⌄ 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.

```
1  !git clone https://github.com/mlvlab/ProMetaR.git
2  %cd ProMetaR/
3
4  !git clone https://github.com/KaiyangZhou/Dassl.pytorch.git
5  %cd Dassl.pytorch/
6
7  # Install dependencies
8  !pip install -r requirements.txt
9  !cp -r dassl ../
10 # Install this library (no need to re-build if the source code is modified)
11 # !python setup.py develop
12 %cd ..
13
14 !pip install -r requirements.txt
15
16 %mkdir outputs
17 %mkdir data
18
19 %cd data
20 %mkdir eurosat
21 !wget http://madm.dfki.de/files/sentinel/EuroSAT.zip -O EuroSAT.zip
22
23 !unzip -o EuroSAT.zip -d eurosat/
```

```python
24 %cd eurosat
25 !gdown 1Ip7yaCWFi0eaOFUGga0lUdVi_DDQth1o
26
27 %cd ../../
28
29 import os.path as osp
30 from collections import OrderedDict
31 import math
32 import torch
33 import torch.nn as nn
34 from torch.nn import functional as F
35 from torch.cuda.amp import GradScaler, autocast
36 from PIL import Image
37 import torchvision.transforms as transforms
38 import torch
39 from clip import clip
40 from clip.simple_tokenizer import SimpleTokenizer as _Tokenizer
41 import time
42 from tqdm import tqdm
43 import datetime
44 import argparse
45 from dassl.utils import setup_logger, set_random_seed, collect_env_info
46 from dassl.config import get_cfg_default
47 from dassl.engine import build_trainer
48 from dassl.engine import TRAINER_REGISTRY, TrainerX
49 from dassl.metrics import compute_accuracy
50 from dassl.utils import load_pretrained_weights, load_checkpoint
51 from dassl.optim import build_optimizer, build_lr_scheduler
52
53 # custom
54 import datasets.oxford_pets
55 import datasets.oxford_flowers
56 import datasets.fgvc_aircraft
57 import datasets.dtd
58 import datasets.eurosat
59 import datasets.stanford_cars
60 import datasets.food101
61 import datasets.sun397
62 import datasets.caltech101
63 import datasets.ucf101
64 import datasets.imagenet
65 import datasets.imagenet_sketch
66 import datasets.imagenetv2
67 import datasets.imagenet_a
68 import datasets.imagenet_r
69
70 def print_args(args, cfg):
71     print("***************")
```

```
72        print("** Arguments **")
73        print("**************")
74        optkeys = list(args.__dict__.keys())
75        optkeys.sort()
76        for key in optkeys:
77            print("{}: {}".format(key, args.__dict__[key]))
78        print("************")
79        print("** Config **")
80        print("************")
81        print(cfg)
82
83  def reset_cfg(cfg, args):
84        if args.root:
85            cfg.DATASET.ROOT = args.root
86        if args.output_dir:
87            cfg.OUTPUT_DIR = args.output_dir
88        if args.seed:
89            cfg.SEED = args.seed
90        if args.trainer:
91            cfg.TRAINER.NAME = args.trainer
92        cfg.DATASET.NUM_SHOTS = 16
93        cfg.DATASET.SUBSAMPLE_CLASSES = args.subsample_classes
94        cfg.DATALOADER.TRAIN_X.BATCH_SIZE = args.train_batch_size
95        cfg.OPTIM.MAX_EPOCH = args.epoch
96
97  def extend_cfg(cfg):
98        """
