02. Selenium

CODE

1) Crawling

```
def main():
    crawling()
    saveCSV()

### 1. Crawling
def crawling():
    file_path = './2022.12/12.09_d48_image/data/selenium'
    chromedriver_path = './2022.12/chromedriver.exe'
    os.makedirs(file_path, exist_ok=True)

    N = 200  # 데이터 수집 개수
    SEARCH_LIST = ['T-Shirt', "Trouser", "Dress", "Bag", "Sandal"]
    driver = webdriver.Chrome(chromedriver_path)
    driver.implicitly_wait(10)

    for search in SEARCH_LIST:
        img_dir = os.path.join(file_path, search)
        os.makedirs(img_dir, exist_ok=True)
        driver.get('https://www.google.com')
        elem = driver.find_element(By.NAME, 'q')
        elem.clear()
        elem.send_keys(search)          # 검색
        elem.send_keys(Keys.RETURN)     # 엔터
        assert "No results found." not in driver.page_source

        # 이미지 메뉴 누르기
        driver.find_element(By.XPATH, '//*[@id="hdtb-
msb"]/div[1]/div/div[2]/a').click()
        selenium_scroll_option(driver)
        img_srcs = driver.find_elements(By.CLASS_NAME, 'rg_i')

        url_list = []
        last = 0
        for idx, img_src in enumerate(img_srcs):
            base64_image = img_src.get_attribute('src')
            try:
                if base64_image:  # 64일 때
                    if 'base64' in base64_image:
                        img =
Image.open(BytesIO(base64.b64decode(base64_image.split(',')[-1])))  # decode
후 저장
                        img.save(os.path.join(img_dir, search+str(idx)+'.png'))
```

```
                        last = idx + 1
                    else:
                        url_list.append(base64_image)
            except PIL.UnidentifiedImageError:
                print(base64_image)
            except AttributeError:
                print(base64_image)
                continue
            if int(N) == idx:
                break
        [urllib.request.urlretrieve(url, os.path.join(img_dir,
search+str(last+i)+'.png')) for i, url in enumerate(set(url_list))]
    driver.close()
```

2) Resize and Save CSV file

```
#### 2. Image Reszie & Save CSV file
def saveCSV():
    SEARCH_LIST = ['T-Shirt', "Trouser", "Dress", "Bag", "Sandal"]
    TRAIN_PERCENTAGE = 0.8  # training 비율
    IMAGE_SIZE = 28         # 이미지 사이즈

    file_path = './2022.12/12.09_d48_image/data/selenium'
    df_dict_train= {
    'file_name':[],
    'label':[]
    }
    df_dict_test= {
    'file_name':[],
    'label':[]
    }

    os.makedirs(os.path.join(file_path, 'train'), exist_ok=True)
    os.makedirs(os.path.join(file_path, 'test') , exist_ok=True)

    for idx in range(len(SEARCH_LIST)):
        img_dir = os.path.join(file_path, SEARCH_LIST[idx])  # image file path
        for index, item in enumerate(os.listdir(img_dir)):
            image = cv2.imread(os.path.join(img_dir,item),
cv2.IMREAD_GRAYSCALE)    # gray scale
            image =
cv2.resize(image,(IMAGE_SIZE,IMAGE_SIZE))                          # resize
            if int(index) < len(os.listdir(img_dir)) * TRAIN_PERCENTAGE :   #
train 비율 80%
                cv2.imwrite( os.path.join(file_path, 'train', item), image)
                df_dict_train['file_name'].append(item)
                df_dict_train['label'].append(idx)
```

```
            else:                                                       #
test  비율 20%
                cv2.imwrite( os.path.join(file_path, 'test', item), image)
                df_dict_test['file_name'].append(item)
                df_dict_test['label'].append(idx)
        df_train = pd.DataFrame(df_dict_train)
        df_test  = pd.DataFrame(df_dict_test)

    df_train.to_csv(os.path.join(file_path, 'annotation_train.csv'))
    df_test.to_csv (os.path.join(file_path, 'annotation_test.csv'))

if __name__ =='__main__':
    main()
```

