

✓ Preparation

```

!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 EuroSAT.zip

!unzip -o EuroSAT.zip -d eurosat/
%cd eurosat
!gdown 1Ip7yaCWF0ea0FUGga0LUdVi_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.TRAINER.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.COOCOOP = CN()
    cfg.TRAINER.COOCOOP.N_CTX = 4 # number of context vectors
    cfg.TRAINER.COOCOOP.CTX_INIT = "a photo of a" # initialization words
    cfg.TRAINER.COOCOOP.PREC = "fp16" # fp16, fp32, amp
    cfg.TRAINER.PROMETAR = CN()
    cfg.TRAINER.PROMETAR.N_CTX_VISION = 4 # number of context vectors at the vision branch
    cfg.TRAINER.PROMETAR.N_CTX_TEXT = 4 # number of context vectors at the language branch
    cfg.TRAINER.PROMETAR.CTX_INIT = "a photo of a" # initialization words
    cfg.TRAINER.PROMETAR.PREC = "fp16" # fp16, fp32, amp
    cfg.TRAINER.PROMETAR.PROMPT_DEPTH_VISION = 9 # Max 12, minimum 0, for 0 it will be using shallow IVLP prompting (J=1)
    cfg.TRAINER.PROMETAR.PROMPT_DEPTH_TEXT = 9 # Max 12, minimum 0, for 0 it will be using shallow IVLP prompting (J=1)
    cfg.DATASET.SUBSAMPLE_CLASSES = "all" # all, base or new
    cfg.TRAINER.PROMETAR.ADAPT_LR = 0.0005
    cfg.TRAINER.PROMETAR.LR_RATIO = 0.0005
    cfg.TRAINER.PROMETAR.FAST_ADAPTATION = False
    cfg.TRAINER.PROMETAR.MIXUP_ALPHA = 0.5
    cfg.TRAINER.PROMETAR.MIXUP_BETA = 0.5
    cfg.TRAINER.PROMETAR.DIM_RATE=8
    cfg.OPTIM_VNET = CN()
    cfg.OPTIM_VNET.NAME = "adam"
    cfg.OPTIM_VNET.LR = 0.0003
    cfg.OPTIM_VNET.WEIGHT_DECAY = 5e-4
    cfg.OPTIM_VNET.MOMENTUM = 0.9
    cfg.OPTIM_VNET.SGD_DAMPNING = 0
    cfg.OPTIM_VNET.SGD_NESTEROV = False
    cfg.OPTIM_VNET.RMSPROP_ALPHA = 0.99
    cfg.OPTIM_VNET.ADAM_BETA1 = 0.9
    cfg.OPTIM_VNET.ADAM_BETA2 = 0.999
    cfg.OPTIM_VNET.STAGED_LR = False
    cfg.OPTIM_VNET.NEW_LAYERS = ()
    cfg.OPTIM_VNET.BASE_LR_MULT = 0.1
    # Learning rate scheduler
    cfg.OPTIM_VNET.LR_SCHEDULER = "single_step"
    # -1 or 0 means the stepsize is equal to max_epoch
    cfg.OPTIM_VNET.STEPSIZE = (-1, )
    cfg.OPTIM_VNET.GAMMA = 0.1
    cfg.OPTIM_VNET.MAX_EPOCH = 10
    # Set WARMUP_EPOCH larger than 0 to activate warmup training
    cfg.OPTIM_VNET.WARMUP_EPOCH = -1
    # Either linear or constant
    cfg.OPTIM_VNET.WARMUP_TYPE = "linear"
    # Constant learning rate when type=constant
    cfg.OPTIM_VNET.WARMUP_CONS_LR = 1e-5
    # Minimum learning rate when type=linear
    cfg.OPTIM_VNET.WARMUP_MIN_LR = 1e-5
    # Recount epoch for the next scheduler (last_epoch=-1)
    # Otherwise last_epoch=warmup_epoch
    cfg.OPTIM_VNET.WARMUP_RECOUNT = True

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

_tokenizer = _Tokenizer()

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

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

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

    if cfg.TRAINER.NAME == "":
        design_trainer = "CoOp"
    else:
        design_trainer = cfg.TRAINER.NAME
    design_details = {"trainer": design_trainer,
                      "vision_depth": 0,
                      "language_depth": 0, "vision_ctx": 0,

