Practicing Machine Vision

2020-04-16

Neuro-Machine Augmented Intelligence Lab

Overview

In this lab, we will review what we have learned in the previous labs and take a look at the next assignment in which you have to create your own image classifier!

What you'll learn

- ✓ Overview of current machine vision field technology
- ✓ (Review of Lab 1&2) Pipeline of model training procedure for image classification
- ✓ Introduce the assignment

Introduction

- Various signal processing technologies are now widespread.
- Especially, robotics is one of the most important fields right now

Introduction

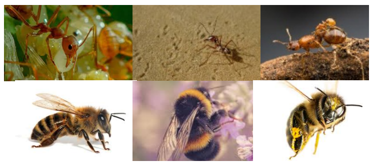
- When you design a robot system, you can choose various sensors for your purpose
- Out of these sensors, the camera is one of the more critical sensor platforms you can use to build your robot system.

KAIST NMAIL Neuro-Machine Augmented Intelligence Lab

Image classification Pipeline

Training

Training data



Forward propagation → Loss gradient calculation → Weights update

Repeat many times

Testing

New data

Trained Model

Output prediction

95% bee

5% ant

Pipeline Code

Main function (transforms)

```
train_transform = transforms.Compose([
        transforms.RandomResizedCrop(input_dim),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

test_transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(input_dim),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
```

RandomResizedCrop = randomly crop picture to the input size
RandomHorizontalFlip = Flip some pictures randomly
ToTensor = transform pixel data to Pytorch type data (named tensor)
Normalize = Normalization of dataset
CenterCrop = Crop the center of picture data
Resize = Change size of the image to the input size

Pipeline Code

Main function (data loading)

```
train_dataset = datasets.ImageFolder(root_dir + '/train', train_transform)
test_dataset  = datasets.ImageFolder(root_dir + '/test' , test_transform)


train_loader = torch.utils.data.DataLoader(train_dataset, batch_size = batch_size,
               shuffle = True , num_workers = num_workers)
test_loader  = torch.utils.data.DataLoader(test_dataset , batch_size = batch_size,
               shuffle = False, num_workers = num_workers)
```

ImageFolder = Given a path to a dataset folder, create a dataset of all the images within the folder (https://pytorch.org/docs/stable/torchvision/datasets.html)
DataLoader = generate processed dataset and set parameters for train and test

Pipeline Code

Main function (model part)

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = Custom_Network().to(device)
optimizer = optim.Adam(model.parameters(), lr = 0.001)
scheduler = StepLR(optimizer, step_size = 1, gamma = 0.8)
```

Device = Using GPU if available, CPU otherwise
Optimizer/scheduler = Setting what optimizer the model uses for loss gradient calculation and other parameters that affect the training procedure
Model = A custom model

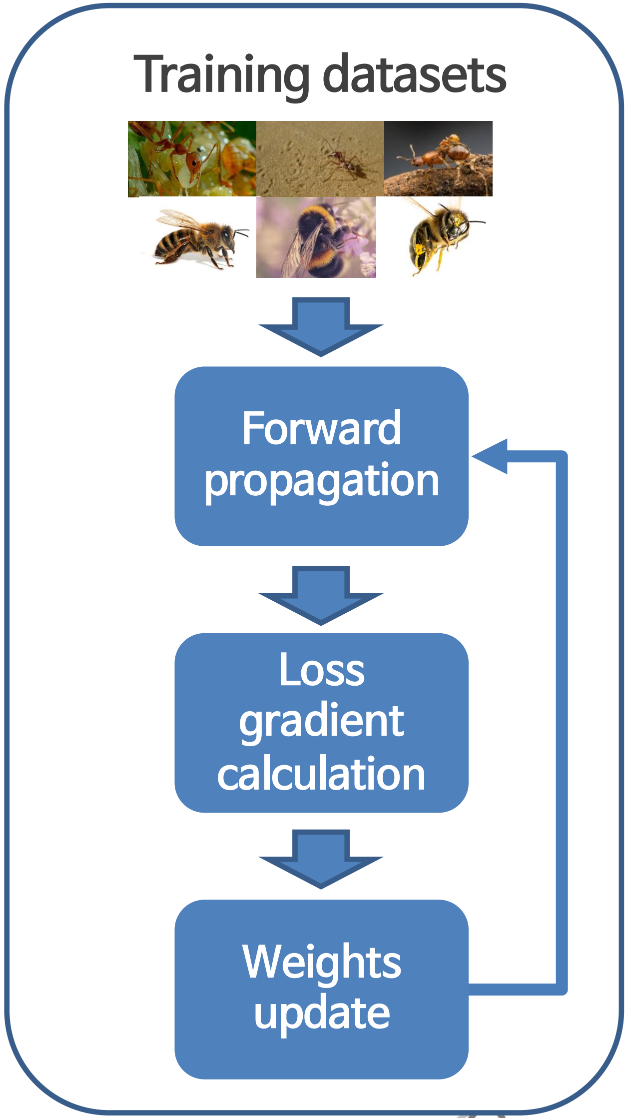
Pipeline Code

Train function

