

✓ Homework 4

✓ 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.

```
!git clone https://github.com/mlvlab/ProMetaR.git
%cd ProMetaR/

!git clone https://github.com/KaiyangZhou/Dassl.pytorch.git
%cd Dassl.pytorch/

# Install dependencies
!pip install -r requirements.txt
!cp -r dassl ../
# Install this library (no need to re-build if the source code is modified)
# !python setup.py develop
%cd ..

!pip install -r requirements.txt

%mkdir outputs
%mkdir data

%cd data
%mkdir eurosat
!wget http://madm.dfki.de/files/sentinel/EuroSAT.zip -O EuroSAT.zip

!unzip -o EuroSAT.zip -d eurosat/
%cd eurosat
!gdown 1Ip7yaCWFi0ea0FUGga0lUdVi_DDQth1o

%cd ../../

import os.path as osp
from collections import OrderedDict
import math
import torch
import torch.nn as nn
from torch.nn import functional as F
from torch.cuda.amp import GradScaler, autocast
from PIL import Image
import torchvision.transforms as transforms
import torch
from clip import clip
from clip.simple_tokenizer import SimpleTokenizer as _Tokenizer
import time
from tqdm import tqdm
import datetime
import argparse
from dassl.utils import setup_logger, set_random_seed, collect_env_info
from dassl.config import get_cfg_default
from dassl.engine import build_trainer
from dassl.engine import TRAINER_REGISTRY, TrainerX
from dassl.metrics import compute_accuracy
from dassl.utils import load_pretrained_weights, load_checkpoint
from dassl.optim import build_optimizer, build_lr_scheduler

# custom
import datasets.oxford_pets
import datasets.oxford_flowers
import datasets.fgvc_aircraft
import datasets.dtd
import datasets.eurosat
import datasets.stanford_cars
import datasets.food101
import datasets.sun397
import datasets.caltech101
import datasets.ucf101
import datasets.imagenet
import datasets.imagenet_sketch
import datasets.imagenetv2
import datasets.imagenet_a
import datasets.imagenet_r

def print_args(args, cfg):
```

```

print("*****")
print("** Arguments **")
print("*****")
optkeys = list(args.__dict__.keys())
optkeys.sort()
for key in optkeys:
    print("{}: {}".format(key, args.__dict__[key]))
print("*****")
print("** Config **")
print("*****")
print(cfg)

def reset_cfg(cfg, args):
    if args.root:
        cfg.DATASET.ROOT = args.root
    if args.output_dir:
        cfg.OUTPUT_DIR = args.output_dir
    if args.seed:
        cfg.SEED = args.seed
    if args.trainer:
        cfg.TRAINER.NAME = args.trainer
    cfg.DATASET.NUM_SHOTS = 16
    cfg.DATASET.SUBSAMPLE_CLASSES = args.subsample_classes
    cfg.DATALOADER.TRAIN_X.BATCH_SIZE = args.train_batch_size
    cfg.OPTIM.MAX_EPOCH = args.epoch

def extend_cfg(cfg):
    """
    Add new config variables.
    """
    from yacs.config import CfgNode as CN
    cfg.TRAINER.COOP = CN()
    cfg.TRAINER.COOP.N_CTX = 16 # number of context vectors
    cfg.TRAINER.COOP.CSC = False # class-specific context
    cfg.TRAINER.COOP.CTX_INIT = "" # initialization words
    cfg.TRAINER.COOP.PREC = "fp16" # fp16, fp32, amp
    cfg.TRAINER.COOP.CLASS_TOKEN_POSITION = "end" # 'middle' or 'end' or 'front'
    cfg.TRAINER.COCOOP = CN()
    cfg.TRAINER.COCOOP.N_CTX = 4 # number of context vectors
    cfg.TRAINER.COCOOP.CTX_INIT = "a photo of a" # initialization words
    cfg.TRAINER.COCOOP.PREC = "fp16" # fp16, fp32, amp
    cfg.TRAINER.PROMETAR = CN()
    cfg.TRAINER.PROMETAR.N_CTX_VISION = 4 # number of context vectors at the vision branch
    cfg.TRAINER.PROMETAR.N_CTX_TEXT = 4 # number of context vectors at the language branch
    cfg.TRAINER.PROMETAR.CTX_INIT = "a photo of a" # initialization words
    cfg.TRAINER.PROMETAR.PREC = "fp16" # fp16, fp32, amp
    cfg.TRAINER.PROMETAR.PROMPT_DEPTH_VISION = 9 # Max 12, minimum 0, for 0 it will be using shallow IVLP prompting (J=1)
    cfg.TRAINER.PROMETAR.PROMPT_DEPTH_TEXT = 9 # Max 12, minimum 0, for 0 it will be using shallow IVLP prompting (J=1)
    cfg.DATASET.SUBSAMPLE_CLASSES = "all" # all, base or new
    cfg.TRAINER.PROMETAR.ADAPT_LR = 0.0005
    cfg.TRAINER.PROMETAR.LR_RATIO = 0.0005
    cfg.TRAINER.PROMETAR.FAST_ADAPTATION = False
    cfg.TRAINER.PROMETAR.MIXUP_ALPHA = 0.5
    cfg.TRAINER.PROMETAR.MIXUP_BETA = 0.5
    cfg.TRAINER.PROMETAR.DIM_RATE=8
    cfg.OPTIM_VNET = CN()
    cfg.OPTIM_VNET.NAME = "adam"
    cfg.OPTIM_VNET.LR = 0.0003
    cfg.OPTIM_VNET.WEIGHT_DECAY = 5e-4
    cfg.OPTIM_VNET.MOMENTUM = 0.9
    cfg.OPTIM_VNET.SGD_DAMPNING = 0
    cfg.OPTIM_VNET.SGD_NESTEROV = False
    cfg.OPTIM_VNET.RMSPROP_ALPHA = 0.99
    cfg.OPTIM_VNET.ADAM_BETA1 = 0.9
    cfg.OPTIM_VNET.ADAM_BETA2 = 0.999
    cfg.OPTIM_VNET.STAGED_LR = False
    cfg.OPTIM_VNET.NEW_LAYERS = ()
    cfg.OPTIM_VNET.BASE_LR_MULT = 0.1
    # Learning rate scheduler
    cfg.OPTIM_VNET.LR_SCHEDULER = "single_step"
    # -1 or 0 means the stepsize is equal to max_epoch
    cfg.OPTIM_VNET.STEPSIZE = (-1, )
    cfg.OPTIM_VNET.GAMMA = 0.1
    cfg.OPTIM_VNET.MAX_EPOCH = 10
    # Set WARMUP_EPOCH larger than 0 to activate warmup training
    cfg.OPTIM_VNET.WARMUP_EPOCH = -1
    # Either linear or constant
    cfg.OPTIM_VNET.WARMUP_TYPE = "linear"
    # Constant learning rate when type=constant
    cfg.OPTIM_VNET.WARMUP_CONS_LR = 1e-5
    # Minimum learning rate when type=linear
    cfg.OPTIM_VNET.WARMUP_MIN_LR = 1e-5
    # Recount epoch for the next scheduler (last_epoch=-1)

