In [ ]:

```
from __future__ import absolute_import, division, print_function

from options import LiteMonoOptions
from trainer import Trainer


from __future__ import absolute_import, division, print_function


import time
import torch.optim as optim
from torch.utils.data import DataLoader
from tensorboardX import SummaryWriter

import json

from utils import *
from kitti_utils import *
from layers import *

import datasets
import networks
from linear_warmup_cosine_annealing_warm_restarts_weight_decay import ChainedScheduler
import albumentations as A
from albumentations.pytorch import ToTensorV2


# torch.backends.cudnn.benchmark = True

import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.cuda.amp import GradScaler, autocast
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
from pathlib import Path
import cv2 as cv
from PIL import Image
import segmentation_models_pytorch as smp
from sklearn.model_selection import train_test_split
import albumentations as A
from albumentations.pytorch import ToTensorV2
from torchmetrics.image import StructuralSimilarityIndexMeasure as SSIM
from torchmetrics.regression import MeanSquaredError as MSE
from torchmetrics.collections import MetricCollection
import gc
from torchvision.transforms import Normalize


def time_sync():
    # PyTorch-accurate time
    if torch.cuda.is_available():
        torch.cuda.synchronize()
    return time.time()

sample_tfms = [
    A.HorizontalFlip(),
    A.GaussNoise(p=0.2),
    A.OneOf([
        A.MotionBlur(p=.3),
        A.MedianBlur(blur_limit=3, p=0.3),
        A.Blur(blur_limit=3, p=0.5),
    ], p=0.3),
    A.RGBShift(),
    A.RandomBrightnessContrast(),
    A.RandomResizedCrop(384,384),
    A.ColorJitter(),
    A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.3, rotate_limit=45, p=0.5),
    A.HueSaturationValue(p=0.3),
]
train_tfms = A.Compose([
    *sample_tfms,
    A.Resize(224,224),
    A.Normalize(always_apply=True),
    ToTensorV2()
])
valid_tfms = A.Compose([
    A.Resize(224,224),
    A.Normalize(always_apply=True),
    ToTensorV2()
])


class Dataset:
    def __init__(self,df,tfms):
        self.df = df
        self.tfms=tfms
    def open_im(self,p,gray=False):
        im = cv.imread(str(p))
        im = cv.cvtColor(im,cv.COLOR_BGR2GRAY if gray else cv.COLOR_BGR2RGB)
        return im

    def __len__(self,):
        return len(self.df)

    def __getitem__(self,idx):
        s = self.df.iloc[idx,:]
        im, dp = s[0],s[1]
        im, dp = self.open_im(im), self.open_im(dp,True)
        augs = self.tfms(image=im,mask=dp)
        im, dp = augs['image'], augs['mask'] / 255.
        return im, dp.unsqueeze(0)

train_csv = Path('./nyu-depth-v2/nyu_data/data/nyu2_train.csv')
train_ims_path = Path('./nyu-depth-v2/nyu_data/data/nyu2_train')
base_path = Path('./nyu-depth-v2/nyu_data')

df = pd.read_csv(train_csv,header=None)
df[0] = df[0].map(lambda x:base_path/x)
df[1] = df[1].map(lambda x:base_path/x)
df.head()

train_df, val_df = train_test_split(df,test_size=0.1,shuffle=True)
val_df, test_df = train_test_split(val_df, test_size=0.1,shuffle=True)
train_df.reset_index(drop=True,inplace=True)
val_df.reset_index(drop=True,inplace=True)
test_df.reset_index(drop=True,inplace=True)
len(train_df),len(val_df), len(test_df)

train_ds = Dataset(train_df,train_tfms)
val_ds = Dataset(val_df,valid_tfms)
test_ds = Dataset(test_df, valid_tfms)
len(train_ds), len(val_ds), len(test_ds)

fn_loss = nn.MSELoss()

class Trainer:
    def __init__(self, options):
        self.opt = options
        self.log_path = os.path.join(self.opt.log_dir, self.opt.model_name)

        # checking height and width are multiples of 32
        assert self.opt.height % 32 == 0, "'height' must be a multiple of 32"
        assert self.opt.width % 32 == 0, "'width' must be a multiple of 32"

        self.models = {}
        self.models_pose = {}
        self.parameters_to_train = []
        self.parameters_to_train_pose = []

        self.device = torch.device("cpu" if self.opt.no_cuda else "cuda")
        self.profile = self.opt.profile

        self.num_scales = len(self.opt.scales)
        self.frame_ids = len(self.opt.frame_ids)
        self.num_pose_frames = 2 if self.opt.pose_model_input == "pairs" else self.num_input_frames

        assert self.opt.frame_ids[0] == 0, "frame_ids must start with 0"

        self.use_pose_net = not (self.opt.use_stereo and self.opt.frame_ids == [0])

        if self.opt.use_stereo:
            self.opt.frame_ids.append("s")

        self.models["encoder"] = networks.LiteMono(model=self.opt.model,
                                                   drop_path_rate=self.opt.drop_path,
                                                   width=self.opt.width, height=self.opt.height)
        self.models["encoder"].to(self.device)
        self.parameters_to_train += list(self.models["encoder"].parameters())

        self.models["depth"] = networks.DepthDecoder(self.models["encoder"].num_ch_enc,
                                                     self.opt.scales)
        self.models["depth"].to(self.device)
        self.parameters_to_train += list(self.models["depth"].parameters())

        if self.use_pose_net:
            if self.opt.pose_model_type == "separate_resnet":
                self.models_pose["pose_encoder"] = networks.ResnetEncoder(
                    self.opt.num_layers,
                    self.opt.weights_init == "pretrained",
                    num_input_images=self.num_pose_frames)

                self.models_pose["pose_encoder"].to(self.device)
                self.parameters_to_train_pose += list(self.models_pose["pose_encoder"].parameters(

                self.models_pose["pose"] = networks.PoseDecoder(
                    self.models_pose["pose_encoder"].num_ch_enc,
                    num_input_features=1,
                    num_frames_to_predict_for=2)

            elif self.opt.pose_model_type == "shared":
                self.models_pose["pose"] = networks.PoseDecoder(
                    self.models["encoder"].num_ch_enc, self.num_pose_frames)

            elif self.opt.pose_model_type == "posecnn":
                self.models_pose["pose"] = networks.PoseCNN(
                    self.num_input_frames if self.opt.pose_model_input == "all" else 2)

            self.models_pose["pose"].to(self.device)
            self.parameters_to_train_pose += list(self.models_pose["pose"].parameters())

        if self.opt.predictive_mask:
            assert self.opt.disable_automasking, \
                "When using predictive_mask, please disable automasking with --disable_automasking'

            # Our implementation of the predictive masking baseline has the the same architecture
            # as our depth decoder. We predict a separate mask for each source frame.
            self.models["predictive_mask"] = networks.DepthDecoder(
                self.models["encoder"].num_ch_enc, self.opt.scales,
                num_output_channels=(len(self.opt.frame_ids) - 1))
            self.models["predictive_mask"].to(self.device)
            self.parameters_to_train += list(self.models["predictive_mask"].parameters())

        self.model_optimizer = optim.AdamW(self.parameters_to_train, self.opt.lr[0], weight_decay=
        if self.use_pose_net:
            self.model_pose_optimizer = optim.AdamW(self.parameters_to_train_pose, self.opt.lr[3],

        self.model_lr_scheduler = ChainedScheduler(
                            self.model_optimizer,
                            T_0=int(self.opt.lr[2]),
                            T_mul=1,
                            eta_min=self.opt.lr[1],
                            last_epoch=-1,
                            max_lr=self.opt.lr[0],
                            warmup_steps=0,
                            gamma=0.9
                        )
        self.model_pose_lr_scheduler = ChainedScheduler(
            self.model_pose_optimizer,
            T_0=int(self.opt.lr[5]),
            T_mul=1,
            eta_min=self.opt.lr[4],
            last_epoch=-1,
            max_lr=self.opt.lr[3],
            warmup_steps=0,
            gamma=0.9
        )

        if self.opt.load_weights_folder is not None:
            self.load_model()

        if self.opt.mypretrain is not None:
            self.load_pretrain()


        # data
        datasets_dict = {"kitti": datasets.KITTIRAWDataset,
                         "kitti_odom": datasets.KITTIOdomDataset}
        self.dataset = datasets_dict[self.opt.dataset]

        fpath = os.path.join(par_dir, "splits", self.opt.split, "{}_files.txt")

        train_filenames = readlines(fpath.format("train"))
        val_filenames = readlines(fpath.format("val"))
        img_ext = '.png' if self.opt.png else '.jpg'

        num_train_samples = len(train_filenames)
        self.num_total_steps = num_train_samples // self.opt.batch_size * self.opt.num_epochs
        ################
        train_dataset = train_ds
        #self.dataset(
            # self.opt.data_path, train_filenames, self.opt.height, self.opt.width,
            # self.opt.frame_ids, 4, is_train=True, img_ext=img_ext)
        self.train_loader = DataLoader(
            train_dataset, self.opt.batch_size, True,
            num_workers=self.opt.num_workers, pin_memory=True, drop_last=True)
        val_dataset = val_ds
        # self.dataset(
        #     self.opt.data_path, val_filenames, self.opt.height, self.opt.width,
        #     self.opt.frame_ids, 4, is_train=False, img_ext=img_ext)
        self.val_loader = DataLoader(
            val_dataset, self.opt.batch_size, True,
            num_workers=self.opt.num_workers, pin_memory=True, drop_last=True)
        self.val_iter = iter(self.val_loader)

        self.writers = {}
        for mode in ["train", "val"]:
            self.writers[mode] = SummaryWriter(os.path.join(self.log_path, mode))

        if not self.opt.no_ssim:
            self.ssim = SSIM()
            self.ssim.to(self.device)

        self.backproject_depth = {}
        self.project_3d = {}
        for scale in self.opt.scales:
            h = self.opt.height // (2 ** scale)
            w = self.opt.width // (2 ** scale)

            self.backproject_depth[scale] = BackprojectDepth(self.opt.batch_size, h, w)
            self.backproject_depth[scale].to(self.device)

            self.project_3d[scale] = Project3D(self.opt.batch_size, h, w)
            self.project_3d[scale].to(self.device)

        self.depth_metric_names = [
            "de/abs_rel", "de/sq_rel", "de/rms", "de/log_rms", "da/a1", "da/a2", "da/a3"]

        print("There are {:d} training items and {:d} validation items\n".format(
            len(train_dataset), len(val_dataset)))

        self.save_opts()

    def set_train(self):
        """Convert all models to training mode
        """
        for m in self.models.values():
            m.train()

    def set_eval(self):
        """Convert all models to testing/evaluation mode
        """
        for m in self.models.values():
            m.eval()

    def train(self):
        """Run the entire training pipeline
        """
        self.epoch = 0
        self.step = 0
        self.start_time = time.time()
        for self.epoch in range(self.opt.num_epochs):
            self.run_epoch()
            if (self.epoch + 1) % self.opt.save_frequency == 0:
                self.save_model()
    def run_epoch(self):
        """Run a single epoch of training and validation
        """

        print("Training")
        self.set_train()

        self.model_lr_scheduler.step()
        if self.use_pose_net:
            self.model_pose_lr_scheduler.step()

        for batch_idx, inputs in enumerate(self.train_loader):

            before_op_time = time.time()

            outputs, losses = self.process_batch(inputs)

            # self.model_optimizer.zero_grad()
            # if self.use_pose_net:
            #     self.model_pose_optimizer.zero_grad()
            losses.backward()
            self.model_optimizer.step()
            # if self.use_pose_net:
            #     self.model_pose_optimizer.step()

            duration = time.time() - before_op_time

            # log less frequently after the first 2000 steps to save time & disk space
            early_phase = batch_idx % self.opt.log_frequency == 0 and self.step < 20000
            late_phase = self.step % 2000 == 0

            if early_phase or late_phase:
                self.log_time(batch_idx, duration, losses.cpu().data)

                # if "depth_gt" in inputs:
                #     self.compute_depth_losses(inputs, outputs, losses)

                # self.log("train", inputs, outputs, losses)
                self.val()

            self.step += 1

    def process_batch(self, inputs):
        """Pass a minibatch through the network and generate images and losses
        """
        losses = 0.
