Homework 4

Instructions

- This homework focuses on understanding and applying CoCoOp for CLIP prompt tuning. It consists of **four questions** designed to assess both theoretical understanding and practical application.
- Please organize your answers and results for the questions below and submit this jupyter notebook as **a .pdf file**.
- **Deadline: 11/26 (Sat) 23:59**

Preparation

- Run the code below before proceeding with the homework.
- If an error occurs, click 'Run Session Again' and then restart the runtime from the beginning.

↳ 숨겨진 셀 1개

Q1. Understanding and implementing CoCoOp

- We have learned how to define CoOp in Lab Session 4.
- The main difference between CoOp and CoCoOp is **meta network** to extract image tokens that is added to the text prompt.
- Based on the CoOp code given in Lab Session 4, fill-in-the-blank exercise (4 blanks!!) to test your understanding of critical parts of the CoCoOp.

Understanding CoCoOp

CoOp과 다르게 prompt가 dynamic하다. 입력 이미지로부터 정보를 추출해 이를 기반으로 텍스트를 prompting하여 이미지마다 다른 text prompt를 사용한다.

```
import torch.nn as nn

class CoCoOpPromptLearner(nn.Module):
    def __init__(self, cfg, classnames, clip_model):
        super().__init__()
        n_cls = len(classnames)
        n_ctx = cfg.TRAINER.COCOOP.N_CTX
        ctx_init = cfg.TRAINER.COCOOP.CTX_INIT
        dtype = clip_model.dtype
        ctx_dim = clip_model.ln_final.weight.shape[0]
        vis_dim = clip_model.visual.output_dim
        clip_imsize = clip_model.visual.input_resolution
        cfg_imsize = cfg.INPUT.SIZE[0]
        assert cfg_imsize == clip_imsize, f"cfg_imsize ({cfg_imsize}) must equal to clip_imsize ({clip_imsize})"

        if ctx_init:
            # use given words to initialize context vectors
            ctx_init = ctx_init.replace("_", " ")
            n_ctx = len(ctx_init.split(" "))
            prompt = clip.tokenize(ctx_init)
            with torch.no_grad():
                embedding = clip_model.token_embedding(prompt).type(dtype)
            ctx_vectors = embedding[0, 1: 1 + n_ctx, :]
            prompt_prefix = ctx_init
        else:
            # random initialization
            ctx_vectors = torch.empty(n_ctx, ctx_dim, dtype=dtype)
            nn.init.normal_(ctx_vectors, std=0.02)
            prompt_prefix = " ".join(["X"] * n_ctx)

        print(f'Initial context: "{prompt_prefix}"')
        print(f"Number of context words (tokens): {n_ctx}")

        self.ctx = nn.Parameter(ctx_vectors)  # Wrap the initialized prompts above as parameters to make them trainable.

        ### Tokenize ###
        classnames = [name.replace("_", " ") for name in classnames]  # 예) "Forest"
        name_lens = [len(_tokenizer.encode(name)) for name in classnames]
        prompts = [prompt_prefix + " " + name + "." for name in classnames] # 예) "A photo of Forest."

        tokenized_prompts = torch.cat([clip.tokenize(p) for p in prompts]) # 예) [49406, 320, 1125, 539...]
        ####################################
        ####### Q1. Fill in the blank #######
        ########## Define Meta Net ##########
        self.meta_net = nn.Sequential(OrderedDict([
            ("linear1", nn.Linear(vis_dim, vis_dim // 16)),
            ("relu", nn.ReLU(inplace=True)),
            ("linear2", nn.Linear(vis_dim // 16, ctx_dim))
        ]))
        ####################################
        ## Hint: meta network is composed to linear layer, relu activation, and linear layer.



        if cfg.TRAINER.COCOOP.PREC == "fp16":
            self.meta_net.half()

        with torch.no_grad():
            embedding = clip_model.token_embedding(tokenized_prompts).type(dtype)