```

```

# Otherwise last_epoch=warmup_epoch
cfg.OPTIM_VNET.WARMUP_RECOUNT = True

def setup_cfg(args):
    cfg = get_cfg_default()
    extend_cfg(cfg)
    # 1. From the dataset config file
    if args.dataset_config_file:
        cfg.merge_from_file(args.dataset_config_file)
    # 2. From the method config file
    if args.config_file:
        cfg.merge_from_file(args.config_file)
    # 3. From input arguments
    reset_cfg(cfg, args)
    cfg.freeze()
    return cfg

_tokenizer = _Tokenizer()

def load_clip_to_cpu(cfg): # Load CLIP
    backbone_name = cfg.MODEL.BACKBONE.NAME
    url = clip._MODELS[backbone_name]
    model_path = clip._download(url)

    try:
        # loading JIT archive
        model = torch.jit.load(model_path, map_location="cpu").eval()
        state_dict = None

    except RuntimeError:
        state_dict = torch.load(model_path, map_location="cpu")

    if cfg.TRAINER.NAME == "":
        design_trainer = "CoOp"
    else:
        design_trainer = cfg.TRAINER.NAME
    design_details = {"trainer": design_trainer,
                      "vision_depth": 0,
                      "language_depth": 0, "vision_ctx": 0,
                      "language_ctx": 0}
    model = clip.build_model(state_dict or model.state_dict(), design_details)

    return model

from dassl.config import get_cfg_default
cfg = get_cfg_default()
cfg.MODEL.BACKBONE.NAME = "ViT-B/16" # Set the vision encoder backbone of CLIP to ViT.
clip_model = load_clip_to_cpu(cfg)

class TextEncoder(nn.Module):
    def __init__(self, clip_model): # 초기화 하는 함수
        super().__init__()
        self.transformer = clip_model.transformer
        self.positional_embedding = clip_model.positional_embedding
        self.ln_final = clip_model.ln_final
        self.text_projection = clip_model.text_projection
        self.dtype = clip_model.dtype

    def forward(self, prompts, tokenized_prompts): # 모델 호출
        x = prompts + self.positional_embedding.type(self.dtype)
        x = x.permute(1, 0, 2) # NLD -> LND
        x = self.transformer(x)
        x = x.permute(1, 0, 2) # LND -> NLD
        x = self.ln_final(x).type(self.dtype)

        # x.shape = [batch_size, n_ctx, transformer.width]
        # take features from the eot embedding (eot_token is the highest number in each sequence)
        x = x[torch.arange(x.shape[0]), tokenized_prompts.argmax(dim=-1)] @ self.text_projection

        return x

@TRAINER_REGISTRY.register(force=True)
class CoCoOp(TrainerX):
    def check_cfg(self, cfg):
        assert cfg.TRAINER.COOCOOP.PREC in ["fp16", "fp32", "amp"]

    def build_model(self):
        cfg = self.cfg
        classnames = self.dm.dataset.classnames
        print(f"Loading CLIP (backbone: {cfg.MODEL.BACKBONE.NAME})")

