

✓ 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/
24 %cd eurosat
25 !gdown 1lp7yaCWFf0ea0FUGga0IUdVi_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 'front'
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 the vision branch
114    cfg.TRAINER.PROMETAR.N_CTX_TEXT = 4 # number of context vectors at the language branch
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 it will be using s
118    cfg.TRAINER.PROMETAR.PROMPT_DEPTH_TEXT = 9 # Max 12, minimum 0, for 0 it will be using sha
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)
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_details)
```

```

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 CLIP to ViT.
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 number in each sequence)
223         x = x[torch.arange(x.shape[0]), tokenized_prompts.argmax(dim=-1)] @ self.text_projection
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 == "amp":
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.INIT_WEIGHTS)
262
263     self.model.to(self.device)
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, self.optim, self.sched)
268
269     self.scaler = GradScaler() if cfg.TRAINER.COCOOP.PREC == "amp" else None
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 of them!")
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     def forward_backward(self, batch):
288         image, label = self.parse_batch_train(batch)
289
290         model = self.model
291         optim = self.optim
292         scaler = self.scaler
293
294         prec = self.cfg.TRAINER.COCOOP.PREC
295         loss = model(image, label) # Input image 모델 통과
296         optim.zero_grad()
297         loss.backward() # Backward (역전파)
298         optim.step() # 모델 parameter update
299
300         loss_summary = {"loss": loss.item()}
301
302         if (self.batch_idx + 1) == self.num_batches:
303             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)
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 is given")
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(model_path))
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(name, model_path, epoch))
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()

```

```

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 dataset")
382 parser.add_argument("--output-dir", type=str, default="outputs/cocoop3", help="output directory")
383 parser.add_argument(
384     "--seed", type=int, default=1, help="only positive value enables a fixed seed"
385 )
386 parser.add_argument(
387     "--config-file", type=str, default="configs/trainers/ProMetaR/vit_b16_c2_ep10_batch4_4+4ct",
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 trainer")
396 parser.add_argument("--eval-only", action="store_true", help="evaluation only")
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 epoch for evaluation"
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_1380.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1855.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1427.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2295.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_968.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1046.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2153.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2182.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1730.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1097.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_2244.jpg
inflating: eurosat/2750/PermanentCrop/PermanentCrop_1351.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]
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 equal to clip_imsize
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 above as parameter
35
36     ### Tokenize ###
37     classnames = [name.replace("_", " ") for name in classnames] # 예) "Forest"
38     name_lens = [len(_tokenizer.encode(name)) for name in classnames]
39     prompts = [prompt_prefix + " " + name + "." for name in classnames] # 예) "A photo of F
40
41     tokenized_prompts = torch.cat([clip.tokenize(p) for p in prompts]) # 예) [49406, 320, 1
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, and linear layer.
55
56
57
58     if cfg.TRAINER.COCOOP.PREC == "fp16":
59         self.meta_net.half()
60
61     with torch.no_grad():
62         embedding = clip_model.token_embedding(tokenized_prompts).type(dtype)
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:, :]) # CLS, EOS
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 testing)
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)
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, n_tkn, ctx_dim)
118         prompts.append(pts_i)
119     prompts = torch.stack(prompts)
120
121     return prompts

```



```

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=True)
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, keepdim=True)
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. Training CoCoOp

In this task, you will train CoCoOp on the EuroSAT dataset. If your implementation of CoCoOp in Question 1 is correct, the following code should execute without errors. Please submit the execution file so we can evaluate whether your code runs without any issues.

```

1 # Train on the Base Classes Train split and evaluate accuracy on the Base Classes Test split.
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)

```



```

Loading trainer: CoCoOp
Loading dataset: EuroSAT
Reading split from /content/ProMetaR/data/eurosat/split_zhou_EuroSAT.json
Creating a 16-shot dataset
Creating a 4-shot dataset
Saving preprocessed few-shot data to /content/ProMetaR/data/eurosat/split_fewshot/shot_16-s
SUBSAMPLE BASE CLASSES!
Building transform_train
+ random resized crop (size=(224, 224), scale=(0.08, 1.0))
+ random flip
+ to torch tensor of range [0, 1]
+ normalization (mean=[0.48145466, 0.4578275, 0.40821073], std=[0.26862954, 0.26130258, 0.2
Building transform_test
+ resize the smaller edge to 224
+ 224x224 center crop
+ to torch tensor of range [0, 1]
+ normalization (mean=[0.48145466, 0.4578275, 0.40821073], std=[0.26862954, 0.26130258, 0.2

```

```

Dataset      EuroSAT
# classes    5
# train_x    80
# val        20
# test       4,200

```

Loading CLIP (backbone: ViT-B/16)

/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:617: UserWarning: Th
warnings.warn(

Building custom CLIP

Initial context: "a photo of a"

Number of context words (tokens): 4

Turning off gradients in both the image and the text encoder

Parameters to be updated: {'prompt_learner.meta_net.linear1.bias', 'prompt_learner.meta_net