99        Add new config variables.
100       """
101       from yacs.config import CfgNode as CN
102       cfg.TRAINER.COOP = CN()
103       cfg.TRAINER.COOP.N_CTX = 16  # number of context vectors
104       cfg.TRAINER.COOP.CSC = False  # class-specific context
105       cfg.TRAINER.COOP.CTX_INIT = ""  # initialization words
106       cfg.TRAINER.COOP.PREC = "fp16"  # fp16, fp32, amp
107       cfg.TRAINER.COOP.CLASS_TOKEN_POSITION = "end"  # 'middle' or 'end' or '1
108       cfg.TRAINER.COCOOP = CN()
109       cfg.TRAINER.COCOOP.N_CTX = 4  # number of context vectors
110       cfg.TRAINER.COCOOP.CTX_INIT = "a photo of a"  # initialization words
111       cfg.TRAINER.COCOOP.PREC = "fp16"  # fp16, fp32, amp
112       cfg.TRAINER.PROMETAR = CN()
113       cfg.TRAINER.PROMETAR.N_CTX_VISION = 4  # number of context vectors at th
114       cfg.TRAINER.PROMETAR.N_CTX_TEXT = 4  # number of context vectors at the
115       cfg.TRAINER.PROMETAR.CTX_INIT = "a photo of a"  # initialization words
116       cfg.TRAINER.PROMETAR.PREC = "fp16"  # fp16, fp32, amp
117       cfg.TRAINER.PROMETAR.PROMPT_DEPTH_VISION = 9  # Max 12, minimum 0, for 0
118       cfg.TRAINER.PROMETAR.PROMPT_DEPTH_TEXT = 9  # Max 12, minimum 0, for 0 i
119       cfg.DATASET.SUBSAMPLE_CLASSES = "all"  # all, base or new
```

```
120        cfg.TRAINER.PROMETAR.ADAPT_LR = 0.0005
121        cfg.TRAINER.PROMETAR.LR_RATIO = 0.0005
122        cfg.TRAINER.PROMETAR.FAST_ADAPTATION = False
123        cfg.TRAINER.PROMETAR.MIXUP_ALPHA = 0.5
124        cfg.TRAINER.PROMETAR.MIXUP_BETA = 0.5
125        cfg.TRAINER.PROMETAR.DIM_RATE=8
126        cfg.OPTIM_VNET = CN()
127        cfg.OPTIM_VNET.NAME = "adam"
128        cfg.OPTIM_VNET.LR = 0.0003
129        cfg.OPTIM_VNET.WEIGHT_DECAY = 5e-4
130        cfg.OPTIM_VNET.MOMENTUM = 0.9
131        cfg.OPTIM_VNET.SGD_DAMPNING = 0
132        cfg.OPTIM_VNET.SGD_NESTEROV = False
133        cfg.OPTIM_VNET.RMSPROP_ALPHA = 0.99
134        cfg.OPTIM_VNET.ADAM_BETA1 = 0.9
135        cfg.OPTIM_VNET.ADAM_BETA2 = 0.999
136        cfg.OPTIM_VNET.STAGED_LR = False
137        cfg.OPTIM_VNET.NEW_LAYERS = ()
138        cfg.OPTIM_VNET.BASE_LR_MULT = 0.1
139        # Learning rate scheduler
140        cfg.OPTIM_VNET.LR_SCHEDULER = "single_step"
141        # -1 or 0 means the stepsize is equal to max_epoch
142        cfg.OPTIM_VNET.STEPSIZE = (-1, )
143        cfg.OPTIM_VNET.GAMMA = 0.1
144        cfg.OPTIM_VNET.MAX_EPOCH = 10
145        # Set WARMUP_EPOCH larger than 0 to activate warmup training
146        cfg.OPTIM_VNET.WARMUP_EPOCH = -1
147        # Either linear or constant
148        cfg.OPTIM_VNET.WARMUP_TYPE = "linear"
149        # Constant learning rate when type=constant
150        cfg.OPTIM_VNET.WARMUP_CONS_LR = 1e-5
151        # Minimum learning rate when type=linear
152        cfg.OPTIM_VNET.WARMUP_MIN_LR = 1e-5
153        # Recount epoch for the next scheduler (last_epoch=-1)
154        # Otherwise last_epoch=warmup_epoch
155        cfg.OPTIM_VNET.WARMUP_RECOUNT = True
156
157 def setup_cfg(args):
158        cfg = get_cfg_default()
159        extend_cfg(cfg)
160        # 1. From the dataset config file
161        if args.dataset_config_file:
162            cfg.merge_from_file(args.dataset_config_file)
163        # 2. From the method config file
164        if args.config_file:
165            cfg.merge_from_file(args.config_file)
166        # 3. From input arguments
167        reset_cfg(cfg, args)
```

```python
168        cfg.freeze()
169        return cfg
170
171 _tokenizer = _Tokenizer()
172
173 def load_clip_to_cpu(cfg): # Load CLIP
174     backbone_name = cfg.MODEL.BACKBONE.NAME
175     url = clip._MODELS[backbone_name]
176     model_path = clip._download(url)