```

```

print('Loading CLIP (backbone: {cfg.MODEL.BACKBONE_NAME}), ,
clip_model = load_clip_to_cpu(cfg)

if cfg.TRAINER.COCOOP.PREC == "fp32" or cfg.TRAINER.COCOOP.PREC == "amp":
    # CLIP's default precision is fp16
    clip_model.float()

print("Building custom CLIP")
self.model = CoCoOpCustomCLIP(cfg, classnames, clip_model)

print("Turning off gradients in both the image and the text encoder")
name_to_update = "prompt_learner"

for name, param in self.model.named_parameters():
    if name_to_update not in name:
        param.requires_grad_(False)

# Double check
enabled = set()
for name, param in self.model.named_parameters():
    if param.requires_grad:
        enabled.add(name)
print(f"Parameters to be updated: {enabled}")

if cfg.MODEL.INIT_WEIGHTS:
    load_pretrained_weights(self.model.prompt_learner, cfg.MODEL.INIT_WEIGHTS)

self.model.to(self.device)
# NOTE: only give prompt_learner to the optimizer
self.optim = build_optimizer(self.model.prompt_learner, cfg.OPTIM)
self.sched = build_lr_scheduler(self.optim, cfg.OPTIM)
self.register_model("prompt_learner", self.model.prompt_learner, self.optim, self.sched)

self.scaler = GradScaler() if cfg.TRAINER.COCOOP.PREC == "amp" else None

# Note that multi-gpu training could be slow because CLIP's size is
# big, which slows down the copy operation in DataParallel
device_count = torch.cuda.device_count()
if device_count > 1:
    print(f"Multiple GPUs detected (n_gpus={device_count}), use all of them!")
    self.model = nn.DataParallel(self.model)

def before_train(self):
    directory = self.cfg.OUTPUT_DIR
    if self.cfg.RESUME:
        directory = self.cfg.RESUME
    self.start_epoch = self.resume_model_if_exist(directory)

    # Remember the starting time (for computing the elapsed time)
    self.time_start = time.time()

def forward_backward(self, batch):
    image, label = self.parse_batch_train(batch)

    model = self.model
    optim = self.optim
    scaler = self.scaler

    prec = self.cfg.TRAINER.COCOOP.PREC
    loss = model(image, label) # Input image 모델 통과
    optim.zero_grad()
    loss.backward() # Backward (역전파)
    optim.step() # 모델 parameter update

    loss_summary = {"loss": loss.item()}

    if (self.batch_idx + 1) == self.num_batches:
        self.update_lr()

    return loss_summary

def parse_batch_train(self, batch):
    input = batch["img"]
    label = batch["label"]
    input = input.to(self.device)
    label = label.to(self.device)
    return input, label

def load_model(self, directory, epoch=None):
    if not directory:
        print("Note that load_model() is skipped as no pretrained model is given")
        return