Loading evaluator: Classification

No checkpoint found, train from scratch

/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The ve
warnings.warn(

```

epoch [1/100] batch [20/20] time 0.102 (0.250) data 0.000 (0.028) loss 0.2744 (1.1881) lr 2
epoch [2/100] batch [20/20] time 0.090 (0.124) data 0.000 (0.017) loss 0.8384 (0.8970) lr 2
epoch [3/100] batch [20/20] time 0.136 (0.183) data 0.000 (0.034) loss 0.6382 (0.7859) lr 2
epoch [4/100] batch [20/20] time 0.103 (0.124) data 0.000 (0.017) loss 0.5044 (0.7151) lr 2
epoch [5/100] batch [20/20] time 0.090 (0.123) data 0.000 (0.017) loss 0.5703 (0.6317) lr 2
epoch [6/100] batch [20/20] time 0.090 (0.121) data 0.000 (0.020) loss 0.6060 (0.6009) lr 2
epoch [7/100] batch [20/20] time 0.127 (0.129) data 0.000 (0.020) loss 0.3853 (0.6638) lr 2
epoch [8/100] batch [20/20] time 0.133 (0.191) data 0.000 (0.034) loss 1.4082 (0.6633) lr 2
epoch [9/100] batch [20/20] time 0.091 (0.123) data 0.000 (0.017) loss 0.1780 (0.4582) lr 2
epoch [10/100] batch [20/20] time 0.101 (0.123) data 0.000 (0.020) loss 1.2285 (0.5051) lr
epoch [11/100] batch [20/20] time 0.090 (0.124) data 0.000 (0.020) loss 0.2539 (0.5013) lr
epoch [12/100] batch [20/20] time 0.120 (0.143) data 0.000 (0.024) loss 1.1484 (0.4657) lr
epoch [13/100] batch [20/20] time 0.133 (0.189) data 0.000 (0.037) loss 0.8467 (0.5009) lr
epoch [14/100] batch [20/20] time 0.091 (0.124) data 0.000 (0.019) loss 0.5547 (0.4495) lr
epoch [15/100] batch [20/20] time 0.100 (0.127) data 0.000 (0.019) loss 1.0430 (0.5549) lr
epoch [16/100] batch [20/20] time 0.092 (0.124) data 0.000 (0.018) loss 1.3906 (0.4799) lr
epoch [17/100] batch [20/20] time 0.122 (0.133) data 0.000 (0.016) loss 0.0238 (0.3497) lr
epoch [18/100] batch [20/20] time 0.131 (0.222) data 0.000 (0.032) loss 0.1337 (0.2804) lr
epoch [19/100] batch [20/20] time 0.092 (0.125) data 0.000 (0.020) loss 1.0420 (0.3864) lr
epoch [20/100] batch [20/20] time 0.097 (0.123) data 0.000 (0.016) loss 0.3484 (0.4984) lr
epoch [21/100] batch [20/20] time 0.091 (0.123) data 0.000 (0.017) loss 0.8184 (0.3434) lr

```

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/data/eurosat/split_zhou_EuroSAT.json

Loading preprocessed few-shot data from /content/ProMetaR/data/eurosat/split_fewshot/shot_16-
SUBSAMPLE NEW CLASSES!

Building transform_train

+ random resized crop (size=(224, 224), scale=(0.08, 1.0))

+ random flip

+ to torch tensor of range [0, 1]

```

+ normalization (mean=[0.48145466, 0.4578275, 0.40821073], std=[0.26862954, 0.26130258, 0.275
Building transform_test
+ resize the smaller edge to 224
+ 224x224 center crop
+ to torch tensor of range [0, 1]
+ normalization (mean=[0.48145466, 0.4578275, 0.40821073], std=[0.26862954, 0.26130258, 0.275
-----
Dataset      EuroSAT
# classes    5
# train_x    80
# val        20
# test       3,900
-----

Loading CLIP (backbone: ViT-B/16)
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:617: UserWarning: This
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verb
warnings.warn(
/content/ProMetaR/dassl/utils/torchtools.py:102: FutureWarning: You are using `torch.load` wi
checkpoint = torch.load(fpath, map_location=map_location)
Building custom CLIP
Initial context: "a photo of a"
Number of context words (tokens): 4
Turning off gradients in both the image and the text encoder
Parameters to be updated: {'prompt_learner.meta_net.linear1.bias', 'prompt_learner.meta_net.l
Loading evaluator: Classification
Loading weights to prompt_learner from "outputs/cocoop/prompt_learner/model.pth.tar-100" (epo
Evaluate on the *test* set
100%|██████████| 39/39 [00:59<00:00, 1.53s/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.

For the base classes, COOP showed an accuracy of 91.4%, while CoCoOP achieved 90.8%. Although there is a slight difference, both demonstrated high performance. For the new classes, COOP achieved 51.3%, whereas CoCoOP reached 43.3%. Contrary to general expectations, COOP performed better in this case, which is believed to be due to the difference in the number of context tokens. In the COOP experiment, 16 tokens were used, whereas CoCoOP used only 4.

Discussion : In the experiments with COOP and CoCoOP, the performance difference was smaller than expected given the difference in the number of tokens. This can be attributed to CoCoOP's use of a meta-network, which is designed to better learn inter-class differences,

making it more advantageous for generalization. However, in terms of time efficiency, there is a trade-off, and depending on the type of dataset, COOP may show better performance. Therefore, selecting the appropriate model for the task is essential.