```

```

        language_ctx": 0, 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

```

```

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

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

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

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

return loss_summary

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

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

    names = self.get_model_names()

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

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

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

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

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

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

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

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

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

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

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

    # Close writer
    self.close_writer()
    return acc

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

parser = argparse.ArgumentParser()
parser.add_argument("--root", type=str, default="data/", help="path to dataset")
parser.add_argument("--output-dir", type=str, default="outputs/cocoop3", help="output directory")
parser.add_argument(
    "--seed", type=int, default=1, help="only positive value enables a fixed seed"
)
parser.add_argument(
    "--config-file", type=str, default="configs/trainers/ProMetaR/vit_b16_c2_ep10_batch4_4+4ctx.yaml", help="path to config file"
)
parser.add_argument(
    "-----"

```

```
    "--dataset-config-file",
    type=str,
    default="configs/datasets/eurosat.yaml",
    help="path to config file for dataset setup",
)
parser.add_argument("--trainer", type=str, default="CoOp", help="name of trainer")
parser.add_argument("--eval-only", action="store_true", help="evaluation only")
parser.add_argument(
    "--model-dir",
    type=str,
    default="",
    help="load model from this directory for eval-only mode",
)
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
```



[illegible]

- Use another architecture for CoCoOp's Meta Net - residual

Key changes made to the meta network:

Added a deeper architecture with three layers:

First layer: Linear(vis_dim \rightarrow hidden_dim) + ReLU Second layer: Linear(hidden_dim \rightarrow hidden_dim) + ReLU Third layer: Linear(hidden_dim \rightarrow ctx_dim)

Added residual connection:

Created a residual projection layer to match dimensions Added the residual connection by summing the meta network output with the projected input

The residual connection helps with:

Better gradient flow Maintaining original feature information Potentially improved training stability

```

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.COOCOOP.N_CTX
        ctx_init = cfg.TRAINER.COOCOOP.CTX_INIT
        dtype = clip_model.dtype
        ctx_dim = clip_model.ln_final.weight.shape[0]
        vis_dim = clip_model.visual.output_dim
        clip_imsize = clip_model.visual.input_resolution
        cfg_imsize = cfg.INPUT.SIZE[0]
        assert cfg_imsize == clip_imsize, f"cfg_imsize ({cfg_imsize}) must equal to clip_imsize ({clip_imsize})"

        if ctx_init:
            # use given words to initialize context vectors
            ctx_init = ctx_init.replace("_", " ")
            n_ctx = len(ctx_init.split(" "))
            prompt = clip.tokenize(ctx_init)
            with torch.no_grad():
                embedding = clip_model.token_embedding(prompt).type(dtype)
            ctx_vectors = embedding[0, 1: 1 + n_ctx, :]
            prompt_prefix = ctx_init
        else:
            # random initialization
            ctx_vectors = torch.empty(n_ctx, ctx_dim, dtype=dtype)
            nn.init.normal_(ctx_vectors, std=0.02)
            prompt_prefix = " ".join(["X"] * n_ctx)

        print(f'Initial context: "{prompt_prefix}"')
        print(f"Number of context words (tokens): {n_ctx}")

        self.ctx = nn.Parameter(ctx_vectors)

        # Tokenize
        classnames = [name.replace("_", " ") for name in classnames]
        name_lens = [len(_tokenizer.encode(name)) for name in classnames]
        prompts = [prompt_prefix + " " + name + "." for name in classnames]
        tokenized_prompts = torch.cat([clip.tokenize(p) for p in prompts])

        # Define Meta Net with Residual Connections
        hidden_dim = vis_dim // 16
        self.meta_net = nn.Sequential(OrderedDict([
            ("layer1", nn.Sequential(
                nn.Linear(vis_dim, hidden_dim),
                nn.ReLU(inplace=True),
            )),
            ("layer2", nn.Sequential(
                nn.Linear(hidden_dim, hidden_dim),
                nn.ReLU(inplace=True),
            )),
            ("layer3", nn.Linear(hidden_dim, ctx_dim))
        ]))

        # Residual projection for input to match output dimension
        self.residual_proj = nn.Linear(vis_dim, ctx_dim)

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

        with torch.no_grad():
            embedding = clip_model.token_embedding(tokenized_prompts).type(dtype)

        self.register_buffer("token_prefix", embedding[:, :1, :]) # SOS
        self.register_buffer("token_suffix", embedding[:, 1 + n_ctx:, :]) # CLS, EOS
        self.n_cls = n_cls

