# 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.

```
!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 promptin
    cfg.TRAINER.PROMETAR.PROMPT_DEPTH_TEXT = 9  # Max 12, minimum 0, for 0 it will be using shallow IVLP prompting
    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 t
)
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
```

숨겨진 출력 표시

## 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, :] # context vecotr 생성
            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 trai

        ### 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...]
        # 토큰화된 프롬프트들이 하나의 텐서로 저장됨



        ####################################
        ####### Q1. Fill in the blank #######
        ########## Define Meta Net: extract image tokens that is added to the text prompt ##########
        self.meta_net = nn.Sequential(OrderedDict([
            ("linear1", nn.Linear(vis_dim, vis_dim // 16)), # 입력 데이터를 줄여 더 간단한 표현을 학습
            ("relu", nn.ReLU(inplace=True)),                # nonlinear 변환을 통해 모델이 더 복잡한 관계를 학습할 수 있도록
            ("linear2", nn.Linear(vis_dim // 16, ctx_dim))  # 압축된 특징을 최종 컨텍스트 벡터로 변환
        ]))
        ####################################
        ## Hint: meta network is composed to linear layer, relu activation, and linear layer.



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

        # CLIP 모델 토큰 임베딩 생성
        with torch.no_grad():
            # 프롬프트를 모델이 이해할 수 있는 임베딩 형태로 변환
            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                              # 클래스의 수를 저장
            self.n_ctx = n_ctx                              # 컨텍스트 벡터의 길이(토큰의 수)를 저장
            self.tokenized_prompts = tokenized_prompts  # torch.Tensor # 토크나이즈된 프롬프트를 저장
            self.name_lens = name_lens                      # 각 클래스 이름의 토큰 길이를 저장

        # 최종 프롬프트를 생성하는 함수
        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)

            # label이 주어진 경우, prefix와 suffix에서 해당 레이블의 데이터만 선택
            if label is not None:
                prefix = prefix[label]
                suffix = suffix[label]

            # 컨텍스트 벡터와 시작/종료 토큰을 결합하여 최종 프롬프트를 생성
            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)
            # im_features는 이미지 특징으로, 원래는 vis_dim이라는 시각적 특징 공간에 있음.
            # im_features는 meta_net에 의해 ctx_dim 차원으로 매핑
            bias = bias.unsqueeze(1)  # (batch, 1, ctx_dim) # bias의 차원을 (batch, 1, ctx_dim)로 확장하여 컨텍스트 벡터와의 계산
            ctx = ctx.unsqueeze(0)  # (1, n_ctx, ctx_dim)
            # ctx의 차원을 (1, n_ctx, ctx_dim)으로 확장하여 배치 크기와 맞춤.
            # ctx는 clip_model의 텍스트 임베딩 차원(ctx_dim)을 따름 -> ctx_dim은 프롬프트 학습에 사용되는 학습 가능한 벡터의 기본 차원
            ctx_shifted = ctx + bias  # (batch, n_ctx, ctx_dim)
            # 각 배치의 bias를 컨텍스트 벡터에 더해 조정된 컨텍스트 벡터(ctx_shifted)를 생성
            ##########################################
            ##########################################



            # 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)
                # ctx_shifted_i의 차원을 (n_cls, n_ctx, ctx_dim)으로 확장하여 모든 클래스에 적용 가능하게 함.
                pts_i = self.construct_prompts(ctx_i, prefix, suffix)  # (n_cls, n_tkn, ctx_dim)
                # 확장된 컨텍스트 벡터(ctx_i)와 시작 토큰(prefix), 종료 토큰(suffix)을 결합해 프롬프트를 생성함.
                prompts.append(pts_i)
                # 생성된 프롬프트를 리스트에 추가
            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()

        # 이미지를 CLIP의 이미지 인코더에 전달하여 임베딩 벡터를 생성
        image_features = self.image_encoder(image.type(self.dtype))
        # L2 정규화를 수행하여 이미지 임베딩 벡터를 단위 벡터로
        image_features = image_features / image_features.norm(dim=-1, keepdim=True)


        ############################################
        ########## Q4. Fill in the blank #########
        prompts = self.prompt_learner(image_features)
        ############################################
        # self.prompt_learner는 이미지 특징을 기반으로 프롬프트를 생성함. 즉, image_features를 self.prompt_learner에 전달해야 함.
        ############################################


        logits = []
        for pts_i, imf_i in zip(prompts, image_features):
            text_features = self.text_encoder(pts_i, tokenized_prompts) # 각 프롬프트(pts_i)를 텍스트 임베딩(text_featur
            text_features = text_features / text_features.norm(dim=-1, keepdim=True)
            l_i = logit_scale * imf_i @ text_features.t() # 이미지 임베딩(imf_i)과 텍스트 임베딩(text_features)의 내적을 계
            logits.append(l_i) # 각 배치에 대해 매칭 점수(l_i)를 logits 리스트에 추가
        logits = torch.stack(logits)