```

```

names = self.get_model_names()

# By default, the best model is loaded
model_file = "model-best.pth.tar"

if epoch is not None:
    model_file = "model.pth.tar-" + str(epoch)

for name in names:
    model_path = osp.join(directory, name, model_file)

    if not osp.exists(model_path):
        raise FileNotFoundError('Model not found at {}'.format(model_path))

    checkpoint = load_checkpoint(model_path)
    state_dict = checkpoint["state_dict"]
    epoch = checkpoint["epoch"]

    # Ignore fixed token vectors
    if "token_prefix" in state_dict:
        del state_dict["token_prefix"]

    if "token_suffix" in state_dict:
        del state_dict["token_suffix"]

    print("Loading weights to {} " 'from "{}' (epoch = {})'.format(name, model_path, epoch))
    # set strict=False
    self._models[name].load_state_dict(state_dict, strict=False)

def after_train(self):
    print("Finish training")

    do_test = not self.cfg.TEST.NO_TEST
    if do_test:
        if self.cfg.TEST.FINAL_MODEL == "best_val":
            print("Deploy the model with the best val performance")
            self.load_model(self.output_dir)
        else:
            print("Deploy the last-epoch model")
        acc = self.test()

    # Show elapsed time
    elapsed = round(time.time() - self.time_start)
    elapsed = str(datetime.timedelta(seconds=elapsed))
    print(f"Elapsed: {elapsed}")

    # Close writer
    self.close_writer()
    return acc

def train(self):
    """Generic training loops."""
    self.before_train()
    for self.epoch in range(self.start_epoch, self.max_epoch):
        self.before_epoch()
        self.run_epoch()
        self.after_epoch()
    acc = self.after_train()
    return acc

parser = argparse.ArgumentParser()
parser.add_argument("--root", type=str, default="data/", help="path to dataset")
parser.add_argument("--output-dir", type=str, default="outputs/cocoop3", help="output directory")
parser.add_argument(
    "--seed", type=int, default=1, help="only positive value enables a fixed seed"
)
parser.add_argument(
    "--config-file", type=str, default="configs/trainers/ProMetaR/vit_b16_c2_ep10_batch4_4+4ctx.yaml", help="path to config file"
)
parser.add_argument(
    "--dataset-config-file",
    type=str,
    default="configs/datasets/eurosat.yaml",
    help="path to config file for dataset setup",
)
parser.add_argument("--trainer", type=str, default="CoOp", help="name of trainer")
parser.add_argument("--eval-only", action="store_true", help="evaluation only")
parser.add_argument(
    "--model-dir",
    type=str,
    default="",
    help="load model from this directory for eval-only mode",
)

```

```
→ inflating: eurosat/2750/PermanentCrop/PermanentCrop_1398.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_163.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_970.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_502.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2472.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1567.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1915.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2013.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_828.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1106.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1670.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1211.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2304.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_273.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1088.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_612.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1438.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_164.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1059.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_595.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_977.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2475.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1912.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1560.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2014.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1101.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1677.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_19.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1216.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2303.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1753.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1332.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1495.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2227.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_118.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1444.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1836.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2130.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1782.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_579.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1025.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2409.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_853.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_421.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_386.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2068.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_882.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_357.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_65.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_736.jpg
/content/ProMetaR/data/eurosat
Downloading...
From: https://drive.google.com/uc?id=1IpZyaCWFfI0eaOFUGga0lUdVi\_DD0th1o
To: /content/ProMetaR/data/eurosat/split_zhou_EuroSAT.json
100% 3.01M/3.01M [00:00<00:00, 228MB/s]
/content/ProMetaR
100%
```

6/11

왜 CoOp는 문맥 벡터를 X 들로 초기화하고, CoCoOp는 왜 문맥 벡터를 CLIP과 같은 "a photo of a {classname}"으로 하는가?

pretrained CLIP을 사용하므로 CLIP의 텍스트 인코더, 이미지 인코더, 프로젝션 레이어의 파라미터 값들이 학습된 결과를 사용해야 하므로 CLIP의 학습 데이터였던 "a photo of a {classname}" 사용하는 것은 납득이 가지만, CoOp의 경우 완전히 새롭게 문맥 벡터를 학습하겠다는 것인데 이 방법이 효과적인가?