        # These token vectors will be saved when in save_model(),
        # but they should be ignored in load_model() as we want to use
        # those computed using the current class names
        self.register_buffer("token_prefix", embedding[:, :1, :])  # SOS
        self.register_buffer("token_suffix", embedding[:, 1 + n_ctx:, :])  # CLS, EOS
        self.n_cls = n_cls
        self.n_ctx = n_ctx
        self.tokenized_prompts = tokenized_prompts  # torch.Tensor
        self.name_lens = name_lens

    def construct_prompts(self, ctx, prefix, suffix, label=None):
        # dim0 is either batch_size (during training) or n_cls (during testing)
        # ctx: context tokens, with shape of (dim0, n_ctx, ctx_dim)
        # prefix: the sos token, with shape of (n_cls, 1, ctx_dim)
        # suffix: remaining tokens, with shape of (n_cls, *, ctx_dim)

        if label is not None:
            prefix = prefix[label]
            suffix = suffix[label]

        prompts = torch.cat(
            [
                prefix,  # (dim0, 1, dim)
                ctx,  # (dim0, n_ctx, dim)
                suffix,  # (dim0, *, dim)
            ],
            dim=1,
        )

        return prompts

    def forward(self, im_features):
        prefix = self.token_prefix
        suffix = self.token_suffix
        ctx = self.ctx  # (n_ctx, ctx_dim)


        ###########################################
        ########## Q2,3. Fill in the blank #########
        bias = self.meta_net(im_features)  # (batch, ctx_dim)
        bias = bias.unsqueeze(1)  # (batch, 1, ctx_dim)
        ctx = ctx.unsqueeze(0)  # (1, n_ctx, ctx_dim)
        ctx_shifted = ctx + bias  # (batch, n_ctx, ctx_dim)
        ###########################################
        ###########################################



        # Use instance-conditioned context tokens for all classes
        prompts = []
        for ctx_shifted_i in ctx_shifted:
            ctx_i = ctx_shifted_i.unsqueeze(0).expand(self.n_cls, -1, -1)
            pts_i = self.construct_prompts(ctx_i, prefix, suffix)  # (n_cls, n_tkn, ctx_dim)
            prompts.append(pts_i)
        prompts = torch.stack(prompts)

        return prompts


class CoCoOpCustomCLIP(nn.Module):
    def __init__(self, cfg, classnames, clip_model):
        super().__init__()
        self.prompt_learner = CoCoOpPromptLearner(cfg, classnames, clip_model)
        self.tokenized_prompts = self.prompt_learner.tokenized_prompts
        self.image_encoder = clip_model.visual
        self.text_encoder = TextEncoder(clip_model)
        self.logit_scale = clip_model.logit_scale
        self.dtype = clip_model.dtype

    def forward(self, image, label=None):
        tokenized_prompts = self.tokenized_prompts
        logit_scale = self.logit_scale.exp()

        image_features = self.image_encoder(image.type(self.dtype))
        image_features = image_features / image_features.norm(dim=-1, keepdim=True)


        ############################################
        ########## Q4. Fill in the blank #########
        prompts = self.prompt_learner(image_features)
        ############################################
        ############################################


        logits = []
        for pts_i, imf_i in zip(prompts, image_features):
            text_features = self.text_encoder(pts_i, tokenized_prompts)
            text_features = text_features / text_features.norm(dim=-1, keepdim=True)
            l_i = logit_scale * imf_i @ text_features.t()
            logits.append(l_i)
        logits = torch.stack(logits)

        if self.prompt_learner.training:
            return F.cross_entropy(logits, label)

        return logits
```