        # 매칭 점수(logits)와 레이블(label)을 기반으로 손실 계산
        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.

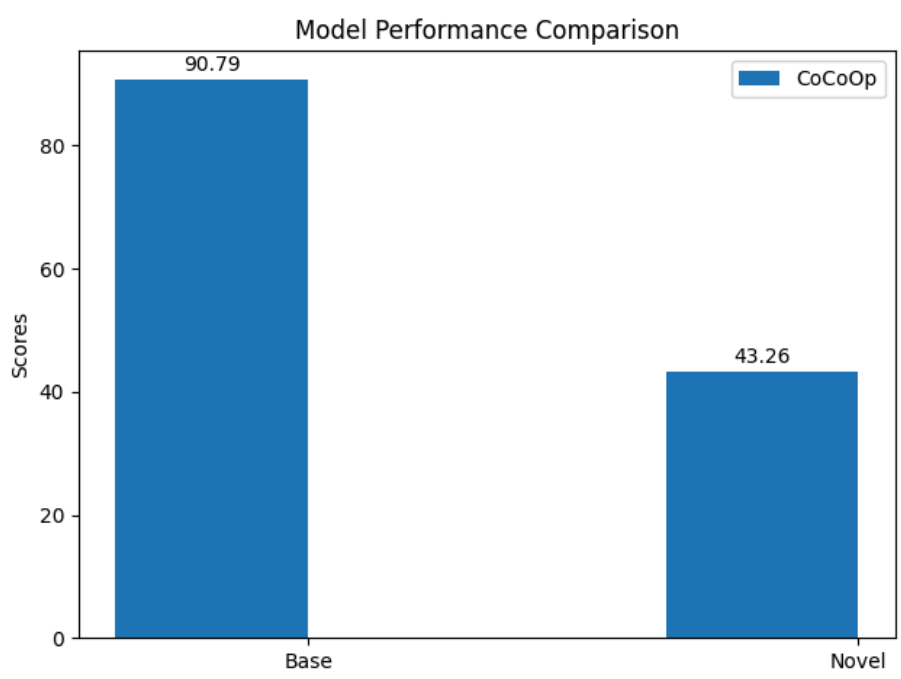
```
# Train on the Base Classes Train split and evaluate accuracy on the Base Classes Test split.
args.trainer = "CoCoOp"
args.train_batch_size = 4
args.epoch = 100
args.output_dir = "outputs/cocoop"

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

```
epoch [85/100] batch [20/20] time 0.097 (0.127) data 0.000 (0.023) loss 0.0314 (0.1640) lr 1.5462e-04 eta
epoch [86/100] batch [20/20] time 0.096 (0.128) data 0.000 (0.025) loss 0.0459 (0.1491) lr 1.3624e-04 eta
epoch [87/100] batch [20/20] time 0.096 (0.131) data 0.000 (0.016) loss 0.2108 (0.1862) lr 1.1897e-04 eta
epoch [88/100] batch [20/20] time 0.137 (0.148) data 0.000 (0.021) loss 0.1178 (0.2581) lr 1.0281e-04 eta
epoch [89/100] batch [20/20] time 0.140 (0.195) data 0.000 (0.037) loss 0.0460 (0.2158) lr 8.7779e-05 eta
epoch [90/100] batch [20/20] time 0.098 (0.128) data 0.000 (0.022) loss 0.0492 (0.1039) lr 7.3899e-05 eta
epoch [91/100] batch [20/20] time 0.101 (0.130) data 0.000 (0.018) loss 0.2791 (0.1459) lr 6.1179e-05 eta
epoch [92/100] batch [20/20] time 0.091 (0.129) data 0.000 (0.017) loss 0.0514 (0.1019) lr 4.9633e-05 eta
epoch [93/100] batch [20/20] time 0.134 (0.149) data 0.000 (0.021) loss 0.1763 (0.2449) lr 3.9271e-05 eta
epoch [94/100] batch [20/20] time 0.174 (0.203) data 0.000 (0.030) loss 0.2859 (0.2261) lr 3.0104e-05 eta
epoch [95/100] batch [20/20] time 0.093 (0.128) data 0.000 (0.019) loss 0.1564 (0.1853) lr 2.2141e-05 eta
epoch [96/100] batch [20/20] time 0.097 (0.130) data 0.000 (0.019) loss 0.4089 (0.1330) lr 1.5390e-05 eta
epoch [97/100] batch [20/20] time 0.095 (0.132) data 0.000 (0.022) loss 0.0698 (0.1542) lr 9.8566e-06 eta
epoch [98/100] batch [20/20] time 0.144 (0.147) data 0.000 (0.019) loss 0.2188 (0.2041) lr 5.5475e-06 eta
epoch [99/100] batch [20/20] time 0.140 (0.195) data 0.000 (0.035) loss 0.0691 (0.1264) lr 2.4666e-06 eta
epoch [100/100] batch [20/20] time 0.105 (0.129) data 0.000 (0.022) loss 0.0025 (0.1101) lr 6.1680e-07 eta
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 [01:01<00:00,  1.47s/it]=> result
* total: 4,200
* correct: 3,813
* accuracy: 90.8%
* error: 9.2%
* macro_f1: 90.9%
Elapsed: 0:06:32
```

