Homework 4

2022320009 이수현

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.

```
!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 1Ip7yaCWFi0eaOFUGga0lUdVi_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.COCOOP.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})")
        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 fil
)
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",
)
parser.add_argument("--train-batch-size", type=int, default=4)
parser.add_argument("--epoch", type=int, default=10)
parser.add_argument("--subsample-classes", type=str, default="base")
parser.add_argument(
    "--load-epoch", type=int, default=0, help="load model weights at this epoch for evaluation"
)
args = parser.parse_args([])

def main(args):
    cfg = setup_cfg(args)
    if cfg.SEED >= 0:
        set_random_seed(cfg.SEED)

    if torch.cuda.is_available() and cfg.USE_CUDA:
        torch.backends.cudnn.benchmark = True

    trainer = build_trainer(cfg)
    if args.eval_only:
        trainer.load_model(args.model_dir, epoch=args.load_epoch)
        acc = trainer.test()
        return acc

```

```
    acc = trainer.train()
    return acc
```

```
inflating: eurosat/2750/PermanentCrop/PermanentCrop_615.jpg
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_505.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/ProMetaR/data/eurosat
Downloading...
From: https://drive.google.com/uc?id=1Ip7yaCWFi0ea0FUGga0IUdVi_DDQth1o
To: /content/ProMetaR/ProMetaR/data/eurosat/split_zhou_EuroSAT.json
100% 3.01M/3.01M [00:00<00:00, 132MB/s]
/content/ProMetaR/ProMetaR
```

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 ({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 #######   #이 부분이 meta netwrok 정의
        ########## Define Meta Net ##########
        self.meta_net = nn.Sequential(OrderedDict([
            ("linear1", nn.Linear(vis_dim, vis_dim // 16)), #Q1 #이미지 faature를 압축. vis_dim : 이미지 특징 벡터의 차원.
            ("relu", nn.ReLU(inplace=True)),  #relu activation func
            ("linear2", nn.Linear(vis_dim // 16, ctx_dim)) #vis_dim -> ctx_dim(:컨텍스트 차원)으로 매핑
        ]))
        ######################################
        ## Hint: meta network is composed to linear layer, relu activation, and linear layer.
        #input 이미지의 특징을 컨텍스트 차원으로 매핑. 2개의 linear layer와 relu함수를 사용



        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): #im_features : input image feature vector. 이미지 인코더의 출력값
        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)#Q2.  Hint: Image feature is given as input to meta network")  # (batch, ctx_dim)
        bias = bias.unsqueeze(1)  # 차원 확장 (batch_size, ctx_dim) -> (batch, 1, ctx_dim)
        ctx = ctx.unsqueeze(0)  # 차원 확장 -> (1, n_ctx, ctx_dim)
        ctx_shifted = ctx + bias #Q3. "Fill in here, Hint: Add meta token to context token"  # (batch, n_ctx, ctx_dim)
        ############################################
        ############################################
        # bias는 이미지 조건부로 생성된 벡터
        # ctx는 컨텍스트 벡터. 학습 가능한 초기값
        # ctx는 고정된 컨텍스트 벡터로 모든 이미지에 동일하게 적용. CoOp는 이걸로 각 이미지의 특징을 반영하지 못함
        # 그래서 나온게 bias
        # bias는 meta network가 이미지 특징 벡터를 입력받아 생성한 각 이미지 특징이 반영된 벡터
        # CoCoOp는 bias를 ctx에 더해 조정된 ctx를 이용해서 이미지별로 특징을 반영할 수 있음.



        # 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)) #이미지 특징 벡터 추출
        image_features = image_features / image_features.norm(dim=-1, keepdim=True)


        ############################################
        ########## Q4. Fill in the blank #########
        prompts = self.prompt_learner(image_features) #Q4.이미지특징을 prompt learner로 전달
        ############################################
        ############################################
        #이미지특징을 prompt learner로 전달,
        #그럼 이제 prompl_learner는 이미지 특징 벡터를 받아 각 이미지별로, 클래스별로 조정된 텍스트 프롬포트를 return
        #prompts크기: (batch_size, n_cls, n_tkn, ctx_dim)