```

```

self.n_ctx = n_ctx
self.tokenized_prompts = tokenized_prompts
self.name_lens = name_lens

def construct_prompts(self, ctx, prefix, suffix, label=None):
    if label is not None:
        prefix = prefix[label]
        suffix = suffix[label]

    prompts = torch.cat(
        [
            prefix,
            ctx,
            suffix,
        ],
        dim=1,
    )
    return prompts

def forward(self, im_features):
    prefix = self.token_prefix
    suffix = self.token_suffix
    ctx = self.ctx

    # Forward pass through meta network with residual connection
    meta_output = self.meta_net(im_features)
    residual = self.residual_proj(im_features)
    bias = meta_output + residual # Add residual connection

    bias = bias.unsqueeze(1)
    ctx = ctx.unsqueeze(0)
    ctx_shifted = ctx + bias

    # Use instance-conditioned context tokens for all classes
    prompts = []
    for ctx_shifted_i in ctx_shifted:
        ctx_i = ctx_shifted_i.unsqueeze(0).expand(self.n_cls, -1, -1)
        pts_i = self.construct_prompts(ctx_i, prefix, suffix)
        prompts.append(pts_i)
    prompts = torch.stack(prompts)

    return prompts

class CoCoOpCustomCLIP(nn.Module):
    def __init__(self, cfg, classnames, clip_model):
        super().__init__()
        self.prompt_learner = CoCoOpPromptLearner(cfg, classnames, clip_model)
        self.tokenized_prompts = self.prompt_learner.tokenized_prompts
        self.image_encoder = clip_model.visual
        self.text_encoder = TextEncoder(clip_model)
        self.logit_scale = clip_model.logit_scale
        self.dtype = clip_model.dtype

    def forward(self, image, label=None):
        tokenized_prompts = self.tokenized_prompts
        logit_scale = self.logit_scale.exp()

        image_features = self.image_encoder(image.type(self.dtype))
        image_features = image_features / image_features.norm(dim=-1, keepdim=True)

        #####
        prompts = self.prompt_learner(image_features)
        #####

        logits = []
        for pts_i, imf_i in zip(prompts, image_features):
            text_features = self.text_encoder(pts_i, tokenized_prompts)
            text_features = text_features / text_features.norm(dim=-1, keepdim=True)
            l_i = logit_scale * imf_i @ text_features.t()
            logits.append(l_i)
        logits = torch.stack(logits)

        if self.prompt_learner.training:
            return F.cross_entropy(logits, label)

        return logits

```

▼ Training

- Train modified version of CoCoOp (residualp) on the EuroSAT dataset.

```
!pwd
!ls
```



