ELEC 5630 Assignment 0

Introduction to Python, Image Classification, and Semantic Segmentation

Professor: TAN, Ping

Due Sep. 28, 2025

1 Introduction of Python

In this project, you will set up a Python environment and run two toy demos: handwritten character recognition and semantic segmentation. Python is a high-level, interpreted, and general-purpose programming language. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its syntax allows programmers to express concepts in fewer lines of code compared to languages like C or Java.

Why Python for Machine Learning? Python's popularity in the field of machine learning is due to several factors: *Simplicity*: Python's syntax is clean and easy to understand, making it an excellent choice for beginners and experts alike. *Community Support*: Python has a large and active community, which means plenty of resources, tutorials, and pre-built libraries are available. *Rich Ecosystem*: Python boasts a multitude of libraries and frameworks such as PyTorch, Tensorflow, and Keras, which are specifically designed for machine learning tasks. *Interdisciplinary Usability*: Python is not just for machine learning; it's also great for data analysis, visualization, and automating tasks, making it a versatile tool in a data scientist's toolkit.

The MNIST Dataset: A Benchmark for Handwritten Digit Classification The Modified National Institute of Standards and Technology (MNIST) database is a classic dataset in the field of machine learning. It consists of 70,000 images of handwritten digits, each labeled with the digit they represent (0 through 9). The dataset is split into 60,000 training images and 10,000 test images. The MNIST dataset is widely used for training and testing image processing systems. It has become a de facto benchmark for handwritten digit classification in the same way that the Iris dataset is a benchmark for multi-class classification or the CIFAR-10 dataset is for image classification with 10 different classes.

Cityscapes: A Benchmark for Semantic Segmentation This dataset targets the semantic understanding of urban street scenes. It includes stereo video sequences collected across 50 cities, with pixel-accurate annotations for 5,000 frames and an additional 20,000 weakly annotated frames. In this assignment, we will use Cityscapes for semantic segmentation and rely on the high-quality pixel-level annotations. We provide the processed data.

In this project, we provide a skeleton code for image classification and semantic segmentation. You should complete the rest by yourself. And you also need to upload a screenshot of the output showing that your Python environment has

been created successfully.

Post questions to Canvas so everybody can share unless the questions are private. Please look at Canvas first if similar questions have been posted.

2 TASK

2.1 Objective

This assignment aims to familiarize you with Python programming and the necessary environment setup for a machine learning project. You should install Python on your computer first and we recommend using Anaconda to manage the Python environment. Then, you will use Python to develop the machine-learning models, like MLP and CNN. We provide the skeleton codes for you and you can finish the rest and run the training code.

2.2 Introduction to Conda

Conda is an open-source package management system and environment management system that allows you to install, run, and update packages and manage their dependencies. It is cross-platform and works on Windows, macOS, and Linux. Conda is particularly popular in the scientific computing community due to its ability to handle complex dependencies and create isolated environments for different projects.

Conda can be installed in two ways: by downloading the Miniconda installer or the Anaconda installer. Miniconda is a minimal installer for Conda, while Anaconda includes a large collection of pre-installed packages. Here we show the install details of Anaconda.

More Efficient Python package management? If you are familiar with Conda, you may find that Conda is heavy and may cause some problems when you have a large number of packages in the environment. You can choose to Install libMamba for a huge quality of life improvement when using Conda.

2.2.1 Common Steps Before Installation

You check the following steps before installing the Anaconda.

Backup Your Data: Before starting the installation process, ensure you have a backup of your important data.

Check System Requirements: Ensure your system meets the minimum requirements for Anaconda. Generally, you need a few GB of free disk space and an internet connection for the download.

2.2.2 Download and Install Anaconda on Different Platforms

For laptops with different platforms, you should choose the corresponding tutorial guidance for installation. (Note: we recommend you click the "Add Anaconda3 to my PATH environment variable" during installation.)

Windows: You can refer to the installation tutorial for installation and environment variable setup on Windows Laptops via https://docs.anaconda.com/anaconda/install/windows/.

MacOS: You can refer to the installation tutorial for installation and environment variable setup on Mac Laptops via https://docs.anaconda.com/anaconda/install/mac-os/.

Linux: You can refer to the installation tutorial for installation and environment variable setup on Linux Laptops via https://docs.anaconda.com/anaconda/install/linux/.

2.2.3 Verify the Installation

Open a new terminal or command prompt. Type "conda –version" and press Enter. You should see the version number if the installation was successful.

