

SL Project - Scratch

December 14, 2022

```
[1]: import torchvision
import torch
import torch.nn as nn
from torchsummary import summary
```

```
[2]: from torchvision import transforms, datasets
import matplotlib.pyplot as plt
from tqdm import tqdm
```

```
[3]: transform = transforms.Compose([
    transforms.ToTensor()
])
```

```
[4]: PATH = "../..data/asl_alphabet_train/"
```

```
[5]: dataset = datasets.ImageFolder(PATH, transform=transform)
```

```
[6]: n = len(dataset)
```

```
[7]: torch.manual_seed(1)
indices = torch.randperm(n)
```

```
[8]: test_proportion = 0.2 # 20 percent of data used for testing
test_size = int(n * test_proportion)
```

```
[9]: train_dataset = torch.utils.data.Subset(dataset, indices[test_size:])
test_dataset = torch.utils.data.Subset(dataset, indices[:test_size])
```

```
[10]: len(train_dataset)
```

```
[10]: 69600
```

```
[11]: len(test_dataset)
```

```
[11]: 17400
```

```
[12]: train_dataloader = torch.utils.data.DataLoader(dataset=train_dataset,
                                                    batch_size=32,
```

```

shuffle=True,
num_workers=4)

test_dataloader = torch.utils.data.DataLoader(dataset=test_dataset,
                                              batch_size=32,
                                              shuffle=False,
                                              num_workers=4)

```

```
[13]: classes = dataset.classes
```

```
[14]: class ASLNeuralNet(nn.Module):

    def __init__(self, num_classes=1000):
        super(ASLNeuralNet, self).__init__()
        # declare all the layers for feature extraction
        self.features = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=11, stride=4,
↳padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(in_channels=64, out_channels=192, kernel_size=3,
↳padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(in_channels=192, out_channels=256, kernel_size=3,
↳padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),)
        # declare all the layers for classification
        self.classifier = nn.Sequential(
            nn.Linear(6 * 6 * 256, 200),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Linear(200, 500),
            nn.ReLU(inplace=True),
            nn.Linear(500, num_classes),)

    def forward(self, x):
        # apply the feature extractor in the input
        x = self.features(x)
        # squeeze the spatial dimensions in one
        x = x.view(-1, 6 * 6 * 256)
        # classify the images
        x = self.classifier(x)
        return x

```

```
[15]: model = ASLNeuralNet().to('cuda')
```

```
[16]: # sanity check
print(model)

ASLNeuralNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceiling_mode=False)
    (3): Conv2d(64, 192, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceiling_mode=False)
    (6): Conv2d(192, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceiling_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=9216, out_features=200, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=200, out_features=500, bias=True)
    (4): ReLU(inplace=True)
    (5): Linear(in_features=500, out_features=1000, bias=True)
  )
)
```

```
[17]: from torchsummary import summary
summary(model, (3, 200, 200))
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 49, 49]	23,296
ReLU-2	[-1, 64, 49, 49]	0
MaxPool2d-3	[-1, 64, 24, 24]	0
Conv2d-4	[-1, 192, 26, 26]	110,784
ReLU-5	[-1, 192, 26, 26]	0
MaxPool2d-6	[-1, 192, 13, 13]	0
Conv2d-7	[-1, 256, 13, 13]	442,624
ReLU-8	[-1, 256, 13, 13]	0
MaxPool2d-9	[-1, 256, 6, 6]	0
Linear-10	[-1, 200]	1,843,400
ReLU-11	[-1, 200]	0
Dropout-12	[-1, 200]	0
Linear-13	[-1, 500]	100,500
ReLU-14	[-1, 500]	0

Linear-15	[-1, 1000]	501,000
-----------	------------	---------

```

=====
Total params: 3,021,604
Trainable params: 3,021,604
Non-trainable params: 0
-----
Input size (MB): 0.46
Forward/backward pass size (MB): 5.60
Params size (MB): 11.53
Estimated Total Size (MB): 17.59
-----

```

```
[18]: batch_size = 64
```

```
[19]: def train_one_epoch(train_loader, model, device, optimizer, log_interval,
    ↪epoch):
    model.train()
    losses = []
    counter = []

    for i, (img, label) in enumerate(train_loader):
        img, label = img.to(device), label.to(device)
        optimizer.zero_grad()
        output = model(img)
        loss = criterion(output, label)
        loss.backward()
        optimizer.step()
        if (i+1) % log_interval == 0:
            losses.append(loss.item())
            counter.append(
                (i * batch_size) + img.size(0) + epoch * len(train_loader.
    ↪dataset))

    return losses, counter
```

```
[20]: def test_one_epoch(test_loader, model, device):
    model.eval()
    test_loss = 0
    num_correct = 0

    with torch.no_grad():
        for i, (img, label) in enumerate(test_loader):
            img, label = img.to(device), label.to(device)
            output = model(img)
            pred = torch.argmax(output, dim=1)
            num_correct += (pred == label).sum().item()
            test_loss += criterion(output, label).item()
```

```
test_loss /= len(test_loader.dataset)
return test_loss, num_correct
```

```
[21]: criterion = nn.CrossEntropyLoss()
```

```
[22]: no_of_epochs = 10
```

```
[23]: lr = 0.01

log_interval = 100

optimizer = torch.optim.SGD(model.parameters(), lr=lr)

train_losses = []
train_counter = []
test_losses = []
test_correct = []
train_correct = []
for epoch in tqdm(range(no_of_epochs)):
    train_loss, counter = train_one_epoch(train_dataloader, model, "cuda",
    optimizer, log_interval, epoch)
    test_loss, num_correct = test_one_epoch(test_dataloader, model, "cuda")
    _, num_correct_train = test_one_epoch(train_dataloader, model, "cuda")

    # record results
    train_losses.extend(train_loss)
    train_counter.extend(counter)
    test_losses.append(test_loss)
    test_correct.append(num_correct)
    train_correct.append(num_correct_train)
```

```
100%|          | 10/10 [29:18<00:00, 175.81s/it]
```

```
[24]: print(f"test accuracy: {test_correct[-1]/len(test_dataloader.dataset)}")
print(f"train accuracy: {train_correct[-1]/len(train_dataloader.dataset)}")
```

```
test accuracy: 0.9916091954022989
train accuracy: 0.9939511494252874
```

```
[25]: SAVE_PATH = "../..data/googlenet_asl_v0.pth"
```

```
[26]: torch.save(model, SAVE_PATH)
```