/content/ProMetaR

clip	dassl	datasets	meta_learning	README.md	trainers
clip_words.csv	Dassl.pytorch	docs	outputs	requirements.txt	train.py
configs	data	LICENSE	parse_test_res.py	scripts	

```
# 1 마저 실행 eurosat 디렉토리까지
```

```
# 1. 먼저 현재 디렉토리 구조를 보여
!rm -rf data/eurosat/*
```

```
# 2. 작업 디렉토리 이동
%cd data/eurosat
```

```
# 3. EuroSAT 데이터셋 다운로드 (직접적인 다운로드 URL 사용)
!wget http://madm.dfki.de/files/sentinel/EuroSAT.zip
```

```
# 4. 압축 해제
!unzip -o EuroSAT.zip
```

```
# 5. split 파일 다운로드
!gdown 1Ip7yaCWF0ea0FUGga0lUdVi_DD0th1o
```

```
# 6. 원래 디렉토리로 돌아가기
%cd ../../
```

```
# 7. 확인
!ls data/eurosat/2750
```



```
inflating: 2750/PermanentCrop/PermanentCrop_1398.jpg
inflating: 2750/PermanentCrop/PermanentCrop_163.jpg
inflating: 2750/PermanentCrop/PermanentCrop_970.jpg
inflating: 2750/PermanentCrop/PermanentCrop_502.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2472.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1567.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1915.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2013.jpg
inflating: 2750/PermanentCrop/PermanentCrop_828.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1106.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1670.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1211.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2304.jpg
inflating: 2750/PermanentCrop/PermanentCrop_273.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1088.jpg
inflating: 2750/PermanentCrop/PermanentCrop_612.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1438.jpg
inflating: 2750/PermanentCrop/PermanentCrop_164.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1059.jpg
inflating: 2750/PermanentCrop/PermanentCrop_505.jpg
inflating: 2750/PermanentCrop/PermanentCrop_977.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2475.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1912.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1560.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2014.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1101.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1677.jpg
inflating: 2750/PermanentCrop/PermanentCrop_19.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1216.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2303.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1753.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1332.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1495.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2227.jpg
inflating: 2750/PermanentCrop/PermanentCrop_118.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1444.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1836.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2130.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1782.jpg
inflating: 2750/PermanentCrop/PermanentCrop_579.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1025.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2409.jpg
inflating: 2750/PermanentCrop/PermanentCrop_853.jpg
inflating: 2750/PermanentCrop/PermanentCrop_421.jpg
inflating: 2750/PermanentCrop/PermanentCrop_386.jpg
inflating: 2750/PermanentCrop/PermanentCrop_2068.jpg
inflating: 2750/PermanentCrop/PermanentCrop_882.jpg
inflating: 2750/PermanentCrop/PermanentCrop_357.jpg
inflating: 2750/PermanentCrop/PermanentCrop_1.jpg
inflating: 2750/PermanentCrop/PermanentCrop_65.jpg
inflating: 2750/PermanentCrop/PermanentCrop_736.jpg
```

Downloading...

From: https://drive.google.com/uc?id=1Ip7yaCWF0ea0FUGga0lUdVi_DD0th1o

To: /content/ProMetaR/data/eurosat/split_zhou_EuroSAT.json

100% 3.01M/3.01M [00:00<00:00, 225MB/s]

