Assignment : 6

DATE

31

- **Aim**

Object detection using Transfer learning of CNN architecture.

a. Load in pre-trained CNN model trained on large dataset
b. Freeze parameters (weight) in the model's lower convolutional layers.
c. Add a custom classifier with several layers of trainable parameter to model
d. Train classifier layer on training data available for the task
e. fine-tune hyperparameters and unfreeze more layers as needed.

- **Course objectives**

1. To apply the algorithms to a real-world problem, optimise the models learned and report on the expected accuracy that can be achieved by applying the models.

- **Course-outcomes**

CO3: Apply Deep learning technique like CNN and RNN Auto encoders to solve real-world problems
CO4: Evaluate the performance of the model built using Deep learning.

FOR EDUCATIONAL USE

- Softwares and hardware requirements

sr. no	Hardwares and hardware	version / specification
1.	Jupyter notebook	V. 7.13.008
2.	computer / PC	IS version, 64 bits, 8GB RAM

- Theory

Steps / Algorithm of object deetection using Transfer learning of CNN architecture.

step 1 :

Dataset link and libraries

~~step2~~ :

/ test
/ class 1
/ class 2

Libraries required :

PyTorch

torchvision important transform

pip install pytorch
(for installing pytorch)

FOR EDUCATIONAL USE

step 2 :

Prepare the dataset in splitting in three directory Train, validation and test with 50 25 ~~50~~ 25

step 3 :

Do pre-processing on data with transform from pytorch training dataset transformation as follows:
transform. compose ([
transform. RandomResized Crop (size = 256, scale = (0.8, 1.0)),
transform . Random Rotation (degrees = 15), transform. (color Jitter()
transform · RandomHorizontalFlip (), transforms . Center Crop (size = 224,

step 4 :

Create Datasets and Loaders : data = {
'train': (our name given to train data set dir created)
datasets · ImageFolder (root = traindir, transform = image_transforms ['train']), 'valid':
dataset · ImageFolder (root = validdir, transform = image_transform ['valid']),

DataLoaders = {
'train': DataLoader (data ['train'], batch_size = batch_size, shuffle = True), 'val':
DataLoader (data ['valid'], batch_size = batch_size, shuffle = True) }

FOR EDUCATIONAL USE

step 5 :

Load Pretrain model : from torchvision import models
model = model.vgg16 (pretrained = True)

step 6 :

freeze all the model weights
for param in model.parameters(): param.requires grad = False.

step 7 :

Add our own custom classifier with following parameters: fully connected Relu Activation function

step 8 :

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_feature = 4096, out_feature = 4096, bias = ~~True~~ True)
(4) ~~(47)~~: Relu (inplace)
(5): Dropout (p=0.5
(6): sequential(

FOR EDUCATIONAL USE

step 9 :

Initialize the loss and optimizer criterization = nn.NLLLoss()
optimizer = optim.Adam (model.parameters())

step 10:

Train the model using PyTorch for epoch in Range (n-epochs): for data, targets in trainloader:

# Generate prediction out = model (data)
# calculate loss
loss = criterion (out, targets)
# Back propagation loss.backward()
# update model parameters optimizer.step()

step 11 :

Perform Early stopping

step 12 :

Draw performance curve

step 13 :

Calculate Accuracy
Pred = torch.max(ps, ~~dim~~ dim=1) equals = pred == targets
# calculate accuracy
accuracy = torch.mean(equals)

FOR EDUCATIONAL USE

Conclusion

In this practical, I have learned about pre-trained CNN model is adapted for object detection by freezing lower layers, adding a custom classifier, and fine-tuning hyperparameters. unfreezing layers can enhance performance for the specific task.

FOR EDUCATIONAL USE