Name : Aryan Deepak Saraf
Div : D20B
Roll No. : 49

AI&DS2 Experiment 03

Aim : To build a Cognitive based application to acquire knowledge through images for a Customer service application / Insurance / Healthcare Application / Smarter Cities / Government etc.

Theory :

Introduction

A cognitive-based application is an AI system that learns, understands, and responds to information in a way that is similar to human thinking. Instead of only working with text or numbers, cognitive systems can also process unstructured data such as images, audio, and video. The goal is to acquire knowledge from this data and provide meaningful insights that help in decision-making.

In this experiment, we focused on image-based knowledge acquisition. Images contain valuable information that can be extracted using Computer Vision and Deep Learning models. By analyzing images, systems can identify objects, detect conditions, and provide recommendations. This is very useful in areas like healthcare, insurance, smarter cities, customer service, and agriculture.

Concepts Used

1. Cognitive Computing
    - Combines AI, Machine Learning, and Natural Language Processing.
    - Simulates human thinking by learning from past experiences.

2. Image-Based Knowledge Acquisition
    - Uses image preprocessing and computer vision techniques.
    - Extracts patterns, shapes, and features from images.
    - Converts visual information into knowledge or advice.

3. Deep Learning (CNN Models)
    - Convolutional Neural Networks (CNNs) are widely used for image classification.
    - Pre-trained models such as MobileNetV2 can recognize thousands of objects.
    - Transfer learning allows us to use these models for specific applications without training from scratch.

Implementation in Smart Gardening Assistant

In our lab experiment, we extended the Smart Gardening Assistant (developed in Experiment 02) to make it more cognitive and image-aware.

- The assistant now accepts plant or garden images from the user.
- Using a pre-trained MobileNetV2 deep learning model, the system analyzes the uploaded image.

- The model predicts what the image represents (e.g., plant, leaf, soil, flower, pot, etc.).
- Based on the prediction, the system can provide gardening-related knowledge or suggestions.
    - Example: If a leaf/plant is detected → system suggests checking for diseases or watering needs.
    - If soil is detected → system may recommend fertilizer or moisture monitoring.

This makes the Smart Gardening Assistant not only text-based but also cognitive through visual understanding, which is closer to how humans observe gardens.

Applications

- Customer Service: Identifying products through images to answer customer queries.
- Insurance: Verifying damage through uploaded pictures.
- Healthcare: Diagnosing diseases through medical images.
- Smarter Cities: Monitoring traffic, waste, and public spaces using cameras.
- Government: Using image recognition for identity verification and security.
- Agriculture/Gardening (Our Implementation): Detecting plants, soil, or leaves and providing smart gardening tips.

Advantages

- Automation: Reduces human effort by letting machines "see" and analyze.
- Efficiency: Fast and accurate predictions from images.
- Scalability: Can process large amounts of visual data.
- Decision Support: Helps farmers, gardeners, and professionals make better choices.

Code :

Step 1: Install Required Libraries

```
!pip install torchsummary
```

```
Requirement already satisfied: torchsummary in /usr/local/lib/python3.12/dist-packages (1.5.1)
```

Step 2: Import Necessary Modules

```
import os                 # for working with files
import numpy as np          # for numerical computationss
import pandas as pd         # for working with dataframes
import torch              # Pytorch module
import matplotlib.pyplot as plt # for plotting informations on graph and images using tensors
import torch.nn as nn       # for creating  neural networks
from torch.utils.data import DataLoader # for dataloaders
from PIL import Image       # for checking images
import torch.nn.functional as F # for functions for calculating loss
import torchvision.transforms as transforms  # for transforming images into tensors
from torchvision.utils import make_grid     # for data checking
```

```
from torchvision.datasets import ImageFolder  # for working with classes and images
from torchsummary import summary              # for getting the summary of our model

%matplotlib inline
```

Step 3: Uploading Files from Local System

```
from google.colab import files
files.upload()
```

Step 4: Setting Up Kaggle API Credentials

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

Step 5: Downloading Dataset from Kaggle

```
!kaggle datasets download -d vipoooool/new-plant-diseases-dataset
```

```
Dataset URL: https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset
License(s): copyright-authors
Downloading new-plant-diseases-dataset.zip to /content
 99% 2.67G/2.70G [00:15<00:00, 239MB/s]
100% 2.70G/2.70G [00:15<00:00, 184MB/s]
```