/content/ProMetaR

AnnualCrop	HerbaceousVegetation	Industrial	PermanentCrop	River
Forest	Highway	Pasture	Residential	SeaLake

```
# 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 [55/100] batch [20/20] time 0.098 (0.133) data 0.000 (0.017) loss 0.2678 (0.1540) lr 1.0933e-03 eta 0:01:59
epoch [56/100] batch [20/20] time 0.110 (0.136) data 0.000 (0.022) loss 0.0080 (0.1010) lr 1.0545e-03 eta 0:01:59
epoch [57/100] batch [20/20] time 0.153 (0.166) data 0.000 (0.019) loss 0.1194 (0.1408) lr 1.0158e-03 eta 0:02:22
epoch [58/100] batch [20/20] time 0.098 (0.145) data 0.000 (0.036) loss 0.3020 (0.2257) lr 9.7732e-04 eta 0:02:01
epoch [59/100] batch [20/20] time 0.099 (0.132) data 0.000 (0.019) loss 0.2274 (0.2011) lr 9.3914e-04 eta 0:01:48
epoch [60/100] batch [20/20] time 0.100 (0.135) data 0.000 (0.017) loss 0.0399 (0.1352) lr 9.0126e-04 eta 0:01:47
epoch [61/100] batch [20/20] time 0.126 (0.152) data 0.000 (0.018) loss 0.2100 (0.1550) lr 8.6373e-04 eta 0:01:58
epoch [62/100] batch [20/20] time 0.158 (0.210) data 0.000 (0.039) loss 0.0305 (0.1789) lr 8.2658e-04 eta 0:02:39
epoch [63/100] batch [20/20] time 0.105 (0.138) data 0.000 (0.021) loss 0.3735 (0.1725) lr 7.8984e-04 eta 0:01:41
epoch [64/100] batch [20/20] time 0.099 (0.144) data 0.000 (0.036) loss 0.0437 (0.1502) lr 7.5357e-04 eta 0:01:43
epoch [65/100] batch [20/20] time 0.103 (0.134) data 0.000 (0.017) loss 0.0601 (0.1843) lr 7.1778e-04 eta 0:01:34
epoch [66/100] batch [20/20] time 0.177 (0.173) data 0.000 (0.023) loss 0.2734 (0.1050) lr 6.8251e-04 eta 0:01:57
epoch [67/100] batch [20/20] time 0.100 (0.140) data 0.000 (0.021) loss 0.2366 (0.1219) lr 6.4781e-04 eta 0:01:32
epoch [68/100] batch [20/20] time 0.102 (0.136) data 0.000 (0.018) loss 0.2382 (0.1447) lr 6.1370e-04 eta 0:01:26
epoch [69/100] batch [20/20] time 0.108 (0.134) data 0.000 (0.024) loss 0.0374 (0.1116) lr 5.8022e-04 eta 0:01:23
epoch [70/100] batch [20/20] time 0.148 (0.166) data 0.000 (0.022) loss 0.1333 (0.1070) lr 5.4740e-04 eta 0:01:39
epoch [71/100] batch [20/20] time 0.099 (0.143) data 0.000 (0.030) loss 0.0980 (0.1058) lr 5.1527e-04 eta 0:01:23
epoch [72/100] batch [20/20] time 0.102 (0.136) data 0.000 (0.024) loss 0.0648 (0.1556) lr 4.8387e-04 eta 0:01:16
epoch [73/100] batch [20/20] time 0.151 (0.141) data 0.000 (0.019) loss 0.1855 (0.2329) lr 4.5322e-04 eta 0:01:16
epoch [74/100] batch [20/20] time 0.159 (0.160) data 0.000 (0.023) loss 0.2744 (0.1687) lr 4.2336e-04 eta 0:01:23
epoch [75/100] batch [20/20] time 0.099 (0.148) data 0.000 (0.037) loss 0.0623 (0.1367) lr 3.9432e-04 eta 0:01:13
epoch [76/100] batch [20/20] time 0.107 (0.137) data 0.000 (0.027) loss 0.0266 (0.1396) lr 3.6612e-04 eta 0:01:05
epoch [77/100] batch [20/20] time 0.102 (0.135) data 0.000 (0.020) loss 0.2035 (0.0984) lr 3.3879e-04 eta 0:01:02
epoch [78/100] batch [20/20] time 0.129 (0.156) data 0.000 (0.023) loss 0.0498 (0.0697) lr 3.1236e-04 eta 0:01:08
epoch [79/100] batch [20/20] time 0.159 (0.217) data 0.000 (0.046) loss 0.0063 (0.1408) lr 2.8686e-04 eta 0:01:31
epoch [80/100] batch [20/20] time 0.103 (0.139) data 0.000 (0.023) loss 0.1410 (0.1516) lr 2.6231e-04 eta 0:00:55
epoch [81/100] batch [20/20] time 0.099 (0.136) data 0.000 (0.024) loss 0.5015 (0.1617) lr 2.3873e-04 eta 0:00:51
epoch [82/100] batch [20/20] time 0.113 (0.134) data 0.000 (0.017) loss 0.1656 (0.0967) lr 2.1615e-04 eta 0:00:48