        #이미지와 텍스트 특징 매칭
        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
```

takeout messages

- CoOp : 고정된 text prompt로 모델이 학습하도록 최적화
- CoCoOp : 추가적으로 이미지의 특성에 따라 동적인 text context를 생성. meta network를 사용하여 image toekn을 추출하고 추출한 것을 text prompt와 결합

- ctx는 모든 이미지에 동일하게 적용되는 초기 컨텍스트
- bias는 meta network가 이미지 특징 벡터를 입력받아 생성한 각 이미지의 특성을 반영한 벡터. 각 이미지마다 다른 값을 가짐

- ctx랑 bias가 더하기에는 차원이 안맞아서 조정해줘야함(unsqueeze)
- ctx랑 bias를 알맞게 차원확장해주고
- 브로드캐스팅 규칙에 따라 더함 -> shifted_ctx
- 여기서 확장할 때 1로 확장한 이유는 1은 상대방의 차원을 그대로 복사해오기 때문

- CoOp는 ctx만 이용해서 각 이미지의 특징을 반영하지 못한다는 단점이 있음
- CoCoOp 는 ctx에 bias를 더해 shifted_ctx를 이용함으로써 각 이미지의 특징을 반영할 수 있음.

- 이미지 특징을 promp learner로 전달해서 각 이미지별로, 클래스별로 알맞은 택스트 프롬프트를 return함
- 특징 매칭

[same in english]

- CoOp : Optimizes model to learn with fixed text prompt
- CoCoOp: Additionally, a dynamic text context is created based on the characteristics of the image. Image toekn is extracted using the meta network and combined with the text prompt

---

- ctx is an initial context that applies equally to all images
- The bias is a vector that reflects the characteristics of each image generated by the metannetwork by receiving the image feature vector. Each image has a different value

---

- Ctx and bias are not the right dimensions to add, so they need to be adjusted (unsqueeze)
- It expands the size of CTX and Bias
- Added according to broadcasting rules -> shifted_ctx
- The reason why I expanded to 1 when expanding here is because 1 copies the other person's dimensions

---

- CoOp has the disadvantage of not reflecting the features of each image using only ctx
- CoCoOp can reflect the characteristics of each image by using shifted_ctx by adding bias to ctx.

- Transfer image features to the prompeller to return the appropriate tact prompt for each image and for each class
- feature matching

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 [62/100] batch [20/20] time 0.120 (0.143) data 0.000 (0.019) loss 0.0041 (0.2124) lr 8.2658e-04 eta 0:01:48
epoch [63/100] batch [20/20] time 0.171 (0.193) data 0.000 (0.023) loss 0.1748 (0.2624) lr 7.8984e-04 eta 0:02:22
epoch [64/100] batch [20/20] time 0.108 (0.148) data 0.000 (0.025) loss 0.2600 (0.1714) lr 7.5357e-04 eta 0:01:46
epoch [65/100] batch [20/20] time 0.108 (0.144) data 0.000 (0.017) loss 0.5747 (0.2100) lr 7.1778e-04 eta 0:01:40
epoch [66/100] batch [20/20] time 0.114 (0.144) data 0.000 (0.024) loss 0.1279 (0.1686) lr 6.8251e-04 eta 0:01:38
epoch [67/100] batch [20/20] time 0.275 (0.236) data 0.000 (0.018) loss 0.0054 (0.2219) lr 6.4781e-04 eta 0:02:35
epoch [68/100] batch [20/20] time 0.113 (0.193) data 0.000 (0.027) loss 0.2773 (0.2684) lr 6.1370e-04 eta 0:02:03
epoch [69/100] batch [20/20] time 0.109 (0.143) data 0.000 (0.019) loss 0.0228 (0.2471) lr 5.8022e-04 eta 0:01:28
epoch [70/100] batch [20/20] time 0.108 (0.145) data 0.000 (0.019) loss 0.2318 (0.1503) lr 5.4740e-04 eta 0:01:26
epoch [71/100] batch [20/20] time 0.171 (0.224) data 0.000 (0.030) loss 0.0285 (0.1188) lr 5.1527e-04 eta 0:02:10
epoch [72/100] batch [20/20] time 0.110 (0.145) data 0.000 (0.023) loss 0.1163 (0.2144) lr 4.8387e-04 eta 0:01:21
epoch [73/100] batch [20/20] time 0.107 (0.145) data 0.000 (0.021) loss 0.0424 (0.1745) lr 4.5322e-04 eta 0:01:18
epoch [74/100] batch [20/20] time 0.109 (0.145) data 0.000 (0.030) loss 0.1774 (0.1305) lr 4.2336e-04 eta 0:01:15