        # print(inputs[0].shape, inputs[1].shape, len(inputs))
        for key, ipt in enumerate(inputs):
            inputs[key] = ipt.to(self.device)

#         if self.opt.pose_model_type == "shared":
#             # If we are using a shared encoder for both depth and pose (as advocated
#             # in monodepthv1), then all images are fed separately through the depth encoder.
#             all_color_aug = torch.cat([inputs[("color_aug", i, 0)] for i in self.opt.frame_ids])
#             all_features = self.models["encoder"](all_color_aug)
#             all_features = [torch.split(f, self.opt.batch_size) for f in all_features]

#             features = {}
#             for i, k in enumerate(self.opt.frame_ids):
#                 features[k] = [f[i] for f in all_features]

#             outputs = self.models["depth"](features[0])
#         else:
#             # Otherwise, we only feed the image with frame_id 0 through the depth encoder

        features = self.models["encoder"](inputs[0])#["color_aug", 0, 0]

        outputs = self.models["depth"](features)

        # if self.opt.predictive_mask:
        #     outputs["predictive_mask"] = self.models["predictive_mask"](features)

        # if self.use_pose_net:
        #     outputs.update(self.predict_poses(inputs, features))

        # self.generate_images_pred(inputs[0], outputs)
        # print(outputs.keys())
        # print(list(outputs.values())[-1].shape)
        # print(list(outputs.values())[-1])
        # losses = self.compute_losses(inputs[1], list(outputs.values())[-1])
        gts = inputs[1]
        prds = list(outputs.values())[-1]
        for prd, gt in zip(gts, prds):
            loss = fn_loss(gts, prds)
            losses += loss
        # losses = nn.MSELoss(inputs[1], list(outputs.values())[-1])

        return outputs, losses

    def predict_poses(self, inputs, features):
        """Predict poses between input frames for monocular sequences.
        """
        outputs = {}
        if self.num_pose_frames == 2:
            # In this setting, we compute the pose to each source frame via a
            # separate forward pass through the pose network.

            # select what features the pose network takes as input
            if self.opt.pose_model_type == "shared":
                pose_feats = {f_i: features[f_i] for f_i in self.opt.frame_ids}
            else:
                pose_feats = {f_i: inputs["color_aug", f_i, 0] for f_i in self.opt.frame_ids}

            for f_i in self.opt.frame_ids[1:]:
                if f_i != "s":
                    # To maintain ordering we always pass frames in temporal order
                    if f_i < 0:
                        pose_inputs = [pose_feats[f_i], pose_feats[0]]
                    else:
                        pose_inputs = [pose_feats[0], pose_feats[f_i]]

                    if self.opt.pose_model_type == "separate_resnet":
                        pose_inputs = [self.models_pose["pose_encoder"](torch.cat(pose_inputs, 1))]
                    elif self.opt.pose_model_type == "posecnn":
                        pose_inputs = torch.cat(pose_inputs, 1)

                    axisangle, translation = self.models_pose["pose"](pose_inputs)
                    outputs[("axisangle", 0, f_i)] = axisangle
                    outputs[("translation", 0, f_i)] = translation

                    # Invert the matrix if the frame id is negative
                    outputs[("cam_T_cam", 0, f_i)] = transformation_from_parameters(
                        axisangle[:, 0], translation[:, 0], invert=(f_i < 0))

        else:
            # Here we input all frames to the pose net (and predict all poses) together
            if self.opt.pose_model_type in ["separate_resnet", "posecnn"]:
                pose_inputs = torch.cat(
                    [inputs[("color_aug", i, 0)] for i in self.opt.frame_ids if i != "s"], 1)

                if self.opt.pose_model_type == "separate_resnet":
                    pose_inputs = [self.models["pose_encoder"](pose_inputs)]

            elif self.opt.pose_model_type == "shared":
                pose_inputs = [features[i] for i in self.opt.frame_ids if i != "s"]

            axisangle, translation = self.models_pose["pose"](pose_inputs)

            for i, f_i in enumerate(self.opt.frame_ids[1:]):
                if f_i != "s":
                    outputs[("axisangle", 0, f_i)] = axisangle
                    outputs[("translation", 0, f_i)] = translation
                    outputs[("cam_T_cam", 0, f_i)] = transformation_from_parameters(
                        axisangle[:, i], translation[:, i])

        return outputs

    def val(self):
        """Validate the model on a single minibatch
        """
        self.set_eval()
        try:
            inputs = self.val_iter.next()
        except StopIteration:
            self.val_iter = iter(self.val_loader)
            inputs = self.val_iter.next()

        with torch.no_grad():
            outputs, losses = self.process_batch(inputs)
            if "depth_gt" in inputs:
                self.compute_depth_losses(inputs, outputs, losses)

            # self.log("val", inputs, outputs, losses)
            del inputs, outputs, losses

        self.set_train()

    def generate_images_pred(self, inputs, outputs):
        """Generate the warped (reprojected) color images for a minibatch.
        Generated images are saved into the `outputs` dictionary.
        """
        for scale in self.opt.scales:
            disp = outputs[("disp", scale)]
            if self.opt.v1_multiscale:
                source_scale = scale
            else:
                disp = F.interpolate(
                    disp, [self.opt.height, self.opt.width], mode="bilinear", align_corners=False)
                source_scale = 0

            _, depth = disp_to_depth(disp, self.opt.min_depth, self.opt.max_depth)

            outputs[("depth", 0, scale)] = depth

            for i, frame_id in enumerate(self.opt.frame_ids[1:]):

                if frame_id == "s":
                    T = inputs["stereo_T"]
                else:
                    T = outputs[("cam_T_cam", 0, frame_id)]

                # from the authors of https://arxiv.org/abs/1712.00175
                if self.opt.pose_model_type == "posecnn":

                    axisangle = outputs[("axisangle", 0, frame_id)]
                    translation = outputs[("translation", 0, frame_id)]

                    inv_depth = 1 / depth
                    mean_inv_depth = inv_depth.mean(3, True).mean(2, True)

                    T = transformation_from_parameters(
                        axisangle[:, 0], translation[:, 0] * mean_inv_depth[:, 0], frame_id < 0)

                cam_points = self.backproject_depth[source_scale](
                    depth, inputs[("inv_K", source_scale)])
                pix_coords = self.project_3d[source_scale](
                    cam_points, inputs[("K", source_scale)], T)

                outputs[("sample", frame_id, scale)] = pix_coords

                outputs[("color", frame_id, scale)] = F.grid_sample(
                    inputs[("color", frame_id, source_scale)],
                    outputs[("sample", frame_id, scale)],
                    padding_mode="border", align_corners=True)

                if not self.opt.disable_automasking:
                    outputs[("color_identity", frame_id, scale)] = \
                        inputs[("color", frame_id, source_scale)]

    def compute_reprojection_loss(self, pred, target):
        """Computes reprojection loss between a batch of predicted and target images
        """
        abs_diff = torch.abs(target - pred)
        l1_loss = abs_diff.mean(1, True)

        if self.opt.no_ssim:
            reprojection_loss = l1_loss
        else:
            ssim_loss = self.ssim(pred, target).mean(1, True)
            reprojection_loss = 0.85 * ssim_loss + 0.15 * l1_loss

        return reprojection_loss

    def compute_losses(self, inputs, outputs):
        """Compute the reprojection and smoothness losses for a minibatch
        """

        losses = {}
        total_loss = 0
        for scale in self.opt.scales:
            loss = 0
            reprojection_losses = []

            if self.opt.v1_multiscale:
                source_scale = scale
            else:
                source_scale = 0

            disp = outputs[("disp", scale)]
            color = inputs[("color", 0, scale)]
            target = inputs[("color", 0, source_scale)]

            for frame_id in self.opt.frame_ids[1:]:
                pred = outputs[("color", frame_id, scale)]
                reprojection_losses.append(self.compute_reprojection_loss(pred, target))

            reprojection_losses = torch.cat(reprojection_losses, 1)

            if not self.opt.disable_automasking:
                identity_reprojection_losses = []
                for frame_id in self.opt.frame_ids[1:]:
                    pred = inputs[("color", frame_id, source_scale)]
                    identity_reprojection_losses.append(
                        self.compute_reprojection_loss(pred, target))

                identity_reprojection_losses = torch.cat(identity_reprojection_losses, 1)

                if self.opt.avg_reprojection:
                    identity_reprojection_loss = identity_reprojection_losses.mean(1, keepdim=True)
                else:
                    # save both images, and do min all at once below
                    identity_reprojection_loss = identity_reprojection_losses

            elif self.opt.predictive_mask:
                # use the predicted mask
                mask = outputs["predictive_mask"]["disp", scale]
                if not self.opt.v1_multiscale:
                    mask = F.interpolate(
                        mask, [self.opt.height, self.opt.width],
                        mode="bilinear", align_corners=False)

                reprojection_losses *= mask

                # add a loss pushing mask to 1 (using nn.BCELoss for stability)
                weighting_loss = 0.2 * nn.BCELoss()(mask, torch.ones(mask.shape).cuda())
                loss += weighting_loss.mean()

            if self.opt.avg_reprojection:
                reprojection_loss = reprojection_losses.mean(1, keepdim=True)
            else:
                reprojection_loss = reprojection_losses

            if not self.opt.disable_automasking:
                # add random numbers to break ties
                identity_reprojection_loss += torch.randn(
                    identity_reprojection_loss.shape, device=self.device) * 0.00001

                combined = torch.cat((identity_reprojection_loss, reprojection_loss), dim=1)
            else:
                combined = reprojection_loss

            if combined.shape[1] == 1:
                to_optimise = combined
            else:
                to_optimise, idxs = torch.min(combined, dim=1)

            if not self.opt.disable_automasking:
                outputs["identity_selection/{}".format(scale)] = (
                    idxs > identity_reprojection_loss.shape[1] - 1).float()

            loss += to_optimise.mean()

            mean_disp = disp.mean(2, True).mean(3, True)
            norm_disp = disp / (mean_disp + 1e-7)
            smooth_loss = get_smooth_loss(norm_disp, color)

            loss += self.opt.disparity_smoothness * smooth_loss / (2 ** scale)
            total_loss += loss
            losses["loss/{}".format(scale)] = loss

        total_loss /= self.num_scales
        losses["loss"] = total_loss
        return losses

    def compute_depth_losses(self, inputs, outputs, losses):
        """Compute depth metrics, to allow monitoring during training

        This isn't particularly accurate as it averages over the entire batch,
        so is only used to give an indication of validation performance
        """
        depth_pred = outputs[("depth", 0, 0)]
        depth_pred = torch.clamp(F.interpolate(
            depth_pred, [375, 1242], mode="bilinear", align_corners=False), 1e-3, 80)
        depth_pred = depth_pred.detach()

        depth_gt = inputs["depth_gt"]
        mask = depth_gt > 0

        # garg/eigen crop
        crop_mask = torch.zeros_like(mask)
        crop_mask[:, :, 153:371, 44:1197] = 1
        mask = mask * crop_mask

        depth_gt = depth_gt[mask]
        depth_pred = depth_pred[mask]
        depth_pred *= torch.median(depth_gt) / torch.median(depth_pred)

        depth_pred = torch.clamp(depth_pred, min=1e-3, max=80)

        depth_errors = compute_depth_errors(depth_gt, depth_pred)

        for i, metric in enumerate(self.depth_metric_names):
            losses[metric] = np.array(depth_errors[i].cpu())

    def log_time(self, batch_idx, duration, loss):
        """Print a logging statement to the terminal
        """
        samples_per_sec = self.opt.batch_size / duration
        time_sofar = time.time() - self.start_time
        training_time_left = (
            self.num_total_steps / self.step - 1.0) * time_sofar if self.step > 0 else 0
        print_string = "epoch {:>3} | lr {:.6f} |lr_p {:.6f} | batch {:>6} | examples/s: {:5.1f}" +
            " | loss: {:.5f} | time elapsed: {} | time left: {}"
        print(print_string.format(self.epoch, self.model_optimizer.state_dict()['param_groups'][0]
                                  self.model_pose_optimizer.state_dict()['param_groups'][0]['lr'],
                                  batch_idx, samples_per_sec, loss,
                                  sec_to_hm_str(time_sofar), sec_to_hm_str(training_time_left)))

    def log(self, mode, inputs, outputs, losses):
        """Write an event to the tensorboard events file
        """
        writer = self.writers[mode]
        for l, v in losses.items():
            writer.add_scalar("{}".format(l), v, self.step)

        for j in range(min(4, self.opt.batch_size)):  # write a maxmimum of four images
            for s in self.opt.scales:
                for frame_id in self.opt.frame_ids:
                    writer.add_image(
                        "color_{}_{}/{}".format(frame_id, s, j),
                        inputs[("color", frame_id, s)][j].data, self.step)
                    if s == 0 and frame_id != 0:
                        writer.add_image(
                            "color_pred_{}_{}/{}".format(frame_id, s, j),
                            outputs[("color", frame_id, s)][j].data, self.step)

                writer.add_image(
                    "disp_{}/{}".format(s, j),
                    normalize_image(outputs[("disp", s)][j]), self.step)

                if self.opt.predictive_mask:
                    for f_idx, frame_id in enumerate(self.opt.frame_ids[1:]):
                        writer.add_image(
                            "predictive_mask_{}_{}/{}".format(frame_id, s, j),
                            outputs["predictive_mask"][("disp", s)][j, f_idx][None, ...],
                            self.step)

                elif not self.opt.disable_automasking:
                    writer.add_image(
                        "automask_{}/{}".format(s, j),
                        outputs["identity_selection/{}".format(s)][j][None, ...], self.step)

    def save_opts(self):
        """Save options to disk so we know what we ran this experiment with
        """
        models_dir = os.path.join(self.log_path, "models")
        if not os.path.exists(models_dir):
            os.makedirs(models_dir)
        to_save = self.opt.__dict__.copy()

        with open(os.path.join(models_dir, 'opt.json'), 'w') as f:
            json.dump(to_save, f, indent=2)

    def save_model(self):
        """Save model weights to disk
        """
        save_folder = os.path.join(self.log_path, "models", "weights_{}".format(self.epoch))
        if not os.path.exists(save_folder):
            os.makedirs(save_folder)

        for model_name, model in self.models.items():
            save_path = os.path.join(save_folder, "{}.pth".format(model_name))
            to_save = model.state_dict()
            if model_name == 'encoder':
                # save the sizes - these are needed at prediction time
                to_save['height'] = self.opt.height
                to_save['width'] = self.opt.width
                to_save['use_stereo'] = self.opt.use_stereo
            torch.save(to_save, save_path)

        for model_name, model in self.models_pose.items():
            save_path = os.path.join(save_folder, "{}.pth".format(model_name))
            to_save = model.state_dict()
            torch.save(to_save, save_path)

        save_path = os.path.join(save_folder, "{}.pth".format("adam"))
        torch.save(self.model_optimizer.state_dict(), save_path)

        save_path = os.path.join(save_folder, "{}.pth".format("adam_pose"))
        if self.use_pose_net:
            torch.save(self.model_pose_optimizer.state_dict(), save_path)

    def load_pretrain(self):
        self.opt.mypretrain = os.path.expanduser(self.opt.mypretrain)
        path = self.opt.mypretrain
        model_dict = self.models["encoder"].state_dict()
        pretrained_dict = torch.load(path)['model']
        pretrained_dict = {k: v for k, v in pretrained_dict.items() if (k in model_dict and not k.s
        model_dict.update(pretrained_dict)
        self.models["encoder"].load_state_dict(model_dict)
        print('mypretrain loaded.')

    def load_model(self):
        """Load model(s) from disk
        """
        self.opt.load_weights_folder = os.path.expanduser(self.opt.load_weights_folder)

        assert os.path.isdir(self.opt.load_weights_folder), \
            "Cannot find folder {}".format(self.opt.load_weights_folder)
        print("loading model from folder {}".format(self.opt.load_weights_folder))

        for n in self.opt.models_to_load:
            print("Loading {} weights...".format(n))
            path = os.path.join(self.opt.load_weights_folder, "{}.pth".format(n))

            if n in ['pose_encoder', 'pose']:
                model_dict = self.models_pose[n].state_dict()
                pretrained_dict = torch.load(path)
                pretrained_dict = {k: v for k, v in pretrained_dict.items() if k in model_dict}
                model_dict.update(pretrained_dict)
                self.models_pose[n].load_state_dict(model_dict)
            else:
                model_dict = self.models[n].state_dict()
                pretrained_dict = torch.load(path)
                pretrained_dict = {k: v for k, v in pretrained_dict.items() if k in model_dict}
                model_dict.update(pretrained_dict)
                self.models[n].load_state_dict(model_dict)
        # loading adam state

        optimizer_load_path = os.path.join(self.opt.load_weights_folder, "adam.pth")
        optimizer_pose_load_path = os.path.join(self.opt.load_weights_folder, "adam_pose.pth")
        if os.path.isfile(optimizer_load_path):
            print("Loading Adam weights")
            optimizer_dict = torch.load(optimizer_load_path)
            optimizer_pose_dict = torch.load(optimizer_pose_load_path)
            self.model_optimizer.load_state_dict(optimizer_dict)
            self.model_pose_optimizer.load_state_dict(optimizer_pose_dict)
        else:
            print("Cannot find Adam weights so Adam is randomly initialized")





options = LiteMonoOptions()
opts = options.parse()
par_dir = os.getcwd()
print(par_dir)

trainer = Trainer(opts)
trainer.train()
```

```
/mnt/workspace/sunqiao/mymono
/home/pai/lib/python3.9/site-packages/torchvision/models/_utils.py:252: UserWarning: Accessing the
model URLs via the internal dictionary of the module is deprecated since 0.13 and will be removed
in 0.15. Please access them via the appropriate Weights Enum instead.
  warnings.warn(
Training model named:
   lite-mono
Models and tensorboard events files are saved to:
   ./tmp
Training is using:
   cuda
Using split:
   eigen_zhou
There are 45619 training items and 4562 validation items

Training
/mnt/workspace/sunqiao/mymono/networks/depth_encoder.py:35: UserWarning: __floordiv__ is deprecate
d, and its behavior will change in a future version of pytorch. It currently rounds toward 0 (like
the 'trunc' function NOT 'floor'). This results in incorrect rounding for negative values. To keep
the current behavior, use torch.div(a, b, rounding_mode='trunc'), or for actual floor division, us
e torch.div(a, b, rounding_mode='floor').