```
# 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
cocoop_novel_acc = main(args)
```

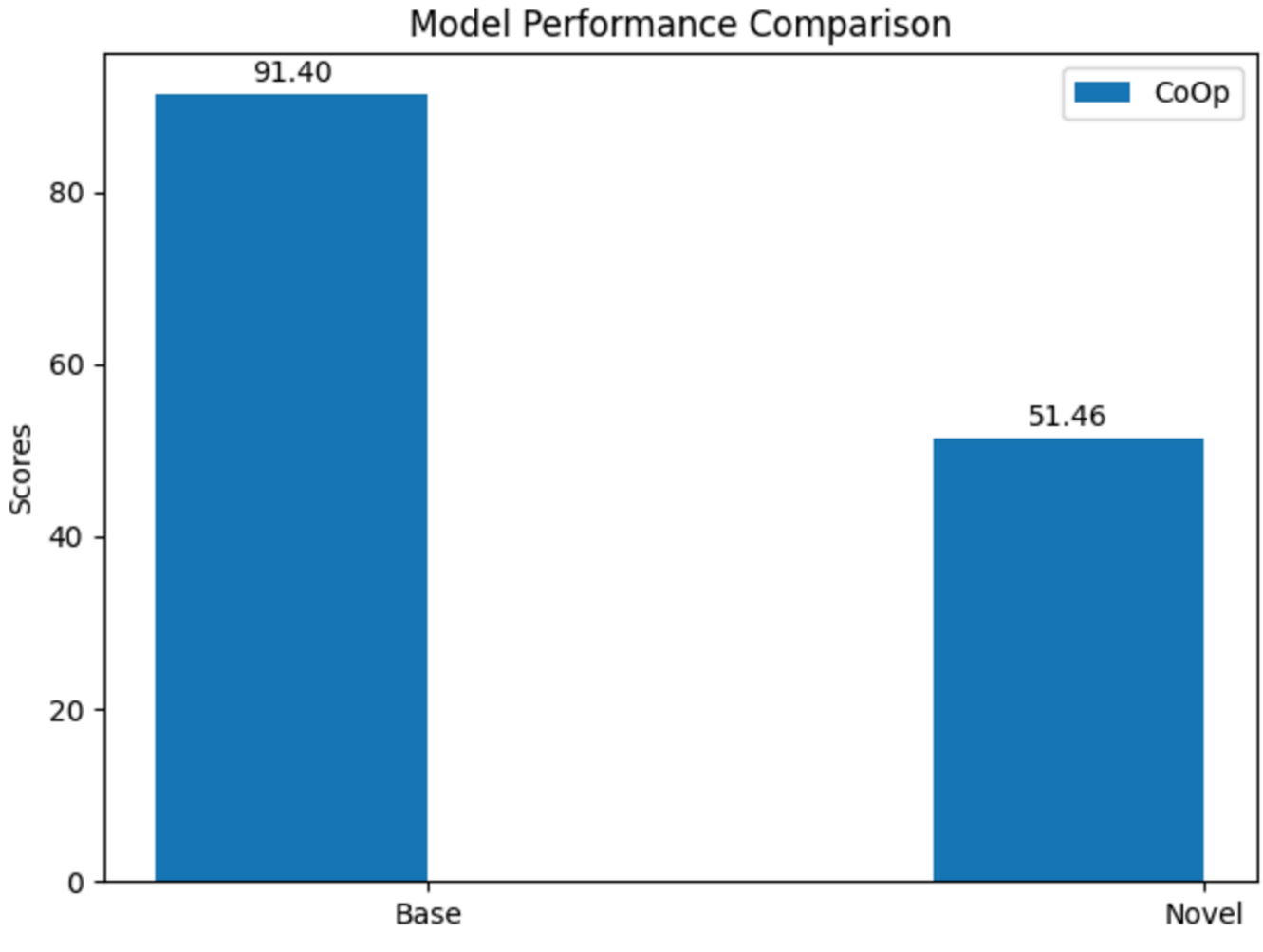
```
Loading trainer: CoCoOp
Loading dataset: EuroSAT
Reading split from /content/ProMetaR/data/eurosat/split_zhou_EuroSAT.json
Loading preprocessed few-shot data from /content/ProMetaR/data/eurosat/split_fewshot/shot_16-seed_1.pkl
SUBSAMPLE NEW CLASSES!
Building transform_train
+ random resized crop (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
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is
  warnings.warn(
/content/ProMetaR/dassl/utils/torchtools.py:102: FutureWarning: You are using `torch.load` with `weights_only=
  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.linear1.bias', '
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 [01:03<00:00,  1.62s/it]=> result
* total: 3,900
* correct: 1,687
* accuracy: 43.3%
* error: 56.7%
* macro_f1: 39.0%
```