epoch [75/100] batch [20/20] time 0.158 (0.194) data 0.000 (0.021) loss 0.0523 (0.1880) lr 3.9432e-04 eta 0:01:36
epoch [76/100] batch [20/20] time 0.112 (0.147) data 0.000 (0.026) loss 0.0109 (0.1781) lr 3.6612e-04 eta 0:01:10
epoch [77/100] batch [20/20] time 0.105 (0.145) data 0.000 (0.020) loss 0.0092 (0.1832) lr 3.3879e-04 eta 0:01:06
epoch [78/100] batch [20/20] time 0.107 (0.143) data 0.000 (0.022) loss 0.1420 (0.2149) lr 3.1236e-04 eta 0:01:02
epoch [79/100] batch [20/20] time 0.157 (0.195) data 0.000 (0.019) loss 0.6455 (0.2502) lr 2.8686e-04 eta 0:01:22
epoch [80/100] batch [20/20] time 0.109 (0.145) data 0.000 (0.024) loss 0.1262 (0.1671) lr 2.6231e-04 eta 0:00:58
epoch [81/100] batch [20/20] time 0.107 (0.144) data 0.000 (0.018) loss 0.1049 (0.1736) lr 2.3873e-04 eta 0:00:54
epoch [82/100] batch [20/20] time 0.109 (0.145) data 0.000 (0.022) loss 0.5278 (0.1947) lr 2.1615e-04 eta 0:00:52
epoch [83/100] batch [20/20] time 0.158 (0.194) data 0.000 (0.018) loss 0.1053 (0.1895) lr 1.9459e-04 eta 0:01:06
epoch [84/100] batch [20/20] time 0.345 (0.178) data 0.000 (0.022) loss 0.1261 (0.1526) lr 1.7407e-04 eta 0:00:56
epoch [85/100] batch [20/20] time 0.113 (0.144) data 0.000 (0.026) loss 0.0314 (0.1640) lr 1.5462e-04 eta 0:00:43
epoch [86/100] batch [20/20] time 0.112 (0.145) data 0.000 (0.021) loss 0.0459 (0.1491) lr 1.3624e-04 eta 0:00:40
epoch [87/100] batch [20/20] time 0.162 (0.224) data 0.000 (0.036) loss 0.2108 (0.1862) lr 1.1897e-04 eta 0:00:58
epoch [88/100] batch [20/20] time 0.110 (0.145) data 0.000 (0.019) loss 0.1178 (0.2581) lr 1.0281e-04 eta 0:00:34
epoch [89/100] batch [20/20] time 0.110 (0.145) data 0.000 (0.018) loss 0.0460 (0.2158) lr 8.7779e-05 eta 0:00:31
epoch [90/100] batch [20/20] time 0.109 (0.144) data 0.000 (0.026) loss 0.0492 (0.1039) lr 7.3899e-05 eta 0:00:28
epoch [91/100] batch [20/20] time 0.159 (0.197) data 0.000 (0.019) loss 0.2791 (0.1459) lr 6.1179e-05 eta 0:00:35
epoch [92/100] batch [20/20] time 0.113 (0.146) data 0.000 (0.024) loss 0.0514 (0.1019) lr 4.9633e-05 eta 0:00:23
epoch [93/100] batch [20/20] time 0.112 (0.154) data 0.000 (0.021) loss 0.1763 (0.2449) lr 3.9271e-05 eta 0:00:21
epoch [94/100] batch [20/20] time 0.108 (0.147) data 0.000 (0.018) loss 0.2859 (0.2261) lr 3.0104e-05 eta 0:00:17
epoch [95/100] batch [20/20] time 0.166 (0.202) data 0.000 (0.023) loss 0.1564 (0.1853) lr 2.2141e-05 eta 0:00:20
epoch [96/100] batch [20/20] time 0.110 (0.145) data 0.000 (0.021) loss 0.4089 (0.1330) lr 1.5390e-05 eta 0:00:11
epoch [97/100] batch [20/20] time 0.109 (0.144) data 0.000 (0.018) loss 0.0698 (0.1542) lr 9.8566e-06 eta 0:00:08
epoch [98/100] batch [20/20] time 0.112 (0.145) data 0.000 (0.020) loss 0.2188 (0.2041) lr 5.5475e-06 eta 0:00:05
epoch [99/100] batch [20/20] time 0.157 (0.194) data 0.000 (0.019) loss 0.0691 (0.1264) lr 2.4666e-06 eta 0:00:03
epoch [100/100] batch [20/20] time 0.116 (0.146) data 0.000 (0.027) loss 0.0025 (0.1101) lr 6.1680e-07 eta 0:00:00
Checkpoint saved to outputs/cocoop/prompt_learner/model.pth.tar-100
Finish training
Deploy the last-epoch model
Evaluate on the *test* set
100%|██████████| 42/42 [01:11<00:00,  1.71s/it]=> result
* total: 4,200
* correct: 3,813
* accuracy: 90.8%
* error: 9.2%
* macro_f1: 90.9%
Elapsed: 0:07:00
```

```