✓ 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
```

```
# 프롬프트를 만들어내는 모듈
```

```
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 # 문맥 벡터: 프롬프트 튜닝에서 학습 대상이 되는 학습 가능한 벡터
        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, :] # SOS 토큰이 있기 때문에 index 1부터를 가져온다. 차원은 (배치 인덱스, 토큰 인덱스, 토큰 임베딩 크기)
            prompt_prefix = ctx_init # 프롬프트의 앞부분은 문맥 벡터로 고정됨(문맥 벡터의 위치 고정)
    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 #####
```

```
# meta_net은 이미지의 특징에 따라 문맥 벡터를 조정할 수 있도록 학습시킬 수 있는 신경망, linear function을 이용해서 이미지-텍스트 벡터의
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": # 정밀도가 16bit Float-Point로 configure되어 있으면
    self.meta_net.half() # 16-bit 부동소수점 연산으로 변환함
```

```
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 - 값이 변경되어서는 안 되므로 register_buffer에 저장
self.register_buffer("token_suffix", embedding[:, 1 + n_ctx:, :]) # CLS, EOS - 마찬가지로 이유로 register_buffer에 저장
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) ctx에다가 bias를 더해 이미지 고유의 프롬프트를 만들어준다. 즉, 이미지마다 프롬프트가 다르다.

    #####
    ##### 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

```

정리

COCOOP은 clip 모델에서 텍스트(프롬프트)를 튜닝할 수 있도록 한 것이다. 기본 CLIP 모델은 항상 텍스트로 A picture of {class}이런 식으로 썼다. 그러나 단순히 모두가 공통된 프롬프트 양식을 갖는 것은 최적의 성능을 뽑아내기에는 한계가 있는 방식이었고 이에 프롬프트를 학습하자는 취지에서 도입된 것이 COCOOP이다. 이 때 기존 CLIP의 A picture of 에 해당하는 부분을 context vector라고 하여 학습 가능한 부분으로 설정하였다. 이미지마다 서로 다른 프롬프트(context vector)를 갖게 되는데, 그 방법은 다음과 같다.

이미지 인코더를 통해 추출된 image feature를 linear layer(activation function은 ReLU)에 통과시켜서 context vector의 길이와 같은 벡터인 bias를 만든다. 그리고 bias와 모두가 공유하는 default context vector("a photo of a {classname}")를 더하여 customized된 context vector를 만드는 것이다. 여기서 bias는 이미지들의 공통된 특징과 개별 이미지 인스턴스의 특징을 모두 반영하고 있다.

질문

1. CLS의 위치가 항상 시퀀스의 마지막에 위치하도록 설계되었는데, CLS의 위치가 고정인 데에는 성능 상의 이점이 있는 것인가?
2. context vector와 bias를 더하는 방법으로는 고정된 context vector만을 생성할 수 있는데 이로 인한 성능 상의 제약은 없는가? 있다면 어떻게 유동적인 길이의 프롬프트 튜닝을 할 수 있는가?


```

class CoCoOpCustomCLIP(nn.Module):
    def __init__(self, cfg, classnames, clip_model):
        super().__init__()
        self.prompt_learner = CoCoOpPromptLearner(cfg, classnames, clip_model) # 위에서 만든 PromptLearner를 받아온다
        self.tokenized_prompts = self.prompt_learner.tokenized_prompts
        self.image_encoder = clip_model.visual
        self.text_encoder = TextEncoder(clip_model) # CLIP 모델의 텍스트 인코더를 extend하여 프롬프트 튜닝을 지원하기 위함
        self.logit_scale = clip_model.logit_scale # logit을 계산할 때 따로 곱해주는 계수, logit이 소프트맥스에 의해 확률로 변환된다는 점을 고려
        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)
        #####
        #####