3) Train & Test

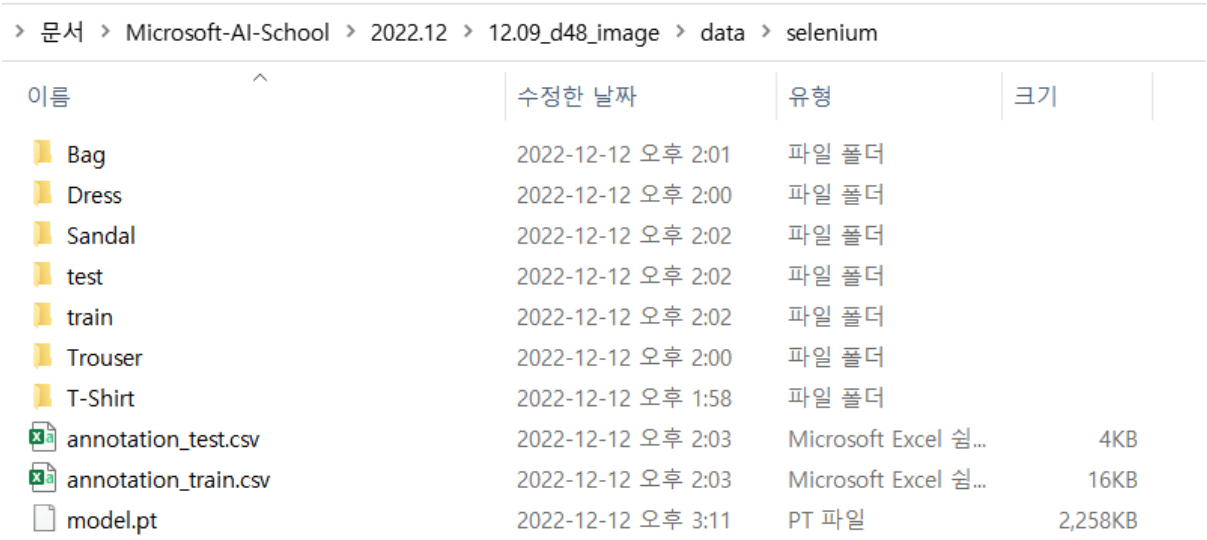
```
file_path = './2022.12/12.09_d48_image/data/selenium'
IMAGE_SIZE = 28

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None,
target_transform=None):
        self.img_labels = pd.read_csv(annotations_file, names=['file_name',
'label'], skiprows=[0])
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

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

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        try:
            image = read_image(img_path)
        except:
            print(self.img_labels.iloc[idx, 0])
            exit()
        label = int(self.img_labels.iloc[idx, 1])
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label

#### Define Neural Netowrk model
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 512)  # input 28x28 = 784
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, 32)
        self.fc6 = nn.Linear(32, 10)    # Output 0~9 = 10 labels

    def forward(self, x):
        x = x.float()
        h1 = F.relu(self.fc1(x.view(-1, 784)))
        h2 = F.relu(self.fc2(h1))
        h3 = F.relu(self.fc3(h2))
        h4 = F.relu(self.fc4(h3))
        h5 = F.relu(self.fc5(h4))
        h6 = self.fc6(h5)
        return F.log_softmax(h6, dim=1)

#### Prepare Data Loader for Training and Validation
transform = transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                        ])

#### vars and deivce 설정
epochs = 10         # 몇번 학습
lr = 0.001          # learning rate
momentum = 0.5      # optimizer 최적화 함수에 들어가는 관성계수
no_cuda = True      # cuda 인지
seed = 1            # random seed
log_interval = 5

use_cuda = not no_cuda and torch.cuda.is_available()
torch.manual_seed(seed)
# cuda 면 cuda 쓰고 아니면 cpu 사용
device = torch.device("cuda" if use_cuda else "cpu")
kwargs = {'num_workers': 1, 'pin_memory': True} if use_cuda else{}
                # 1: 사용 프로세서 1 로설정

batch_size = 16    # 한번 학습할때 몇개
test_batch_size = 16

dataset_train = CustomImageDataset(
    annotations_file= os.path.join(file_path, 'annotation_train.csv'),
    img_dir= file_path + '/train',
    )
```

```
dataset_test = CustomImageDataset(
    annotations_file= os.path.join(file_path, 'annotation_test.csv'),
    img_dir= file_path + '/test',
    )

train_loader = torch.utils.data.DataLoader(dataset_train, batch_size=
batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset_test,
batch_size=test_batch_size, shuffle=True, **kwargs)

model = Net().to(device)  # devce : cpu or gpu ?? 나는 cpu
optimizer = optim.SGD(model.parameters(), lr= lr, momentum= momentum)

def train(log_interval, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

def test(log_interval, model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').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)))

for epoch in range(1, epochs+1):
    print(epoch)
    train(log_interval, model, device, train_loader, optimizer, epoch)
test(log_interval, model, device, test_loader)
torch.save(model, file_path + '/model.pt')  # 가중치
```