2.3 MNIST

After the installation of Python, you will develop a machine-learning model to classify handwritten digits using the MNIST dataset. You are required to download both the training and testing datasets. Upon completion, submit both your code and a detailed report outlining your process and results. You have the flexibility to utilize either TensorFlow or PyTorch for this assignment, and we provide a skeleton code based on PyTorch. The dataset can be accessed at http://yann.lecun.com/exdb/mnist/, and we provide the skeleton code for downloading the dataset automatically.

2.3.1 Install the Required Packages

First, go to the PyTorch official website to find the installation command that matches your system configuration (OS, package manager, Python version, and CUDA version if

applicable). For example, if you are using Python 3.12 on macOS without CUDA, the installation command might look like this: "pip install torch torchvision torchaudio". For Windows or Linux, the command might include additional parameters. Use the command provided by the PyTorch website's installation guide.

Install Numpy For a complete MNIST training setup, you might also need other packages such as NumPy, which can be installed using: "pip install numpy"

Install Matplotlib (Optional) While not required for training, Matplotlib is useful for visualizing the data and results. The installation command might look like this: "pip install matplotlib"

Once you have completed these steps, you should have all the necessary packages to start training a model on the MNIST dataset. If you encounter any errors during installation, make sure your pip is up to date by running "pip install –upgrade pip" and refer to the package's documentation for troubleshooting steps.

2.3.2 Task0: Visualization and Preprocess

The MNIST dataset is a large database of handwritten digits that is commonly used for training various image processing systems. It has a training set of 60,000 examples and a testing set of 10,000 examples. Each image is a 28x28 grayscale image containing a handwritten digit from 0 to 9. This requirement document outlines the specifications for a visualization tool that will allow users to explore and understand the MNIST dataset.

You are required to: (1) Randomly pick some samples and view individual images and corresponding labels from the dataset; (2) Analyze the distribution of digits in the dataset; (3) Generate statistical summaries of the dataset.

2.3.3 Task1: Finish Different Types of Neural Networks

This requirement document outlines the specifications for implementing three different models: Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and LeNet-5 for the classification of the MNIST dataset.

You are required to develop the following vision models on the MNIST dataset: (1)MLP: A simple feedforward neural network with one or more layers of nodes; (2)CNN: A deep learning model that excels at image recognition tasks; (3)LeNet-5: A classic convolutional neural network architecture designed specifically for handwritten digit recognition. For the

MLP and CNN, you are allowed to use ANY hyperparameters, like the number of hidden layers.

2.3.4 Task2: Finish the Loss

Cross-entropy loss, also known as log loss, is a crucial function used in classification problems with two or more classes. It measures the performance of a classification model whose output is a probability value between 0 and 1. You can also choose the L2 loss for supervision.

$$L_{CrossEntropy} = -\sum_{c=1}^{M} x \log(\hat{x}),$$

where x is the ground truth and $\hat{x}$ is the prediction results.

2.3.5 Run Code

Once you have finished the above setup, you can run the script using the following command: "python mnist.py".

2.4 Cityscapes

You will train a U-Net for semantic segmentation on the Cityscapes dataset. We follow the official script of Cityscapes, which uses 19 categories for training. Pixels that are unlabeled or do not belong to the 19 categories will be ignored when computing losses. Since the original dataset does not provide the ground truth label in the testing set, we remove three scenes from the training set and use them as the testing set.

2.4.1 Project Overview

There are six files.

- dataset.py: defines the Cityscapes dataloader. The dataset has been re-split.
- network.py: defines the neural network. We use the U-Net for this task.
- train.py: trains the neural network.
- visualize.py: visualizes the model's predictions.

- test.py: evaluates the model's performance.
- utils.py: includes some functions used to evaluate the network.

2.4.2 Task1: Data Augmentation

The dataloader returns an input image and its ground-truth label. Labels are integer class IDs. Unlabeled pixels use the ignore ID 255. Please complete dataset.py by implementing the data-augmentation functions *random_resize* and *random_crop*.

Skeleton to fill:

```
def random_resize(img: Image.Image, label: Image.Image, scale_range=(0.5,
    2.0)):
    # Randomly resize image

    return img, label

def random_crop(img: Image.Image, label: Image.Image, size: Tuple[int, int
    ]):
    # Random crop to the target size; pad if smaller than desired size
    # The input size is a tuple (width, height)

    return img, label
```

random_resize: sample a scale factor within scale_range and resize both the image and the label by this factor.
random_crop: choose a random location in the input 'img' and crop a patch of size specified by the input 'size' (width, height). If the image is smaller than the requested crop in either dimension, pad the image with 0 and the label with 255 before cropping.

2.4.3 Task2: Training

Training iterates over the dataset: forward pass, loss computation, and parameter updates. Complete train.py by implementing *cross_entropy_loss* and the forward pass.

