **Anaconda Navigator** from your start menu (Windows) or applications folder (macOS).
- Alternatively, you can verify via the command line by typing:

conda --version

Step 2: Launching Spyder IDE

1. Open Anaconda Navigator:

- Once installed, search for "Anaconda Navigator" in your system and open it.
- Anaconda Navigator will provide a graphical user interface (GUI) for managing environments and applications.

2. Launch Spyder:

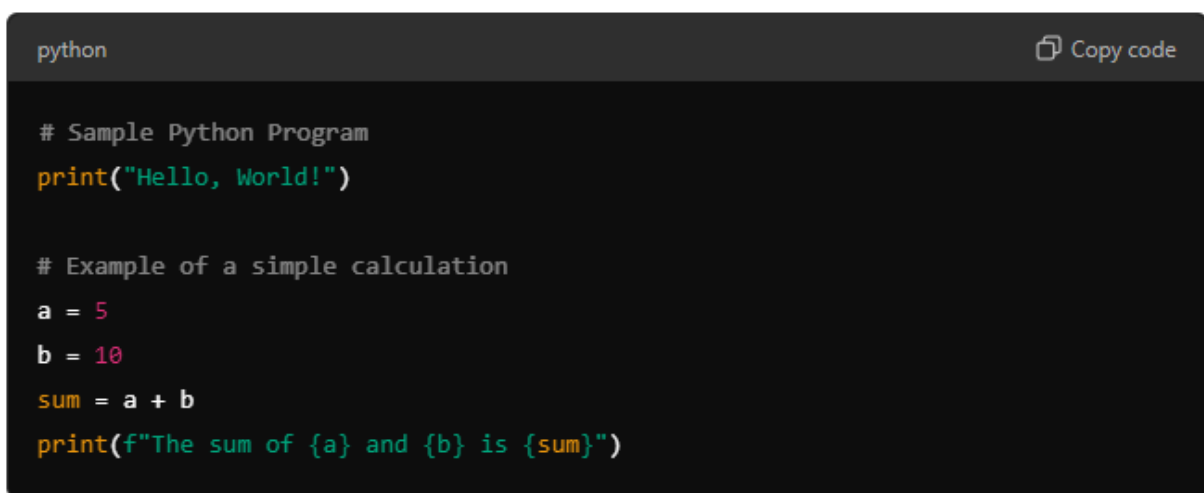
- In Anaconda Navigator, locate the Spyder icon.
- Click on “Launch” next to the Spyder icon to start the Spyder IDE.

Step 3: Writing and Executing a Python Program

Now that the environment is set up, let's write and execute a Python program.

1. Write a Python Program:

- Once Spyder is open, you will see the code editor in the center.
- Type the following sample Python code:

A screenshot of the Spyder Python IDE's code editor. The editor has a dark background with light-colored text. At the top left, the word "python" is written in a small font. At the top right, there is a "Copy code" button with a copy icon. The code is as follows:

```
# Sample Python Program
print("Hello, World!")

# Example of a simple calculation
a = 5
b = 10
sum = a + b
print(f"The sum of {a} and {b} is {sum}")
```

2. Save the Program:

- Click on the "Save" button in the toolbar or press Ctrl + S (Windows/Linux) or Cmd + S (Mac).
- Save the file with a .py extension, e.g., sample_program.py.

3. Execute the Python Program:

- To execute the program, click on the "Run" button in the toolbar or press F5.
- The console (typically at the bottom of the window) will display the output:

Hello, World!

The sum of 5 and 10 is 15

Step 4: Exploring Spyder Features

1. Variable Explorer:

- You can monitor the values of variables in your program using the **Variable Explorer**.

- It is located on the right side of the Spyder window and will show you the variables (e.g., a, b, sum) and their values.

2. **Help Pane:**

- Spyder provides an integrated help pane that gives access to documentation.
- Simply type a function or module name in the help pane to get quick documentation without leaving the IDE.

