Problem Statement : Build CNN Model for Classification Of Flowers

```
!gdown --id 1xkynpL15pt6KT3YSlDimu4A5iRU9qYck
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was deprecated in version 4.3.1 and wi
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1xkynpL15pt6KT3YSlDimu4A5iRU9qYck
To: /content/Flowers-Dataset.zip
100% 236M/236M [00:00<00:00, 319MB/s]
```

```
!unzip Flowers-Dataset.zip
```

```
Archive:  Flowers-Dataset.zip
  inflating: flowers/daisy/100080576_f52e8ee070_n.jpg
  inflating: flowers/daisy/10140303196_b88d3d6cec.jpg
  inflating: flowers/daisy/10172379554_b296050f82_n.jpg
  inflating: flowers/daisy/10172567486_2748826a8b.jpg
  inflating: flowers/daisy/10172636503_21bededa75_n.jpg
  inflating: flowers/daisy/102841525_bd6628ae3c.jpg
  inflating: flowers/daisy/10300722094_28fa978807_n.jpg
  inflating: flowers/daisy/1031799732_e7f4008c03.jpg
  inflating: flowers/daisy/10391248763_1d16681106_n.jpg
  inflating: flowers/daisy/10437754174_22ec990b77_m.jpg
  inflating: flowers/daisy/10437770546_8bb6f7bdd3_m.jpg
  inflating: flowers/daisy/10437929963_bc13eebe0c.jpg
  inflating: flowers/daisy/10466290366_cc72e33532.jpg
  inflating: flowers/daisy/10466558316_a7198b87e2.jpg
  inflating: flowers/daisy/10555749515_13a12a026e.jpg
  inflating: flowers/daisy/10555815624_dc211569b0.jpg
  inflating: flowers/daisy/10555826524_423eb8bf71_n.jpg
  inflating: flowers/daisy/10559679065_50d2b16f6d.jpg
  inflating: flowers/daisy/105806915_a9c13e2106_n.jpg
  inflating: flowers/daisy/10712722853_5632165b04.jpg
  inflating: flowers/daisy/107592979_aaa9cdfe78_m.jpg
  inflating: flowers/daisy/10770585085_4742b9dac3_n.jpg
  inflating: flowers/daisy/10841136265_af473efc60.jpg
  inflating: flowers/daisy/10993710036_2033222c91.jpg
  inflating: flowers/daisy/10993818044_4c19b86c82.jpg
  inflating: flowers/daisy/10994032453_ac7f8d9e2e.jpg
  inflating: flowers/daisy/11023214096_b5b39fab08.jpg
  inflating: flowers/daisy/11023272144_fce94401f2_m.jpg
  inflating: flowers/daisy/11023277956_8980d53169_m.jpg
  inflating: flowers/daisy/11124324295_503f3a0804.jpg
  inflating: flowers/daisy/1140299375_3aa7024466.jpg
  inflating: flowers/daisy/11439894966_dca877f0cd.jpg
  inflating: flowers/daisy/1150395827_6f94a5c6e4_n.jpg
  inflating: flowers/daisy/11642632_1e7627a2cc.jpg
  inflating: flowers/daisy/11834945233_a53b7a92ac_m.jpg
  inflating: flowers/daisy/11870378973_2ec1919f12.jpg
  inflating: flowers/daisy/11891885265_ccefec7284_n.jpg
  inflating: flowers/daisy/12193032636_b50ae7db35_n.jpg
  inflating: flowers/daisy/12348343085_d4c396e5b5_m.jpg
  inflating: flowers/daisy/12585131704_0f64b17059_m.jpg
  inflating: flowers/daisy/12601254324_3cb62c254a_m.jpg
  inflating: flowers/daisy/1265350143_6e2b276ec9.jpg
  inflating: flowers/daisy/12701063955_4840594ea6_n.jpg
  inflating: flowers/daisy/1285423653_18926dc2c8_n.jpg
  inflating: flowers/daisy/1286274236_1d7ac84efb_n.jpg
  inflating: flowers/daisy/12891819633_e4c82b51e8.jpg
  inflating: flowers/daisy/1299501272_59d9da5510_n.jpg
  inflating: flowers/daisy/1306119996_ab8ae14d72_n.jpg
  inflating: flowers/daisy/1314069875_da8dc023c6_m.jpg
  inflating: flowers/daisy/1342002397_9503c97b49.jpg
  inflating: flowers/daisy/134409839_71069a95d1_m.jpg
  inflating: flowers/daisy/1344985627_c3115e2d71_n.jpg
  inflating: flowers/daisy/13491959645_2cd9df44d6_n.jpg
  inflating: flowers/daisy/1354396826_2868631432_m.jpg
  inflating: flowers/daisy/1355787476_32e9f2a30b.jpg
  inflating: flowers/daisy/13583238844_573df2de8e_m.jpg
  inflating: flowers/daisy/1374193928_a52320eafa.jpg
```

```
import numpy as np
import pandas as pd
import os
import torch
import torchvision
import tarfile
import torchvision
from torch.utils.data import random_split
```

```
from torchvision import transforms
from torchvision.transforms import ToTensor
from torch.utils.data.dataloader import DataLoader
import torch.nn as nn
from torchvision.utils import make_grid
import torchvision.models as models
import torch.nn.functional as F
import matplotlib.pyplot as plt
%matplotlib inline
```

```
data_dir ="flowers"
```