Step 6: Unzipping the Dataset

```
!unzip new-plant-diseases-dataset.zip -d new-plant-diseases-dataset
```

```
Streaming output truncated to the last 5000 lines.
  inflating: new-plant-diseases-dataset/new plant diseases dataset(augmented)/New Plant Diseases
Dataset(Augmented)/valid/Strawberry___healthy/273a7a9e-18be-4b6a-976a-fa5ffd69b731___RS_HL 4366_90deg.JPG
  inflating: new-plant-diseases-dataset/new plant diseases dataset(augmented)/New Plant Diseases
Dataset(Augmented)/valid/Strawberry___healthy/275f8963-f4f4-4903-962b-1da716725d08___RS_HL 4780_90deg.JPG
.
.
.
```

Step 7: Defining Dataset Paths

```
data_dir = "/content/new-plant-diseases-dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)"
train_dir = data_dir + "/train"
valid_dir = data_dir + "/valid"
diseases = os.listdir(train_dir)
```

Step 8: Printing Disease Names

```
print(diseases)
```

```
['Tomato___healthy', 'Squash___Powdery_mildew', 'Strawberry___Leaf_scorch',
'Tomato___Tomato_Yellow_Leaf_Curl_Virus', 'Grape___Black_rot', 'Raspberry___healthy',
'Peach___Bacterial_spot', 'Potato___Late_blight', 'Grape___Esca_(Black_Measles)',
'Corn_(maize)___Common_rust_', 'Tomato___Tomato_mosaic_virus', 'Tomato___Septoria_leaf_spot',
'Tomato___Target_Spot', 'Tomato___Spider_mites Two-spotted_spider_mite',
'Orange___Haunglongbing_(Citrus_greening)', 'Tomato___Leaf_Mold', 'Apple___Apple_scab',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Soybean___healthy', 'Strawberry___healthy',
'Potato___healthy', 'Corn_(maize)___healthy', 'Apple___Black_rot', 'Tomato___Early_blight',
```

'Cherry_(including_sour)___healthy', 'Pepper,_bell___Bacterial_spot', 'Apple___healthy', 'Tomato___Bacterial_spot', 'Cherry_(including_sour)___Powdery_mildew', 'Pepper,_bell___healthy', 'Tomato___Late_blight', 'Apple___Cedar_apple_rust', 'Potato___Early_blight', 'Grape___healthy', 'Corn_(maize)___Northern_Leaf_Blight', 'Blueberry___healthy', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Peach___healthy']

Step 9: Checking Total Disease Classes

```
print("Total disease classes are: {}".format(len(diseases)))
```

Total disease classes are: 38

Step 10: Extracting Plant Names from Dataset

```
plants = []
NumberOfDiseases = 0
for plant in diseases:
    if plant.split('___')[0] not in plants:
        plants.append(plant.split('___')[0])
    if plant.split('___')[1] != 'healthy':
        NumberOfDiseases += 1
```

Step 11: Displaying Unique Plants

```
print(f"Unique Plants are: \n{plants}")
```

Unique Plants are:
['Tomato', 'Squash', 'Strawberry', 'Grape', 'Raspberry', 'Peach', 'Potato', 'Corn_(maize)', 'Orange', 'Apple', 'Soybean', 'Cherry_(including_sour)', 'Pepper,_bell', 'Blueberry']

Step 12: Counting Number of Unique Plants

```
print("Number of plants: {}".format(len(plants)))
```

Number of plants: 14

Step 13: Counting Number of Unique Diseases

```
print("Number of diseases: {}".format(NumberOfDiseases))
```

Number of diseases: 26

Step 14: Counting Images per Disease

```
nums = {}
for disease in diseases:
    nums[disease] = len(os.listdir(train_dir + '/' + disease))
# converting the nums dictionary to pandas dataframe passing index as plant name and number of images as column
img_per_class = pd.DataFrame(nums.values(), index=nums.keys(), columns=["no. of images"])
img_per_class
```