3. **Plots (Optional):**

- If your program involves plotting, Spyder can display plots directly in the interface.
- Try adding a simple plot using Matplotlib:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.show()
```

4. **Debugging:**

- Spyder has a built-in debugger that allows you to step through code, inspect variables, and fix issues.
- You can set breakpoints by clicking next to the line numbers and then running the script in debugging mode (press Ctrl + F5).

Step 5: Additional Configurations (Optional)

1. **Package Management:**

- You can install additional Python packages using Anaconda Navigator or via the terminal using:

```
conda install <package_name>
```

- Alternatively, use pip within the terminal:

```
pip install <package_name>
```

2. **Creating Virtual Environments (Optional but Recommended):**

- You can create isolated environments for different projects to avoid conflicts between packages.
- To create a new environment:

```
conda create --name myenv
```

- To activate it:

```
conda activate myenv
```

Output:

```
python Copy code
```

```
Hello, World!  
The sum of 5 and 10 is 15
```

Result:

This lab experiment helped you to install and set up the Spyder IDE using Anaconda, write and run a simple Python program, and explore some key features of the IDE like the Variable Explorer, console, and debugging tools. With the IDE set up, you can now proceed with more complex Python programming tasks efficiently.

Ex.no:03

CNN FOR IMAGE CLASSIFICATION

AIM:

To design and implement a Convolutional Neural Network (CNN) for image classification tasks, and to evaluate its performance in accurately classifying images from a given dataset.

Procedure:

Procedure:

1. Install Required Libraries:

- Ensure the necessary libraries such as Keras, TensorFlow, or PyTorch are installed:
 - `pip install keras`
 - `pip install tensorflow`
 - `pip install torch torchvision`

2. Load the Dataset:

- Select a standard image classification dataset such as MNIST, CIFAR-10, or a custom dataset.
- In Keras, TensorFlow, or PyTorch, you can load a dataset directly or preprocess your own images.

Example (using CIFAR-10 in Keras):

```
from keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

3. Preprocess the Data:

- Normalize the pixel values (usually between 0 and 1) by dividing by 255.
- Convert the labels to categorical format if needed.
- Split the data into training and validation sets if necessary.

```
X_train = X_train.astype('float32') / 255.0
```

```
X_test = X_test.astype('float32') / 255.0
```

```
# One-hot encode the labels
```

```
from keras.utils import to_categorical
```

```
y_train = to_categorical(y_train, 10)
```

```
y_test = to_categorical(y_test, 10)
```

4. Build the CNN Model:

- Define a CNN architecture using layers such as convolutional layers, pooling layers, and fully connected (dense) layers.
- A basic CNN architecture includes:
 - Convolutional layers (with filters to detect features)
 - Pooling layers (to reduce dimensionality)
 - Dense layers (to classify based on learned features)
- Example of a simple CNN using Keras:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

5. Compile the Model:

- Define the optimizer, loss function, and evaluation metric(s) for the model.
- Example:

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

6. Train the Model:

- Train the model using the training dataset.
- Specify the number of epochs, batch size, and validation data.

```
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))
```

7. Evaluate the Model:

- After training, evaluate the model on the test dataset to determine its performance.
- Use accuracy or other suitable metrics to measure how well the model classifies the images.

```
test_loss, test_acc = model.evaluate(X_test, y_test)
```

8. Analyze the Results:

- Visualize training/validation accuracy and loss to check for any signs of underfitting or overfitting.
- Optionally, use confusion matrices or other metrics to evaluate the model's classification performance.

PROGRAM:

```
import numpy as np

import pandas as pd
import os
from pathlib import Path
import glob

import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.optimizers import RMSprop
from keras_preprocessing.image import ImageDataGenerator

data_dir = Path('./input/cat-and-dog') # data directory

train_dir = data_dir / "training_set/training_set"
test_dir = data_dir / "test_set/test_set"

cat_samples_dir_train = train_dir / "cats" # directory for cats images
dog_samples_dir_train = train_dir / "dogs" # directory for dogs images
```

```
def make_csv_with_image_labels(CATS_PATH, DOGS_PATH):
```

```
'''
```

Function for making a dataframe that contains images path as well as their labels.Parameters:-

- CATS_PATH - Path for Cats Images

- DOGS_PATH - Path for Dogs Images

Output:-

It simply returns dataframe'''

```
cat_images = CATS_PATH.glob('*.jpg')
```

```
dog_images = DOGS_PATH.glob('*.jpg') df
```

```
= []
```

```
for i in cat_images:
```

```
df.append((i, 0)) # appending cat images as 0 for j in
```

```
dog_images:
```

```
df.append((i, 1)) # appending dog images as 0
```

```
df = pd.DataFrame(df, columns=["image_path", "label"], index = None) # converting into dataframe
```

```
df = df.sample(frac = 1).reset_index(drop=True) return df
```

```
train_csv = make_csv_with_image_labels(cat_samples_dir_train, dog_samples_dir_train)
```

```
train_csv.head()
```