Q2. Trainining CoCoOp

In this task, you will train CoCoOp on the EuroSAT dataset. If your implementation of CoCoOp in Question 1 is correct, the following code should execute without errors. Please submit the execution file so we can evaluate whether your code runs without any issues.

```
# Train on the Base Classes Train split and evaluate accuracy on the Base Classes Test split.
args.trainer = "CoCoOp"
args.train_batch_size = 4
args.epoch = 100
args.output_dir = "outputs/cocoop"

args.subsample_classes = "base"
args.eval_only = False
cocoop_base_acc = main(args)
```

```
epoch [66/100] batch [20/20] time 0.108 (0.130) data 0.000 (0.022) loss 0.1279 (0.1686) lr 6.8251e-04 eta 0:01:28
epoch [67/100] batch [20/20] time 0.096 (0.138) data 0.000 (0.033) loss 0.0054 (0.2219) lr 6.4781e-04 eta 0:01:30
epoch [68/100] batch [20/20] time 0.107 (0.129) data 0.000 (0.022) loss 0.2773 (0.2684) lr 6.1370e-04 eta 0:01:22
epoch [69/100] batch [20/20] time 0.156 (0.153) data 0.000 (0.018) loss 0.0228 (0.2471) lr 5.8022e-04 eta 0:01:34
epoch [70/100] batch [20/20] time 0.161 (0.200) data 0.000 (0.041) loss 0.2318 (0.1503) lr 5.4740e-04 eta 0:01:59
epoch [71/100] batch [20/20] time 0.099 (0.128) data 0.000 (0.018) loss 0.0285 (0.1188) lr 5.1527e-04 eta 0:01:14
epoch [72/100] batch [20/20] time 0.096 (0.129) data 0.000 (0.017) loss 0.1163 (0.2144) lr 4.8387e-04 eta 0:01:12
epoch [73/100] batch [20/20] time 0.098 (0.129) data 0.000 (0.017) loss 0.0424 (0.1745) lr 4.5322e-04 eta 0:01:09
epoch [74/100] batch [20/20] time 0.163 (0.148) data 0.000 (0.018) loss 0.1774 (0.1305) lr 4.2336e-04 eta 0:01:17
epoch [75/100] batch [20/20] time 0.144 (0.204) data 0.000 (0.037) loss 0.0523 (0.1880) lr 3.9432e-04 eta 0:01:41
epoch [76/100] batch [20/20] time 0.097 (0.139) data 0.000 (0.023) loss 0.0109 (0.1781) lr 3.6612e-04 eta 0:01:06
epoch [77/100] batch [20/20] time 0.101 (0.128) data 0.000 (0.027) loss 0.0092 (0.1832) lr 3.3879e-04 eta 0:00:58
epoch [78/100] batch [20/20] time 0.098 (0.129) data 0.000 (0.018) loss 0.1420 (0.2149) lr 3.1236e-04 eta 0:00:56
epoch [79/100] batch [20/20] time 0.150 (0.155) data 0.000 (0.022) loss 0.6455 (0.2502) lr 2.8686e-04 eta 0:01:05
epoch [80/100] batch [20/20] time 0.160 (0.204) data 0.000 (0.039) loss 0.1262 (0.1671) lr 2.6231e-04 eta 0:01:21
epoch [81/100] batch [20/20] time 0.108 (0.134) data 0.000 (0.025) loss 0.1049 (0.1736) lr 2.3873e-04 eta 0:00:50
epoch [82/100] batch [20/20] time 0.114 (0.135) data 0.000 (0.025) loss 0.5278 (0.1947) lr 2.1615e-04 eta 0:00:48
epoch [83/100] batch [20/20] time 0.121 (0.143) data 0.000 (0.022) loss 0.1053 (0.1895) lr 1.9459e-04 eta 0:00:48
epoch [84/100] batch [20/20] time 0.157 (0.186) data 0.000 (0.021) loss 0.1261 (0.1526) lr 1.7407e-04 eta 0:00:59
epoch [85/100] batch [20/20] time 0.121 (0.158) data 0.000 (0.026) loss 0.0314 (0.1640) lr 1.5462e-04 eta 0:00:47
epoch [86/100] batch [20/20] time 0.101 (0.143) data 0.000 (0.024) loss 0.0459 (0.1491) lr 1.3624e-04 eta 0:00:39
epoch [87/100] batch [20/20] time 0.111 (0.144) data 0.000 (0.020) loss 0.2108 (0.1862) lr 1.1897e-04 eta 0:00:37
