Sharif University of Technology

Computer Vision

HW05: Scene Recognition Using Neural Networks

Author:
Saeed Razavi (98106542)

July 16, 2022

question1

in this notebook we want to train cnn models with different architectures .

- PART 1 : **using first convolutional layer and two last fully connected layer of Alexnet architecture .** note that we use **max-pooling** layer with **kernel-size=4** and **stride=4** in first layer .

- PART 2 : **using three first convolutions layer and three last fully connected layer of Alexnet architecture** . in the **third layer** , we change the conv2d layer with depth of 384 to 256 .

- PART 3 : using **whole Alexnet architecture** and just change the last fully-connected layer with depth of 1000 to 15 but **without any pre-trained weights.** more specifically , we don't use pre-trained parameters that derived for architecture from imagenet dataset .

- PART 4 : using **whole Alexnet architecture** and just change the last fully-connected layer with depth of 1000 to 15 . we **use pre-trained weights** for all the layers expect for new classifier layer .during training , **we just let the parameters of new classifier layer to update** in each epoch and freeze all other parameters.

- PART 5 : using **whole Alexnet architecture** and just change the last fully-connected layer with depth of 1000 to 15 . we **use pre-trained weights** for all the layers expect for new classifier layer .during training , we **let all parameters of all layers to update in each epoch .**

hyperparameters

we determine hyperparameters as follows :

```

config = {
    '''
    note that all files are saved in folders.
    Pictures of the same category are stored in each folder.
    The folder name is class name

    '''

    'extracted_root' : "/content/Data", #path of extracted zip data
    'train_data_path' : "/content/Data/Train" , #path of extracted train data
    'test_data_path' : "/content/Data/Test" ,  # path of extracted test data
    'batch_size' : 64 ,
    'num_workers' : 1 ,
    'num_classes' : 15 ,
    'eopches' : 50 ,
    'learning_rate' : 0.0005
    'optimizer' : "torch.optim.SGD"
}
```

Listing 1: hyperparameters

- **batch-size** : set to **64** because for greater value the less update we have for the weights (doesn't converge well) and for smaller value we lost the generality of the model (the gradient is not trust-able)
- **learning rate** : set to **0.0005** and gradually decrease in next epoch
- **num-classes** : set to **15** because our classification problem has 15 class
- **epoches** : number of epoches set to **50** for this problem
- **num-workers** : indicate how multi-processing we have in each batch . is set it to **1**
- **train**$_{d}ata - path : pathofextractedtraindatathatwehavepassto$‘**datasets.ImageFolder**
- **test-data-path** : path of extracted test data that we have pass to **torchvision.datasets.ImageFolder**

Transforms

we set different transforms for training and test set

- training set : for this , we define 4 different transform in a sequential structure .
  1. resize each image (**transforms.Resize**) to **227×227**
  2. augment training set using **transforms.RandomHorizontalFlip** to mirror each image horizontally . note that because our objective is **scene recognition** , many of transforms like rotating , mirror vertically and ... lead to bad training and bad dataset . so we only use **transforms.RandomVerticalFlip**. we set the p(probability) to 1 to have one new training dataset alongside the original training dataset
  3. convert dataset to tensor
  4. normalize dataset based on alexnet mean and std
- test set : all of test-set transforms are like training-set transforms except that we don't need augmentation in test-set

```
#define normalization transform
normalize = transforms.Normalize(
        mean=[0.4914, 0.4822, 0.4465],
        std=[0.2023, 0.1994, 0.2010],)


#define train_set transform
train_transform1 = transforms.Compose([
            transforms.Resize((227,227)),
            transforms.ToTensor(),
            normalize,])

#--------------------------

train_transform2 = transforms.Compose([
            transforms.Resize((227,227)),
            transforms.RandomHorizontalFlip(p=1),
            transforms.ToTensor(),
            normalize,])
# #---------------------------

#define test_set transform
test_transform = transforms.Compose([
            transforms.Resize((227,227)),
            transforms.ToTensor(),
            normalize,])
```

