Open in Colab

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`
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1xkynpL15pt6KT3YSlDimu4A5iRU9qYck
To: /content/Flowers-Dataset.zip
100% 236M/236M [00:02<00:00, 109MB/s]
```

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

```
  inflating: flowers/tulip/8712270243_8512cf4fbd.jpg
  inflating: flowers/tulip/8712270665_57b5bda0a2_n.jpg
  inflating: flowers/tulip/8712282563_3819afb7bc.jpg
  inflating: flowers/tulip/8713357842_9964a93473_n.jpg
  inflating: flowers/tulip/8713387500_6a9138b41b_n.jpg
  inflating: flowers/tulip/8713388322_e5ae26263b_n.jpg
  inflating: flowers/tulip/8713389178_66bceb71a8_n.jpg
  inflating: flowers/tulip/8713390684_041148dd3e_n.jpg
  inflating: flowers/tulip/8713391394_4b679ea1e3_n.jpg
  inflating: flowers/tulip/8713392604_90631fb809_n.jpg
  inflating: flowers/tulip/8713394070_b24561b0a9.jpg
  inflating: flowers/tulip/8713396140_5af8136136.jpg
  inflating: flowers/tulip/8713397358_0505cc0176_n.jpg
  inflating: flowers/tulip/8713397694_bcbcbba2c2_n.jpg
  inflating: flowers/tulip/8713398114_bc96f1b624_n.jpg
  inflating: flowers/tulip/8713398614_88202e452e_n.jpg
  inflating: flowers/tulip/8713398906_28e59a225a_n.jpg
  inflating: flowers/tulip/8713407768_f880df361f.jpg
  inflating: flowers/tulip/8717900362_2aa508e9e5.jpg
  inflating: flowers/tulip/8722514702_7ecc68691c.jpg
  inflating: flowers/tulip/8723767533_9145dec4bd_n.jpg
  inflating: flowers/tulip/8729501081_b993185542_m.jpg
  inflating: flowers/tulip/8733586143_3139db6e9e_n.jpg
  inflating: flowers/tulip/8748266132_5298a91dcf_n.jpg
  inflating: flowers/tulip/8750288831_5e49a9f29b.jpg
  inflating: flowers/tulip/8757486380_90952c5377.jpg
  inflating: flowers/tulip/8758464923_75a5ffe320_n.jpg
  inflating: flowers/tulip/8758519201_16e8d2d781_n.jpg
  inflating: flowers/tulip/8759594528_2534c0ec65_n.jpg
  inflating: flowers/tulip/8759597778_7fca5d434b_n.jpg
  inflating: flowers/tulip/8759601388_36e2a50d98_n.jpg
  inflating: flowers/tulip/8759606166_8e475013fa_n.jpg
  inflating: flowers/tulip/8759618746_f5e39fdbf8_n.jpg
  inflating: flowers/tulip/8762189906_8223cef62f.jpg
  inflating: flowers/tulip/8762193202_0fbf2f6a81.jpg
  inflating: flowers/tulip/8768645961_8f1e097170_n.jpg
```

```
          inflating: flowers/tulip/8817622133_a42bb90e38_n.jpg
          inflating: flowers/tulip/8838347159_746d14e6c1_m.jpg
          inflating: flowers/tulip/8838354855_c474fc66a3_m.jpg
          inflating: flowers/tulip/8838914676_8ef4db7f50_n.jpg
          inflating: flowers/tulip/8838975946_f54194894e_m.jpg
          inflating: flowers/tulip/8838983024_5c1a767878_n.jpg
          inflating: flowers/tulip/8892851067_79242a7362_n.jpg
          inflating: flowers/tulip/8904780994_8867d64155_n.jpg
          inflating: flowers/tulip/8908062479_449200a1b4.jpg
          inflating: flowers/tulip/8908097235_c3e746d36e_n.jpg
          inflating: flowers/tulip/9019694597_2d3bbedb17.jpg
          inflating: flowers/tulip/9030467406_05e93ff171_n.jpg
          inflating: flowers/tulip/9048307967_40a164a459_m.jpg
          inflating: flowers/tulip/924782410_94ed7913ca_m.jpg
          inflating: flowers/tulip/9378657435_89fabf13c9_n.jpg
          inflating: flowers/tulip/9444202147_405290415b_n.jpg
          inflating: flowers/tulip/9446982168_06c4d71da3_n.jpg
          inflating: flowers/tulip/9831362123_5aac525a99_n.jpg
          inflating: flowers/tulip/9870557734_88eb3b9e3b_n.jpg
          inflating: flowers/tulip/9947374414_fdf1d0861c_n.jpg
          inflating: flowers/tulip/9947385346_3a8cacea02_n.jpg
          inflating: flowers/tulip/9976515506_d496c5e72c.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.datasets import ImageFolder
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),
```

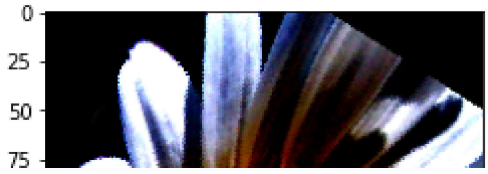
```
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize(
            mean=[0.4914, 0.4822, 0.4465], std=[0.2023, 0.1994, 0.2010]
        ),
    ]
)
database = ImageFolder(data_dir, transform=transformer)
```