  dim_t = self.temperature ** (2 * (dim_t // 2) / self.hidden_dim)
```

```
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch      0 | examples/s:   1.1 | loss: 0.20615 | time e
lapsed: 00h00m04s | time left: 00h00m00s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch    250 | examples/s: 104.3 | loss: 0.12414 | time e
lapsed: 00h00m13s | time left: 07h41m26s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch    500 | examples/s: 103.4 | loss: 0.23405 | time e
lapsed: 00h00m23s | time left: 06h34m19s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch    750 | examples/s: 100.9 | loss: 0.62217 | time e
lapsed: 00h00m33s | time left: 06h11m04s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   1000 | examples/s: 106.4 | loss: 0.27663 | time e
lapsed: 00h00m43s | time left: 05h58m49s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   1250 | examples/s:  99.6 | loss: 0.23921 | time e
lapsed: 00h00m53s | time left: 05h51m50s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   1500 | examples/s: 105.8 | loss: 0.30374 | time e
lapsed: 00h01m03s | time left: 05h47m40s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   1750 | examples/s:  91.6 | loss: 0.32157 | time e
lapsed: 00h01m12s | time left: 05h44m05s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   2000 | examples/s: 104.3 | loss: 0.51153 | time e
lapsed: 00h01m22s | time left: 05h41m27s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   2250 | examples/s: 103.6 | loss: 0.34248 | time e
lapsed: 00h01m32s | time left: 05h39m11s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   2500 | examples/s: 106.5 | loss: 0.45199 | time e
lapsed: 00h01m42s | time left: 05h37m25s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   2750 | examples/s: 102.5 | loss: 0.36953 | time e
lapsed: 00h01m51s | time left: 05h35m43s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   3000 | examples/s: 103.9 | loss: 0.45303 | time e
lapsed: 00h02m01s | time left: 05h34m25s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   3250 | examples/s: 101.6 | loss: 0.50665 | time e
lapsed: 00h02m11s | time left: 05h33m13s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   3500 | examples/s:  90.4 | loss: 0.61825 | time e
lapsed: 00h02m21s | time left: 05h32m04s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   3750 | examples/s: 106.5 | loss: 0.60815 | time e
lapsed: 00h02m30s | time left: 05h31m12s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   4000 | examples/s: 105.6 | loss: 0.46399 | time e
lapsed: 00h02m40s | time left: 05h30m48s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   4250 | examples/s: 104.2 | loss: 0.75837 | time e
lapsed: 00h02m50s | time left: 05h29m48s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   4500 | examples/s: 105.3 | loss: 0.80678 | time e
lapsed: 00h03m00s | time left: 05h29m01s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   4750 | examples/s: 104.7 | loss: 0.39219 | time e
lapsed: 00h03m09s | time left: 05h28m30s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   5000 | examples/s: 105.9 | loss: 0.69590 | time e
lapsed: 00h03m20s | time left: 05h28m26s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   5250 | examples/s: 101.2 | loss: 0.23884 | time e
lapsed: 00h03m29s | time left: 05h27m51s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   5500 | examples/s:  98.9 | loss: 0.52098 | time e
lapsed: 00h03m39s | time left: 05h27m34s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   5750 | examples/s: 101.5 | loss: 0.37412 | time e
lapsed: 00h03m49s | time left: 05h27m06s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   6000 | examples/s: 103.0 | loss: 0.97765 | time e
lapsed: 00h03m59s | time left: 05h26m38s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   6250 | examples/s: 104.0 | loss: 0.22806 | time e
lapsed: 00h04m09s | time left: 05h26m15s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   6500 | examples/s: 100.6 | loss: 0.43808 | time e
lapsed: 00h04m18s | time left: 05h25m50s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   6750 | examples/s: 100.8 | loss: 0.54961 | time e
lapsed: 00h04m28s | time left: 05h25m25s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   7000 | examples/s: 104.1 | loss: 0.53456 | time e
lapsed: 00h04m38s | time left: 05h25m03s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   7250 | examples/s: 104.8 | loss: 0.47826 | time e
lapsed: 00h04m48s | time left: 05h24m44s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   7500 | examples/s: 103.3 | loss: 0.35346 | time e
lapsed: 00h04m57s | time left: 05h24m26s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   7750 | examples/s: 104.4 | loss: 0.14338 | time e
lapsed: 00h05m07s | time left: 05h24m14s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   8000 | examples/s: 101.7 | loss: 0.38821 | time e
lapsed: 00h05m17s | time left: 05h23m56s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   8250 | examples/s:  90.9 | loss: 0.37182 | time e
lapsed: 00h05m27s | time left: 05h23m36s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   8500 | examples/s: 104.3 | loss: 0.39300 | time e
lapsed: 00h05m37s | time left: 05h23m16s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   8750 | examples/s: 106.5 | loss: 0.54625 | time e
lapsed: 00h05m46s | time left: 05h22m54s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   9000 | examples/s: 101.1 | loss: 0.68417 | time e
lapsed: 00h05m56s | time left: 05h22m36s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   9250 | examples/s: 104.8 | loss: 0.50373 | time e
lapsed: 00h06m06s | time left: 05h22m20s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   9500 | examples/s: 104.6 | loss: 0.48334 | time e
lapsed: 00h06m16s | time left: 05h22m06s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch   9750 | examples/s: 106.1 | loss: 0.59760 | time e
lapsed: 00h06m25s | time left: 05h21m52s
```

```
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch  10000 | examples/s: 103.0 | loss: 0.31928 | time e
lapsed: 00h06m35s | time left: 05h21m37s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch  10250 | examples/s:  99.1 | loss: 0.65071 | time e
lapsed: 00h06m45s | time left: 05h21m26s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch  10500 | examples/s: 108.1 | loss: 0.72714 | time e
lapsed: 00h06m55s | time left: 05h21m14s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch  10750 | examples/s: 100.5 | loss: 0.39611 | time e
lapsed: 00h07m05s | time left: 05h20m57s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch  11000 | examples/s: 104.1 | loss: 0.52416 | time e
lapsed: 00h07m15s | time left: 05h20m53s
epoch   0 | lr 0.000100 |lr_p 0.000100 | batch  11250 | examples/s: 102.8 | loss: 0.41693 | time e
lapsed: 00h07m25s | time left: 05h20m40s
Training
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch      0 | examples/s:  21.1 | loss: 0.55769 | time e
lapsed: 00h07m33s | time left: 05h22m05s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch    250 | examples/s:  94.2 | loss: 0.38379 | time e
lapsed: 00h07m43s | time left: 05h21m57s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch    500 | examples/s: 101.7 | loss: 0.57161 | time e
lapsed: 00h07m53s | time left: 05h21m42s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch    596 | examples/s:  99.8 | loss: 0.87279 | time e
lapsed: 00h07m56s | time left: 05h21m38s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch    750 | examples/s: 102.4 | loss: 0.41252 | time e
lapsed: 00h08m02s | time left: 05h21m30s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   1000 | examples/s:  98.9 | loss: 0.39082 | time e
lapsed: 00h08m12s | time left: 05h21m16s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   1250 | examples/s:  99.1 | loss: 0.38651 | time e
lapsed: 00h08m22s | time left: 05h21m03s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   1500 | examples/s: 104.1 | loss: 0.46981 | time e
lapsed: 00h08m32s | time left: 05h20m51s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   1750 | examples/s: 107.3 | loss: 0.70182 | time e
lapsed: 00h08m42s | time left: 05h20m40s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   2000 | examples/s: 105.0 | loss: 0.52628 | time e
lapsed: 00h08m52s | time left: 05h20m30s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   2250 | examples/s: 106.0 | loss: 0.54541 | time e
lapsed: 00h09m02s | time left: 05h20m17s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   2500 | examples/s:  99.9 | loss: 0.33003 | time e
lapsed: 00h09m12s | time left: 05h20m07s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   2596 | examples/s: 101.9 | loss: 0.60328 | time e
lapsed: 00h09m15s | time left: 05h20m03s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   2750 | examples/s: 100.7 | loss: 0.96149 | time e