```
import matplotlib.pyplot as plt
import numpy as np

metrics = ['Base', 'Novel']

coop_acc_list = [cocoop_base_acc, cocoop_novel_acc]

bar_width = 0.35
index = np.arange(len(metrics))
fig, ax = plt.subplots()
bar1 = ax.bar(index, coop_acc_list, bar_width, label='CoCoOp')

ax.set_ylabel('Scores')
ax.set_title('Model Performance Comparison')
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(metrics)
ax.legend()

def add_value_labels(bars):
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 2),  # 2 points vertical offset
                    textcoords='offset points',
                    ha='center', va='bottom')

add_value_labels(bar1)
plt.tight_layout()
plt.show()
```



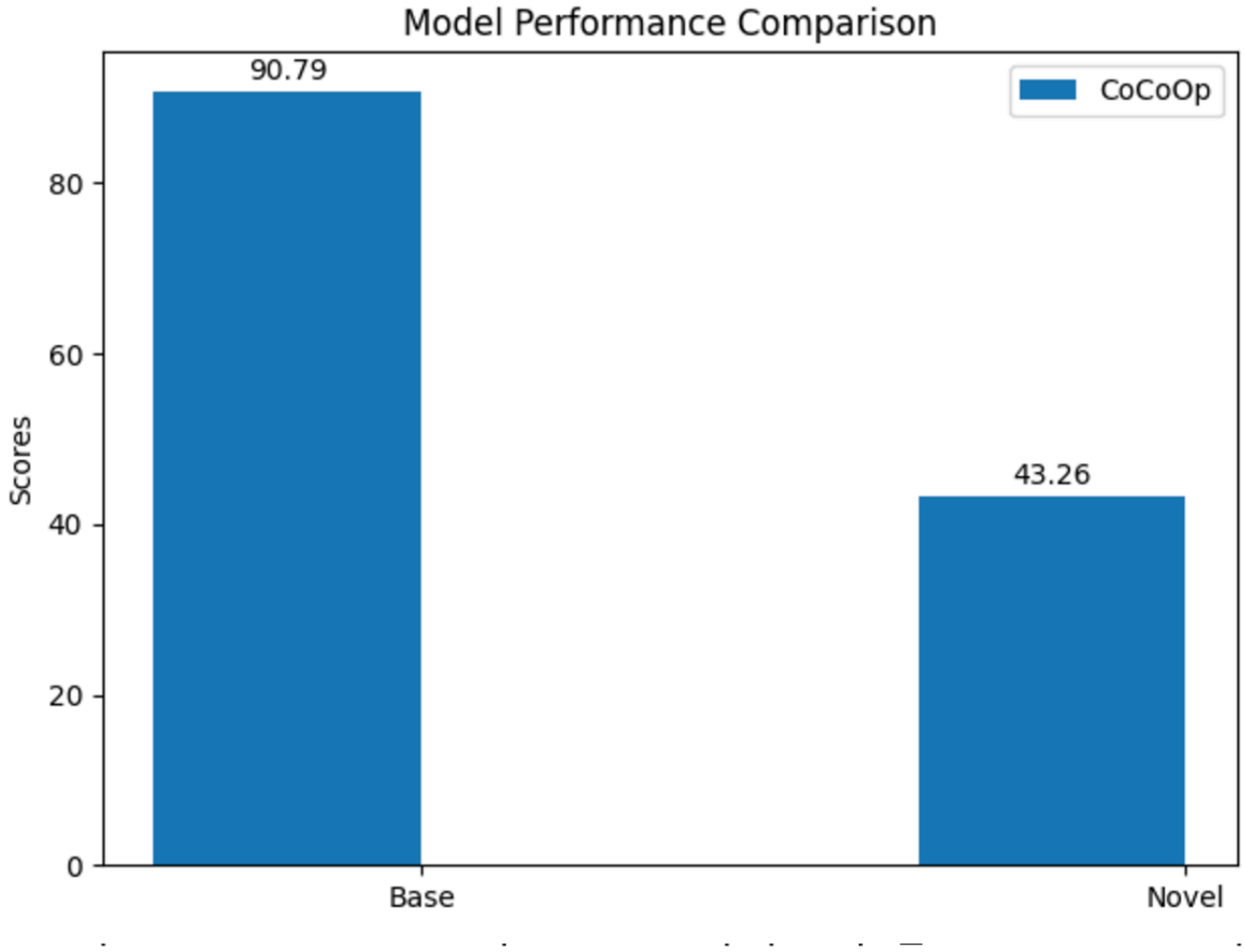

## Q3. Analyzing the results of CoCoOp

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.