```
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 d
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 d
```

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([-0.0264,  0.0084,  0.0092, -0.0073,  0.0259],
             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': 1.608038306236267, 'val_acc': 0.21145835518836975}
```

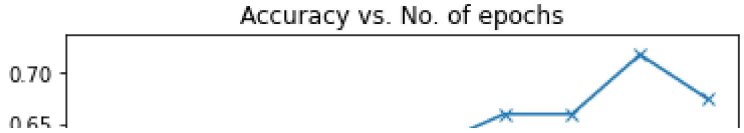
```
num_epochs = 10
opt_func = torch.optim.Adam
lr = 0.001
```

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

```
Epoch [0], train_loss: 1.5893, val_loss: 1.3745, val_acc: 0.3333
Epoch [1], train_loss: 1.2759, val_loss: 1.2067, val_acc: 0.5042
Epoch [2], train_loss: 1.1504, val_loss: 1.0649, val_acc: 0.5865
Epoch [3], train_loss: 1.0730, val_loss: 0.9798, val_acc: 0.5865
Epoch [4], train_loss: 1.0116, val_loss: 0.9407, val_acc: 0.6219
Epoch [5], train_loss: 0.9555, val_loss: 0.8839, val_acc: 0.6302
Epoch [6], train_loss: 0.8876, val_loss: 0.9227, val_acc: 0.6594
Epoch [7], train_loss: 0.8574, val_loss: 0.8727, val_acc: 0.6594
Epoch [8], train_loss: 0.8345, val_loss: 0.7943, val_acc: 0.7167
Epoch [9], train_loss: 0.8237, val_loss: 0.8049, val_acc: 0.6740
CPU times: user 6min 49s, sys: 4.88 s, total: 6min 54s
Wall time: 6min 55s
```

```
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');
```

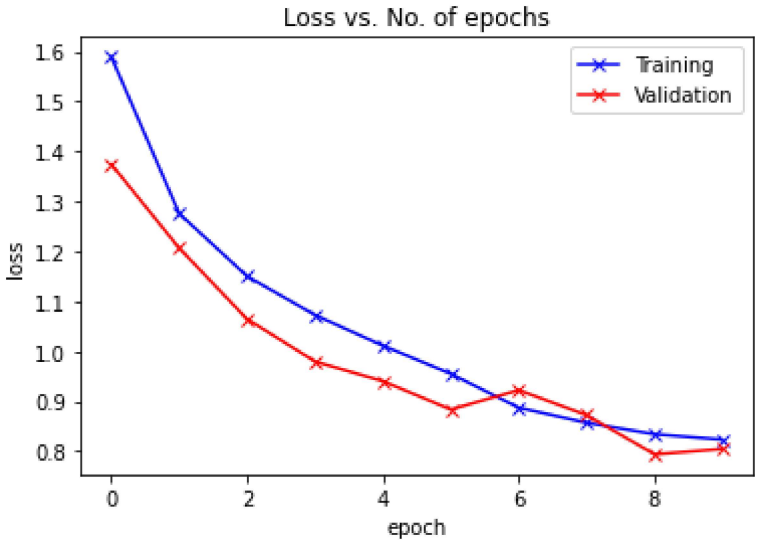
```
plot_accuracies(try1)
```




```
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');
```

epoch

```
plot_losses(try1)
```




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

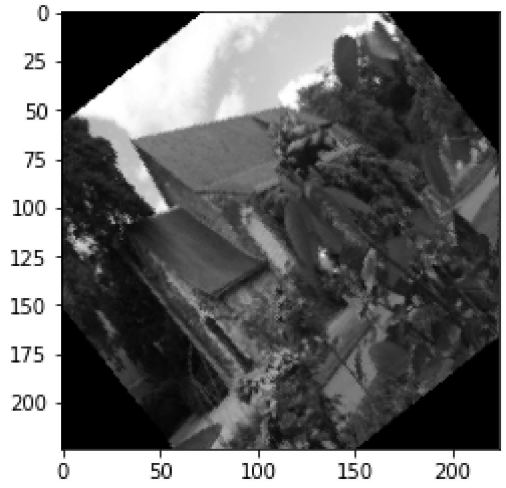
```
{'val_loss': 0.8072760105133057, 'val_acc': 0.6830357313156128}
```

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()]
```

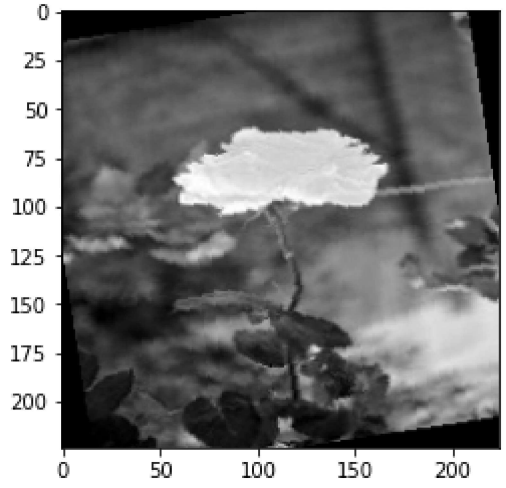
```
img, label = test_ds[1]
plt.imshow(img[0], cmap='gray')
print('Label:', database.classes[label], ', Predicted:', predict_image(img, FlowerModel())
```

```
Label: rose , Predicted: sunflower
```




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

```
Label: rose , Predicted: tulip
```



Saving the model

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

Colab paid products - Cancel contracts here

✓ 4s completed at 3:42 PM