lapsed: 00h09m22s | time left: 05h19m57s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   3000 | examples/s: 107.0 | loss: 0.67061 | time e
lapsed: 00h09m32s | time left: 05h19m52s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   3250 | examples/s: 102.5 | loss: 0.55636 | time e
lapsed: 00h09m41s | time left: 05h19m39s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   3500 | examples/s:  93.7 | loss: 0.56087 | time e
lapsed: 00h09m51s | time left: 05h19m24s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   3750 | examples/s: 106.1 | loss: 0.24713 | time e
lapsed: 00h10m01s | time left: 05h19m12s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   4000 | examples/s: 105.0 | loss: 0.28251 | time e
lapsed: 00h10m11s | time left: 05h19m00s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   4250 | examples/s:  98.6 | loss: 0.82811 | time e
lapsed: 00h10m21s | time left: 05h18m51s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   4500 | examples/s: 105.8 | loss: 0.80209 | time e
lapsed: 00h10m31s | time left: 05h18m41s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   4596 | examples/s: 103.1 | loss: 1.03606 | time e
lapsed: 00h10m35s | time left: 05h18m35s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   4750 | examples/s: 104.1 | loss: 0.48541 | time e
lapsed: 00h10m41s | time left: 05h18m29s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   5000 | examples/s:  92.9 | loss: 0.43020 | time e
lapsed: 00h10m51s | time left: 05h18m17s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   5250 | examples/s:  99.8 | loss: 0.63685 | time e
lapsed: 00h11m00s | time left: 05h18m04s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   5500 | examples/s:  96.7 | loss: 0.92930 | time e
lapsed: 00h11m10s | time left: 05h17m51s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   5750 | examples/s: 102.4 | loss: 0.59154 | time e
lapsed: 00h11m20s | time left: 05h17m39s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   6000 | examples/s: 103.0 | loss: 0.54017 | time e
lapsed: 00h11m30s | time left: 05h17m27s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   6250 | examples/s: 104.0 | loss: 0.45813 | time e
lapsed: 00h11m40s | time left: 05h17m16s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   6500 | examples/s: 102.6 | loss: 0.52609 | time e
lapsed: 00h11m50s | time left: 05h17m05s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   6596 | examples/s: 103.4 | loss: 0.48732 | time e
lapsed: 00h11m53s | time left: 05h17m00s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   6750 | examples/s:  97.6 | loss: 0.43162 | time e
lapsed: 00h11m59s | time left: 05h16m53s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   7000 | examples/s:  98.1 | loss: 0.60409 | time e
lapsed: 00h12m09s | time left: 05h16m42s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   7250 | examples/s: 102.1 | loss: 0.50888 | time e
```

```
lapsed: 00h12m19s | time left: 05h16m32s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   7500 | examples/s: 101.5 | loss: 0.33882 | time e
lapsed: 00h12m29s | time left: 05h16m21s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   7750 | examples/s: 102.2 | loss: 0.82656 | time e
lapsed: 00h12m39s | time left: 05h16m09s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   8000 | examples/s:  99.8 | loss: 0.33599 | time e
lapsed: 00h12m49s | time left: 05h15m59s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   8250 | examples/s: 103.2 | loss: 0.45911 | time e
lapsed: 00h12m59s | time left: 05h15m49s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   8500 | examples/s: 102.0 | loss: 0.71029 | time e
lapsed: 00h13m09s | time left: 05h15m38s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch   8596 | examples/s:  94.3 | loss: 0.51233 | time e
lapsed: 00h13m12s | time left: 05h15m34s
epoch   1 | lr 0.000099 |lr_p 0.000099 | batch  10596 | examples/s: 105.9 | loss: 0.33377 | time e
lapsed: 00h14m31s | time left: 05h14m03s
Training
epoch   2 | lr 0.000098 |lr_p 0.000098 | batch   1192 | examples/s: 101.1 | loss: 0.34211 | time e
lapsed: 00h15m52s | time left: 05h13m17s
epoch   2 | lr 0.000098 |lr_p 0.000098 | batch   3192 | examples/s:  99.2 | loss: 0.85638 | time e
lapsed: 00h17m11s | time left: 05h11m46s
epoch   2 | lr 0.000098 |lr_p 0.000098 | batch   5192 | examples/s: 103.6 | loss: 0.31263 | time e
lapsed: 00h18m29s | time left: 05h10m10s
epoch   2 | lr 0.000098 |lr_p 0.000098 | batch   7192 | examples/s: 102.8 | loss: 0.57371 | time e
lapsed: 00h19m47s | time left: 05h08m26s
epoch   2 | lr 0.000098 |lr_p 0.000098 | batch   9192 | examples/s: 107.2 | loss: 0.98736 | time e
lapsed: 00h21m05s | time left: 05h06m50s
epoch   2 | lr 0.000098 |lr_p 0.000098 | batch  11192 | examples/s:  96.6 | loss: 0.44031 | time e
lapsed: 00h22m23s | time left: 05h05m15s
Training
epoch   3 | lr 0.000096 |lr_p 0.000096 | batch   1788 | examples/s:  99.7 | loss: 0.51225 | time e
lapsed: 00h23m45s | time left: 05h04m32s
epoch   3 | lr 0.000096 |lr_p 0.000096 | batch   3788 | examples/s: 100.5 | loss: 0.34387 | time e
lapsed: 00h25m03s | time left: 05h03m09s
epoch   3 | lr 0.000096 |lr_p 0.000096 | batch   5788 | examples/s: 104.8 | loss: 0.41121 | time e
lapsed: 00h26m22s | time left: 05h01m44s
epoch   3 | lr 0.000096 |lr_p 0.000096 | batch   7788 | examples/s: 102.5 | loss: 0.64221 | time e
lapsed: 00h27m40s | time left: 05h00m17s
epoch   3 | lr 0.000096 |lr_p 0.000096 | batch   9788 | examples/s:  99.8 | loss: 0.56477 | time e
lapsed: 00h29m00s | time left: 04h59m05s
Training
epoch   4 | lr 0.000094 |lr_p 0.000094 | batch    384 | examples/s:  99.7 | loss: 0.36104 | time e
lapsed: 00h30m22s | time left: 04h58m11s
epoch   4 | lr 0.000094 |lr_p 0.000094 | batch   2384 | examples/s: 104.5 | loss: 0.62724 | time e
lapsed: 00h31m41s | time left: 04h56m50s
epoch   4 | lr 0.000094 |lr_p 0.000094 | batch   4384 | examples/s: 103.9 | loss: 0.44416 | time e
lapsed: 00h33m00s | time left: 04h55m27s
epoch   4 | lr 0.000094 |lr_p 0.000094 | batch   6384 | examples/s: 101.2 | loss: 0.57530 | time e
lapsed: 00h34m19s | time left: 04h54m07s
epoch   4 | lr 0.000094 |lr_p 0.000094 | batch   8384 | examples/s: 103.8 | loss: 0.32936 | time e
lapsed: 00h35m39s | time left: 04h52m51s
epoch   4 | lr 0.000094 |lr_p 0.000094 | batch  10384 | examples/s:  94.6 | loss: 0.52077 | time e
lapsed: 00h36m57s | time left: 04h51m29s
Training
epoch   5 | lr 0.000091 |lr_p 0.000092 | batch    980 | examples/s:  96.4 | loss: 0.56453 | time e
lapsed: 00h38m19s | time left: 04h50m29s
epoch   5 | lr 0.000091 |lr_p 0.000092 | batch   2980 | examples/s: 102.1 | loss: 0.51823 | time e
lapsed: 00h39m38s | time left: 04h49m10s
epoch   5 | lr 0.000091 |lr_p 0.000092 | batch   4980 | examples/s: 102.4 | loss: 0.43740 | time e
lapsed: 00h40m57s | time left: 04h47m48s
epoch   5 | lr 0.000091 |lr_p 0.000092 | batch   6980 | examples/s: 103.3 | loss: 0.65279 | time e
lapsed: 00h42m18s | time left: 04h46m34s
epoch   5 | lr 0.000091 |lr_p 0.000092 | batch   8980 | examples/s:  95.9 | loss: 0.76437 | time e
lapsed: 00h43m37s | time left: 04h45m18s
epoch   5 | lr 0.000091 |lr_p 0.000092 | batch  10980 | examples/s:  97.3 | loss: 0.37074 | time e
lapsed: 00h44m57s | time left: 04h44m03s
Training
epoch   6 | lr 0.000089 |lr_p 0.000089 | batch   1576 | examples/s:  95.4 | loss: 0.42072 | time e
lapsed: 00h46m19s | time left: 04h42m58s
epoch   6 | lr 0.000089 |lr_p 0.000089 | batch   3576 | examples/s: 106.7 | loss: 0.57686 | time e
lapsed: 00h47m39s | time left: 04h41m41s
epoch   6 | lr 0.000089 |lr_p 0.000089 | batch   5576 | examples/s:  90.7 | loss: 0.37693 | time e
lapsed: 00h48m58s | time left: 04h40m20s
epoch   6 | lr 0.000089 |lr_p 0.000089 | batch   7576 | examples/s:  96.7 | loss: 0.47295 | time e
lapsed: 00h50m18s | time left: 04h39m05s
epoch   6 | lr 0.000089 |lr_p 0.000089 | batch   9576 | examples/s: 100.4 | loss: 0.27683 | time e
lapsed: 00h51m38s | time left: 04h37m46s
Training
epoch   7 | lr 0.000085 |lr_p 0.000086 | batch    172 | examples/s: 105.0 | loss: 0.32946 | time e
lapsed: 00h53m00s | time left: 04h36m40s
epoch   7 | lr 0.000085 |lr_p 0.000086 | batch   2172 | examples/s: 100.1 | loss: 0.65826 | time e
```

```
lapsed: 00h54m19s | time left: 04h35m18s
epoch   7 | lr 0.000085 |lr_p 0.000086 | batch   4172 | examples/s:  99.8 | loss: 0.23194 | time e
lapsed: 00h55m38s | time left: 04h33m59s
epoch   7 | lr 0.000085 |lr_p 0.000086 | batch   6172 | examples/s:  95.7 | loss: 0.61961 | time e
lapsed: 00h56m58s | time left: 04h32m41s
epoch   7 | lr 0.000085 |lr_p 0.000086 | batch   8172 | examples/s: 101.6 | loss: 0.38886 | time e
lapsed: 00h58m18s | time left: 04h31m22s
epoch   7 | lr 0.000085 |lr_p 0.000086 | batch  10172 | examples/s: 102.1 | loss: 0.39918 | time e
lapsed: 00h59m37s | time left: 04h30m02s