# 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/ProMetaR/data/eurosat/split_zhou_EuroSAT.json
Loading preprocessed few-shot data from /content/ProMetaR/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.ctx', 'prompt_learner.meta_net.linear1.weight', 'prompt_learner.meta_net.linear2.weight', 'prompt_lea
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 [01:06<00:00,  1.69s/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.

1. Base Class 성능비교

Metric	CoOp	CoCoOp
Total	4,200	4,200
Correct	3,839	3,813
Accuracy (%)	91.4%	90.8%
Error (%)	8.6%	9.2%
Macro F1 (%)	91.5%	90.9%
Elapsed Time	0:03:10	0:07:00

- CoOp가 accuracy와 macro F1을 봤을 때 더 높은 값을 가지고, error는 더 적은 것으로 보아 CoOp의 성능이 더 좋다.
- CoOp는 고정된 ctx를 사용하므로 학습 데이터에 더 최적화 되어있다.
- 반면 CoCoOp는 ctx에 bias를 더해 shifted_ctx를 사용하여 동적으로 프롬프트를 생성하는데, 이 과정에서 base class에 상대적으로 덜 최적화 되기 때문에 CoOp보다 떨어지는 성능을 보인다고 생각할 수 있다.
- Elapsed Time을 보면 CoCoOp가 더 오래 걸리는데, 이는 각 이미지의 특징을 반영하기위해 meta network를 사용하기 때문에 더 오랜 시간이 걸리는 것이다.
- 반면 CoOp는 고정된 컨텍스트를 사용하여 추가연산과정이 없기 때문에 더 빠르다

2. New Class 성능 비교

Metric	CoOp	CoCoOp
Total	3,900	3,900
Correct	2,007	1,687
Accuracy (%)	51.5%	43.3%
Error (%)	48.5%	56.7%
Macro F1 (%)	45.6%	39.0%

- accuracy나 marco F1을 봤을 때 CoOp가 더 높은 퍼센트를 가지고, error도 CoOp가 더 낮다.
- 각 이미지별 특징을 반영할 수 있는 CoCoOp의 성능이 더 높을 것이라는 예상과 다르게 new class에서도 CoOp가 더 좋은 성능을 보인다.
- CoCoOp의 동적 텍스트 컨텍스트 생성 과정이 new class에 기대만큼 효과적으로 작동하지는 않았음을 보여준다.
- 기대만큼의 성능이 나오지 않은 이유를 다음과 같이 생각해 볼 수 있다.
  1. meta network에서 imgae feature를 텍스트 컨텍스트로 매핑할 때, new class의 특징을 잘 반영하지 못했을 가능성
  2. 데이터셋에서 base class와 new class에 큰 차이가 있을 경우, 동적 프롬프트 생성방식이 비효율적으로 작용했을 가능성.
  3. CoCoOp의 동적 프롬프트 생성이 base class에 과적합되어, new class에 일반화되지 않았을 가능성.

[same in english]

1. Base Class Performance Comparison
   - CoOp's performance is better because CoOp has a higher value and fewer errors when looking at acuracy and macro F1.
   - CoOp is more optimized for training data as it uses fixed ctx.
   - On the other hand, CoCoOp dynamically generates prompts using shifted_ctx by adding bias to ctx, which can be considered to perform worse than CoOp because it is relatively less optimized for the base class.
   - Looking at Elapsed Time, CoCoOp takes longer, which takes longer because the meta network is used to reflect the features of each image.
   - CoOp, on the other hand, is faster because there is no additional computation process using a fixed context

2. New Class Performance Comparison
    - When looking at accuracy or marco F1, CoOp has a higher percentage and error is lower.
    - Contrary to the expectation that CoCoOp's performance, which can reflect the characteristics of each image, will be higher, CoOp shows better performance even in the new class.
    - We show that the dynamic text context generation process of CoCoOp did not work as effectively as expected for the new class.
    - The reason why the performance did not come out as expected can be considered as follows.
        1. When mapping the image feature to text context in the meta network, it is likely that it did not reflect the new class's features well
        2. If there is a large difference between base class and new class in the dataset, it is possible that the dynamic prompt generation method worked inefficiently.
        3. The possibility that the dynamic prompt generation of CoCoOp was overfitting the base class and not generalized to the new class.