Loss function to implement:

```
def cross_entropy_loss(
    prediction: torch.Tensor, labels: torch.Tensor, ignore_index=255
):
```

```
    # prediction: [B, C, H, W], float32
    # labels: [B, H, W], int64
    # return the mean cross entropy loss
    pass
```

The loss function takes a 4D tensor [batchsize, channels, height, width] 'prediction' and a 3D tensor [batchsize, height, width] 'labels', and outputs the mean cross-entropy loss. The 'prediction' is the raw output of the neural network. Each pixel in 'prediction' has a C-dimensional vector, and C is the number of categories. 'labels' contains the ground truth class ID for each pixel. Please note that some pixels on the image are unlabeled or do not belong to the 19 categories; the label of these pixels is 'ignore_index', i.e., 255.

The function *cross_entropy_loss* returns the mean cross-entropy loss. It first applies softmax on the input and then computes $L_{CrossEntropy}$. Please use the valid pixels (label≠ignore_index) to compute the mean loss.

Training code to implement:

```
for epoch in range(1, args.epochs + 1):
        model.train()
        running_loss = 0.0
        tic = time.time()
        for i, batch in enumerate(train_loader):
            imgs = batch["image"].to(device)  # [B, C, H, W], float32
            labels = batch["label"].to(device)  # [B, H, W], int64

            optimizer.zero_grad()

            # Task2.2: Feed forward
            ############ fill the code ##########
            loss : torch.Tensor # Please compute the scalar loss

            ######################################

            loss.backward()
            optimizer.step()
```

Run the script:

```
python train.py --data_root PATH/TO/THE/DATASET
```

for example:

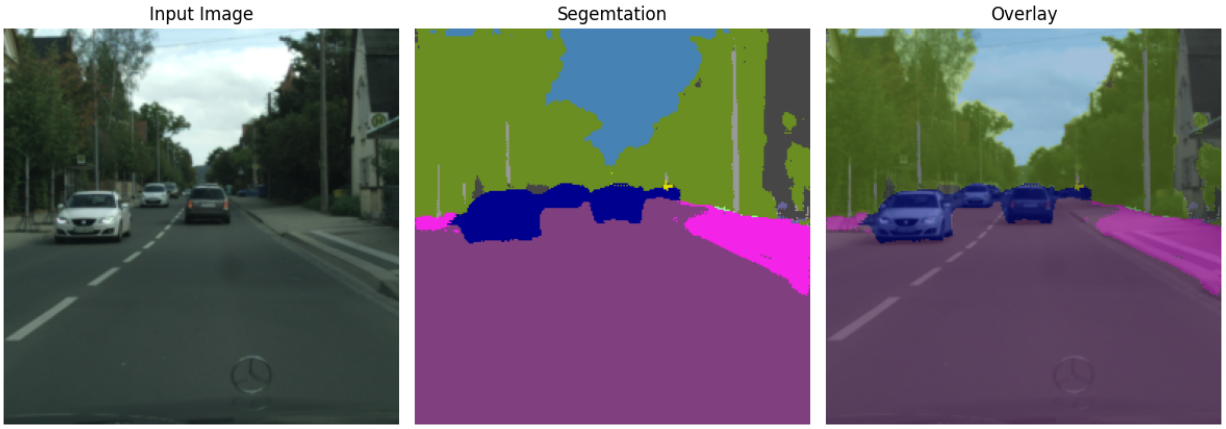
python train.py –data_root ./citycapes

2.4.4 Visualization and Testing

Run the following commands. The scripts will visualize and evaluate the predictions on the testing set. Please report the results in your submission.

```
python visualize.py --data_root PATH/TO/THE/DATASET
python test.py --data_root PATH/TO/THE/DATASET
```

Example of the results:



3 Submission

You only need to upload **ONE** zip file containing two folders. Each folder contains the code, the README file, the results, and the screenshot of your Python program. **Please do not upload the full dataset. You can prepare an empty folder called *"/data"* as a placeholder.** The code should be in Python format. Implementing another language is also allowed in this assignment. You are allowed to use the high-level function provided in Python.

You should write the necessary information in the README file, including the command,

the output, and maybe some analysis, etc. You should make the code in the submission self-contained. The script output should be matched with the file in the results folder.

Grading will only consider the completeness of the implementation, not the complexity of the implementation or the performance of the model. Cite the paper, GitHub repo, or code URL if you use or reference the code online. Please keep academic integrity; plagiarism is not tolerated in this course.

4 Tips

You can know more about the best practices of Python via **Zen of Python**.