epoch [88/100] batch [20/20] time 0.157 (0.187) data 0.000 (0.021) loss 0.1178 (0.2581) lr 1.0281e-04 eta 0:00:44
epoch [89/100] batch [20/20] time 0.109 (0.138) data 0.000 (0.022) loss 0.0460 (0.2158) lr 8.7779e-05 eta 0:00:30
```

```python
# Accuracy on the New Classes.
args.model_dir = "outputs/cocoop"
args.output_dir = "outputs/cocoop/new_classes"
args.subsample_classes = "new"
args.load_epoch = 100
args.eval_only = True
coop_novel_acc = main(args)
```

```
Loading trainer: CoCoOp
Loading dataset: EuroSAT
Reading split from /content/ProMetaR/data/eurosat/split_zhou_EuroSAT.json
Loading preprocessed few-shot data from /content/ProMetaR/data/eurosat/split_fewshot/shot_16-seed_1.pkl
SUBSAMPLE NEW CLASSES!
Building transform_train
+ random resized crop (size=(224, 224), scale=(0.08, 1.0))
+ random flip
+ to torch tensor of range [0, 1]
+ normalization (mean=[0.48145466, 0.4578275, 0.40821073], std=[0.26862954, 0.26130258, 0.27577711])
Building transform_test
+ resize the smaller edge to 224
+ 224x224 center crop
+ to torch tensor of range [0, 1]
+ normalization (mean=[0.48145466, 0.4578275, 0.40821073], std=[0.26862954, 0.26130258, 0.27577711])
---------  -------
Dataset    EuroSAT
# classes  5
# train_x  80
# val      20
# test     3,900
---------  -------
Loading CLIP (backbone: ViT-B/16)
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:617: UserWarning: This DataLoader will create 8 worker processes in total
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr
  warnings.warn(
/content/ProMetaR/dassl/utils/torchtools.py:102: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value)
  checkpoint = torch.load(fpath, map_location=map_location)
Building custom CLIP
Initial context: "a photo of a"
Number of context words (tokens): 4
Turning off gradients in both the image and the text encoder
Parameters to be updated: {'prompt_learner.meta_net.linear1.bias', 'prompt_learner.meta_net.linear2.bias', 'prompt_learner.meta_net.linear2.weig
Loading evaluator: Classification
Loading weights to prompt_learner from "outputs/cocoop/prompt_learner/model.pth.tar-100" (epoch = 100)
Evaluate on the *test* set
100%|██████████| 39/39 [01:00<00:00,  1.56s/it]=> result
* total: 3,900
* correct: 1,687
* accuracy: 43.3%
* error: 56.7%
* macro_f1: 39.0%
```

Q3. Analyzing the results of CoCoOp

Compare the results of CoCoOp with those of CoOp that we trained in Lab Session 4. Discuss possible reasons for the performance differences observed between CoCoOp and CoOp.

Base: 91.40% -> 90.8% Accuracy Novel: 51.46 -> 43.3% Accuracy

CoCoOp에서의 model performance가 CoOp에 비해 조금 떨어진 정확도를 가지는 것을 볼 수 있었다. 예상 가능한 원인은 meta network = 동적 prompting의 한계이다.

CoCoOp에서는 generalization = 새로운 데이터 클래스에 대한 분류를 더 잘하기 위해 동적 prompting을 사용하는데, base class의 특징을 기반으로 prompting을 하는 과정에서 base 데이터셋에 오히려 과적합 될 가능성이 존재할 수 있다. 따라서, Novel 데이터셋에 일반적이지 않은 모델이 될 수 있다. (CoOp에서는 고정된 학습 가능한 prompt를 사용한다.) 또한, base와 novel의 데이터 분포가 크다면, meta network가 novel data에서 적합한 prompt를 생성해 내지 못할 수도 있다.

코딩을 시작하거나 AI로 코드를 생성하세요.