Training
epoch   8 | lr 0.000082 |lr_p 0.000083 | batch    768 | examples/s: 103.7 | loss: 0.37604 | time e
lapsed: 01h00m58s | time left: 04h28m50s
epoch   8 | lr 0.000082 |lr_p 0.000083 | batch   2768 | examples/s:  96.8 | loss: 0.33777 | time e
lapsed: 01h02m19s | time left: 04h27m36s
epoch   8 | lr 0.000082 |lr_p 0.000083 | batch   4768 | examples/s:  99.1 | loss: 0.26284 | time e
lapsed: 01h03m39s | time left: 04h26m16s
epoch   8 | lr 0.000082 |lr_p 0.000083 | batch   6768 | examples/s: 104.3 | loss: 0.71091 | time e
lapsed: 01h04m59s | time left: 04h24m58s
epoch   8 | lr 0.000082 |lr_p 0.000083 | batch   8768 | examples/s: 104.4 | loss: 0.37017 | time e
lapsed: 01h06m18s | time left: 04h23m40s
epoch   8 | lr 0.000082 |lr_p 0.000083 | batch  10768 | examples/s:  96.0 | loss: 0.28578 | time e
lapsed: 01h07m37s | time left: 04h22m17s
Training
epoch   9 | lr 0.000078 |lr_p 0.000079 | batch   1364 | examples/s: 104.9 | loss: 0.30243 | time e
lapsed: 01h08m58s | time left: 04h21m04s
epoch   9 | lr 0.000078 |lr_p 0.000079 | batch   3364 | examples/s: 101.7 | loss: 0.36172 | time e
lapsed: 01h10m18s | time left: 04h19m44s
epoch   9 | lr 0.000078 |lr_p 0.000079 | batch   5364 | examples/s: 104.3 | loss: 0.42035 | time e
lapsed: 01h11m33s | time left: 04h18m08s
epoch   9 | lr 0.000078 |lr_p 0.000079 | batch   7364 | examples/s: 112.3 | loss: 0.46136 | time e
lapsed: 01h12m44s | time left: 04h16m17s
epoch   9 | lr 0.000078 |lr_p 0.000079 | batch   9364 | examples/s: 115.1 | loss: 1.12097 | time e
lapsed: 01h13m54s | time left: 04h14m27s
epoch   9 | lr 0.000078 |lr_p 0.000079 | batch  11364 | examples/s: 109.6 | loss: 0.50139 | time e
lapsed: 01h15m05s | time left: 04h12m39s
Training
epoch  10 | lr 0.000073 |lr_p 0.000075 | batch   1960 | examples/s: 111.4 | loss: 0.52033 | time e
lapsed: 01h16m18s | time left: 04h11m01s
epoch  10 | lr 0.000073 |lr_p 0.000075 | batch   3960 | examples/s: 113.3 | loss: 0.73017 | time e
lapsed: 01h17m29s | time left: 04h09m18s
epoch  10 | lr 0.000073 |lr_p 0.000075 | batch   5960 | examples/s: 116.0 | loss: 0.39833 | time e
lapsed: 01h18m41s | time left: 04h07m36s
epoch  10 | lr 0.000073 |lr_p 0.000075 | batch   7960 | examples/s: 110.1 | loss: 0.46344 | time e
lapsed: 01h19m53s | time left: 04h05m57s
epoch  10 | lr 0.000073 |lr_p 0.000075 | batch   9960 | examples/s: 117.6 | loss: 0.38397 | time e
lapsed: 01h21m05s | time left: 04h04m18s
Training
epoch  11 | lr 0.000069 |lr_p 0.000071 | batch    556 | examples/s: 111.0 | loss: 0.59284 | time e
lapsed: 01h22m19s | time left: 04h02m46s
epoch  11 | lr 0.000069 |lr_p 0.000071 | batch   2556 | examples/s: 111.8 | loss: 0.40175 | time e
lapsed: 01h23m32s | time left: 04h01m13s
epoch  11 | lr 0.000069 |lr_p 0.000071 | batch   4556 | examples/s: 112.1 | loss: 0.40041 | time e
lapsed: 01h24m45s | time left: 03h59m41s
epoch  11 | lr 0.000069 |lr_p 0.000071 | batch   6556 | examples/s: 108.3 | loss: 0.38449 | time e
lapsed: 01h25m58s | time left: 03h58m08s
epoch  11 | lr 0.000069 |lr_p 0.000071 | batch   8556 | examples/s: 103.2 | loss: 0.51516 | time e
lapsed: 01h27m11s | time left: 03h56m34s
epoch  11 | lr 0.000069 |lr_p 0.000071 | batch  10556 | examples/s: 111.2 | loss: 0.76401 | time e
lapsed: 01h28m23s | time left: 03h55m00s
Training
epoch  12 | lr 0.000064 |lr_p 0.000066 | batch   1152 | examples/s: 106.6 | loss: 0.94782 | time e
lapsed: 01h29m38s | time left: 03h53m34s
epoch  12 | lr 0.000064 |lr_p 0.000066 | batch   3152 | examples/s: 112.9 | loss: 0.52948 | time e
lapsed: 01h30m50s | time left: 03h52m02s
epoch  12 | lr 0.000064 |lr_p 0.000066 | batch   5152 | examples/s:  92.4 | loss: 0.46954 | time e
lapsed: 01h32m03s | time left: 03h50m31s
epoch  12 | lr 0.000064 |lr_p 0.000066 | batch   7152 | examples/s: 114.0 | loss: 0.38311 | time e
lapsed: 01h33m15s | time left: 03h48m59s
epoch  12 | lr 0.000064 |lr_p 0.000066 | batch   9152 | examples/s: 115.5 | loss: 0.45565 | time e
lapsed: 01h34m27s | time left: 03h47m27s
epoch  12 | lr 0.000064 |lr_p 0.000066 | batch  11152 | examples/s: 116.9 | loss: 0.37749 | time e
lapsed: 01h35m38s | time left: 03h45m56s
Training
epoch  13 | lr 0.000060 |lr_p 0.000062 | batch   1748 | examples/s: 113.2 | loss: 0.46985 | time e
lapsed: 01h36m53s | time left: 03h44m31s
epoch  13 | lr 0.000060 |lr_p 0.000062 | batch   3748 | examples/s: 114.1 | loss: 0.54206 | time e
lapsed: 01h38m06s | time left: 03h43m04s
epoch  13 | lr 0.000060 |lr_p 0.000062 | batch   5748 | examples/s: 109.0 | loss: 0.22653 | time e
lapsed: 01h39m19s | time left: 03h41m35s
epoch  13 | lr 0.000060 |lr_p 0.000062 | batch   7748 | examples/s: 115.8 | loss: 0.65166 | time e
```

```
lapsed: 01h40m30s | time left: 03h40m06s
epoch  13 | lr 0.000060 |lr_p 0.000062 | batch   9748 | examples/s: 112.4 | loss: 0.43550 | time e
lapsed: 01h41m42s | time left: 03h38m36s
Training
epoch  14 | lr 0.000055 |lr_p 0.000057 | batch    344 | examples/s: 106.5 | loss: 0.35515 | time e
lapsed: 01h42m57s | time left: 03h37m13s
epoch  14 | lr 0.000055 |lr_p 0.000057 | batch   2344 | examples/s: 109.8 | loss: 0.22190 | time e
lapsed: 01h44m11s | time left: 03h35m49s
epoch  14 | lr 0.000055 |lr_p 0.000057 | batch   4344 | examples/s: 106.4 | loss: 0.55782 | time e
lapsed: 01h45m23s | time left: 03h34m23s
epoch  14 | lr 0.000055 |lr_p 0.000057 | batch   6344 | examples/s: 111.4 | loss: 0.32657 | time e
lapsed: 01h46m35s | time left: 03h32m56s
epoch  14 | lr 0.000055 |lr_p 0.000057 | batch   8344 | examples/s: 113.8 | loss: 0.40553 | time e
lapsed: 01h47m47s | time left: 03h31m28s
epoch  14 | lr 0.000055 |lr_p 0.000057 | batch  10344 | examples/s: 112.1 | loss: 0.51742 | time e
lapsed: 01h48m58s | time left: 03h30m00s
Training
epoch  15 | lr 0.000050 |lr_p 0.000053 | batch    940 | examples/s: 105.7 | loss: 0.56769 | time e
lapsed: 01h50m13s | time left: 03h28m40s
epoch  15 | lr 0.000050 |lr_p 0.000053 | batch   2940 | examples/s: 103.5 | loss: 0.34662 | time e
lapsed: 01h51m26s | time left: 03h27m15s
epoch  15 | lr 0.000050 |lr_p 0.000053 | batch   4940 | examples/s:  92.9 | loss: 0.53990 | time e
lapsed: 01h52m39s | time left: 03h25m50s
epoch  15 | lr 0.000050 |lr_p 0.000053 | batch   6940 | examples/s: 113.8 | loss: 0.43283 | time e
lapsed: 01h53m50s | time left: 03h24m24s
epoch  15 | lr 0.000050 |lr_p 0.000053 | batch   8940 | examples/s: 113.4 | loss: 0.39722 | time e
lapsed: 01h55m03s | time left: 03h23m00s
epoch  15 | lr 0.000050 |lr_p 0.000053 | batch  10940 | examples/s: 111.0 | loss: 0.66041 | time e
lapsed: 01h56m15s | time left: 03h21m35s
Training
epoch  16 | lr 0.000045 |lr_p 0.000048 | batch   1536 | examples/s: 105.4 | loss: 0.56372 | time e
lapsed: 01h57m30s | time left: 03h20m15s
epoch  16 | lr 0.000045 |lr_p 0.000048 | batch   3536 | examples/s: 113.1 | loss: 0.31197 | time e
lapsed: 01h58m43s | time left: 03h18m53s
epoch  16 | lr 0.000045 |lr_p 0.000048 | batch   5536 | examples/s: 107.3 | loss: 0.47659 | time e
lapsed: 01h59m55s | time left: 03h17m30s
epoch  16 | lr 0.000045 |lr_p 0.000048 | batch   7536 | examples/s: 112.9 | loss: 0.63046 | time e
lapsed: 02h01m09s | time left: 03h16m08s
epoch  16 | lr 0.000045 |lr_p 0.000048 | batch   9536 | examples/s: 111.3 | loss: 0.57245 | time e
lapsed: 02h02m21s | time left: 03h14m44s
Training
epoch  17 | lr 0.000041 |lr_p 0.000044 | batch    132 | examples/s: 110.6 | loss: 0.50303 | time e
lapsed: 02h03m34s | time left: 03h13m24s
epoch  17 | lr 0.000041 |lr_p 0.000044 | batch   2132 | examples/s: 106.9 | loss: 0.39004 | time e
lapsed: 02h04m48s | time left: 03h12m03s
epoch  17 | lr 0.000041 |lr_p 0.000044 | batch   4132 | examples/s: 105.1 | loss: 0.71872 | time e
lapsed: 02h06m01s | time left: 03h10m41s
epoch  17 | lr 0.000041 |lr_p 0.000044 | batch   6132 | examples/s: 114.0 | loss: 0.67266 | time e
lapsed: 02h07m15s | time left: 03h09m21s
epoch  17 | lr 0.000041 |lr_p 0.000044 | batch   8132 | examples/s: 113.8 | loss: 0.37485 | time e
lapsed: 02h08m28s | time left: 03h08m00s
epoch  17 | lr 0.000041 |lr_p 0.000044 | batch  10132 | examples/s:  96.1 | loss: 0.23352 | time e
lapsed: 02h09m40s | time left: 03h06m38s
Training
epoch  18 | lr 0.000036 |lr_p 0.000039 | batch    728 | examples/s: 108.7 | loss: 0.30353 | time e
lapsed: 02h10m55s | time left: 03h05m19s
epoch  18 | lr 0.000036 |lr_p 0.000039 | batch   2728 | examples/s: 111.0 | loss: 0.74875 | time e
lapsed: 02h12m08s | time left: 03h03m58s
epoch  18 | lr 0.000036 |lr_p 0.000039 | batch   4728 | examples/s: 109.7 | loss: 0.55153 | time e
lapsed: 02h13m21s | time left: 03h02m37s
epoch  18 | lr 0.000036 |lr_p 0.000039 | batch   6728 | examples/s:  91.7 | loss: 0.45575 | time e
lapsed: 02h14m33s | time left: 03h01m16s
epoch  18 | lr 0.000036 |lr_p 0.000039 | batch   8728 | examples/s: 107.3 | loss: 0.32482 | time e
lapsed: 02h15m45s | time left: 02h59m54s