177
178     try:
179         # loading JIT archive
180         model = torch.jit.load(model_path, map_location="cpu").eval()
181         state_dict = None
182
183     except RuntimeError:
184         state_dict = torch.load(model_path, map_location="cpu")
185
186     if cfg.TRAINER.NAME == "":
187       design_trainer = "CoOp"
188     else:
189       design_trainer = cfg.TRAINER.NAME
190     design_details = {"trainer": design_trainer,
191                       "vision_depth": 0,
192                       "language_depth": 0, "vision_ctx": 0,
193                       "language_ctx": 0}
194     model = clip.build_model(state_dict or model.state_dict(), design_detail
195
196     return model
197
198 from dassl.config import get_cfg_default
199 cfg = get_cfg_default()
200 cfg.MODEL.BACKBONE.NAME = "ViT-B/16" # Set the vision encoder backbone of Cl
201 clip_model = load_clip_to_cpu(cfg)
202
203
204
205 class TextEncoder(nn.Module):
206     def __init__(self, clip_model): # 초기화 하는 함수
207         super().__init__()
208         self.transformer = clip_model.transformer
209         self.positional_embedding = clip_model.positional_embedding
210         self.ln_final = clip_model.ln_final
211         self.text_projection = clip_model.text_projection
212         self.dtype = clip_model.dtype
213
214     def forward(self, prompts, tokenized_prompts): # 모델 호출
215         x = prompts + self.positional_embedding.type(self.dtype)
```

```
216            x = x.permute(1, 0, 2)  # NLD -> LND
217            x = self.transformer(x)
218            x = x.permute(1, 0, 2)  # LND -> NLD
219            x = self.ln_final(x).type(self.dtype)
220
221            # x.shape = [batch_size, n_ctx, transformer.width]
222            # take features from the eot embedding (eot_token is the highest num
223            x = x[torch.arange(x.shape[0]), tokenized_prompts.argmax(dim=-1)] @
224
225            return x
226
227
228 @TRAINER_REGISTRY.register(force=True)
229 class CoCoOp(TrainerX):
230     def check_cfg(self, cfg):
231         assert cfg.TRAINER.COCOOP.PREC in ["fp16", "fp32", "amp"]
232
233     def build_model(self):
234         cfg = self.cfg
235         classnames = self.dm.dataset.classnames
236         print(f"Loading CLIP (backbone: {cfg.MODEL.BACKBONE.NAME})")
237         clip_model = load_clip_to_cpu(cfg)
238
239         if cfg.TRAINER.COCOOP.PREC == "fp32" or cfg.TRAINER.COCOOP.PREC == "
240             # CLIP's default precision is fp16
241             clip_model.float()
242
243         print("Building custom CLIP")
244         self.model = CoCoOpCustomCLIP(cfg, classnames, clip_model)
245
246         print("Turning off gradients in both the image and the text encoder"
247         name_to_update = "prompt_learner"
248
249         for name, param in self.model.named_parameters():
250             if name_to_update not in name:
251                 param.requires_grad_(False)
252
253         # Double check
254         enabled = set()
255         for name, param in self.model.named_parameters():
256             if param.requires_grad:
257                 enabled.add(name)
258         print(f"Parameters to be updated: {enabled}")
259
260         if cfg.MODEL.INIT_WEIGHTS:
261             load_pretrained_weights(self.model.prompt_learner, cfg.MODEL.INI
262
263         self.model.to(self.device)
```

```python
264            # NOTE: only give prompt_learner to the optimizer
265            self.optim = build_optimizer(self.model.prompt_learner, cfg.OPTIM)
266            self.sched = build_lr_scheduler(self.optim, cfg.OPTIM)
267            self.register_model("prompt_learner", self.model.prompt_learner, sel
268
269            self.scaler = GradScaler() if cfg.TRAINER.COCOOP.PREC == "amp" else
