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.

```
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 ({

        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 t

    ### 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 Fore

    tokenized_prompts = torch.cat([clip.tokenize(p) for p in prompts]) # 예) [49406, 320, 1125



    ######################################
    ####### 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)
```

```
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_1
SUBSAMPLE BASE 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, 0.2
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, 0.2
---------  -------
Dataset    EuroSAT
# classes  5
# train_x  80
# val      20
# test     4,200
---------  -------
Loading CLIP (backbone: ViT-B/16)
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:617: UserWarning: Th
  warnings.warn(
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
Loading evaluator: Classification
No checkpoint found, train from scratch
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The ve
  warnings.warn(
epoch [1/100] batch [20/20] time 0.092 (0.261) data 0.000 (0.036) loss 0.2744 (1.1881) lr 2
epoch [2/100] batch [20/20] time 0.142 (0.163) data 0.000 (0.024) loss 0.8384 (0.8970) lr 2
epoch [3/100] batch [20/20] time 0.169 (0.200) data 0.000 (0.039) loss 0.6382 (0.7859) lr 2
epoch [4/100] batch [20/20] time 0.140 (0.131) data 0.000 (0.023) loss 0.5044 (0.7151) lr 2
epoch [5/100] batch [20/20] time 0.094 (0.131) data 0.000 (0.023) loss 0.5703 (0.6317) lr 2
epoch [6/100] batch [20/20] time 0.096 (0.123) data 0.000 (0.022) loss 0.6060 (0.6009) lr 2
epoch [7/100] batch [20/20] time 0.154 (0.143) data 0.000 (0.018) loss 0.3853 (0.6638) lr 2
epoch [8/100] batch [20/20] time 0.143 (0.208) data 0.000 (0.034) loss 1.4082 (0.6633) lr 2
epoch [9/100] batch [20/20] time 0.090 (0.125) data 0.000 (0.023) loss 0.1780 (0.4582) lr 2
epoch [10/100] batch [20/20] time 0.104 (0.125) data 0.000 (0.016) loss 1.2285 (0.5051) lr
epoch [11/100] batch [20/20] time 0.093 (0.125) data 0.000 (0.020) loss 0.2539 (0.5013) lr
epoch [12/100] batch [20/20] time 0.125 (0.139) data 0.000 (0.020) loss 1.1484 (0.4657) lr
epoch [13/100] batch [20/20] time 0.170 (0.193) data 0.000 (0.034) loss 0.8467 (0.5009) lr
epoch [14/100] batch [20/20] time 0.093 (0.134) data 0.000 (0.024) loss 0.5547 (0.4495) lr
epoch [15/100] batch [20/20] time 0.106 (0.127) data 0.000 (0.021) loss 1.0430 (0.5549) lr
epoch [16/100] batch [20/20] time 0.092 (0.124) data 0.000 (0.017) loss 1.3906 (0.4799) lr
epoch [17/100] batch [20/20] time 0.121 (0.141) data 0.000 (0.020) loss 0.0238 (0.3497) lr
epoch [18/100] batch [20/20] time 0.167 (0.198) data 0.000 (0.042) loss 0.1337 (0.2804) lr
epoch [19/100] batch [20/20] time 0.097 (0.127) data 0.000 (0.018) loss 1.0420 (0.3864) lr
epoch [20/100] batch [20/20] time 0.093 (0.123) data 0.000 (0.016) loss 0.3484 (0.4984) lr
epoch [21/100] batch [20/20] time 0.096 (0.127) data 0.000 (0.019) loss 0.8184 (0.3434) lr
epoch [22/100] batch [20/20] time 0.118 (0.140) data 0.000 (0.020) loss 0.2090 (0.4361) lr
epoch [23/100] batch [20/20] time 0.137 (0.193) data 0.000 (0.029) loss 0.0860 (0.3027) lr
```

```
# 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-
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, 0.275
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, 0.275
---------  -------
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
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verb
  warnings.warn(
/content/ProMetaR/dassl/utils/torchtools.py:102: FutureWarning: You are using `torch.load` wi
  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.l
Loading evaluator: Classification
Loading weights to prompt_learner from "outputs/cocoop/prompt_learner/model.pth.tar-100" (epo
Evaluate on the *test* set
100%|██████████| 39/39 [01:02<00:00,  1.59s/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 클래스 정확도

  > coop: 91.4%, cocoop: 90.8%

- new 클래스 정확도

  > coop: 51.5%, cocoop: 43.3%

base와 new클래스 모두에서 coop이 cocoop보다 높은 성능을 보였다. 이는 cocoop이 조건부 context를 사용하기 때문에 더 높은 성능이 기록할 것이라는 기대에 상반된 결과이다. 이러한 결과가 나온 주요한 이유는 coop과 cocoop을 훈련할때 사용한 context 토큰의 갯수 차이에 기인한다고 볼 수 있다. coop은 16개의 토큰을 사용한 반면 cocoop은 4개의 토큰만을 사용했기에 coop이 조금 더 높은 성능을 기록한 것이다.

하지만 cocoop이 4개의 토큰으로만 학습했음에도 base클래스에 대해 coop과 거의 동일한 성능을 낸 것으로 보아 cocoop은 context 토큰의 수가 많지 않아도 동적 조정을 통해 충분한 표현력을 얻었다고 볼 수 있다. coop이 고정된 프롬프트 구조에서 표현력을 높이기 위해 더 많은 context 토큰을 사용하는 것에 비해 cocoop은 적은 context 토큰으로 충분한 context 표현력을 가지는 동시에 학습 시간 단축과 더 높은 메모리 효율성을 달성한다.

discussion

> coop: 동적 조정 없이 고정된 프롬프트로 학습 진행

- 소량의 데이터에서도 안정적
- 추가적인 메타 네트워크가 없어서 학습 속도가 빠르고, 계산 비용이 낮음
- 개별 클래스의 최적화와 동적 조정이 부족해 다양한 이미지 도메인에서는 성능이 떨어짐.

> cocoop : 입력 이미지에 따라 동적으로 조정되는 컨텍스트

- 메타 네트워크를 통해 입력 이미지의 특징을 반영 => 모델의 복잡성이 증가하고 학습 시간이 길어짐
- 데이터셋의 분포가 불균형하거나 충분하지 않으면, 메타 네트워크가 적절히 학습되지 않아 coop보다 성능이 낮아질 수 있음