```python
def train(model, device, train_loader, criterion, optimizer, epoch):
    model.train()
    processed = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        processed += len(data)
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % 10 ==0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, processed, len(train_loader.dataset),
                100. * (batch_idx + 1) / len(train_loader), loss.item()))
```

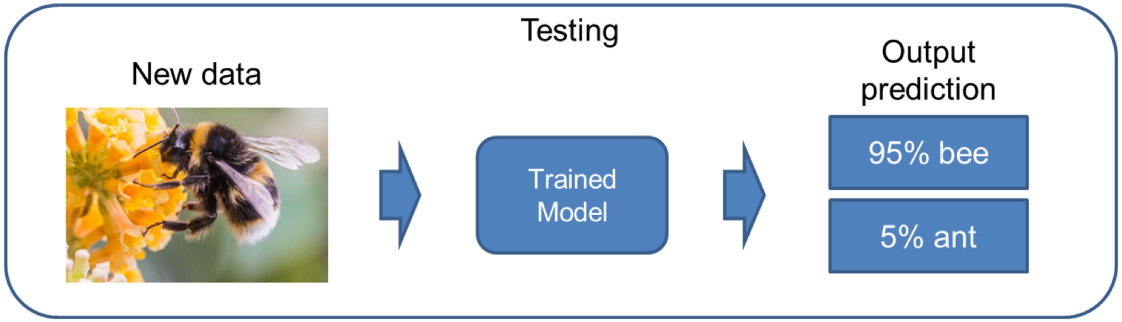


Pipeline Code

Test function

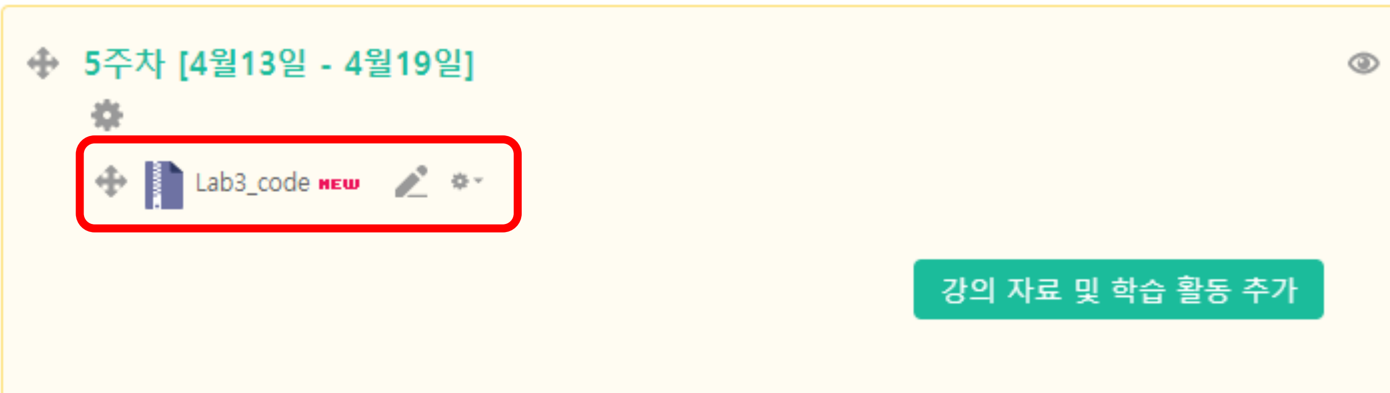
```python
def test(model, device, test_loader, criterion):
    model.eval()
    test_loss = 0
    correct = 0
    # no_grad() prevents codes from tracking records or using memories
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)