270
271            # Note that multi-gpu training could be slow because CLIP's size is
272            # big, which slows down the copy operation in DataParallel
273            device_count = torch.cuda.device_count()
274            if device_count > 1:
275                print(f"Multiple GPUs detected (n_gpus={device_count}), use all
276                self.model = nn.DataParallel(self.model)
277
278    def before_train(self):
279        directory = self.cfg.OUTPUT_DIR
280        if self.cfg.RESUME:
281            directory = self.cfg.RESUME
282        self.start_epoch = self.resume_model_if_exist(directory)
283
284        # Remember the starting time (for computing the elapsed time)
285        self.time_start = time.time()
286
287
288    def forward_backward(self, batch):
289        image, label = self.parse_batch_train(batch)
290
291        model = self.model
292        optim = self.optim
293        scaler = self.scaler
294
295        prec = self.cfg.TRAINER.COCOOP.PREC
296        loss = model(image, label) # Input image 모델 통과
297        optim.zero_grad()
298        loss.backward() # Backward (역전파)
299        optim.step() # 모델 parameter update
300
301        loss_summary = {"loss": loss.item()}
302
303        if (self.batch_idx + 1) == self.num_batches:
304            self.update_lr()
305
306        return loss_summary
307
308    def parse_batch_train(self, batch):
309        input = batch["img"]
310        label = batch["label"]
311        input = input.to(self.device)
```

```python
312                 label = label.to(self.device)
313             return input, label
314
315     def load_model(self, directory, epoch=None):
316         if not directory:
317             print("Note that load_model() is skipped as no pretrained model
318             return
319
320         names = self.get_model_names()
321
322         # By default, the best model is loaded
323         model_file = "model-best.pth.tar"
324
325         if epoch is not None:
326             model_file = "model.pth.tar-" + str(epoch)
327
328         for name in names:
329             model_path = osp.join(directory, name, model_file)
330
331             if not osp.exists(model_path):
332                 raise FileNotFoundError('Model not found at "{}"'.format(mod
333
334             checkpoint = load_checkpoint(model_path)
335             state_dict = checkpoint["state_dict"]
336             epoch = checkpoint["epoch"]
337
338             # Ignore fixed token vectors
339             if "token_prefix" in state_dict:
340                 del state_dict["token_prefix"]
341
342             if "token_suffix" in state_dict:
343                 del state_dict["token_suffix"]
344
345             print("Loading weights to {} " 'from "{}" (epoch = {})'.format(n
346             # set strict=False
347             self._models[name].load_state_dict(state_dict, strict=False)
348
349     def after_train(self):
350         print("Finish training")
351
352         do_test = not self.cfg.TEST.NO_TEST
353         if do_test:
354             if self.cfg.TEST.FINAL_MODEL == "best_val":
355                 print("Deploy the model with the best val performance")
356                 self.load_model(self.output_dir)
357             else:
358                 print("Deploy the last-epoch model")
359             acc = self.test()
```

```python
360
361        # Show elapsed time
362        elapsed = round(time.time() - self.time_start)
363        elapsed = str(datetime.timedelta(seconds=elapsed))
364        print(f"Elapsed: {elapsed}")
365
366        # Close writer
367        self.close_writer()
368        return acc
369
370    def train(self):
371        """Generic training loops."""