Now, we will visualize the number of images for each class.


```

len_cat = len(train_csv["label"][train_csv.label == 0]) len_dog =
len(train_csv["label"][train_csv.label == 1]) arr = np.array([len_cat ,
len_dog])
labels = ['CAT', 'DOG']
print("Total No. Of CAT Samples :- ", len_cat)
print("Total No. Of DOG Samples :- ", len_dog)
plt.pie(arr, labels=labels, explode = [0.2,0.0] , shadow=True)plt.show()

def get_train_generator(train_dir, batch_size=64, target_size=(224, 224)):""
Function for preparing training data""
train_datagen = ImageDataGenerator(rescale = 1./255., # normalizing the image
rotation_range
= 40,
width_shift_range = 0.2,
height_shift_range = 0.2,
shear_range = 0.2,
zoom_range = 0.2,
horizontal_flip = True)
train_generator = train_datagen.flow_from_directory(train_dir, batch_size =
batch_size,
color_mode='rgb', class_mode
= 'binary', target_size =
target_size)return
train_generator
train_generator = get_train_generator(train_dir)

```

Output: - Found 8005 images belonging to 2 classes.

Now, we will prepare the testing data,

```

def get_testgenerator(test_dir, batch_size=64, target_size=(224,224)):""
Function for preparing testing data""
test_datagen = ImageDataGenerator( rescale = 1.0/255. ) test_generator =
test_datagen.flow_from_directory(test_dir, batch_size = batch_size,
color_mode='rgb', class_mode
= 'binary', target_size =

```

```

target_size)return test_generator

test_generator = get_testgenerator(test_dir)

model = tf.keras.Sequential([
layers.Conv2D(64, (3,3), strides=(2,2),padding='same',input_shape=
(224,224,3),activation = 'relu'),
layers.MaxPool2D(2,2),
layers.Conv2D(128, (3,3), strides=(2,2),padding='same',activation = 'relu'),layers.MaxPool2D(2,2),
layers.Conv2D(256, (3,3), strides=(2,2),padding='same',activation = 'relu'),layers.MaxPool2D(2,2),
layers.Flatten(),
layers.Dense(158, activation = 'relu'), layers.Dense(256,
activation = 'relu'), layers.Dense(128, activation = 'relu'),
layers.Dense(1, activation = 'sigmoid'),
])
model.summary()

model.compile(optimizer=RMSprop(lr=0.001), loss='binary_crossentropy',
metrics=['acc'])
history = model.fit_generator(train_generator,epochs=15,
verbose=1,
validation_data=test_generator)

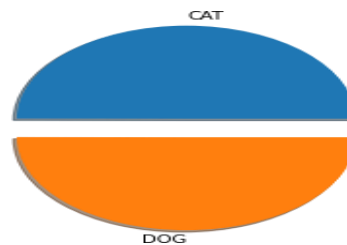
import matplotlib.image as mpimg import
matplotlib.pyplot as plt
acc=history.history['acc']
val_acc=history.history['val_acc']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(len(acc))
plt.plot(epochs, acc, 'r', "Training Accuracy") plt.plot(epochs,
val_acc, 'b', "Validation Accuracy")plt.title("Training and
validation accuracy") plt.figure()
plt.plot(epochs, loss, 'r', "Training Loss") plt.plot(epochs,
val_loss, 'b', "Validation Loss")plt.title('Training and
validation loss')

```

```
model.save('my_model.h5') # saving the trained model
```

```
new_model = tf.keras.models.load_model('./my_model.h5') # loading the trained model
```