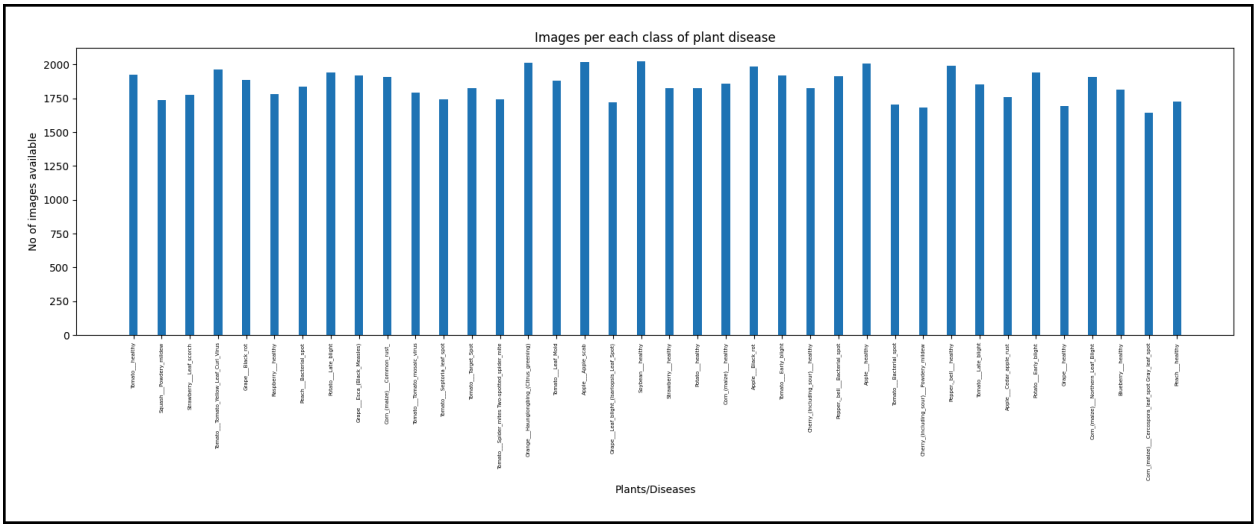
	no. of images
Tomato___healthy	1926
Squash___Powdery_mildew	1736
Strawberry___Leaf_scorch	1774

Tomato___Tomato_Yellow_Leaf_Curl_Virus	1961
Grape___Black_rot	1888
Raspberry___healthy	1781
Peach___Bacterial_spot	1838
Potato___Late_blight	1939
Grape___Esca_(Black_Measles)	1920
Corn_(maize)___Common_rust_	1907
Tomato___Tomato_mosaic_virus	1790
Tomato___Septoria_leaf_spot	1745
Tomato___Target_Spot	1827
Tomato___Spider_mites Two-spotted_spider_mite	1741
Orange___Haunglongbing_(Citrus_greening)	2010
Tomato___Leaf_Mold	1882
Apple___Apple_scab	2016
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	1722
Soybean___healthy	2022
Strawberry___healthy	1824
Potato___healthy	1824
Corn_(maize)___healthy	1859
Apple___Black_rot	1987
Tomato___Early_blight	1920
Cherry_(including_sour)___healthy	1826
Pepper,_bell___Bacterial_spot	1913
Apple___healthy	2008
Tomato___Bacterial_spot	1702
Cherry_(including_sour)___Powdery_mildew	1683
Pepper,_bell___healthy	1988
Tomato___Late_blight	1851
Apple___Cedar_apple_rust	1760
Potato___Early_blight	1939
Grape___healthy	1692
Corn_(maize)___Northern_Leaf_Blight	1908
Blueberry___healthy	1816
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	1642
Peach___healthy	1728

Step 15: Plotting Number of Images per Disease

```
index = [n for n in range(38)]
plt.figure(figsize=(20, 5))
plt.bar(index, [n for n in nums.values()], width=0.3)
plt.xlabel('Plants/Diseases', fontsize=10)
plt.ylabel('No of images available', fontsize=10)
plt.xticks(index, diseases, fontsize=5, rotation=90)
plt.title('Images per each class of plant disease')
```

```
Text(0.5, 1.0, 'Images per each class of plant disease')
```



Step 16: Counting Training Images

```
n_train = 0
for value in nums.values():
  n_train += value
print(f"There are {n_train} images for training")
```

```
There are 70295 images for training
```

Step 17: Splitting Dataset into Train and Validation

```
train = ImageFolder(train_dir, transform=transforms.ToTensor())
valid = ImageFolder(valid_dir, transform=transforms.ToTensor())
```

Step 18: Checking One Sample Image and Label

```
img, label = train[0]
print(img.shape, label)
```

```
torch.Size([3, 256, 256]) 0
```