372        self.before_train()
373        for self.epoch in range(self.start_epoch, self.max_epoch):
374            self.before_epoch()
375            self.run_epoch()
376            self.after_epoch()
377        acc = self.after_train()
378        return acc
379
380 parser = argparse.ArgumentParser()
381 parser.add_argument("--root", type=str, default="data/", help="path to datas
382 parser.add_argument("--output-dir", type=str, default="outputs/cocoop3", hel
383 parser.add_argument(
384     "--seed", type=int, default=1, help="only positive value enables a fixed
385 )
386 parser.add_argument(
387     "--config-file", type=str, default="configs/trainers/ProMetaR/vit_b16_c2
388 )
389 parser.add_argument(
390     "--dataset-config-file",
391     type=str,
392     default="configs/datasets/eurosat.yaml",
393     help="path to config file for dataset setup",
394 )
395 parser.add_argument("--trainer", type=str, default="CoOp", help="name of tra
396 parser.add_argument("--eval-only", action="store_true", help="evaluation onl
397 parser.add_argument(
398     "--model-dir",
399     type=str,
400     default="",
401     help="load model from this directory for eval-only mode",
402 )
403 parser.add_argument("--train-batch-size", type=int, default=4)
404 parser.add_argument("--epoch", type=int, default=10)
405 parser.add_argument("--subsample-classes", type=str, default="base")
406 parser.add_argument(
407     "--load-epoch", type=int, default=0, help="load model weights at this ep
```

```
408 )
409 args = parser.parse_args([])
410
411 def main(args):
412     cfg = setup_cfg(args)
413     if cfg.SEED >= 0:
414         set_random_seed(cfg.SEED)
415
416     if torch.cuda.is_available() and cfg.USE_CUDA:
417         torch.backends.cudnn.benchmark = True
418
419     trainer = build_trainer(cfg)
420     if args.eval_only:
421         trainer.load_model(args.model_dir, epoch=args.load_epoch)
422         acc = trainer.test()
423         return acc
424
425     acc = trainer.train()
426     return acc
```

```
    inflating: eurosat/2750/PermanentCrop/PermanentCrop_350.jpg
    inflating: eurosat/2750/PermanentCrop/PermanentCrop_885.jpg
    inflating: eurosat/2750/PermanentCrop/PermanentCrop_2378.jpg
    inflating: eurosat/2750/PermanentCrop/PermanentCrop_6.jpg
    inflating: eurosat/2750/PermanentCrop/PermanentCrop_731.jpg
    inflating: eurosat/2750/PermanentCrop/PermanentCrop_62.jpg
    inflating: eurosat/2750/PermanentCrop/PermanentCrop_1728.jpg
    inflating: eurosat/2750/PermanentCrop/PermanentCrop_274.jpg
    inflating: eurosat/2750/PermanentCrop/PermanentCrop_1349.jpg
    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
```

⌄ 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.

```
1  import torch.nn as nn
2
3  class CoCoOpPromptLearner(nn.Module):
4      def __init__(self, cfg, classnames, clip_model):
5          super().__init__()
6          n_cls = len(classnames)
7          n_ctx = cfg.TRAINER.COCOOP.N_CTX
8          ctx_init = cfg.TRAINER.COCOOP.CTX_INIT
9          dtype = clip_model.dtype
10         ctx_dim = clip_model.ln_final.weight.shape[0]
```

```python
11          vis_dim = clip_model.visual.output_dim
12          clip_imsize = clip_model.visual.input_resolution
13          cfg_imsize = cfg.INPUT.SIZE[0]
14          assert cfg_imsize == clip_imsize, f"cfg_imsize ({cfg_imsize}) must e
15
16          if ctx_init:
17              # use given words to initialize context vectors
18              ctx_init = ctx_init.replace("_", " ")
19              n_ctx = len(ctx_init.split(" "))
20              prompt = clip.tokenize(ctx_init)
21              with torch.no_grad():
22                  embedding = clip_model.token_embedding(prompt).type(dtype)
23              ctx_vectors = embedding[0, 1: 1 + n_ctx, :]
24              prompt_prefix = ctx_init
25          else:
26              # random initialization
27              ctx_vectors = torch.empty(n_ctx, ctx_dim, dtype=dtype)
28              nn.init.normal_(ctx_vectors, std=0.02)
29              prompt_prefix = " ".join(["X"] * n_ctx)
30
31          print(f'Initial context: "{prompt_prefix}"')
32          print(f"Number of context words (tokens): {n_ctx}")
33
34          self.ctx = nn.Parameter(ctx_vectors)  # Wrap the initialized prompts
35
36          ### Tokenize ###
37          classnames = [name.replace("_", " ") for name in classnames]  # 예)
38          name_lens = [len(_tokenizer.encode(name)) for name in classnames]
39          prompts = [prompt_prefix + " " + name + "." for name in classnames]
40
41          tokenized_prompts = torch.cat([clip.tokenize(p) for p in prompts]) #
42
43
44
45          #######################################
46          ####### Q1. Fill in the blank #######
