BME646 and ECE60146: Homework 4

Spring 2025
Due Date: Tuesday, Feb 11, 2025, 11:59pm
TA: Akshita Kamsali (akamsali@purdue.edu)

Turn in typed solutions via Gradescope. Post questions to Piazza. Additional instructions can be found at the end. Late submissions will be accepted with penalty: -10 points per-late-day, up to 5 days.

1 Introduction

This homework will introduce you to the competition-grade and widely used COCO dataset (MS-COCO: Microsoft Common Objects in Context). COCO is frequently used by researchers to evaluate neural networks in image segmentation, classification, object detection, and more. You will work with a subset of COCO to create your own image classification dataset, which will be used to analyze how the presence and arrangement of objects impact classification performance.

Specifically, you will construct and train CNN models for classification on three different dataset variations:

1. Single-instance dataset: Images containing only one instance of an object.
2. Multi-instance (same object) dataset: Images containing multiple instances of the same object.
3. Multi-instance (different objects) dataset: Images containing multiple objects from different categories.

By training the same CNN model on each dataset, you will assess how classification accuracy is influenced by object quantity, diversity, and whether the object is the primary focus of the image. The dataset creation process and evaluation metrics will be outlined later in this homework.

2 Background

2.1 About the COCO Dataset

The COCO dataset, first published in 2014, remains a key resource in deep learning due to its rich annotations. It has been used to train powerful models like Meta's Segment Anything (SAM) [2] for image segmentation. With its versatility, COCO supports tasks such as classification, object detection, self-supervised learning, and pose estimation.

To understand the motivations behind its inception and to appreciate the challenges faced in constructing the dataset, see the original paper [3] on COCO. You should at least read the Introduction section of the paper.

For this homework, you will download a part of the full COCO dataset and familiarize yourself with the COCO API, which provides a convenient interface to the otherwise complicated annotation files. Finally, you will create your own image dataset for classification using the downloaded COCO files and the COCO API.

2.2 DL Studio

DLStudio's inner class `ExperimentsWithCIFAR` will serve as a sandbox for quickly coming up to speed on this part of the Homework. The network you have to create is likely to be very similar to the two examples — `Net` and `Net2` — shown in that part of DLStudio. To get started, follow these steps:

1. **Install DLStudio**: Download its zip archive from its main documentation page and install the module.

2. **Explore Example Networks**: Experiment with `Net` and `Net2` by modifying convolutional and fully connected layer parameters to observe their impact on classification accuracy.

3. **Run CIFAR-10 Example**: Navigate to the Examples directory and execute the following script:

```
python playing_with_cifar10.py
```

[1]

[1]The CIFAR-10 dataset will be downloaded automatically when you run the script `playing_with_cifar10.py`. The CIFAR image dataset, made available by the University of Toronto, consists of 32×32 images, 50,000 for training, and 10,000 for testing. It can be easily processed on a laptop. For more information, search for "CIFAR-10 dataset."

4. **Understand Network Structure**: The classification network you create will contain multiple convolutional layers followed by one or more fully connected (FC) layers. The final output layer should match the number of image classes (10 in this case).

5. **Analyze Resolution Changes**: Pay attention to how the resolution of the image tensor changes as it moves through the CNN hierarchy.

6. **Consider Kernel Sizes and Padding**: Understand how different convolutional kernel sizes and padding choices affect the network architecture and performance.

3 Programming Tasks

3.1 Using COCO to Create Your Own Image Classification Dataset

Note: Don't be concerned about the initial large size of the complete dataset you download. You will utilize only a subset of the entire dataset, which you will extract following the provided instructions. Make sure to execute these steps accurately, save the subset data, and keep it for future assignments.

Through this exercise, you will create a custom dataset which is a subset of the COCO dataset:

1. The first step is to install the COCO API in your `conda` environment. The Python version of the COCO API — `pycocotools` provides the necessary functionalities for loading the annotation JSON files and accessing images using class names. The `pycocoDemo.ipynb` demo available on the COCO API GitHub repository [1] is a useful resource to familiarize yourself with the COCO API. You can install the `pycocotools` package with the following command [2]:

```
conda install -c conda-forge pycocotools
```

2. Now, you need to download the image files and their annotations. The COCO dataset comes in 2014 and 2017 versions. For this homework, you will be using the **2014 Train images**. You can download them directly from this page:

[2]The following command may change based on your version of conda, please check for the appropriate conda/pip command to install pycocotools

https://cocodataset.org/#download

On the same page, you will also need to download the accompanying annotation files: **2014 Train/Val annotations**. Unzip the two archives you just downloaded.

