Minutes of Meeting

Date: 09-03-2025

Agenda:

1. Understanding PyTorch tensors and basic operations
2. Implementing custom datasets and data loaders
3. Building a CNN model in PyTorch

Discussion Point :

1. Understanding PyTorch tensors and basic operations

```
import torch

arr = [1,2,3,4]

arr1 = [5,4,3,2]

arr

[1, 2, 3, 4]

tens2 = torch.tensor(arr1, dtype = torch.float32, requires_grad =
True)

tens = torch.tensor(arr, dtype = torch.float32, requires_grad = True)

tens.shape

torch.Size([4])

tens @ tens2

tensor(30., grad_fn=<DotBackward0>)

torch.tensor(5)

tensor(5)

for idx, ele in enumerate(tens):
    print(ele, f"time{idx}")

tensor(1., grad_fn=<UnbindBackward0>) time0
tensor(2., grad_fn=<UnbindBackward0>) time1
tensor(3., grad_fn=<UnbindBackward0>) time2
tensor(4., grad_fn=<UnbindBackward0>) time3
```

creating tensor

multiplying tensor

2. Implementing custom datasets and data loaders

```
from torch.utils.data import Dataset, DataLoader

import os
```

```
from PIL import Image
from torchvision import transforms

trans = transforms.Compose([transforms.Resize((226,226)),
    transforms.ToTensor()]
                           )

class CustomDataset(Dataset):
    def __init__(self, base_dir, trans):
        self.dir = base_dir
        self.classes = os.listdir(self.dir)
        self.mapping = {class_name: idx for idx, class_name in
enumerate(classes)}
        self.images = []
        self.labels = []
        self.trans = trans

        for clas in self.classes:
            class_path = f"{self.dir}/{clas}"
            for image_path in os.listdir(class_path):
                self.images.append(f"{class_path}/{image_path}")
                self.labels.append(self.mapping[clas])


    def __len__(self):
        return len(self.images)


    def __getitem__(self, idx):
        image_cur = self.images[idx]
        label_cur = self.labels[idx]
```

```
        image = Image.open(image_cur)
        image = self.trans(image)

        return image , label_cur
```

```
animal_dataset = CustomDataset("/kaggle/input/animals10/raw-img",
trans)

train_loader = DataLoader(animal_dataset, batch_size = 32, shuffle =
True)

bathced_images = next(iter(train_loader))[0]

next(iter(animal_dataset))
```

- Implemented a custom dataset class for an image dataset (CustomDataset).
- Used PIL and torchvision.transforms for image preprocessing.
- Mapped class labels using a dictionary.

3. Building a CNN model in PyTorch

```
img = animal_dataset.__getitem__(50)[0]

img.shape
```

```
torch.Size([3, 226, 226])
```

```
import torch.nn as nn
import torch.nn.functional as F
```

```
class CustomModel(nn.Module):
    def __init__(self,in_chanels, classes = 10):
        super(CustomModel, self).__init__()
        self.in_channels = in_chanels
        self.classes = classes
        self.conv1 = nn.Conv2d(self.in_channels, 8, 5, stride = 2)
        self.conv2 = nn.Conv2d(8, 16, 5, stride = 1)
        self.pool = nn.MaxPool2d(7)
        self.linear = nn.Linear(3600, classes)

        # homework: see the default stirde size for MaxPool2d()

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(x.shape[0], -1)
        return F.softmax(self.linear(x))

model = CustomModel(3, 10)
```

- Created '**CustomModel**' using **torch.nn.Module**.
- Included **Conv2d, MaxPool2d and Linear layers**
- Applied ReLU activation and Softmax