47          ########## Define Meta Net ##########
48          self.meta_net = nn.Sequential(OrderedDict([
49              ("linear1", nn.Linear(vis_dim, vis_dim // 16)),
50              ("relu", nn.ReLU(inplace=True)),
51              ("linear2", nn.Linear(vis_dim // 16, ctx_dim))
52          ]))
53          #######################################
54          ## Hint: meta network is composed to linear layer, relu activation,
55
56
57
58          if cfg.TRAINER.COCOOP.PREC == "fp16":
```

```python
59                  self.meta_net.half()
60
61          with torch.no_grad():
62              embedding = clip_model.token_embedding(tokenized_prompts).type(d
63
64          # These token vectors will be saved when in save_model(),
65          # but they should be ignored in load_model() as we want to use
66          # those computed using the current class names
67          self.register_buffer("token_prefix", embedding[:, :1, :])  # SOS
68          self.register_buffer("token_suffix", embedding[:, 1 + n_ctx:, :])  #
69          self.n_cls = n_cls
70          self.n_ctx = n_ctx
71          self.tokenized_prompts = tokenized_prompts  # torch.Tensor
72          self.name_lens = name_lens
73
74      def construct_prompts(self, ctx, prefix, suffix, label=None):
75          # dim0 is either batch_size (during training) or n_cls (during testi
76          # ctx: context tokens, with shape of (dim0, n_ctx, ctx_dim)
77          # prefix: the sos token, with shape of (n_cls, 1, ctx_dim)
78          # suffix: remaining tokens, with shape of (n_cls, *, ctx_dim)
79
80          if label is not None:
81              prefix = prefix[label]
82              suffix = suffix[label]
83
84          prompts = torch.cat(
85              [
86                  prefix,  # (dim0, 1, dim)
87                  ctx,  # (dim0, n_ctx, dim)
88                  suffix,  # (dim0, *, dim)
89              ],
90              dim=1,
91          )
92
93          return prompts
94
95      def forward(self, im_features):
96          prefix = self.token_prefix
97          suffix = self.token_suffix
98          ctx = self.ctx  # (n_ctx, ctx_dim)
99
100
101
102          #############################################
103          ########## Q2,3. Fill in the blank #########
104          bias = self.meta_net(im_features)  # (batch, ctx_dim)
105          bias = bias.unsqueeze(1)  # (batch, 1, ctx_dim)
106          ctx = ctx.unsqueeze(0)  # (1, n_ctx, ctx_dim)
```

```python
107        ctx_shifted = ctx + bias  # (batch, n_ctx, ctx_dim)
108        ##############################################
109        ##############################################
110
111
112
113        # Use instance-conditioned context tokens for all classes
114        prompts = []
115        for ctx_shifted_i in ctx_shifted:
116            ctx_i = ctx_shifted_i.unsqueeze(0).expand(self.n_cls, -1, -1)
117            pts_i = self.construct_prompts(ctx_i, prefix, suffix)  # (n_cls,
118            prompts.append(pts_i)
119        prompts = torch.stack(prompts)
120
121        return prompts
```

```python
1  class CoCoOpCustomCLIP(nn.Module):
2      def __init__(self, cfg, classnames, clip_model):
3          super().__init__()
4          self.prompt_learner = CoCoOpPromptLearner(cfg, classnames, clip_model
5          self.tokenized_prompts = self.prompt_learner.tokenized_prompts
6          self.image_encoder = clip_model.visual
7          self.text_encoder = TextEncoder(clip_model)
8          self.logit_scale = clip_model.logit_scale
9          self.dtype = clip_model.dtype
10
11     def forward(self, image, label=None):
12         tokenized_prompts = self.tokenized_prompts
13         logit_scale = self.logit_scale.exp()
14
15         image_features = self.image_encoder(image.type(self.dtype))
16         image_features = image_features / image_features.norm(dim=-1, keepdim
17
18
19         ###############################################
20         ########## Q4. Fill in the blank #########
21         prompts = self.prompt_learner(image_features)
22         ###############################################
23         ###############################################
24
25
26         logits = []
27         for pts_i, imf_i in zip(prompts, image_features):
28             text_features = self.text_encoder(pts_i, tokenized_prompts)
29             text_features = text_features / text_features.norm(dim=-1, keepdi
30             l_i = logit_scale * imf_i @ text_features.t()
31             logits.append(l_i)
32         logits = torch.stack(logits)
33
34         if self.prompt_learner.training:
35             return F.cross_entropy(logits, label)
36
37         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.