Do NOT submit any dataset, original or custom.

3.2 Dataset Creation Instructions

Your main task is to use the provided COCO files to create three separate image classification datasets based on object presence and diversity. You will extract images using the COCO API from the `instances_train2014.json` file while ensuring that your datasets meet the following criteria:

1. **Single-instance dataset:** Each image must contain exactly **one instance** of a specified object category.
2. **Multi-instance (same object) dataset:** Each image must contain **multiple instances** of the same object category.
3. **Multi-instance (different objects) dataset:** Each image must contain **multiple objects** from different categories.

For each dataset, ensure the following:

- Each dataset should include at least **400 training** and **100 validation** images for each of **any five object classes of your choice**.

```python
from pycocotools.coco import COCO
import os
import shutil
from PIL import Image

# Set COCO dataset paths
data_dir = "path/to/coco_dataset"
ann_file = os.path.join(data_dir, "annotations/instances_train2017.json")
image_dir = os.path.join(data_dir, "train2017")
output_dir = "output_datasets"

# Load COCO dataset
coco = COCO(ann_file)

# Ensure output directories exist
os.makedirs(output_dir, exist_ok=True)

def save_image(img_info, category_name, dataset_type):
    """Saves the extracted image into a structured output directory."""
    img_path = os.path.join(image_dir, img_info['file_name'])
    save_dir = os.path.join(output_dir, dataset_type, category_name)
    os.makedirs(save_dir, exist_ok=True)

    # Load and resize image
    img = Image.open(img_path).resize((64, 64))
    img.save(os.path.join(save_dir, img_info['file_name']))

def extract_images(cat_names, min_instances=1, max_instances=1, multiple_categories=False, dataset_type="single_instance"):
    """
    Extracts images based on object count conditions.
    - min_instances: Minimum number of object instances required.
    - max_instances: Maximum number of object instances allowed.
    - multiple_categories: If True, selects images with multiple different object types.
    """
    cat_ids = coco.getCatIds(catNms=cat_names)
```

```
    img_ids = coco.getImgIds(catIds=cat_ids)

    extracted = 0  # Counter for extracted images

    for img_id in img_ids:
        img_info = coco.loadImgs(img_id)[0]
        ann_ids = coco.getAnnIds(imgIds=img_id, iscrowd=
                                 False)
        anns = coco.loadAnns(ann_ids)

        # Count objects per category
        obj_counts = {}
        for ann in anns:
            obj_category = coco.loadCats(ann['category_id'
                                 ])[0]['name']
            obj_counts[obj_category] = obj_counts.get(
                                 obj_category, 0) + 1

        if multiple_categories:
            # Ensure multiple object categories are
                                 present
            if len(obj_counts) >= 2:
                save_image(img_info, "multiple_objects",
                                 dataset_type)
                extracted += 1
        else:
            # Ensure the number of instances falls within
                                 the desired range
            if all(obj in obj_counts for obj in cat_names)
                                  and min_instances <=
                                 obj_counts[cat_names[0]] <
                                 = max_instances:
                save_image(img_info, cat_names[0],
                                 dataset_type)
                extracted += 1

        if extracted >= 100:  # Limit to 100 images per
                                 dataset for quick testing
            break


# **Extract Single-instance Dataset**
extract_images(["dog"], min_instances=1, max_instances=1,
                                 dataset_type="
                                 single_instance")

# **Extract Multi-instance (same object) Dataset**
extract_images(["dog"], min_instances=2, dataset_type="
                                 multi_instance_same")
```

```
71
72 # **Extract Multi-instance (different objects) Dataset**
73 extract_images(["dog", "bicycle"], multiple_categories=
                                    True, dataset_type="
                                    multi_instance_different")
74
75 print("Datasets extracted successfully!")
```

- If the selected object classes do not meet this requirement, choose alternative classes that satisfy these criteria across the datasets.
- If no suitable classes are found, apply **data augmentation techniques** from the previous homework to increase the number of images. However, **do not augment more than 100-150 images** per class.
- Extracted images should be sourced exclusively from the **2014 Train images** set.
- Ensure that there are **no overlap images** across train and test datasets. Test dataset must remain unseen during the training process.
- When saving images to disk, resize them to 64×64 pixels using either the `opencv` or `PIL` library.

