

# NYCU Pattern Recognition, HW5

310551079 梁友誠

## Hardware

- CPU: i7-12700K
- GPU: Nvidia GeForce RTX 3090
- RAM: 128G

## Environment

- OS: Ubuntu 22.04 LTS
- Anaconda: 4.10.3
- python: 3.9.12
- cudatoolKit in conda: 11.3.1
- pytorch: 1.11.0
- torchvision: 0.12.0

P.S. The full requirement for anaconda is described in `package-list.txt`

## How to reproduce the work

First create conda environment and activate it.

```
conda create --name HW5 python=3.9.12
conda activate HW5
```

Install pytorch and and ipykernel.

```
conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
conda install -n HW5 ipykernel --update-deps --force-reinstall
```

Install others packages.

```
pip install -r requirements.txt
```

## Implementation Details:

- Data augmentation:

I use torchvision.transforms module to augment images in Cifar-10 dataset. This operation includes image resize and image normalization as below. The mean and standard deviation parameters of normalization are calculated with provided Cifar-10 dataset.

```
# data augmentation with resize and normalization (project data to [-1, 1])
mean, std = get_mean_and_std(x_train)
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])
```

mean	ndarray	(3.)	[0.49156518 0.48238982 0.4469944 ]
std	ndarray	(3.)	[0.24687816 0.24333645 0.26169549]

- Model atchitecture:

I use pretrained model “[DenseNet121](#)” from torchvision and substitute the final fully connected layer (classifier) with a new fully connected layer. The new fully connected layer contains a linear layer with 256 neurons and Relu() activation function, and then use dropout to reduce overfitting. Finally, use a linear layer with 10 neurons to predict the image class.

```
# load a pretrained model (densenet121)
model = models.densenet121(pretrained=True)

# reset final fully connected layer (num_ftrs = 1024)
num_ftrs = model.classifier.in_features

model.classifier = nn.Sequential(
    nn.Linear(num_ftrs, 256),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(256, 10))

# copy weights for futher retraining on full train dataset
model_wts = copy.deepcopy(model.state_dict())
```

- Loss function and optimizer:

Use CrossEntropyLoss() to computes the cross entropy loss between input and target. It is useful for classification problems.

```
# loss function
criterion = nn.CrossEntropyLoss()

# all parameters are being optimized
optimizer = optim.SGD(model.parameters(), lr=lr)
```

- Hyperparameters:

```
num_epochs = 5
batch_size = 64
lr = 0.005
```

- Training result:

```
model = train(model, criterion, optimizer, num_epochs, train_dataloader)
✓ 19m 3.7s

Epoch [1/5]: 100%|██████████| 782/782 [03:58<00:00, 3.28it/s, acc=76.35%, loss=0.86]
Train accuracy: 76.35, loss: 0.86
=====
Epoch [2/5]: 100%|██████████| 782/782 [03:42<00:00, 3.51it/s, acc=93.07%, loss=0.22]
Train accuracy: 93.07, loss: 0.22
=====
Epoch [3/5]: 100%|██████████| 782/782 [03:49<00:00, 3.40it/s, acc=95.92%, loss=0.131]
Train accuracy: 95.92, loss: 0.13
=====
Epoch [4/5]: 100%|██████████| 782/782 [03:49<00:00, 3.41it/s, acc=97.85%, loss=0.077]
Train accuracy: 97.85, loss: 0.08
=====
Epoch [5/5]: 100%|██████████| 782/782 [03:43<00:00, 3.51it/s, acc=99.03%, loss=0.0429]
Train accuracy: 99.03, loss: 0.04
=====
Best train acc: 99.030000
```

- Model result on test dataset:

```
y_pred = model_predict(model, test_dataloader)
✓ 18.7s

assert y_pred.shape == (10000,)
✓ 0.2s

y_test = np.load("y_test.npy")
print("Accuracy of my model on test set: ", accuracy_score(y_test, y_pred))
✓ 0.3s

Accuracy of my model on test set: 0.9562
```

- Reference:  
[DenseNet \(CVPR 2017\)](#)  
[Finetuning torchvision models](#)  
[Kaggle source code](#)