epoch  18 | lr 0.000036 |lr_p 0.000039 | batch  10728 | examples/s: 113.2 | loss: 0.29835 | time e
lapsed: 02h16m57s | time left: 02h58m33s
Training
epoch  19 | lr 0.000032 |lr_p 0.000035 | batch   1324 | examples/s: 114.9 | loss: 0.44576 | time e
lapsed: 02h18m12s | time left: 02h57m16s
epoch  19 | lr 0.000032 |lr_p 0.000035 | batch   3324 | examples/s: 110.6 | loss: 0.53154 | time e
lapsed: 02h19m25s | time left: 02h55m55s
epoch  19 | lr 0.000032 |lr_p 0.000035 | batch   5324 | examples/s: 115.5 | loss: 0.78353 | time e
lapsed: 02h20m37s | time left: 02h54m34s
epoch  19 | lr 0.000032 |lr_p 0.000035 | batch   7324 | examples/s: 106.3 | loss: 0.72005 | time e
lapsed: 02h21m50s | time left: 02h53m14s
epoch  19 | lr 0.000032 |lr_p 0.000035 | batch   9324 | examples/s: 106.5 | loss: 0.29777 | time e
lapsed: 02h23m02s | time left: 02h51m53s
epoch  19 | lr 0.000032 |lr_p 0.000035 | batch  11324 | examples/s: 114.4 | loss: 0.31344 | time e
lapsed: 02h24m14s | time left: 02h50m33s
Training
```

```
epoch  20 | lr 0.000027 |lr_p 0.000031 | batch   1920 | examples/s: 114.1 | loss: 0.30517 | time e
lapsed: 02h25m29s | time left: 02h49m16s
epoch  20 | lr 0.000027 |lr_p 0.000031 | batch   3920 | examples/s: 112.8 | loss: 0.62174 | time e
lapsed: 02h26m42s | time left: 02h47m56s
epoch  20 | lr 0.000027 |lr_p 0.000031 | batch   5920 | examples/s: 114.6 | loss: 0.15017 | time e
lapsed: 02h27m54s | time left: 02h46m36s
epoch  20 | lr 0.000027 |lr_p 0.000031 | batch   7920 | examples/s: 112.7 | loss: 0.35059 | time e
lapsed: 02h29m06s | time left: 02h45m16s
epoch  20 | lr 0.000027 |lr_p 0.000031 | batch   9920 | examples/s: 112.9 | loss: 0.60489 | time e
lapsed: 02h30m17s | time left: 02h43m56s
Training
epoch  21 | lr 0.000023 |lr_p 0.000027 | batch    516 | examples/s: 114.3 | loss: 0.25326 | time e
lapsed: 02h31m32s | time left: 02h42m39s
epoch  21 | lr 0.000023 |lr_p 0.000027 | batch   2516 | examples/s: 114.3 | loss: 0.47149 | time e
lapsed: 02h32m44s | time left: 02h41m19s
epoch  21 | lr 0.000023 |lr_p 0.000027 | batch   4516 | examples/s: 109.7 | loss: 0.51162 | time e
lapsed: 02h33m56s | time left: 02h40m00s
epoch  21 | lr 0.000023 |lr_p 0.000027 | batch   6516 | examples/s: 114.7 | loss: 0.49498 | time e
lapsed: 02h35m09s | time left: 02h38m41s
epoch  21 | lr 0.000023 |lr_p 0.000027 | batch   8516 | examples/s: 112.8 | loss: 0.23164 | time e
lapsed: 02h36m21s | time left: 02h37m21s
epoch  21 | lr 0.000023 |lr_p 0.000027 | batch  10516 | examples/s: 113.7 | loss: 0.57831 | time e
lapsed: 02h37m33s | time left: 02h36m02s
Training
epoch  22 | lr 0.000020 |lr_p 0.000024 | batch   1112 | examples/s: 108.8 | loss: 0.71911 | time e
lapsed: 02h38m48s | time left: 02h34m46s
epoch  22 | lr 0.000020 |lr_p 0.000024 | batch   3112 | examples/s: 107.6 | loss: 0.46787 | time e
lapsed: 02h39m59s | time left: 02h33m26s
epoch  22 | lr 0.000020 |lr_p 0.000024 | batch   5112 | examples/s: 112.8 | loss: 0.45450 | time e
lapsed: 02h41m11s | time left: 02h32m07s
epoch  22 | lr 0.000020 |lr_p 0.000024 | batch   7112 | examples/s: 112.6 | loss: 0.62747 | time e
lapsed: 02h42m23s | time left: 02h30m48s
epoch  22 | lr 0.000020 |lr_p 0.000024 | batch   9112 | examples/s: 115.6 | loss: 0.57191 | time e
lapsed: 02h43m35s | time left: 02h29m30s
epoch  22 | lr 0.000020 |lr_p 0.000024 | batch  11112 | examples/s: 113.8 | loss: 0.42776 | time e
lapsed: 02h44m48s | time left: 02h28m11s
Training
epoch  23 | lr 0.000016 |lr_p 0.000021 | batch   1708 | examples/s: 106.1 | loss: 0.72683 | time e
lapsed: 02h46m03s | time left: 02h26m56s
epoch  23 | lr 0.000016 |lr_p 0.000021 | batch   3708 | examples/s: 116.4 | loss: 0.54990 | time e
lapsed: 02h47m15s | time left: 02h25m37s
epoch  23 | lr 0.000016 |lr_p 0.000021 | batch   5708 | examples/s: 115.7 | loss: 0.41523 | time e
lapsed: 02h48m26s | time left: 02h24m18s
epoch  23 | lr 0.000016 |lr_p 0.000021 | batch   7708 | examples/s: 113.6 | loss: 0.36754 | time e
lapsed: 02h49m38s | time left: 02h22m59s
epoch  23 | lr 0.000016 |lr_p 0.000021 | batch   9708 | examples/s: 111.9 | loss: 0.33181 | time e
lapsed: 02h50m50s | time left: 02h21m41s
Training
epoch  24 | lr 0.000014 |lr_p 0.000018 | batch    304 | examples/s: 112.1 | loss: 0.64990 | time e
lapsed: 02h52m04s | time left: 02h20m25s
epoch  24 | lr 0.000014 |lr_p 0.000018 | batch   2304 | examples/s: 109.3 | loss: 0.46897 | time e
lapsed: 02h53m18s | time left: 02h19m08s
epoch  24 | lr 0.000014 |lr_p 0.000018 | batch   4304 | examples/s: 111.0 | loss: 0.57997 | time e
lapsed: 02h54m31s | time left: 02h17m51s
epoch  24 | lr 0.000014 |lr_p 0.000018 | batch   6304 | examples/s: 109.0 | loss: 0.75668 | time e
lapsed: 02h55m43s | time left: 02h16m34s
epoch  24 | lr 0.000014 |lr_p 0.000018 | batch   8304 | examples/s: 111.1 | loss: 0.34847 | time e
lapsed: 02h56m56s | time left: 02h15m16s
epoch  24 | lr 0.000014 |lr_p 0.000018 | batch  10304 | examples/s: 113.6 | loss: 0.69125 | time e
lapsed: 02h58m08s | time left: 02h13m58s
Training
epoch  25 | lr 0.000011 |lr_p 0.000016 | batch    900 | examples/s: 105.8 | loss: 0.38796 | time e
lapsed: 02h59m23s | time left: 02h12m43s
epoch  25 | lr 0.000011 |lr_p 0.000016 | batch   2900 | examples/s: 110.9 | loss: 0.71663 | time e
lapsed: 03h00m36s | time left: 02h11m26s
epoch  25 | lr 0.000011 |lr_p 0.000016 | batch   4900 | examples/s: 113.7 | loss: 0.99163 | time e
lapsed: 03h01m47s | time left: 02h10m08s
epoch  25 | lr 0.000011 |lr_p 0.000016 | batch   6900 | examples/s: 114.5 | loss: 0.27221 | time e
lapsed: 03h02m59s | time left: 02h08m50s
epoch  25 | lr 0.000011 |lr_p 0.000016 | batch   8900 | examples/s: 113.2 | loss: 0.46326 | time e
lapsed: 03h04m10s | time left: 02h07m32s
epoch  25 | lr 0.000011 |lr_p 0.000016 | batch  10900 | examples/s: 112.6 | loss: 0.61697 | time e
lapsed: 03h05m22s | time left: 02h06m15s
Training
epoch  26 | lr 0.000009 |lr_p 0.000014 | batch   1496 | examples/s: 113.6 | loss: 0.81528 | time e
lapsed: 03h06m37s | time left: 02h05m00s
epoch  26 | lr 0.000009 |lr_p 0.000014 | batch   3496 | examples/s: 109.1 | loss: 0.33061 | time e
lapsed: 03h07m50s | time left: 02h03m43s
epoch  26 | lr 0.000009 |lr_p 0.000014 | batch   5496 | examples/s: 109.8 | loss: 0.42606 | time e
lapsed: 03h09m02s | time left: 02h02m26s
```

```
epoch  26 | lr 0.000009 |lr_p 0.000014 | batch   7496 | examples/s: 114.3 | loss: 0.69500 | time e
lapsed: 03h10m14s | time left: 02h01m09s
epoch  26 | lr 0.000009 |lr_p 0.000014 | batch   9496 | examples/s: 105.0 | loss: 0.44754 | time e
lapsed: 03h11m26s | time left: 01h59m52s
Training
epoch  27 | lr 0.000007 |lr_p 0.000012 | batch     92 | examples/s: 112.9 | loss: 0.59403 | time e
lapsed: 03h12m40s | time left: 01h58m36s
epoch  27 | lr 0.000007 |lr_p 0.000012 | batch   2092 | examples/s: 111.4 | loss: 0.41021 | time e
lapsed: 03h13m54s | time left: 01h57m20s
epoch  27 | lr 0.000007 |lr_p 0.000012 | batch   4092 | examples/s: 111.3 | loss: 0.71705 | time e
lapsed: 03h15m06s | time left: 01h56m04s
epoch  27 | lr 0.000007 |lr_p 0.000012 | batch   6092 | examples/s: 109.9 | loss: 0.60577 | time e
lapsed: 03h16m19s | time left: 01h54m47s
epoch  27 | lr 0.000007 |lr_p 0.000012 | batch   8092 | examples/s: 113.0 | loss: 0.41555 | time e
lapsed: 03h17m31s | time left: 01h53m30s
epoch  27 | lr 0.000007 |lr_p 0.000012 | batch  10092 | examples/s: 114.9 | loss: 0.14415 | time e
lapsed: 03h18m43s | time left: 01h52m14s
Training
epoch  28 | lr 0.000006 |lr_p 0.000011 | batch    688 | examples/s: 113.8 | loss: 0.65281 | time e
lapsed: 03h19m58s | time left: 01h50m59s
epoch  28 | lr 0.000006 |lr_p 0.000011 | batch   2688 | examples/s:  89.0 | loss: 0.57737 | time e
lapsed: 03h21m11s | time left: 01h49m42s
epoch  28 | lr 0.000006 |lr_p 0.000011 | batch   4688 | examples/s: 112.8 | loss: 0.60705 | time e
lapsed: 03h22m22s | time left: 01h48m26s
epoch  28 | lr 0.000006 |lr_p 0.000011 | batch   6688 | examples/s: 108.0 | loss: 0.34504 | time e
lapsed: 03h23m34s | time left: 01h47m09s
epoch  28 | lr 0.000006 |lr_p 0.000011 | batch   8688 | examples/s: 116.2 | loss: 0.59514 | time e
lapsed: 03h24m45s | time left: 01h45m52s
epoch  28 | lr 0.000006 |lr_p 0.000011 | batch  10688 | examples/s: 111.1 | loss: 0.27837 | time e
lapsed: 03h25m58s | time left: 01h44m36s
Training
epoch  29 | lr 0.000005 |lr_p 0.000010 | batch   1284 | examples/s: 106.3 | loss: 0.64420 | time e
lapsed: 03h27m12s | time left: 01h43m21s
epoch  29 | lr 0.000005 |lr_p 0.000010 | batch   3284 | examples/s:  91.3 | loss: 0.20391 | time e
lapsed: 03h28m26s | time left: 01h42m05s
epoch  29 | lr 0.000005 |lr_p 0.000010 | batch   5284 | examples/s: 109.8 | loss: 0.65798 | time e
lapsed: 03h29m39s | time left: 01h40m49s
epoch  29 | lr 0.000005 |lr_p 0.000010 | batch   7284 | examples/s: 115.7 | loss: 0.32823 | time e
lapsed: 03h30m50s | time left: 01h39m33s
