Assignment No.6

Title: Object detection using Transfer Learning of CNN architectures

Aim: Object detection using Transfer Learning of CNN architectures

a. Load in a pre-trained CNN model trained on a large dataset

b. Freeze parameters (weights) in model's lower convolutional layers

c. Add custom classifier with several layers of trainable parameters to model

d. Train classifier layers on training data available for task

e. Fine-tune hyper parameters and unfreeze more layers as needed

Theory:

1) What is Transfer learning ?
2) What are pretrained Neural Network models ?
3) Explain Pytorch library in short.
4) What are advantages of Transfer learning.
5) What are applications of Transfer learning.
6) Explain Caltech 101 images dataset.
7) Explain Imagenet dataset **.**
8) List down basic steps for transfer learning.
9) What is Data augmentation?
10) How and why Data augmentation is done related to transfer learning?
11) Why preprocessing is needed on inputdata in Transfer learning.
12) What is PyTorch Transforms module.Explain following commands w.r.t it :

```
Compose([RandomResizedCrop(size=256, scale=(0.8, 1.0)),RandomRotation(degrees=15),
        ColorJitter(),
        RandomHorizontalFlip(),
        CenterCrop(size=224),  # Image net standards
   .ToTensor(),
        Normalize
```

13) Explain the Validation Transforms steps with Pytorch Transforms.
14) Explain VGG-16 model from Pytorch

Steps/ Algorithm

1. Dataset link and libraries :

https://data.caltech.edu/records/mzrjq-6wc02

separate the data into training, validation, and testing sets with a 50%, 25%, 25% split

andthen structured the directories as follows:

/datadir

/train

/class1

/class2

.

.

/valid

/class1

/class2

.

.

/test

/class1

/class2

.

Libraries required :

PyTorch

torchvision import transforms

torchvision import datasets

torch.utils.data import

DataLoadertorchvision

import models torch.nn as nn

torch import optim

Ref: https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce

1) Prepare the dataset in splitting in three directories Train , alidation and test with 50 25 25

2) Do pre-processing on data with transform from PytorchTraining dataset transformation as follows : transforms.Compose([

    transforms.RandomResizedCrop(size=256,

    scale=(0.8, 1.0)),

    transforms.RandomRotation(degrees=15),

    transforms.ColorJitter(),

    transforms.RandomHorizontalFlip(),

    transforms.CenterCrop(size=224), # Image net

    standards transforms.ToTensor(),

    transforms.Normalize([0.485, 0.456, 0.406],

    [0.229, 0.224, 0.225]) # Imagenet

   standardsValidation Dataset transform as follows :

   transforms.Compose([

    transforms.Resize(size=256)

    ,

    transforms.CenterCrop(size=

    224),transforms.ToTensor(),

    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

3) Create Datasets and

   Loaders :data = {

    'train':(Our name given to train data set dir created)

    datasets.ImageFolder(root=traindir,

    transform=image_transforms['train']),'valid':

    datasets.ImageFolder(root=validdir, transform=image_transforms['valid']),

   }

   dataloaders = {

    'train': DataLoader(data['train'], batch_size=batch_size,

    shuffle=True),'val': DataLoader(data['valid'],

    batch_size=batch_size, shuffle=True)

}

4) Load Pretrain Model : from torchvision import models

model = model.vgg16(pretrained=True)

5) Freez all the Models Weight

for param in

model.parameters():

param.requires_grad =

False

6) Add our own custom classifier with following parameters :

Fully connected with ReLU activation, shape = (n_inputs,

256)Dropout with 40% chance of dropping

Fully connected with log softmax output, shape = (256,

n_classes)import torch.nn as nn

Add on classifier

model.classifier[6] =

nn.Sequential(

nn.Linear(n_inputs,

256),nn.ReLU(),

nn.Dropout(0.4),

nn.Linear(256,

n_classes),

nn.LogSoftmax(dim

=1))

7) Only train the sixth layer of classifier keep remaining layers

off .Sequential(

(0): Linear(in_features=25088, out_features=4096,

bias=True)(1): ReLU(inplace)

(2) : Dropout(p=0.5)

(3) : Linear(in_features=4096, out_features=4096,

bias=True)(4): ReLU(inplace)

```
(5): Dropout(p=0.5)
(6): Sequential(
  (0): Linear(in_features=4096, out_features=256,
  bias=True)(1): ReLU()
  (2): Dropout(p=0.4)
  (3): Linear(in_features=256, out_features=100,
  bias=True)(4): LogSoftmax()
 )
)
```

8) Initialize the loss and

```
optimizercriteration =
nn.NLLLoss()
optimizer = optim.Adam(model.parameters())
```

9) Train the model using

```
Pytorch for epoch in
range(n_epochs): for data,
targets in trainloader:
    # Generate
    predictionsout =
    model(data)
    # Calculate loss
    loss = criterion(out,
    targets)#
    Backpropagation
    loss.backward()
    # Update model
    parameters
    optimizer.step()
```

10) Perform Early stopping

11) Draw performance curve

12) Calculate Accuracy

pred = torch.max(ps,

dim=1)equals = pred

== targets

Calculate accuracy

accuracy = torch.mean(equals)

Conclusion: In this experiment, we were able to see the basics of using PyTorch as well as the concept of transfer learning, an effective method for object recognition. Instead of training a model from scratch, we can use existing architectures that have been trained on a large dataset and then tune them for our task. This reduces the time to train and often results in better overall performance. The outcome of this experiment is knowledge of transfer learning and PyTorch that we can build on to build more complex applications.