RESULT

1) File List



2) Trainset (794개)

	A	B	C	D	E	F	G
		file_name	label				
0		T-Shirt0.pr	0				
1		T-Shirt1.pr	0				
2		T-Shirt10.p	0				
3		T-Shirt100	0				
4		T-Shirt101	0				
5		T-Shirt102	0				

- T-Shirt 160개, Trouser 161개, Dress 160개, Bag 157개, Sandal 156개

3) Testset (197개)

	A	B	C	D	E	F	G
		file_name	label				
0		T-Shirt63.p	0				
1		T-Shirt64.p	0				
2		T-Shirt65.p	0				
3		T-Shirt66.p	0				
4		T-Shirt67.p	0				
5		T-Shirt68.p	0				

- T-Shirt 40개, Trouser 40개, Dress 40개, Bag 39개, Sandal 38개

4) Train & Test

- Train

```
1
Train Epoch: 1 [0/794 (0%)]     Loss: 2.699146
Train Epoch: 1 [80/794 (10%)]   Loss: 2.370903
Train Epoch: 1 [160/794 (20%)]  Loss: 2.479557
Train Epoch: 1 [240/794 (30%)]  Loss: 3.258104
Train Epoch: 1 [320/794 (40%)]  Loss: 4.058762
Train Epoch: 1 [400/794 (50%)]  Loss: 3.149470
Train Epoch: 1 [480/794 (60%)]  Loss: 2.075704
Train Epoch: 1 [560/794 (70%)]  Loss: 2.984457
Train Epoch: 1 [640/794 (80%)]  Loss: 6.260309
Train Epoch: 1 [720/794 (90%)]  Loss: 109.666199
2
Train Epoch: 2 [0/794 (0%)]     Loss: 2.256076
Train Epoch: 2 [80/794 (10%)]   Loss: 2.313601
Train Epoch: 2 [160/794 (20%)]  Loss: 2.243193
Train Epoch: 2 [240/794 (30%)]  Loss: 2.236321
Train Epoch: 2 [320/794 (40%)]  Loss: 2.222329
Train Epoch: 2 [400/794 (50%)]  Loss: 2.194267
Train Epoch: 2 [480/794 (60%)]  Loss: 2.130902
Train Epoch: 2 [560/794 (70%)]  Loss: 2.040957
Train Epoch: 2 [640/794 (80%)]  Loss: 2.049253
Train Epoch: 2 [720/794 (90%)]  Loss: 1.977470
3
Train Epoch: 3 [0/794 (0%)]     Loss: 1.987518
Train Epoch: 3 [80/794 (10%)]   Loss: 2.000035
Train Epoch: 3 [160/794 (20%)]  Loss: 1.864646
Train Epoch: 3 [240/794 (30%)]  Loss: 1.935300
Train Epoch: 3 [320/794 (40%)]  Loss: 1.901992
Train Epoch: 3 [400/794 (50%)]  Loss: 1.803784
Train Epoch: 3 [480/794 (60%)]  Loss: 1.817162
Train Epoch: 3 [560/794 (70%)]  Loss: 1.782257
Train Epoch: 3 [640/794 (80%)]  Loss: 1.775547
Train Epoch: 3 [720/794 (90%)]  Loss: 1.749165
```

```
4
Train Epoch: 4 [0/794 (0%)]     Loss: 1.749975
Train Epoch: 4 [80/794 (10%)]   Loss: 1.695315
Train Epoch: 4 [160/794 (20%)]  Loss: 1.699804
Train Epoch: 4 [240/794 (30%)]  Loss: 1.671278
Train Epoch: 4 [320/794 (40%)]  Loss: 1.652606
Train Epoch: 4 [400/794 (50%)]  Loss: 1.675350
Train Epoch: 4 [480/794 (60%)]  Loss: 1.676652
Train Epoch: 4 [560/794 (70%)]  Loss: 1.648220
Train Epoch: 4 [640/794 (80%)]  Loss: 1.638344
Train Epoch: 4 [720/794 (90%)]  Loss: 1.656271
5
Train Epoch: 5 [0/794 (0%)]     Loss: 1.658601
Train Epoch: 5 [80/794 (10%)]   Loss: 1.622178
Train Epoch: 5 [160/794 (20%)]  Loss: 1.626814
Train Epoch: 5 [240/794 (30%)]  Loss: 1.623847
Train Epoch: 5 [320/794 (40%)]  Loss: 1.633107