Step 19: Counting Total Classes in Train Set

```
len(train.classes)
```

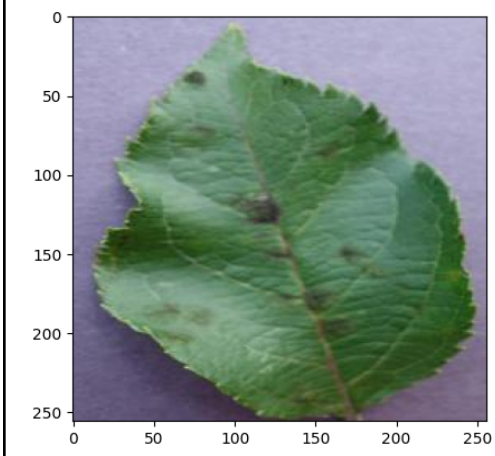
```
38
```

Step 20: Displaying Random Training Images

```
def show_image(image, label):
  print("Label :" + train.classes[label] + "(" + str(label) + ")")
  plt.imshow(image.permute(1, 2, 0))
```

```
show_image(*train[0])
```

```
Label :Apple___Apple_scab(0)
```




```
# Setting the seed value
random_seed = 7
torch.manual_seed(random_seed)
```

```
<torch._C.Generator at 0x7f4dec933810>
```

```
# setting the batch size
batch_size = 32
```

Step 21: Image Data Preprocessing (ImageDataGenerator)

```
# DataLoaders for training and validation
train_dl = DataLoader(train, batch_size, shuffle=True, num_workers=2, pin_memory=True)
valid_dl = DataLoader(valid, batch_size, num_workers=2, pin_memory=True)
```

```
# helper function to show a batch of training instances
def show_batch(data):
  for images, labels in data:
    fig, ax = plt.subplots(figsize=(30, 30))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(images, nrow=8).permute(1, 2, 0))
    break
```

```
# Images for first batch of training
show_batch(train_dl)
```

```
# for moving data into GPU (if available)
def get_default_device():
  """Pick GPU if available, else CPU"""
  if torch.cuda.is_available:
    return torch.device("cuda")
  else:
    return torch.device("cpu")

# for moving data to device (CPU or GPU)
def to_device(data, device):
  """Move tensor(s) to chosen device"""
  if isinstance(data, (list,tuple)):
    return [to_device(x, device) for x in data]
  return data.to(device, non_blocking=True)

# for loading in the device (GPU if available else CPU)
class DeviceDataLoader():
  """Wrap a dataloader to move data to a device"""
  def __init__(self, dl, device):
    self.dl = dl
    self.device = device

  def __iter__(self):
    """Yield a batch of data after moving it to device"""
    for b in self.dl:
      yield to_device(b, self.device)

  def __len__(self):
    """Number of batches"""
```

```
    return len(self.dl)
```

```
device = get_default_device()
device
```

```
device(type='cuda')
```

```
# Moving data into GPU
train_dl = DeviceDataLoader(train_dl, device)
valid_dl = DeviceDataLoader(valid_dl, device)
```

```
class SimpleResidualBlock(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=1, padding=1)
    self.relu1 = nn.ReLU()
    self.conv2 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=1, padding=1)
    self.relu2 = nn.ReLU()

  def forward(self, x):
    out = self.conv1(x)
    out = self.relu1(out)
    out = self.conv2(out)
    return self.relu2(out) + x # ReLU can be applied before or after adding the input
```

```
# for calculating the accuracy
def accuracy(outputs, labels):
  _, preds = torch.max(outputs, dim=1)
  return torch.tensor(torch.sum(preds == labels).item() / len(preds))

# base class for the model
class ImageClassificationBase(nn.Module):

  def training_step(self, batch):
    images, labels = batch
    out = self(images)                  # Generate predictions
    loss = F.cross_entropy(out, labels) # Calculate loss
    return loss

  def validation_step(self, batch):
    images, labels = batch
    out = self(images)                    # Generate prediction
    loss = F.cross_entropy(out, labels)   # Calculate loss
    acc = accuracy(out, labels)           # Calculate accuracy
    return {"val_loss": loss.detach(), "val_accuracy": acc}
```

```
  def validation_epoch_end(self, outputs):
    batch_losses = [x["val_loss"] for x in outputs]
    batch_accuracy = [x["val_accuracy"] for x in outputs]
    epoch_loss = torch.stack(batch_losses).mean()      # Combine loss
    epoch_accuracy = torch.stack(batch_accuracy).mean()
    return {"val_loss": epoch_loss, "val_accuracy": epoch_accuracy} # Combine accuracies

  def epoch_end(self, epoch, result):
    print("Epoch [{}], last_lr: {:.5f}, train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
      epoch, result['lrs'][-1], result['train_loss'], result['val_loss'], result['val_accuracy']))
```