            output = model(data)
            test_loss += criterion(output, target).item()
            pred = output.argmax(dim = 1, keepdim = True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```



Download skeleton file from KLMS

- Access KLMS and download the skeleton file named ‘Lab3_code’



File structure

- Lab3_code
  - logs — Folder of logs
  - models — Folder for saved models
  - torch_template.py — Main training/testing python file
  - model.py — Custom model python file

Pipeline Code

model.py

```python
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F

class Custom_Network(nn.Module):
    def __init__(self):
        #super() function makes class inheritance more manageable and extensible
        super(Custom_Network, self).__init__()
        pass

    def forward(self, x):
        pass
```

In model.py file, you can build your own neural network model

Pipeline Code

Save model function

```
def save_models(model):
    print()
    torch.save(model.state_dict(), "models/trained.model")
    print("****----Checkpoint Saved----****")
    print()
```

Execute code

- When you finish training, you should see a message about the model being saved

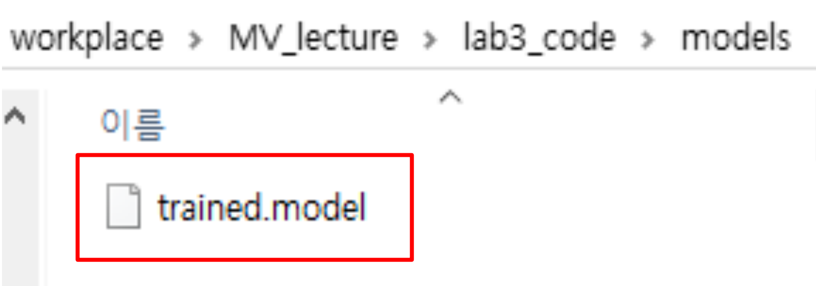
```
Train Epoch: 9 [10/244 (4%)]    Loss: 0.043294
Train Epoch: 9 [110/244 (44%)]  Loss: 0.088104
Train Epoch: 9 [210/244 (84%)]  Loss: 0.008796

Test set: Average loss: 0.0123, Accuracy: 146/153 (95%)


****----Checkpoint Saved----****


C:₩workplace₩MV_lecture₩lab3_code>
```

- You can confirm the new model file in the models folder



Assignment 1

- You will make our own image classifier for this assignment

- Assignment goals
  - ✓ Choose two classes that you want to classify
  - ✓ Collect images for each class to create a training and testing dataset
    - For the training dataset, you can either collect images you find online or take pictures by yourself.
    - For the testing dataset, you must use photos that you took yourself.
      - An image you use in the training dataset should not be used in the testing dataset!
    - Other details are on the Assignment PDF
  - ✓ Preprocess your images
  - ✓ Design your own image classifier model
  - ✓ Train the model and then test it to get an accuracy

Assignment 1

- Submission Guide
  - Submission list
    ① model.py
    ② torch_template.py
    ③ Test dataset folder
    ④ Log folder
    ⑤ Trained model file
    ⑥ Report (under 1 page)
  - Compress it into a .zip file with the name "studentnumber_studentname.zip"

Assignment 1

- Submission Guide
  - In your report, you should write about
    ① How you made each dataset and how many images each dataset has
    ② The type of preprocessing you conducted and why
    ③ The structure of your model and why you chose that structure
    ④ Your final accuracy and why you think you got your accuracy