Listing 2: Transforms

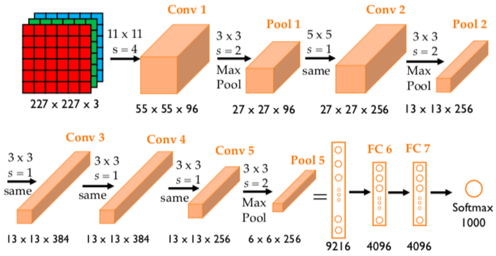
splitting data and augmentation

one good way to split data and pass it to **dataloader** is to use **torchvision.datasets.ImageFolder** . this function get images with their label without using any label file . **Imagefolder** assumes that all files are saved in folders ,Pictures of the same category are stored in each folder and The folder name is class name .
we also use **torch.utils.data.ConcatDataset** for concatenating the new training dataset created by **transforms.RandomHorizontalFlip(p=1)** with original training dataset

```
train_data_aug1 = torchvision.datasets.ImageFolder(root=config['train_data_path'], transform=
    train_transform1)
all_datasets.append(train_data_aug1)
train_data_aug2 = torchvision.datasets.ImageFolder(root=config['train_data_path'], transform=
    train_transform2)
all_datasets.append(train_data_aug2)

train_data_total = torch.utils.data.ConcatDataset(all_datasets)
```

Listing 3: splitting data and augmentation

alexnet architecture



(a) frame450

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

(b) frame270

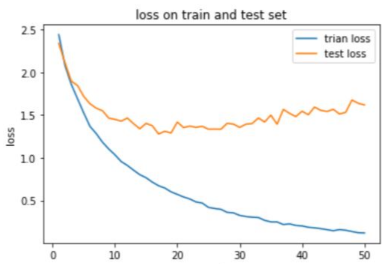
Figure 1

part1

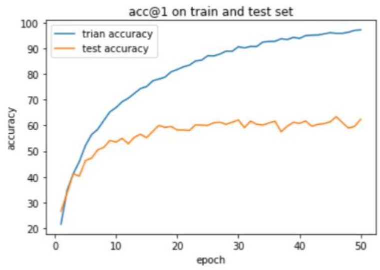
using first convolutional layer and two last fully connected layer of Alexnet architecture . note that we use **max-pooling** layer with **kernel-size=4** and **stride=4** in first layer .
we also add new layer **LocalResponseNorm(2)** **.this operation helps us to reduce input image noises**
below you can see the structure of network :

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=15):
        super(AlexNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=0),
            # nn.BatchNorm2d(96),
            nn.LocalResponseNorm(2), #adding new additional layer
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 4, stride = 4))
        self.fc1 = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(16224, 4096),
            nn.ReLU())
        self.fc2= nn.Sequential(
            nn.Linear(4096, num_classes))

    def forward(self, x):
        out = self.layer1(x)
        out = out.reshape(out.size(0), -1)
        out = self.fc1(out)
        out = self.fc2(out)
        return out
```

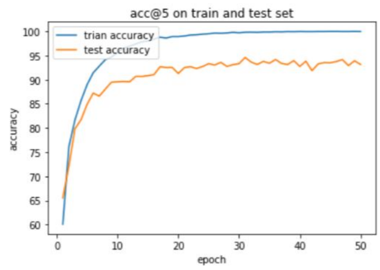
Listing 4: structure of part1



(a) loss



(b) acc@1



(c) acc@5

Figure 2

Results :
maximum acc@1 on test set converges to : 63.4
maximum acc@5 on test set converges to : 93
maximum acc@1 on trian set converges to : 100
maximum acc@5 on train set converges to : 97.2

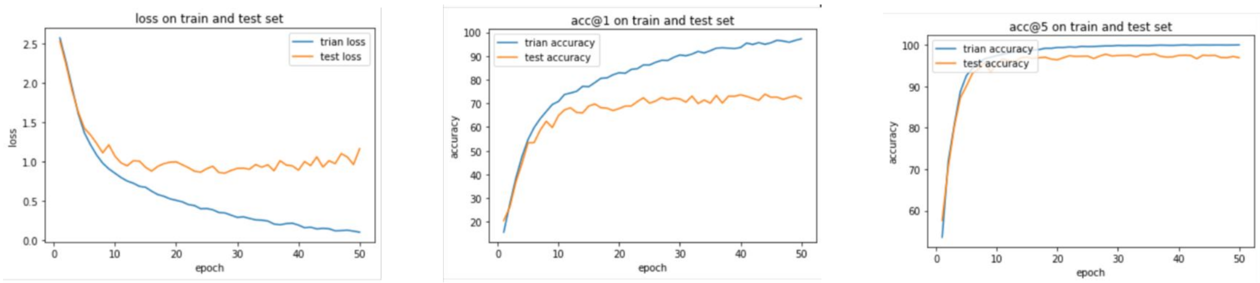
conclusions :
as we can see there is no overfiting in training model . there is a gap between test loss and train loss after some epoches as we expected. also the curve of acc@1-vs-epoch Meet our expectations because there is a gap between test's acc@1 and train's acc@1 over epoches.

part2

using three first convolutions layer and three last fully connected layer of Alexnet architecture . in the **third layer** , we change the conv2d layer with depth of 384 to 256 .
we also add new layer **LocalResponseNorm(2)** .**this operation helps us to reduce input image noises**
below you can see the structure of network :

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=15):
        super(AlexNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=0),
            # nn.BatchNorm2d(96),
            nn.LocalResponseNorm(2), #adding new additional layer
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 3, stride = 2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(96, 256, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 3, stride = 2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.fc = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(9216, 4096),
            nn.ReLU())
        self.fc1 = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU())
        self.fc2= nn.Sequential(
            nn.Linear(4096, num_classes))

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        out = self.fc1(out)
        out = self.fc2(out)
        return out