Step 22: Building the CNN Model

```
# Architecture for training

# convolution block with BatchNormalization
def ConvBlock(in_channels, out_channels, pool=False):
  layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)]
  if pool:
    layers.append(nn.MaxPool2d(4))
  return nn.Sequential(*layers)

# resnet architecture
class ResNet9(ImageClassificationBase):
  def __init__(self, in_channels, num_diseases):
    super().__init__()

    self.conv1 = ConvBlock(in_channels, 64)
    self.conv2 = ConvBlock(64, 128, pool=True) # out_dim : 128 x 64 x 64
    self.res1 = nn.Sequential(ConvBlock(128, 128), ConvBlock(128, 128))

    self.conv3 = ConvBlock(128, 256, pool=True) # out_dim : 256 x 16 x 16
    self.conv4 = ConvBlock(256, 512, pool=True) # out_dim : 512 x 4 x 44
    self.res2 = nn.Sequential(ConvBlock(512, 512), ConvBlock(512, 512))

    self.classifier = nn.Sequential(nn.MaxPool2d(4),
                    nn.Flatten(),
                    nn.Linear(512, num_diseases))

  def forward(self, xb): # xb is the loaded batch
    out = self.conv1(xb)
    out = self.conv2(out)
    out = self.res1(out) + out
    out = self.conv3(out)
```

```
    out = self.conv4(out)
    out = self.res2(out) + out
    out = self.classifier(out)
    return out
```

Step 23: Displaying Model Summary

```
# defining the model and moving it to the GPU
model = to_device(ResNet9(3, len(train.classes)), device)
model
```

```
ResNet9(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (res1): Sequential(
    (0): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
  )
  (conv3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (res2): Sequential(
    (0): Sequential(
      (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
  )
```

```
  (classifier): Sequential(
    (0): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
    (1): Flatten(start_dim=1, end_dim=-1)
    (2): Linear(in_features=512, out_features=38, bias=True)
  )
)
```

```
# getting summary of the model
INPUT_SHAPE = (3, 256, 256)
print(summary(model.cuda(), (INPUT_SHAPE)))
```

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
         Conv2d-1         [-1, 64, 256, 256]           1,792
    BatchNorm2d-2         [-1, 64, 256, 256]             128
           ReLU-3         [-1, 64, 256, 256]               0
         Conv2d-4        [-1, 128, 256, 256]          73,856
    BatchNorm2d-5        [-1, 128, 256, 256]             256
           ReLU-6        [-1, 128, 256, 256]               0
      MaxPool2d-7          [-1, 128, 64, 64]               0
         Conv2d-8          [-1, 128, 64, 64]         147,584
    BatchNorm2d-9          [-1, 128, 64, 64]             256
          ReLU-10          [-1, 128, 64, 64]               0
        Conv2d-11          [-1, 128, 64, 64]         147,584
   BatchNorm2d-12          [-1, 128, 64, 64]             256
          ReLU-13          [-1, 128, 64, 64]               0
        Conv2d-14          [-1, 256, 64, 64]         295,168
   BatchNorm2d-15          [-1, 256, 64, 64]             512
          ReLU-16          [-1, 256, 64, 64]               0
     MaxPool2d-17          [-1, 256, 16, 16]               0
        Conv2d-18          [-1, 512, 16, 16]       1,180,160
   BatchNorm2d-19          [-1, 512, 16, 16]           1,024
          ReLU-20          [-1, 512, 16, 16]               0
     MaxPool2d-21            [-1, 512, 4, 4]               0
        Conv2d-22            [-1, 512, 4, 4]       2,359,808
   BatchNorm2d-23            [-1, 512, 4, 4]           1,024
          ReLU-24            [-1, 512, 4, 4]               0
        Conv2d-25            [-1, 512, 4, 4]       2,359,808
   BatchNorm2d-26            [-1, 512, 4, 4]           1,024
          ReLU-27            [-1, 512, 4, 4]               0
     MaxPool2d-28            [-1, 512, 1, 1]               0
       Flatten-29                  [-1, 512]               0
        Linear-30                   [-1, 38]          19,494
================================================================
Total params: 6,589,734
Trainable params: 6,589,734
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.75
Forward/backward pass size (MB): 343.95
Params size (MB): 25.14
Estimated Total Size (MB): 369.83
----------------------------------------------------------------
None
```