        # 프롬프트 리스트와 이미지 feature 리스트에서 하나씩 꺼내서
        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() # 인덱스가 같은 프롬프트와 이미지 피쳐 간의 유사도를 측정
            logits.append(l_i)
        logits = torch.stack(logits)

        if self.prompt_learner.training:
            return F.cross_entropy(logits, label) # cross-entropy를 이용해서 양성 쌍간의 프롬프트-이미지 유사도를 높이는 데에 초점을 둠

        return logits

```

정리 CoCoOp를 학습하는 방법은 간단하다. 그냥 CLIP에서 했던 것처럼 Image-feature와 그에 해당하는 프롬프트의 유사도를 계산하면 된다. 주의할 점은 pretrained CLIP을 기반으로 CoCoOp가 설계되었으므로 이미 이미지 피쳐와 텍스트 피쳐가 align된 상태이고, contrastive learning이 필요하지 않다는 것이다. CoCoOp 모델의 목적은 음성 쌍의 suppression 보다는 양성 쌍의 유사도를 높이는 방향에 있다. 따라서 CLIP을 학습시킬 때와 달리 양성 쌍끼리만 벡터 내적한 후 logit_scale만큼 곱해 유사도를 계산해준 뒤(logit), crossentropy loss를 구하면 된다.

질문 그렇다면 의도치 않은 결과로 이미지 A에 대한 프롬프트가 이미지 B와 높은 유사도를 가질 가능성은 없는 것인가? CoCoOp에서 따로 음성 쌍에 대한 Suppression을 안 해줘도 성능 상의 장애가 없는 것인가? -> logit에 cross-entropy를 쓴다는 것 자체가 확률을 높인다는 것이고 확률은 합이 1이므로 이는 곧 음성 쌍으로 판정될 확률을 억제하는 것과 같은 효과를 띄게 된다.

✓ Q2. Training 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)

