ML HW5

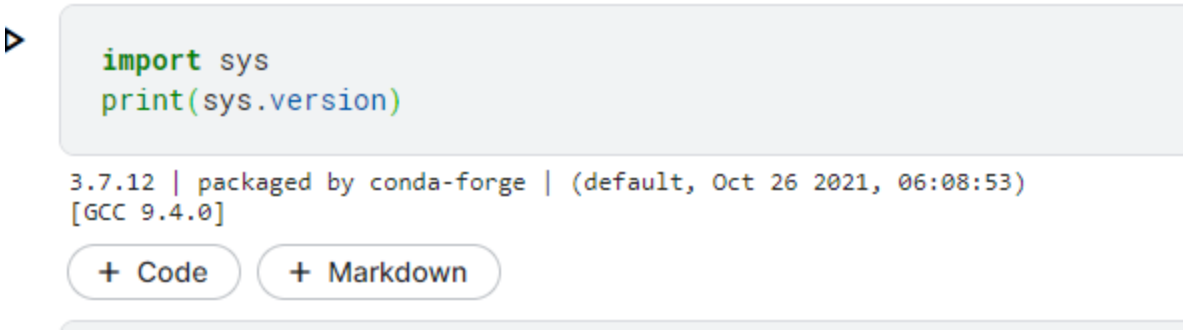
Created	@December 14, 2022 2:11 PM
Tags	

Environment

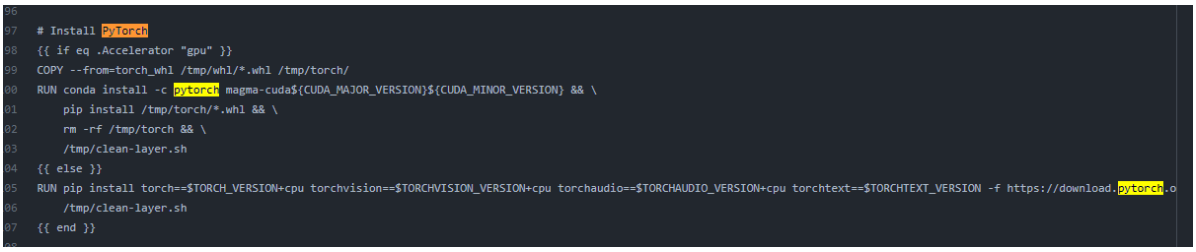
- As the environment kaggle provided, the Docker file is here:

  https://github.com/Kaggle/docker-python/blob/main/Dockerfile.tmpl

- Python Version



- Pytorch (on line 97)



- opencv

```
209       /tmp/clean-layer.sh
210
211   RUN pip install ibis-framework && \
212       pip install gluonnlp && \
213       # b/212703016 4.5.4.62 segfault with readtext.
214       pip install opencv-contrib-python==4.5.4.60 opencv-python==4.5.4.60 && \
215       pip install gluoncv && \
216       /tmp/clean-layer.sh
217
```

- All imported module

```python
# import csv
import os
import random
import string

import cv2
import csv
import numpy as np  # linear algebra
import torch
import torch.nn as nn
import torch.nn.functional as Fun
from torch.utils.data import DataLoader, Dataset
import torchvision.transforms as T
from tqdm import trange
from torchvision import models
import matplotlib.pyplot as plt
```

Weight of my model

- file url :

  https://drive.google.com/drive/folders/1O8zHapzWWEsEnYufsRX6Z-5Cly4mPWeN?usp=sharing

Model Architecture

- The framework I used is Pytorch
- Use the pretrained resnet18 for my model
- modify fully-connected layer’s output feature from 1000 to fit our tasks
- The only difference between models is the output feature

Hyperparameters

- Loss function : nn.CrossEntropyLoss()
- optimizer : torch.optim.Adam
- learning rate : `1e-3` for Task1 and Task2, `2e-3` for Task3

Task1 : 10 classes, 1 output

```
class Mymodel1(nn.Module):
    def __init__(self):
        super(Mymodel1, self).__init__()

        self.model = models.resnet18(pretrained=True)

    # for param in self.model.parameters():
      # param.requires_grad = False

        self.model.fc = nn.Linear(512, 10)
    def forward(self, x):
        logits = self.model(x)
        return logits
model1 = Mymodel1().to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model1.parameters(), lr=1e-3)
```

Task2 : 36 classes, 2 output

```
class Mymodel2(nn.Module):
    def __init__(self):
        super(Mymodel2, self).__init__()

        self.model = models.resnet18(pretrained=True)

    # for param in self.model.parameters():
      # param.requires_grad = False
        num_ftrs = self.model.fc.in_features
        self.model.fc = nn.Identity()
        self.fc1 = nn.Linear(num_ftrs, 36)
        self.fc2 = nn.Linear(num_ftrs, 36)
    def forward(self, x):
        x = self.model(x)
        out1 = self.fc1(x)
        out2 = self.fc2(x)
        return out1, out2
model2 = Mymodel2().to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model2.parameters(), lr=1e-3)
final_acc2 = 0
```

Task3 : 36 classes, 4 output

```
class Mymodel3(nn.Module):
    def __init__(self):
        super(Mymodel3, self).__init__()

        self.model = models.resnet18(pretrained=True)

    # for param in self.model.parameters():
      # param.requires_grad = False

        num_ftrs = self.model.fc.in_features
        self.model.fc = nn.Identity()
        self.fc1 = nn.Linear(num_ftrs, 36)
        self.fc2 = nn.Linear(num_ftrs, 36)
        self.fc3 = nn.Linear(num_ftrs, 36)
        self.fc4 = nn.Linear(num_ftrs, 36)
    def forward(self, x):
        x = self.model(x)
        logits1 = self.fc1(x)
        logits2 = self.fc2(x)
        logits3 = self.fc3(x)
        logits4 = self.fc4(x)
        return logits1, logits2, logits3, logits4
model3 = Mymodel3().to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model3.parameters(), lr=2e-3)
final_acc3 = 0
```