```
# for training
@torch.no_grad()
```

```
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)


def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']

def fit_OneCycle(epochs, max_lr, model, train_loader, val_loader, weight_decay=0,
                 grad_clip=None, opt_func=torch.optim.SGD):
    torch.cuda.empty_cache()
    history = []

    optimizer = opt_func(model.parameters(), max_lr, weight_decay=weight_decay)
    # scheduler for one cycle learniing rate
    sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epochs,
steps_per_epoch=len(train_loader))


    for epoch in range(epochs):
        # Training
        model.train()
        train_losses = []
        lrs = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

            # gradient clipping
            if grad_clip:
                nn.utils.clip_grad_value_(model.parameters(), grad_clip)

            optimizer.step()
            optimizer.zero_grad()

            # recording and updating learning rates
            lrs.append(get_lr(optimizer))
            sched.step()


        # validation
        result = evaluate(model, val_loader)
```

```
    result['train_loss'] = torch.stack(train_losses).mean().item()
    result['lrs'] = lrs
    model.epoch_end(epoch, result)
    history.append(result)

  return history
```

```
%%time
history = [evaluate(model, valid_dl)]
history
```

```
CPU times: user 1min, sys: 3.81 s, total: 1min 4s
Wall time: 1min 9s

[{'val_loss': tensor(3.6378, device='cuda:0'), 'val_accuracy': tensor(0.0282)}]
```

```
epochs = 2
max_lr = 0.01
grad_clip = 0.1
weight_decay = 1e-4
opt_func = torch.optim.Adam
```

Step 24: Verifying Model Output Shape

```
%%time
history += fit_OneCycle(epochs, max_lr, model, train_dl, valid_dl,
                        grad_clip=grad_clip,
                        weight_decay=1e-4,
                        opt_func=opt_func)
```

```
Epoch [0], last_lr: 0.00812, train_loss: 0.7501, val_loss: 0.7137, val_acc: 0.7926
Epoch [1], last_lr: 0.00000, train_loss: 0.1235, val_loss: 0.0265, val_acc: 0.9919
CPU times: user 17min 46s, sys: 15min 7s, total: 32min 53s
Wall time: 33min 17s
```

Step 25: Visualization

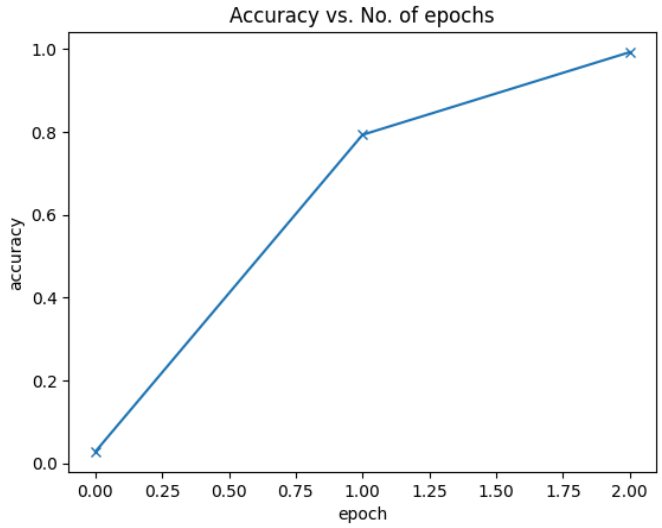
```
def plot_accuracies(history):
  accuracies = [x['val_accuracy'] for x in history]
  plt.plot(accuracies, '-x')
  plt.xlabel('epoch')
  plt.ylabel('accuracy')
  plt.title('Accuracy vs. No. of epochs');