```
Finish training
Deploy the last-epoch model
Evaluate on the *test* set
100%|██████████| 42/42 [00:21<00:00,  1.96it/s]=> result
* total: 4,200
* correct: 3,839
* accuracy: 91.4%
* error: 8.6%
* macro_f1: 91.5%
Elapsed: 0:03:06
```

```
Loading weights to prompt_learner from "outputs/coop/prompt_learner/model.pth.tar-100" (epoch = 100)
Evaluate on the *test* set
100%|██████████| 39/39 [00:15<00:00,  2.60it/s]=> result
* total: 3,900
* correct: 2,007
* accuracy: 51.5%
* error: 48.5%
* macro_f1: 45.6%
```




```
Finish training
Deploy the last-epoch model
Evaluate on the *test* set
100%|██████████| 42/42 [01:01<00:00,  1.47s/it]=> result
* total: 4,200
* correct: 3,813
* accuracy: 90.8%
* error: 9.2%
* macro_f1: 90.9%
Elapsed: 0:06:32
```

```
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,  1.54s/it]=> result
* total: 3,900
* correct: 1,687
* accuracy: 43.3%
```

위의 그림은 CoCoOp와 CoOp의 결과를 나타내는 그래프이다.

우선 base와 novel부터 설명해보자면, base class는 모델을 학습시킬 때 사용된 class를 의미하고, novel class는 새로운 class로, 모델이 학습 과정에서는 보지 못했던 class를 의미한다. CoOp와 CoCoOp의 성능은 모두 Base class에서 score가 Novel class에서보다 더 높았다. 이는 Novel class는 모델이 train 과정에서 보지 못한 새로운 class이기 때문이다.

evaluation 설명에서 나온 다양한 수치 중 Macro F1이란, 모델의 성능을 평가하기 위한 지표로, multi class가 포함된 dataset에서 class 별 F1 Score (precision과 recall의 조화 평균)을 계산한 뒤, 이를 평균 내는 방식이다. Macro F1이 높다는 것은, 모든 클래스에서 F1 score가 고르게 높다는 것을 의미하는데, 이는 모델이 데이터의 class간 차이를 잘 처리하고 있다는 것을 나타낸다. 전체적으로 base와 비교했을 때, novel에서 accuracy와 macro f1이 더 낮다는 것은, 모델이 학습하지 않은 것으로 구성이 되어 이렇게 나타난 것 같다.

다만, CoCoOp가 CoOp에 비해 Base class에선 점수가 비슷했지만, Novel class에서 점수가 더 낮은 이유에 집중해보는 것이 중요하다고 생각한다. CoCoOp에는 image token을 extract하고, 이를 text prompt에 더하는 meta 네트워크 부분이 존재한다. 이러한 meta network를 통해, text prompt가 이미지에 따라 동적으로 조정되며, model이 image와 text 간의 관계를 더 잘 이해할 수 있도록 한다. 아마, novel class에서 점수가 더 낮은 이유는, novel class에서 image 특징이 모델에 맞춰지지 않았기 때문으로 추론할 수 있다. 반면에, CoOp는 고정된 프롬프트를 사용했기 때문에, Novel dataset에서도 안정적으로 동작했다고 추론할 수 있다. 즉, CoCoOp는 이미지 특징에 따라 프롬프트가 동적으로 생성되지만, novel 데이터에서는 이미지 특징이 학습되지 않아 프롬프트가 적절히 조정되지 않았을 것으로 생각된다.