Train Epoch: 5 [400/794 (50%)]  Loss: 1.651696
Train Epoch: 5 [480/794 (60%)]  Loss: 1.584943
Train Epoch: 5 [560/794 (70%)]  Loss: 1.615306
Train Epoch: 5 [640/794 (80%)]  Loss: 1.603913
Train Epoch: 5 [720/794 (90%)]  Loss: 1.613519
6
Train Epoch: 6 [0/794 (0%)]     Loss: 1.675621
Train Epoch: 6 [80/794 (10%)]   Loss: 1.660926
Train Epoch: 6 [160/794 (20%)]  Loss: 1.576978
Train Epoch: 6 [240/794 (30%)]  Loss: 1.576090
Train Epoch: 6 [320/794 (40%)]  Loss: 1.585179
Train Epoch: 6 [400/794 (50%)]  Loss: 1.602170
Train Epoch: 6 [480/794 (60%)]  Loss: 1.637123
Train Epoch: 6 [560/794 (70%)]  Loss: 1.605379
Train Epoch: 6 [640/794 (80%)]  Loss: 1.674768
Train Epoch: 6 [720/794 (90%)]  Loss: 1.700359
```

```
7
Train Epoch: 7 [0/794 (0%)]     Loss: 1.647751
Train Epoch: 7 [80/794 (10%)]   Loss: 1.601263
Train Epoch: 7 [160/794 (20%)]  Loss: 1.653223
Train Epoch: 7 [240/794 (30%)]  Loss: 1.644374
Train Epoch: 7 [320/794 (40%)]  Loss: 1.639955
Train Epoch: 7 [400/794 (50%)]  Loss: 1.663580
Train Epoch: 7 [480/794 (60%)]  Loss: 1.583682
Train Epoch: 7 [560/794 (70%)]  Loss: 1.585771
Train Epoch: 7 [640/794 (80%)]  Loss: 1.605609
Train Epoch: 7 [720/794 (90%)]  Loss: 1.619106
8
Train Epoch: 8 [0/794 (0%)]     Loss: 1.602001
Train Epoch: 8 [80/794 (10%)]   Loss: 1.587150
Train Epoch: 8 [160/794 (20%)]  Loss: 1.606869
Train Epoch: 8 [240/794 (30%)]  Loss: 1.617627
Train Epoch: 8 [320/794 (40%)]  Loss: 1.630340
Train Epoch: 8 [400/794 (50%)]  Loss: 1.573789
Train Epoch: 8 [480/794 (60%)]  Loss: 1.597938
Train Epoch: 8 [560/794 (70%)]  Loss: 1.597730
Train Epoch: 8 [640/794 (80%)]  Loss: 1.586143
Train Epoch: 8 [720/794 (90%)]  Loss: 1.648433
9
Train Epoch: 9 [0/794 (0%)]     Loss: 1.649021
Train Epoch: 9 [80/794 (10%)]   Loss: 1.652404
Train Epoch: 9 [160/794 (20%)]  Loss: 1.669951
Train Epoch: 9 [240/794 (30%)]  Loss: 1.662337
Train Epoch: 9 [320/794 (40%)]  Loss: 1.645265
Train Epoch: 9 [400/794 (50%)]  Loss: 1.599191
Train Epoch: 9 [480/794 (60%)]  Loss: 1.636120
Train Epoch: 9 [560/794 (70%)]  Loss: 1.652933
Train Epoch: 9 [640/794 (80%)]  Loss: 1.585807
Train Epoch: 9 [720/794 (90%)]  Loss: 1.546287
```

```
10
Train Epoch: 10 [0/794 (0%)]    Loss: 1.618852
Train Epoch: 10 [80/794 (10%)]  Loss: 1.672293
Train Epoch: 10 [160/794 (20%)] Loss: 1.618284
Train Epoch: 10 [240/794 (30%)] Loss: 1.608937
Train Epoch: 10 [400/794 (50%)] Loss: 1.631983
Train Epoch: 10 [480/794 (60%)] Loss: 1.591506
Train Epoch: 10 [560/794 (70%)] Loss: 1.582264
Train Epoch: 10 [640/794 (80%)] Loss: 1.608809
Train Epoch: 10 [720/794 (90%)] Loss: 1.651821
```

- Test

```
Test set: Average loss: 1.6138, Accuracy: 39/197 (20%)
```

정확도는 20%로 그냥 찍는 정도입니다....