Applying Data Augmentation

```
transformer = torchvision.transforms.Compose(
    [
        torchvision.transforms.Resize((224, 224)),
        torchvision.transforms.RandomHorizontalFlip(p=0.5),
        torchvision.transforms.RandomVerticalFlip(p=0.5),
        torchvision.transforms.RandomRotation(40),
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize(
            mean=[0.4914, 0.4822, 0.4465], std=[0.2023, 0.1994, 0.2010]
        ),
    ]
)
database = ImageFolder(data_dir, transform=transformer)
```

```
database.classes
```

```
['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

```
flower_name_dict={'daisy':0 ,
                  'dandelion':1,
                  'rose':2 ,
                  'sunflower':3 ,
                  'tulip':4}

def encode_label(img_label):
    return flower_name_dict[img_label]
```

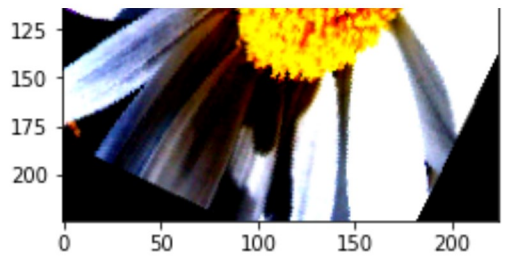
```
flower_label = {
    0: 'daisy',
    1: 'dandelion',
    2: 'rose',
    3: 'sunflower',
    4: 'tulip'
}
```

```
def show_batch(dl,invert=True):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(12, 6))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
        break

def show_sample(image, label,invert=True):
    print("Label :" +database.classes[label] + "(" + str(label) + ")")
    plt.imshow(image.permute(1, 2, 0))
```

```
show_sample(*database[1])
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for i
Label :daisy(0)
```



Splitting the data for validation dataset and train dataset

```
validation_size = 500
training_size = len(database) - validation_size
```

```
train_ds, val_ds_main = random_split(database,[training_size, validation_size])
val_ds, test_ds  = random_split(val_ds_main,[300, 200])
len(train_ds), len(val_ds)
```

```
(3817, 300)
```

```
batch_size=32
```

```
train_dl = DataLoader(train_ds, batch_size , shuffle=True)
val_dl = DataLoader(val_ds,batch_size)
test_dl = DataLoader(test_ds, batch_size)
```

```
show_batch(train_dl,invert=True)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for i
```

1. Training the model for image classification

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

class ImageClassification(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 predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

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

```
class FlowerModel(ImageClassification):
    def __init__(self):
        super().__init__()
```

```
        self.network = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Flatten(),
            nn.Linear(256*28*28, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 5))

    def forward(self, xb):
        return self.network(xb)
```

```
model = FlowerModel()
model
```

```
      FlowerModel(
        (network): Sequential(
          (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): ReLU()
          (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (3): ReLU()
          (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
          (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (6): ReLU()
          (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (8): ReLU()
          (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
          (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (11): ReLU()
          (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (13): ReLU()
          (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
          (15): Flatten(start_dim=1, end_dim=-1)
          (16): Linear(in_features=200704, out_features=1024, bias=True)
          (17): ReLU()
          (18): Linear(in_features=1024, out_features=512, bias=True)
          (19): ReLU()
          (20): Linear(in_features=512, out_features=5, bias=True)
        )
      )
```