def plot_losses(history):
  train_losses = [x.get('train_loss') for x in history]
  val_losses = [x['val_loss'] for x in history]
  plt.plot(train_losses, '-bx')
  plt.plot(val_losses, '-rx')
  plt.xlabel('epoch')
```

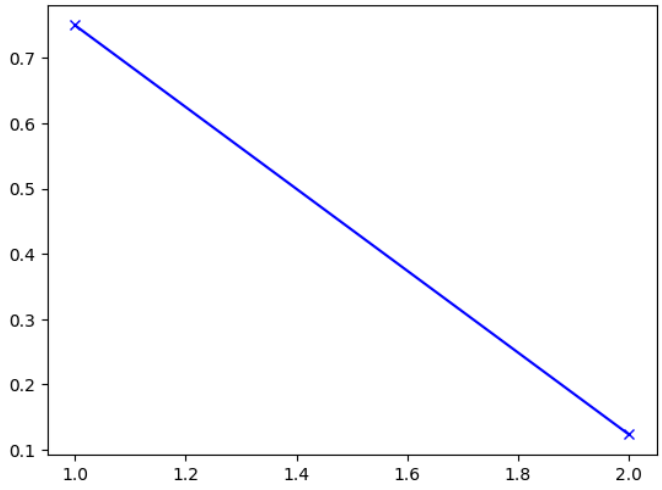
```
    plt.ylabel('loss')
    plt.legend(['Training', 'Validation'])
    plt.title('Loss vs. No. of epochs');

def plot_lrs(history):
    lrs = np.concatenate([x.get('lrs', []) for x in history])
    plt.plot(lrs)
    plt.xlabel('Batch no.')
    plt.ylabel('Learning rate')
    plt.title('Learning Rate vs. Batch no.');
```

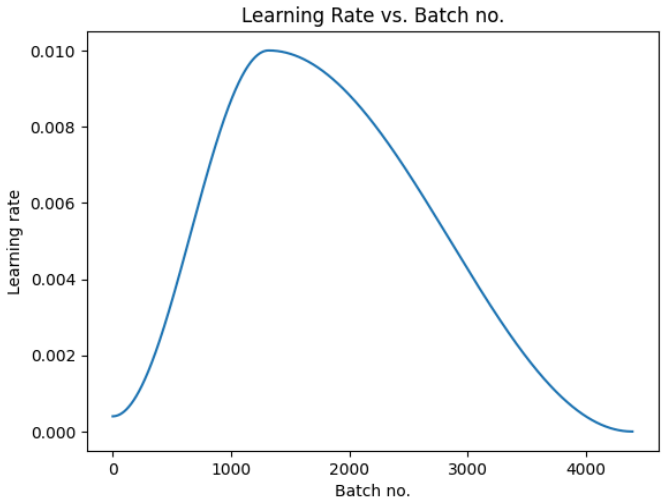
```
plot_accuracies(history)
```




```
plot_losses(history)
```




```
plot_lrs(history)
```




```
test_dir = "/content/new-plant-diseases-dataset/test"
test = ImageFolder(test_dir, transform=transforms.ToTensor())
```

```
test_images = sorted(os.listdir(test_dir + '/test')) # since images in test folder are in alphabetical order
test_images
```

```
['AppleCedarRust1.JPG',
 'AppleCedarRust2.JPG',
 'AppleCedarRust3.JPG',
 'AppleCedarRust4.JPG',
 'AppleScab1.JPG',
 'AppleScab2.JPG',
 'AppleScab3.JPG',
 'CornCommonRust1.JPG',
 'CornCommonRust2.JPG',
 'CornCommonRust3.JPG',
 'PotatoEarlyBlight1.JPG',
 'PotatoEarlyBlight2.JPG',
 'PotatoEarlyBlight3.JPG',
 'PotatoEarlyBlight4.JPG',
 'PotatoEarlyBlight5.JPG',
 'PotatoHealthy1.JPG',
 'PotatoHealthy2.JPG',
 'TomatoEarlyBlight1.JPG',
 'TomatoEarlyBlight2.JPG',
 'TomatoEarlyBlight3.JPG',
 'TomatoEarlyBlight4.JPG',
 'TomatoEarlyBlight5.JPG',
 'TomatoEarlyBlight6.JPG',
 'TomatoHealthy1.JPG',
 'TomatoHealthy2.JPG',
 'TomatoHealthy3.JPG',
 'TomatoHealthy4.JPG',
 'TomatoYellowCurlVirus1.JPG',
 'TomatoYellowCurlVirus2.JPG',
 'TomatoYellowCurlVirus3.JPG',
 'TomatoYellowCurlVirus4.JPG',
 'TomatoYellowCurlVirus5.JPG',
 'TomatoYellowCurlVirus6.JPG']
```