epoch [83/100] batch [20/20] time 0.149 (0.178) data 0.000 (0.029) loss 0.2308 (0.1621) lr 1.9459e-04 eta 0:01:00
epoch [84/100] batch [20/20] time 0.099 (0.147) data 0.000 (0.029) loss 0.1159 (0.1893) lr 1.7407e-04 eta 0:00:47
epoch [85/100] batch [20/20] time 0.104 (0.136) data 0.000 (0.022) loss 0.0194 (0.1052) lr 1.5462e-04 eta 0:00:40
epoch [86/100] batch [20/20] time 0.103 (0.139) data 0.000 (0.027) loss 0.3003 (0.1704) lr 1.3624e-04 eta 0:00:38
epoch [87/100] batch [20/20] time 0.141 (0.161) data 0.000 (0.023) loss 0.0374 (0.1114) lr 1.1897e-04 eta 0:00:41
epoch [88/100] batch [20/20] time 0.147 (0.209) data 0.000 (0.043) loss 0.0457 (0.0935) lr 1.0281e-04 eta 0:00:50
epoch [89/100] batch [20/20] time 0.113 (0.145) data 0.000 (0.032) loss 0.0176 (0.1118) lr 8.7779e-05 eta 0:00:31
epoch [90/100] batch [20/20] time 0.101 (0.137) data 0.000 (0.020) loss 0.0185 (0.1289) lr 7.3899e-05 eta 0:00:27
epoch [91/100] batch [20/20] time 0.102 (0.136) data 0.000 (0.020) loss 0.1214 (0.1812) lr 6.1179e-05 eta 0:00:24
epoch [92/100] batch [20/20] time 0.160 (0.179) data 0.000 (0.022) loss 0.0572 (0.1728) lr 4.9633e-05 eta 0:00:28
epoch [93/100] batch [20/20] time 0.097 (0.147) data 0.000 (0.036) loss 0.1620 (0.1319) lr 3.9271e-05 eta 0:00:20
epoch [94/100] batch [20/20] time 0.098 (0.134) data 0.000 (0.025) loss 0.0190 (0.1702) lr 3.0104e-05 eta 0:00:16
epoch [95/100] batch [20/20] time 0.117 (0.137) data 0.000 (0.020) loss 0.2971 (0.0817) lr 2.2141e-05 eta 0:00:13
epoch [96/100] batch [20/20] time 0.150 (0.159) data 0.000 (0.016) loss 0.0389 (0.0784) lr 1.5390e-05 eta 0:00:12
epoch [97/100] batch [20/20] time 0.189 (0.218) data 0.000 (0.032) loss 0.1016 (0.1338) lr 9.8566e-06 eta 0:00:13
epoch [98/100] batch [20/20] time 0.101 (0.141) data 0.000 (0.024) loss 0.6880 (0.1068) lr 5.5475e-06 eta 0:00:05
epoch [99/100] batch [20/20] time 0.101 (0.134) data 0.000 (0.022) loss 0.2554 (0.1674) lr 2.4666e-06 eta 0:00:02
epoch [100/100] batch [20/20] time 0.098 (0.136) data 0.000 (0.020) loss 0.0033 (0.2557) lr 6.1680e-07 eta 0:00:00
Checkpoint saved to outputs/cocoop/prompt_learner/model.pth.tar-100
Finish training
Deploy the last-epoch model
Evaluate on the *test* set
100%|██████████| 42/42 [01:06<00:00, 1.59s/it]=> result
* total: 4,200
* correct: 3,835
* accuracy: 91.3%
* error: 8.7%
* macro_f1: 91.3%
Elapsed: 0:06:40

```

```

# Accuracy on the New Classes.
args.model_dir = "outputs/cocoop"
args.output_dir = "outputs/cocoop/new_classes"
args.subsample_classes = "new"
args.load_epoch = 100
args.eval_only = True
coop_novel_acc = main(args)

```

```

Loading trainer: CoCoOp
Loading dataset: EuroSAT
Reading split from /content/ProMetaR/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 create
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated
warnings.warn(
/content/ProMetaR/dassl/utils/torchtools.py:102: FutureWarning: You are using `torch.load` with `weights_only=False` which
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.layer1.0.weight', 'prompt_learner.meta_net.layer2.0.weight',
Loading evaluator: Classification
Loading weights to prompt_learner from "outputs/cocoop/prompt_learner/model.pth.tar-100" (epoch = 100)
Evaluate on the *test* set
100%|██████████| 39/39 [01:02<00:00, 1.60s/it]=> result
* total: 3,900
* correct: 2,110
* accuracy: 54.1%
* error: 45.9%
* macro_f1: 47.3%
```