```
for images, labels in train_dl:
  print('images.shape:', images.shape)
  out = model(images)
  print('out.shape:', out.shape)
  print('out[0]:', out[0])
  break
```

```
      images.shape: torch.Size([32, 3, 224, 224])
      out.shape: torch.Size([32, 5])
      out[0]: tensor([-0.0363, -0.0462,  0.0598, -0.0040, -0.0016],
             grad_fn=<SelectBackward0>)
```

```
@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 fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD):
    history = []
    optimizer = opt_func(model.parameters(), lr)
```

```
    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

Get GPU up on running

```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

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)

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 = get_default_device()
device
```

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

Training and Validation Datasets

```
train_dl = DeviceDataLoader(train_dl, device)
val_dl = DeviceDataLoader(val_dl, device)
to_device(model, device);
```

```
model = to_device(FlowerModel(), device)
```

```
evaluate(model, val_dl)
```

```
    {'val_loss': 1.6099056005477905, 'val_acc': 0.21041667461395264}
```

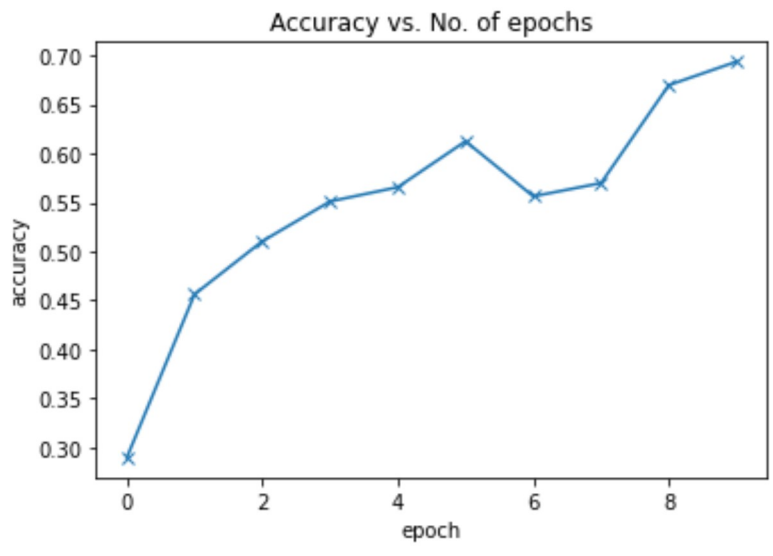
```
num_epochs = 10
opt_func = torch.optim.Adam
lr = 0.001
```

```
%%time
try1 = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

```
Epoch [0], train_loss: 1.5960, val_loss: 1.4159, val_acc: 0.2896
Epoch [1], train_loss: 1.2531, val_loss: 1.2338, val_acc: 0.4563
Epoch [2], train_loss: 1.1317, val_loss: 1.2627, val_acc: 0.5104
Epoch [3], train_loss: 1.0311, val_loss: 1.0987, val_acc: 0.5510
Epoch [4], train_loss: 0.9758, val_loss: 1.2742, val_acc: 0.5656
Epoch [5], train_loss: 0.9406, val_loss: 1.0392, val_acc: 0.6125
Epoch [6], train_loss: 0.8842, val_loss: 1.0742, val_acc: 0.5562
Epoch [7], train_loss: 0.8542, val_loss: 1.1149, val_acc: 0.5698
Epoch [8], train_loss: 0.8431, val_loss: 0.9417, val_acc: 0.6698
Epoch [9], train_loss: 0.7970, val_loss: 0.9862, val_acc: 0.6938
CPU times: user 6min 51s, sys: 4.88 s, total: 6min 56s
Wall time: 6min 58s
```

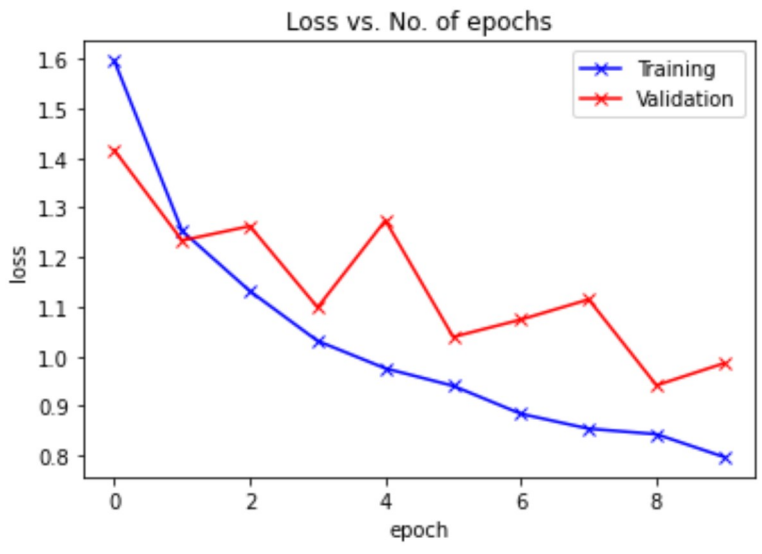
```
def plot_accuracies(try1):
    accuracies = [x['val_acc'] for x in try1]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of epochs');
```