```
def predict_image(img, model):
```

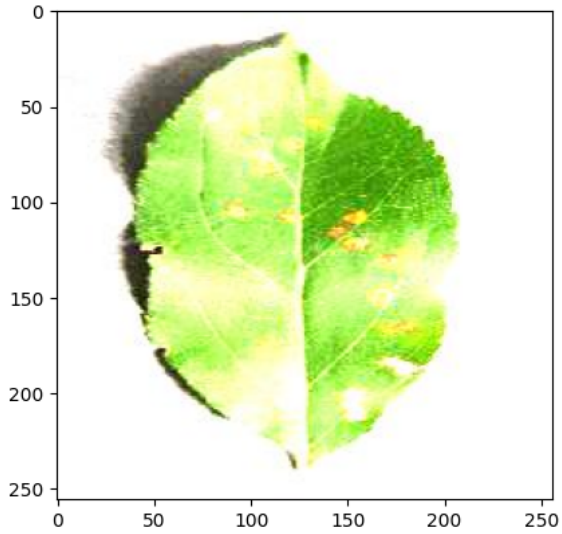
```
    """Converts image to array and return the predicted class
       with highest probability"""
    # Convert to a batch of 1
    xb = to_device(img.unsqueeze(0), device)
    # Get predictions from model
    yb = model(xb)
    # Pick index with highest probability
    _, preds  = torch.max(yb, dim=1)
    # Retrieve the class label

    return train.classes[preds[0].item()]
```

Step 26: Displaying Prediction Result for Image

```
img, label = test[0]
plt.imshow(img.permute(1, 2, 0))
print('Label:', test_images[0], ', Predicted:', predict_image(img, model))
```

Label: AppleCedarRust1.JPG , Predicted: Apple___Cedar_apple_rust



Step 27: Final Accuracy and Performance Summary

```
# getting all predictions (actual label vs predicted)
for i, (img, label) in enumerate(test):
    print('Label:', test_images[i], ', Predicted:', predict_image(img, model))
```

```
Label: AppleCedarRust1.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleCedarRust2.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleCedarRust3.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleCedarRust4.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleScab1.JPG , Predicted: Apple___Apple_scab
Label: AppleScab2.JPG , Predicted: Apple___Apple_scab
Label: AppleScab3.JPG , Predicted: Apple___Apple_scab
Label: CornCommonRust1.JPG , Predicted: Corn_(maize)___Common_rust_
Label: CornCommonRust2.JPG , Predicted: Corn_(maize)___Common_rust_
Label: CornCommonRust3.JPG , Predicted: Corn_(maize)___Common_rust_
Label: PotatoEarlyBlight1.JPG , Predicted: Potato___Early_blight
Label: PotatoEarlyBlight2.JPG , Predicted: Potato___Early_blight
Label: PotatoEarlyBlight3.JPG , Predicted: Potato___Early_blight
```

```
Label: PotatoEarlyBlight4.JPG , Predicted: Potato___Early_blight
Label: PotatoEarlyBlight5.JPG , Predicted: Potato___Early_blight
Label: PotatoHealthy1.JPG , Predicted: Potato___healthy
Label: PotatoHealthy2.JPG , Predicted: Potato___healthy
Label: TomatoEarlyBlight1.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight2.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight3.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight4.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight5.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight6.JPG , Predicted: Tomato___Early_blight
Label: TomatoHealthy1.JPG , Predicted: Tomato___healthy
Label: TomatoHealthy2.JPG , Predicted: Tomato___healthy
Label: TomatoHealthy3.JPG , Predicted: Tomato___healthy
Label: TomatoHealthy4.JPG , Predicted: Tomato___healthy
Label: TomatoYellowCurlVirus1.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus2.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus3.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus4.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus5.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus6.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
```

Conclusion : Through this experiment, we successfully built a cognitive-based application that acquires knowledge from images and extends the functionality of our Smart Gardening Assistant. By integrating deep learning and computer vision, the system can analyze plant or garden images and provide useful advice. This approach demonstrates how cognitive applications can be applied not only in gardening but also in domains like customer service, insurance, healthcare, smarter cities, and government.

49_Prac_03