```

Listing 5: structure of part2



(a) loss (b) acc@1 (c) acc@5

Figure 3

Results :
$maximum\ acc@1\ on\ test\ set\ converges\ to\ :\ 73.91$
$maximum\ acc@5\ on\ test\ set\ converges\ to\ :\ 93.5$
$maximum\ acc@1\ on\ trian\ set\ converges\ to\ :\ 97.27$
$maximum\ acc@5\ on\ train\ set\ converges\ to\ :\ 100$

---

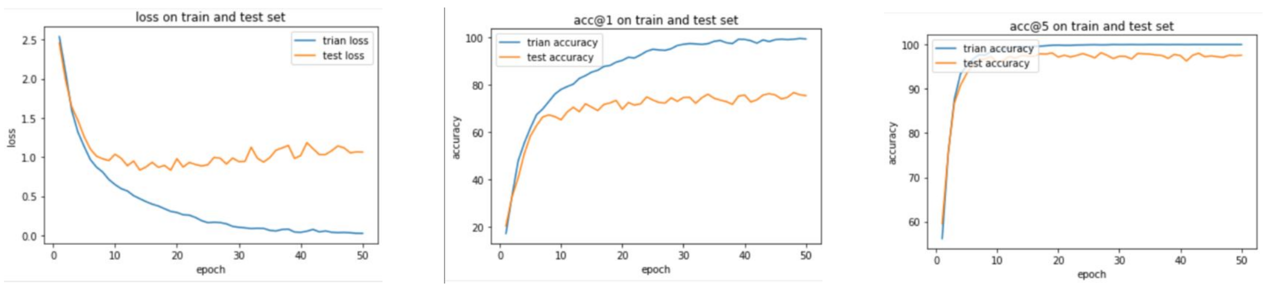
conclusions :
as we can see there is no overfiting in training model , in addition the curve of loss-vs-epoch Meet our expectations because at first , loss of test is bigger than train and over epoches , the train loss becomes less and less but test loss becomes approximately constant after the 20th epoch . there is also a gap between test loss and train loss after some epoches. also the curve of acc@1-vs-epoch Meet our expectations because there is a gap between test's acc@1 and train's acc@1 over epoches.

part3

whole alexnet architecture without using trained weights
below you can see the structure of network :

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=15):
        super(AlexNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=0),
            nn.BatchNorm2d(96),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 3, stride = 2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(96, 256, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 3, stride = 2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(256, 384, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(384),
            nn.ReLU())
        self.layer4 = nn.Sequential(
            nn.Conv2d(384, 384, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(384),
            nn.ReLU())
        self.layer5 = nn.Sequential(
            nn.Conv2d(384, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 3, stride = 2))
        self.fc = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(9216, 4096),
            nn.ReLU())
        self.fc1 = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU())
        self.fc2= nn.Sequential(
            nn.Linear(4096, num_classes))

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.layer5(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        out = self.fc1(out)
        out = self.fc2(out)
        return out

```

Listing 6: structure of part3



(a) loss (b) acc@1 (c) acc@5

Figure 4

Results :
maximum acc@1 on test set converges to : 76.64
maximum acc@5 on test set converges to : 96
maximum acc@1 on trian set converges to : 99.47
maximum acc@5 on train set converges to : 100

---

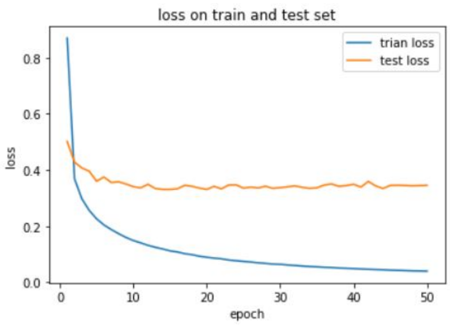
conclusions :
same as previous parts

part4

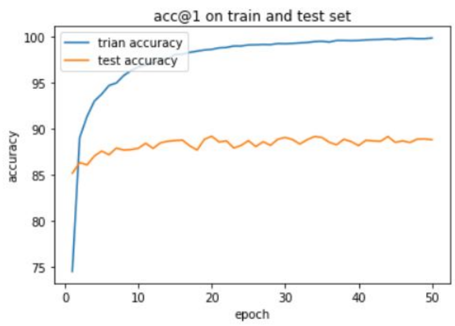
whole alexnet architecture : we squeeze the pre-trained weights ad just learn the new weight of classifier layer :