Dataset Visualization:

1. In your report, include a visualization of your dataset by displaying at least **three images per class** for each dataset variation. To ensure clarity, plot a grid of 5 rows x 3 columns of images for each dataset.
2. Create a 3x5 table where:
   - Rows represent different dataset types.

- Columns represent object classes.
- Each cell contains the count of training images for the corresponding dataset type and class.

3. Create a separate identical table for validation data with the same structure.

3.3 Image Classification using CNNs – Training and Validation

Once you have prepared the datasets, your task is to implement and evaluate a CNN model.

CNN Architecture: The network for this homework, *HW4Net*, consists of convolutional layers followed by fully connected layers. The snippet provided is an example. You are free to modify it.

Network Implementation: Use the following architecture as a starting point:

```
class HW4Net(nn.Module):
    def __init__(self):
        super(HW4Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, 3)

        # fill XXXX and XX

        self.fc1 = nn.Linear(XXXX, 64)
        self.fc2 = nn.Linear(64, XX)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Before you proceed further, identify the number of parameters in each of your network.

Note that in order to train and evaluate your CNNs, you will need to implement your own `torch.utils.data.Dataset` and `DataLoader` classes for loading the images and labels. This is similar to what you have implemented in HW2.

Training Routine: Train and evaluate the model on each dataset using the same optimization setup:

```
net = net.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(
    net.parameters(), lr=1e-3, betas=(0.9, 0.99))
epochs = 7
for epoch in range(epochs):
    running_loss = 0.0
    for i, data in enumerate(train_data_loader):
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if (i+1) % 100 == 0:
            print("[epoch: %d, batch: %5d] loss: %.3f" \
                % (epoch + 1, i + 1, running_loss / 100))
            running_loss = 0.0
```

where the variable `net` is an instance of `HW4Net`.

Analysis and Comparison: Train and evaluate *HW4Net* under three data configurations on a neural network configuration of your choice. Call this the Baseline model. NOTE: Your CNN architecture will remain the same across all datasets and will consistently clas-

sify images into five classes, regardless of dataset variations.

For evaluating the performance of your CNN classifier, you need to write your own code for calculating the confusion matrix.

For the dataset that you created, your confusion matrix will be a 5×5 array of numbers, with both the rows and the columns standing for the 5 classes in the dataset.

The numbers in each row should show how the test samples corresponding to that class were correctly and incorrectly classified. You might find `scikit-learn` and `seaborn` python packages useful for this task.

Report Requirements: Compare results across datasets by plotting:

- Training loss curves for all configurations.
- Confusion matrices for each dataset.
- A table summarizing parameter counts and classification accuracy.

Finally, include your answers to the following questions:

1. By observing your classification accuracies, which dataset think are more difficult to correctly differentiate and why?
2. By observing your confusion matrices, which class or classes do you think are more difficult to correctly differentiate and why?

3. What is one thing that you propose to make the classification performance better?

4 Bonus

Extend your Baseline CNN model by creating two deeper architectures using `nn.Sequential`:

1. HWNet1 with 5 convolutional layers
2. HWNet2 with 10 convolutional layers

Train both models and, similar to the previous section on Single instance dataset, report:

1. Training loss curves for all models
2. Confusion matrices for performance comparison

Analyze whether adding more layers improves classification accuracy. Does a deeper network always result in better performance? Answer with reasoning based on your observations. Avoid a one-word technical explanation — instead, relate your insights to what you learned in HW3 while working with one-neuron and multi-neuron models.

5 Submission Instructions

Include a typed report explaining how you solved the given programming tasks. You may refer to the homework solutions posted at the class website for the previous years for examples of how to structure your report

1. **Turn in a PDF file and mark all pages on gradescope**.
2. Submit your code files(s) as zip file.
3. **Code and Output Placement:** Include the output directly next to the corresponding code block in your submission. Avoid placing the code and output in separate sections as this can make it difficult to follow.
4. **Output Requirement:** Ensure that all your code produces outputs and that these outputs are included in the submitted PDF. Submissions without outputs may not receive full credit, even if the code appears correct.
5. For this homework, you are encouraged to use `.ipynb` for development and the report. If you use `.ipynb`, please convert code to `.py` and submit that as source code. **Do NOT submit .ipynb notebooks.**
6. You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission. **If you are submitting late, do it only once.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.
7. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

References

[1] COCO API - http://cocodataset.org/. URL https://github.com/cocodataset/cocoapi.

[2] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollr, and Ross Girshick. Segment anything, 2023.

[3] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollr. Microsoft coco: Common objects in context, 2015.