```
plot_accuracies(try1)
```




```
def plot_losses(try1):
    train_losses = [x.get('train_loss') for x in try1]
    val_losses = [x['val_loss'] for x in try1]
    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');
```

```
plot_losses(try1)
```




```
test_dl = DeviceDataLoader(test_dl, device)
evaluate(model, test_dl)
```

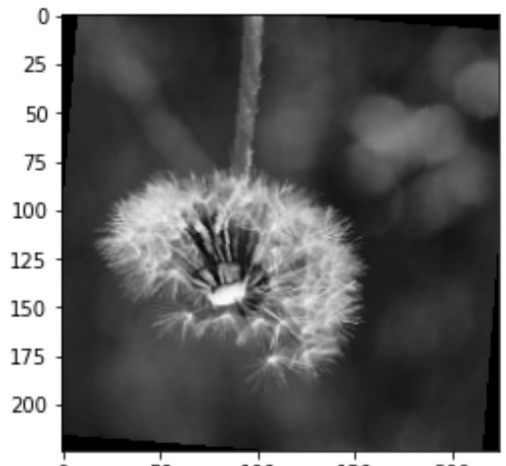
```
{'val_loss': 0.8375752568244934, 'val_acc': 0.7008928656578064}
```

Predict the image

```
def predict_image(img, model):
    xb = img.unsqueeze(0)
    yb = model(xb)
    _, preds  = torch.max(yb, dim=1)
    return flower_label[preds[0].item()]
```

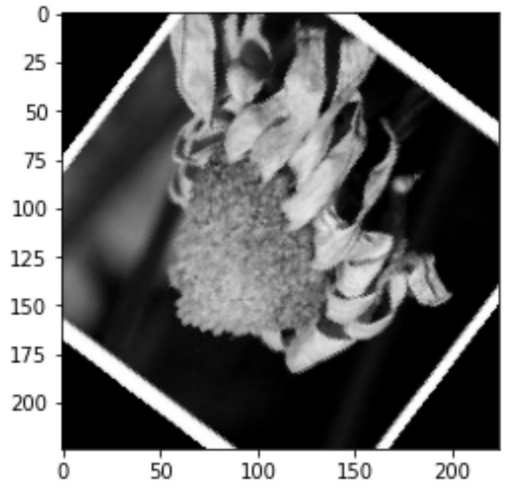
```
img, label = test_ds[1]
plt.imshow(img[0], cmap='gray')
print('Label:', database.classes[label], ', Predicted:', predict_image(img, FlowerModel()))
```

```
Label: dandelion , Predicted: dandelion
```




```
img, label = test_ds[5]
plt.imshow(img[0], cmap='gray')
print('Label:', database.classes[label], ', Predicted:', predict_image(img, FlowerModel()))
```

```
Label: daisy , Predicted: dandelion
```



Saving the model

```
torch.save(model.state_dict(), 'Flower-classification.pt')
```