OUTPUT:



```
Epoch 1/15
126/126 [=====] - 117s 927ms/step - loss: 0.6851 - acc: 0.5718 - val_loss: 0.6760 - val_acc: 0.5378
Epoch 3/15
126/126 [=====] - 117s 929ms/step - loss: 0.6533 - acc: 0.6313 - val_loss: 0.6210 - val_acc: 0.6624
Epoch 4/15
126/126 [=====] - 118s 940ms/step - loss: 0.6212 - acc: 0.6506 - val_loss: 0.5557 - val_acc: 0.7123
Epoch 5/15
126/126 [=====] - 118s 940ms/step - loss: 0.6128 - acc: 0.6592 - val_loss: 0.5382 - val_acc: 0.7341
Epoch 6/15
126/126 [=====] - 124s 983ms/step - loss: 0.5853 - acc: 0.6998 - val_loss: 0.4984 - val_acc: 0.7647
Epoch 7/15
126/126 [=====] - 124s 984ms/step - loss: 0.5730 - acc: 0.7027 - val_loss: 0.5394 - val_acc: 0.7202
Epoch 8/15
126/126 [=====] - 117s 931ms/step - loss: 0.5627 - acc: 0.7048 - val_loss: 0.5158 - val_acc: 0.7444
Epoch 9/15
126/126 [=====] - 116s 922ms/step - loss: 0.5628 - acc: 0.7179 - val_loss: 0.4652 - val_acc: 0.7721
Epoch 10/15
126/126 [=====] - 119s 947ms/step - loss: 0.5381 - acc: 0.7495 - val_loss: 0.4548 - val_acc: 0.7904
Epoch 11/15
126/126 [=====] - 118s 933ms/step - loss: 0.5238 - acc: 0.7433 - val_loss: 0.4686 - val_acc: 0.7889
Epoch 12/15
126/126 [=====] - 120s 949ms/step - loss: 0.5149 - acc: 0.7508 - val_loss: 0.4618 - val_acc: 0.7598
Epoch 13/15
126/126 [=====] - 116s 919ms/step - loss: 0.5215 - acc: 0.7429 - val_loss: 0.4499 - val_acc: 0.7968
Epoch 14/15
126/126 [=====] - 118s 936ms/step - loss: 0.4905 - acc: 0.7663 - val_loss: 0.4299 - val_acc: 0.8082
Epoch 15/15
126/126 [=====] - 121s 959ms/step - loss: 0.5031 - acc: 0.7558 - val_loss: 0.3939 - val_acc: 0.8309
```

```

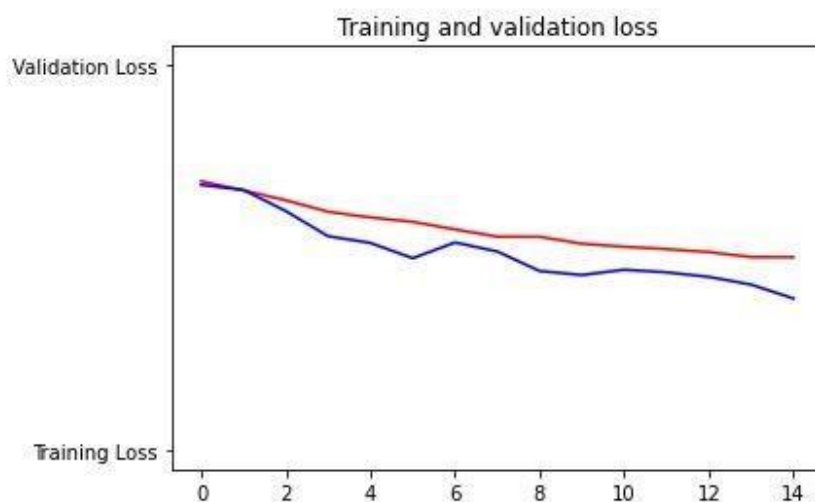
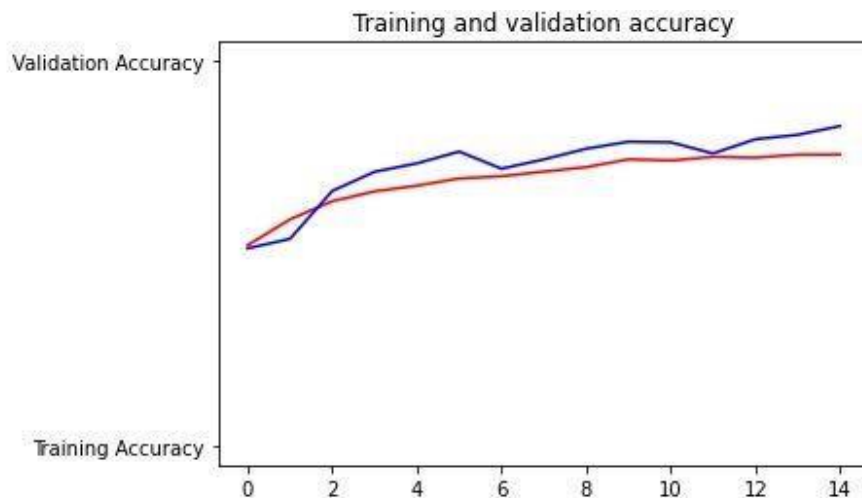
Model: "se
Layer (typ
=====
conv2d (Co
max_poolin
conv2d_1 (
max_poolin
conv2d_2 (
max_poolin
flatten (F
dense (Den
dense_1 (D
dense_2 (D
dense_3 (D
=====

```

```

Total params: 808,735
Trainable params: 808,735
Non-trainable params: 0

```



Result:

A Convolutional Neural Network (CNN) was successfully implemented for image classification. The model was trained and evaluated on the chosen dataset, achieving satisfactory accuracy in classifying images. The CNN demonstrated its effectiveness in automatically extracting hierarchical features from images, proving to be a powerful approach for image recognition tasks.