```

 Building training test
+ resize the smaller edge to 224
+ 224x224 center crop

```

Turning off gradients in both the image and the text encoder
Parameters to be updated: {'prompt_learner.meta_net.linear1.weight', 'prompt_learner.meta_net.linear2.bias', 'prompt_learner.meta_net.linear2.weight'}
Loading evaluator: Classification
No checkpoint found, train from scratch
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use warnings.warn()
epoch [1/100] batch [20/20] time 0.109 (0.263) data 0.000 (0.019) loss 0.2744 (1.1880) lr 2.5000e-03 eta 0:08:40
epoch [2/100] batch [20/20] time 0.129 (0.135) data 0.000 (0.015) loss 0.8413 (0.8959) lr 2.4994e-03 eta 0:04:24
epoch [3/100] batch [20/20] time 0.112 (0.130) data 0.000 (0.015) loss 0.6401 (0.7858) lr 2.4975e-03 eta 0:04:12
epoch [4/100] batch [20/20] time 0.111 (0.128) data 0.000 (0.015) loss 0.5078 (0.7158) lr 2.4945e-03 eta 0:04:05
epoch [5/100] batch [20/20] time 0.111 (0.126) data 0.000 (0.015) loss 0.5728 (0.6316) lr 2.4901e-03 eta 0:04:00
epoch [6/100] batch [20/20] time 0.108 (0.132) data 0.000 (0.014) loss 0.5991 (0.6008) lr 2.4846e-03 eta 0:04:09
epoch [7/100] batch [20/20] time 0.111 (0.128) data 0.000 (0.013) loss 0.3845 (0.6656) lr 2.4779e-03 eta 0:03:57
epoch [8/100] batch [20/20] time 0.113 (0.124) data 0.000 (0.014) loss 1.3896 (0.6626) lr 2.4699e-03 eta 0:03:47
epoch [9/100] batch [20/20] time 0.104 (0.120) data 0.000 (0.013) loss 0.1779 (0.4569) lr 2.4607e-03 eta 0:03:38
epoch [10/100] batch [20/20] time 0.109 (0.123) data 0.000 (0.013) loss 1.2285 (0.5049) lr 2.4504e-03 eta 0:03:40
epoch [11/100] batch [20/20] time 0.112 (0.128) data 0.000 (0.014) loss 0.2566 (0.4992) lr 2.4388e-03 eta 0:03:47
epoch [12/100] batch [20/20] time 0.109 (0.130) data 0.000 (0.016) loss 1.1338 (0.4681) lr 2.4261e-03 eta 0:03:48
epoch [13/100] batch [20/20] time 0.109 (0.125) data 0.000 (0.015) loss 0.8569 (0.5033) lr 2.4122e-03 eta 0:03:37
epoch [14/100] batch [20/20] time 0.110 (0.125) data 0.000 (0.014) loss 0.5000 (0.4536) lr 2.3972e-03 eta 0:03:35
epoch [15/100] batch [20/20] time 0.105 (0.123) data 0.000 (0.014) loss 1.0693 (0.5466) lr 2.3810e-03 eta 0:03:29
epoch [16/100] batch [20/20] time 0.104 (0.121) data 0.000 (0.014) loss 1.3516 (0.4746) lr 2.3638e-03 eta 0:03:23
epoch [17/100] batch [20/20] time 0.105 (0.123) data 0.000 (0.016) loss 0.0210 (0.3349) lr 2.3454e-03 eta 0:03:23
epoch [18/100] batch [20/20] time 0.112 (0.124) data 0.000 (0.014) loss 0.1465 (0.2784) lr 2.3259e-03 eta 0:03:23
epoch [19/100] batch [20/20] time 0.109 (0.126) data 0.000 (0.013) loss 1.0566 (0.3819) lr 2.3054e-03 eta 0:03:24
epoch [20/100] batch [20/20] time 0.109 (0.127) data 0.000 (0.014) loss 0.3176 (0.4950) lr 2.2839e-03 eta 0:03:23
epoch [21/100] batch [20/20] time 0.114 (0.126) data 0.000 (0.014) loss 0.8232 (0.3428) lr 2.2613e-03 eta 0:03:19
epoch [22/100] batch [20/20] time 0.109 (0.127) data 0.000 (0.013) loss 0.2184 (0.4419) lr 2.2377e-03 eta 0:03:18
epoch [23/100] batch [20/20] time 0.104 (0.121) data 0.000 (0.013) loss 0.0971 (0.3105) lr 2.2131e-03 eta 0:03:06
epoch [24/100] batch [20/20] time 0.104 (0.119) data 0.000 (0.013) loss 0.1768 (0.5294) lr 2.1876e-03 eta 0:03:00
epoch [25/100] batch [20/20] time 0.106 (0.120) data 0.000 (0.013) loss 0.4207 (0.3620) lr 2.1612e-03 eta 0:02:59
epoch [26/100] batch [20/20] time 0.110 (0.125) data 0.000 (0.015) loss 0.2383 (0.3636) lr 2.1339e-03 eta 0:03:05
epoch [27/100] batch [20/20] time 0.111 (0.131) data 0.000 (0.015) loss 0.2010 (0.3267) lr 2.1057e-03 eta 0:03:10
epoch [28/100] batch [20/20] time 0.110 (0.126) data 0.000 (0.014) loss 0.3474 (0.3116) lr 2.0766e-03 eta 0:03:01
epoch [29/100] batch [20/20] time 0.108 (0.126) data 0.000 (0.014) loss 0.6992 (0.3488) lr 2.0468e-03 eta 0:02:58
epoch [30/100] batch [20/20] time 0.105 (0.126) data 0.000 (0.014) loss 0.0072 (0.4035) lr 2.0161e-03 eta 0:02:56
epoch [31/100] batch [20/20] time 0.107 (0.123) data 0.000 (0.014) loss 0.7021 (0.3575) lr 1.9847e-03 eta 0:02:49
epoch [32/100] batch [20/20] time 0.105 (0.121) data 0.000 (0.014) loss 0.1962 (0.2706) lr 1.9526e-03 eta 0:02:44
epoch [33/100] batch [20/20] time 0.111 (0.124) data 0.000 (0.013) loss 0.1635 (0.2758) lr 1.9198e-03 eta 0:02:46
epoch [34/100] batch [20/20] time 0.115 (0.126) data 0.000 (0.014) loss 0.0838 (0.3325) lr 1.8863e-03 eta 0:02:46
epoch [35/100] batch [20/20] time 0.110 (0.126) data 0.000 (0.014) loss 0.0181 (0.2818) lr 1.8522e-03 eta 0:02:43

# 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)

[Icon] 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/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use warnings.warn()
/content/ProMetaR/dassl/utils/torchtools.py:102: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default behavior) which can be unsafe. Please use `torch.load` with `weights_only=True` to prevent unsafe loading by default.
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.weight', 'prompt_learner.meta_net.linear2.bias', 'prompt_learner.meta_net.linear2.weight'}
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 [00:40<00:00, 1.03s/it]=> result
* total: 3,900
* correct: 1,766

```

```
* accuracy: 45.3%
* error: 54.7%
* macro_f1: 40.9%
```

✓ 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.