```python
1  # Train on the Base Classes Train split and evaluate accuracy on the Base Cla
```

```
2 args.trainer = "CoCoOp"
3 args.train_batch_size = 4
4 args.epoch = 100
5 args.output_dir = "outputs/cocoop"
6
7 args.subsample_classes = "base"
8 args.eval_only = False
9 cocoop_base_acc = main(args)
```

```
epoch [53/100] batch [20/20] time 0.118 (0.145) data 0.000 (0.020) loss 0
epoch [54/100] batch [20/20] time 0.137 (0.199) data 0.000 (0.038) loss 0
epoch [55/100] batch [20/20] time 0.107 (0.127) data 0.000 (0.017) loss 0
epoch [56/100] batch [20/20] time 0.091 (0.126) data 0.000 (0.029) loss 0
epoch [57/100] batch [20/20] time 0.095 (0.128) data 0.000 (0.020) loss 0
epoch [58/100] batch [20/20] time 0.128 (0.156) data 0.000 (0.022) loss 0
epoch [59/100] batch [20/20] time 0.096 (0.188) data 0.000 (0.037) loss 0
epoch [60/100] batch [20/20] time 0.093 (0.129) data 0.000 (0.019) loss 0
epoch [61/100] batch [20/20] time 0.106 (0.128) data 0.000 (0.018) loss 0
epoch [62/100] batch [20/20] time 0.092 (0.126) data 0.000 (0.021) loss 0
epoch [63/100] batch [20/20] time 0.141 (0.174) data 0.000 (0.022) loss 0
epoch [64/100] batch [20/20] time 0.106 (0.128) data 0.000 (0.025) loss 0
epoch [65/100] batch [20/20] time 0.093 (0.129) data 0.000 (0.021) loss 0
epoch [66/100] batch [20/20] time 0.095 (0.128) data 0.000 (0.026) loss 0
epoch [67/100] batch [20/20] time 0.150 (0.151) data 0.000 (0.021) loss 0
epoch [68/100] batch [20/20] time 0.136 (0.204) data 0.000 (0.047) loss 0
epoch [69/100] batch [20/20] time 0.096 (0.130) data 0.000 (0.021) loss 0
epoch [70/100] batch [20/20] time 0.092 (0.126) data 0.000 (0.024) loss 0
epoch [71/100] batch [20/20] time 0.094 (0.130) data 0.000 (0.027) loss 0
epoch [72/100] batch [20/20] time 0.137 (0.150) data 0.000 (0.021) loss 0
epoch [73/100] batch [20/20] time 0.135 (0.195) data 0.000 (0.035) loss 0
epoch [74/100] batch [20/20] time 0.095 (0.131) data 0.000 (0.024) loss 0
epoch [75/100] batch [20/20] time 0.096 (0.129) data 0.000 (0.019) loss 0
epoch [76/100] batch [20/20] time 0.095 (0.128) data 0.000 (0.018) loss 0
epoch [77/100] batch [20/20] time 0.149 (0.159) data 0.000 (0.023) loss 0
epoch [78/100] batch [20/20] time 0.169 (0.201) data 0.000 (0.040) loss 0
epoch [79/100] batch [20/20] time 0.091 (0.129) data 0.000 (0.019) loss 0
epoch [80/100] batch [20/20] time 0.091 (0.126) data 0.000 (0.019) loss 0
epoch [81/100] batch [20/20] time 0.091 (0.131) data 0.000 (0.027) loss 0
epoch [82/100] batch [20/20] time 0.144 (0.155) data 0.000 (0.024) loss 0
epoch [83/100] batch [20/20] time 0.109 (0.169) data 0.000 (0.034) loss 0
epoch [84/100] batch [20/20] time 0.094 (0.129) data 0.000 (0.025) loss 0
epoch [85/100] batch [20/20] time 0.093 (0.129) data 0.000 (0.020) loss 0
epoch [86/100] batch [20/20] time 0.095 (0.129) data 0.000 (0.024) loss 0