```

AlexNet_model = torch.hub.load('pytorch/vision:v0.6.0', 'alexnet', pretrained=True)
#---description
AlexNet_model.eval()
#-------------------------------------------------------------

for param in AlexNet_model.parameters():
        param.requires_grad = False  #freeze all layers
for param in AlexNet_model.classifier[6].parameters():
        param.requires_grad = True   #unfreez classifer layer
```

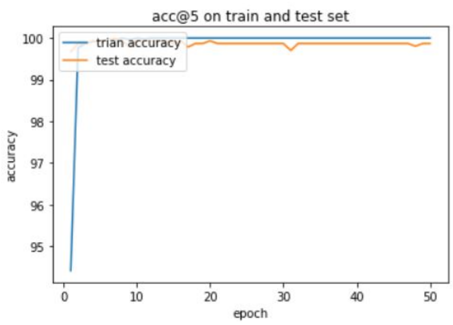
Listing 7: freeze some parameters



(a) loss



(b) acc@1



(c) acc@5

Figure 5

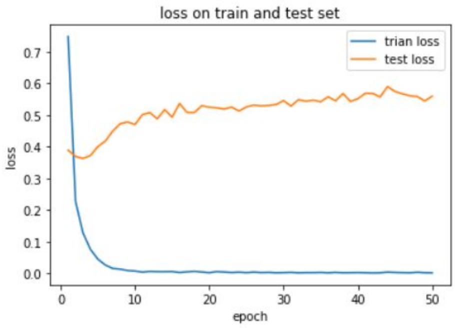
Results :
maximum acc@1 on test set converges to : 89.2
maximum acc@5 on test set converges to : 99.8
maximum acc@1 on trian set converges to : 99.86
maximum acc@5 on train set converges to : 100

---

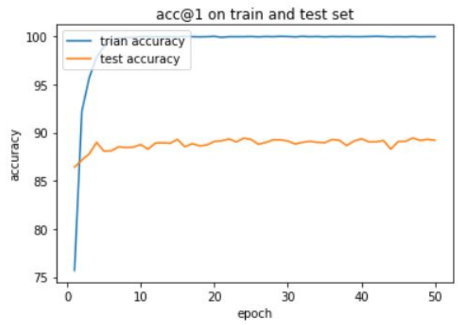
conclusions :
same as previous parts

part5

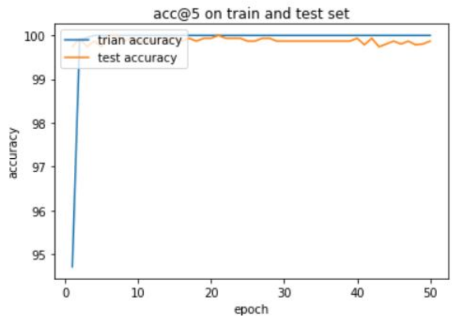
transfer learning : in this part we use trained weights of alexnet model that use for imagenet dataset



(a) loss



(b) acc@1



(c) acc@5

Figure 6

Results :
maximum acc@1 on test set converges to : 89.43
maximum acc@5 on test set converges to : 99.7
maximum acc@1 on trian set converges to : 99.93
maximum acc@5 on train set converges to : 100

---

conclusions :
same as previous parts

summary

results

	optimizer	lr	acc@1	acc@5	augmentation
part1	SGD	5e-4	63.4%	93.0%	2X(horizontal flip)
part2	SGD	5e-4	73.9%	93.5%	2X(horizontal flip)
part3	SGD	5e-4	76.6%	96.0%	2X(horizontal flip)
part4	SGD	5e-4	89.2%	99.8%	2X(horizontal flip)
part5	SGD	5e-4	89.4%	99.7%	2X(horizontal flip)

* number of epoches : 50

* number of training pictures : 5970

Figure 7: summary

This is link of my notebook for this homework: Link.