CoOp는 Context Optimization의 약자이고 CoCoOp는 Conditional Context Optimization의 약자이다.

EuroSAT의 10개 클래스 데이터를 5개, 5개로 쪼개서 각각을 base, new 데이터셋으로 설정한 후 CoCoOp와 CoOp 모두 base 클래스에서 학습 시킨 후 few-shot 학습을 거치고 new 클래스 데이터셋에 대해 평가하였다.

결과를 비교하자면 CoCoOp를 사용했을 때는 정확도가 45.3%, macro_f1이 40.9%가 나온 반면 CoOp를 사용했을 때는 정확도가 61.3%, macro_f1이 59.0%가 나왔다.

CoCoOp가 CoOp의 일반화 능력 한계를 극복하기 위해 고안되었다는 점을 고려할 때 이는 예상된 결과와 상반된다.

이에 대한 원인으로 생각해볼 만한 것들의 후보는 다음과 같다.

1. CoCoOp의 과적합

- EuroSAT의 적은 class 개수
- EuroSAT의 class 데이터 간 데이터 다양성 부족 & CoCoOp의 모델 복잡도에 따른 과적합

EuroSAT은 총 10개의 클래스만을 보유한 데이터셋이고 특히 위성사진이라는 점에서 비교적 데이터의 내재적 특징이 유사하다.(spatial correlated됨) 따라서 데이터 다양성이 부족하다고 할 수 있다. 이렇듯 단순한 데이터 다양성에 비해 CoCoOp는 다소 복잡한 모델이며 이는 CoCoOp가 개별 이미지의 노이즈에 과적합되었을 가능성을 남긴다.

2. CoCoOp의 미적합

- 학습 데이터셋의 부족으로 인한 일반화 학습 부족

CoCoOp의 학습 로그와 CoOp의 전체 100개의 훈련 batch에 대한 훈련 로그로 loss를 비교할 때, CoOp는 전체적으로 loss가 안정적으로 꾸준히 감소하는 추세가 관찰되고 있으므로 수렴하고 있다고 할 수 있다. 또한 마지막 10개 batch에서 모두 0.1 이하의 loss를 보이고 있으며 특히 마지막 100번째 batch에서는 0.0051로 굉장히 낮은 loss값을 보이고 있다. 그러므로 CoOp의 경우 학습이 상당 부분 진행되었다고 추론할 수 있다. 그러나 CoCoOp의 경우 대체적으로 loss가 감소하고는 있으나 비교적 완만하게 손실이 감소하고 있다. 또한 마지막 batch 10개를 봤을 때 maximum loss가 0.2981이지만 minimum loss가 0.0018인 점을 고려하여 아직 수렴이 되지 않았음을 유추할 수 있다. 더군다나 maximum loss를 기록한 batch를 이상치로 볼 수도 없는 것이, second maximum loss가 0.2971이다.

이렇듯 CoCoOp가 CoOp와 다르게 수렴하지 못하는, 즉 미적합 상태인 이유 중 하나는 CoCoOp의 비교적 많은 학습 파라미터이다. CoCoOp의 경우 meta-net을 학습해야 하는데 이는 CoOp가 학습하는 context vector 그 자체보다 훨씬 많은 학습 파라미터를 요구한다. 따라서 CoCoOp가 요구하는 학습량이 CoOp보다 많고 4000개의 훈련 데이터셋이 CoCoOp에서는 비교적 부족했을 것이다.

이상이 CoCoOp의 성능 상 하자에 대한 원인 지목이었다면 아래 서술되는 원인은 이러한 성능 역전의 원인을 CoOp의 성능에서 찾는다.

1. CoOp는 데이터셋의 일반적인 특징을 학습하는데, 문제가 있다면 훈련 데이터로 사용되는 클래스들을 overfitting한다는 것이다. 그러나 EuroSAT 데이터셋을 클래스별로 샘플링하여 관찰하였을 때, EuroSAT의 데이터는 비교적 클래스 간의 차이가 덜한 듯 보였다. 더군다나 이미 지 해상도가 CLIP의 기본 해상도인 224x224로 설정되어 있는데 이는 클래스 간의 차이를 더욱 줄일 수 있다. 이와 같이 클래스 간 차이가 덜하므로 CoOp가 new class data에 대해서도 비교적 우수한 성능을 거둘 수 있었을 것이다.