epoch [87/100] batch [20/20] time 0.161 (0.190) data 0.000 (0.042) loss 0
epoch [88/100] batch [20/20] time 0.093 (0.128) data 0.000 (0.026) loss 0
epoch [89/100] batch [20/20] time 0.094 (0.129) data 0.000 (0.021) loss 0
epoch [90/100] batch [20/20] time 0.091 (0.128) data 0.000 (0.020) loss 0
epoch [91/100] batch [20/20] time 0.117 (0.144) data 0.000 (0.018) loss 0
epoch [92/100] batch [20/20] time 0.170 (0.201) data 0.000 (0.038) loss 0
epoch [93/100] batch [20/20] time 0.093 (0.127) data 0.000 (0.022) loss 0
epoch [94/100] batch [20/20] time 0.091 (0.126) data 0.000 (0.020) loss 0
epoch [95/100] batch [20/20] time 0.093 (0.128) data 0.000 (0.019) loss 0
```

```
epoch [96/100] batch [20/20] time 0.126 (0.152) data 0.000 (0.018) loss 0
epoch [97/100] batch [20/20] time 0.138 (0.198) data 0.000 (0.038) loss 0
epoch [98/100] batch [20/20] time 0.092 (0.126) data 0.000 (0.021) loss 0
epoch [99/100] batch [20/20] time 0.096 (0.129) data 0.000 (0.022) loss 0
epoch [100/100] batch [20/20] time 0.108 (0.131) data 0.000 (0.019) loss
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 [01:04<00:00,  1.53s/it]=> result
* total: 4,200
* correct: 3,813
* accuracy: 90.8%
* error: 9.2%
* macro_f1: 90.9%
```

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

```
Loading trainer: CoCoOp
Loading dataset: EuroSAT
Reading split from /content/ProMetaR/ProMetaR/data/eurosat/split_zhou_EuroS
Loading preprocessed few-shot data from /content/ProMetaR/ProMetaR/data/eur
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,
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,
---------  -------
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:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: Use
  warnings.warn(
/content/ProMetaR/dassl/utils/torchtools.py:102: FutureWarning: You are usi
  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.l
Loading evaluator: Classification
Loading weights to prompt_learner from "outputs/cocoop/prompt_learner/model
Evaluate on the *test* set
100%|██████████| 39/39 [00:58<00:00,  1.51s/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.

원래의 CoCoOp 논문과는 달리 주어진 코드에서는 Seen의 test세트와 Novel test세트 모두에서 CoOp의 성능이 높게 나타난다. 사실 이는, CoOp이 적어도 Novel에서는 성능이 높게 나타날 것이라고 예상했던 것과는 다른 결과였다. Seen에서는 CoCoOp이 90.8, CoOp이 91.4의 성능을 보였으며, Novel에서는 CoCoOp이 43.3, CoOp이 51.5의 정확도를 보이며 모두 CoOp의 성능이 높게 나타났다. 원인을 유추해보자면, CoCoOp에서는 CoOp에 비하여 적은 context token 개수를 사용한다. CoCoOp에서는 Meta-net을 통하여 input-conditional token이 생성되기는 하지만, 이러한 context token 개수 차이가 성능 차이를 야기했을 수 있다. 또한, CoOp은 클래스 별로 고정된 context vector를 생성하는 반면, CoCoOp은 이미지에 따라 context vector가 변화한다. 따라서, 이에 따른 파라미터 수의 증가와 학습 시간 증가라는 문제를 갖게 되는데, 데이터셋이 충분하지 않은 경우 파라미터 수가 더 클 때 overfitting 문제를 겪을 수 있다. 더하여, CoCoOp과 CoOp의 트레이닝 loss가 fluctuation이 상당히 크게 나타난다. 두 모델 모두 약 30epoch대에서 가장 작은 training loss를 보인 후 fluctuation이 심한 것으로 보아 learning rate 조정, 데이터 정규화, 배치 사이즈 조정 등이 필요할 것으로 보인다.