epoch  29 | lr 0.000005 |lr_p 0.000010 | batch   9284 | examples/s: 108.7 | loss: 0.55189 | time e
lapsed: 03h32m02s | time left: 01h38m17s
epoch  29 | lr 0.000005 |lr_p 0.000010 | batch  11284 | examples/s: 111.6 | loss: 0.42040 | time e
lapsed: 03h33m14s | time left: 01h37m01s
Training
epoch  30 | lr 0.000090 |lr_p 0.000090 | batch   1880 | examples/s: 103.5 | loss: 0.59631 | time e
lapsed: 03h34m32s | time left: 01h35m47s
epoch  30 | lr 0.000090 |lr_p 0.000090 | batch   3880 | examples/s: 102.2 | loss: 0.45386 | time e
lapsed: 03h35m50s | time left: 01h34m34s
epoch  30 | lr 0.000090 |lr_p 0.000090 | batch   5880 | examples/s: 102.2 | loss: 0.52599 | time e
lapsed: 03h37m09s | time left: 01h33m21s
epoch  30 | lr 0.000090 |lr_p 0.000090 | batch   7880 | examples/s: 105.8 | loss: 0.90160 | time e
lapsed: 03h38m27s | time left: 01h32m07s
epoch  30 | lr 0.000090 |lr_p 0.000090 | batch   9880 | examples/s:  99.8 | loss: 0.30211 | time e
lapsed: 03h39m45s | time left: 01h30m53s
Training
epoch  31 | lr 0.000090 |lr_p 0.000090 | batch    476 | examples/s: 103.2 | loss: 0.63951 | time e
lapsed: 03h41m06s | time left: 01h29m41s
epoch  31 | lr 0.000090 |lr_p 0.000090 | batch   2476 | examples/s: 102.5 | loss: 0.23640 | time e
lapsed: 03h42m24s | time left: 01h28m27s
epoch  31 | lr 0.000090 |lr_p 0.000090 | batch   4476 | examples/s: 107.8 | loss: 0.54902 | time e
lapsed: 03h43m41s | time left: 01h27m13s
epoch  31 | lr 0.000090 |lr_p 0.000090 | batch   6476 | examples/s: 102.4 | loss: 0.39028 | time e
lapsed: 03h44m59s | time left: 01h25m59s
epoch  31 | lr 0.000090 |lr_p 0.000090 | batch   8476 | examples/s:  99.4 | loss: 0.52520 | time e
lapsed: 03h46m16s | time left: 01h24m45s
epoch  31 | lr 0.000090 |lr_p 0.000090 | batch  10476 | examples/s: 105.8 | loss: 0.45191 | time e
lapsed: 03h47m35s | time left: 01h23m31s
Training
epoch  32 | lr 0.000089 |lr_p 0.000089 | batch   1072 | examples/s: 104.6 | loss: 0.48635 | time e
lapsed: 03h48m57s | time left: 01h22m19s
epoch  32 | lr 0.000089 |lr_p 0.000089 | batch   3072 | examples/s: 102.0 | loss: 0.29880 | time e
lapsed: 03h50m17s | time left: 01h21m06s
epoch  32 | lr 0.000089 |lr_p 0.000089 | batch   5072 | examples/s:  99.3 | loss: 0.34316 | time e
lapsed: 03h51m37s | time left: 01h19m52s
epoch  32 | lr 0.000089 |lr_p 0.000089 | batch   7072 | examples/s: 103.4 | loss: 0.36289 | time e
lapsed: 03h52m56s | time left: 01h18m38s
epoch  32 | lr 0.000089 |lr_p 0.000089 | batch   9072 | examples/s:  99.6 | loss: 0.31858 | time e
lapsed: 03h54m15s | time left: 01h17m25s
epoch  32 | lr 0.000089 |lr_p 0.000089 | batch  11072 | examples/s:  98.9 | loss: 0.69952 | time e
lapsed: 03h55m35s | time left: 01h16m11s
```

```
Training
epoch  33 | lr 0.000088 |lr_p 0.000088 | batch   1668 | examples/s:  92.0 | loss: 0.69925 | time e
lapsed: 03h56m57s | time left: 01h14m58s
epoch  33 | lr 0.000088 |lr_p 0.000088 | batch   3668 | examples/s: 101.9 | loss: 0.66058 | time e
lapsed: 03h58m16s | time left: 01h13m44s
epoch  33 | lr 0.000088 |lr_p 0.000088 | batch   5668 | examples/s: 102.5 | loss: 0.37866 | time e
lapsed: 03h59m35s | time left: 01h12m30s
epoch  33 | lr 0.000088 |lr_p 0.000088 | batch   7668 | examples/s: 101.2 | loss: 0.55513 | time e
lapsed: 04h00m54s | time left: 01h11m16s
epoch  33 | lr 0.000088 |lr_p 0.000088 | batch   9668 | examples/s:  95.8 | loss: 0.34259 | time e
lapsed: 04h02m13s | time left: 01h10m02s
Training
epoch  34 | lr 0.000087 |lr_p 0.000087 | batch    264 | examples/s:  81.7 | loss: 0.60197 | time e
lapsed: 04h03m35s | time left: 01h08m48s
epoch  34 | lr 0.000087 |lr_p 0.000087 | batch   2264 | examples/s: 104.8 | loss: 0.24841 | time e
lapsed: 04h04m54s | time left: 01h07m34s
epoch  34 | lr 0.000087 |lr_p 0.000087 | batch   4264 | examples/s: 104.3 | loss: 0.34349 | time e
lapsed: 04h06m14s | time left: 01h06m19s
epoch  34 | lr 0.000087 |lr_p 0.000087 | batch   6264 | examples/s: 100.3 | loss: 0.95180 | time e
lapsed: 04h07m33s | time left: 01h05m05s
epoch  34 | lr 0.000087 |lr_p 0.000087 | batch   8264 | examples/s: 103.1 | loss: 0.18813 | time e
lapsed: 04h08m52s | time left: 01h03m51s
epoch  34 | lr 0.000087 |lr_p 0.000087 | batch  10264 | examples/s: 101.1 | loss: 0.22310 | time e
lapsed: 04h10m11s | time left: 01h02m36s
Training
epoch  35 | lr 0.000085 |lr_p 0.000085 | batch    860 | examples/s: 101.5 | loss: 0.68612 | time e
lapsed: 04h11m33s | time left: 01h01m22s
epoch  35 | lr 0.000085 |lr_p 0.000085 | batch   2860 | examples/s:  98.4 | loss: 0.34472 | time e
lapsed: 04h12m53s | time left: 01h00m08s
epoch  35 | lr 0.000085 |lr_p 0.000085 | batch   4860 | examples/s:  99.5 | loss: 0.38941 | time e
lapsed: 04h14m13s | time left: 00h58m54s
epoch  35 | lr 0.000085 |lr_p 0.000085 | batch   6860 | examples/s:  93.5 | loss: 0.56871 | time e
lapsed: 04h15m33s | time left: 00h57m39s
epoch  35 | lr 0.000085 |lr_p 0.000085 | batch   8860 | examples/s: 104.9 | loss: 0.28711 | time e
lapsed: 04h16m52s | time left: 00h56m24s
epoch  35 | lr 0.000085 |lr_p 0.000085 | batch  10860 | examples/s:  81.8 | loss: 0.62001 | time e
lapsed: 04h18m11s | time left: 00h55m09s
Training
epoch  36 | lr 0.000082 |lr_p 0.000083 | batch   1456 | examples/s:  96.0 | loss: 0.34747 | time e
lapsed: 04h19m33s | time left: 00h53m55s
epoch  36 | lr 0.000082 |lr_p 0.000083 | batch   3456 | examples/s:  98.6 | loss: 0.61646 | time e
lapsed: 04h20m53s | time left: 00h52m41s
epoch  36 | lr 0.000082 |lr_p 0.000083 | batch   5456 | examples/s:  99.5 | loss: 1.02825 | time e
lapsed: 04h22m13s | time left: 00h51m26s
epoch  36 | lr 0.000082 |lr_p 0.000083 | batch   7456 | examples/s: 101.2 | loss: 0.28100 | time e
lapsed: 04h23m33s | time left: 00h50m11s
epoch  36 | lr 0.000082 |lr_p 0.000083 | batch   9456 | examples/s:  99.0 | loss: 0.41859 | time e
lapsed: 04h24m53s | time left: 00h48m56s
Training
epoch  37 | lr 0.000080 |lr_p 0.000080 | batch     52 | examples/s: 100.2 | loss: 0.64991 | time e
lapsed: 04h26m15s | time left: 00h47m41s
epoch  37 | lr 0.000080 |lr_p 0.000080 | batch   2052 | examples/s: 104.9 | loss: 0.56187 | time e
lapsed: 04h27m34s | time left: 00h46m26s
epoch  37 | lr 0.000080 |lr_p 0.000080 | batch   4052 | examples/s:  98.1 | loss: 0.21505 | time e
lapsed: 04h28m54s | time left: 00h45m11s
epoch  37 | lr 0.000080 |lr_p 0.000080 | batch   6052 | examples/s: 100.9 | loss: 0.24201 | time e
lapsed: 04h30m14s | time left: 00h43m56s
epoch  37 | lr 0.000080 |lr_p 0.000080 | batch   8052 | examples/s:  98.8 | loss: 0.47542 | time e
lapsed: 04h31m32s | time left: 00h42m41s
epoch  37 | lr 0.000080 |lr_p 0.000080 | batch  10052 | examples/s: 105.6 | loss: 0.56635 | time e
lapsed: 04h32m51s | time left: 00h41m26s
Training
epoch  38 | lr 0.000077 |lr_p 0.000078 | batch    648 | examples/s: 102.5 | loss: 0.51054 | time e
lapsed: 04h34m12s | time left: 00h40m11s
epoch  38 | lr 0.000077 |lr_p 0.000078 | batch   2648 | examples/s:  94.6 | loss: 0.53858 | time e
lapsed: 04h35m32s | time left: 00h38m55s
epoch  38 | lr 0.000077 |lr_p 0.000078 | batch   4648 | examples/s:  98.9 | loss: 0.59242 | time e
lapsed: 04h36m53s | time left: 00h37m40s
epoch  38 | lr 0.000077 |lr_p 0.000078 | batch   6648 | examples/s: 100.6 | loss: 0.31681 | time e
lapsed: 04h38m13s | time left: 00h36m25s
epoch  38 | lr 0.000077 |lr_p 0.000078 | batch   8648 | examples/s: 103.5 | loss: 0.36978 | time e
lapsed: 04h39m32s | time left: 00h35m09s
epoch  38 | lr 0.000077 |lr_p 0.000078 | batch  10648 | examples/s:  99.7 | loss: 0.47957 | time e
lapsed: 04h40m52s | time left: 00h33m54s
Training
epoch  39 | lr 0.000074 |lr_p 0.000074 | batch   1244 | examples/s: 100.5 | loss: 0.48269 | time e
lapsed: 04h42m14s | time left: 00h32m39s
epoch  39 | lr 0.000074 |lr_p 0.000074 | batch   3244 | examples/s: 104.4 | loss: 0.33208 | time e
lapsed: 04h43m33s | time left: 00h31m23s
epoch  39 | lr 0.000074 |lr_p 0.000074 | batch   5244 | examples/s: 104.9 | loss: 0.27144 | time e
```

```
lapsed: 04h44m52s | time left: 00h30m08s
epoch  39 | lr 0.000074 |lr_p 0.000074 | batch   7244 | examples/s: 105.3 | loss: 0.87697 | time e
lapsed: 04h46m11s | time left: 00h28m52s
epoch  39 | lr 0.000074 |lr_p 0.000074 | batch   9244 | examples/s: 105.4 | loss: 0.44309 | time e
lapsed: 04h47m29s | time left: 00h27m36s
epoch  39 | lr 0.000074 |lr_p 0.000074 | batch  11244 | examples/s: 105.1 | loss: 0.28501 | time e
lapsed: 04h48m50s | time left: 00h26m21s
Training
epoch  40 | lr 0.000070 |lr_p 0.000071 | batch   1840 | examples/s: 104.9 | loss: 1.26091 | time e
lapsed: 04h50m12s | time left: 00h25m05s
epoch  40 | lr 0.000070 |lr_p 0.000071 | batch   3840 | examples/s:  94.4 | loss: 0.35300 | time e
lapsed: 04h51m31s | time left: 00h23m49s
epoch  40 | lr 0.000070 |lr_p 0.000071 | batch   5840 | examples/s:  96.8 | loss: 0.38632 | time e
lapsed: 04h52m50s | time left: 00h22m33s
epoch  40 | lr 0.000070 |lr_p 0.000071 | batch   7840 | examples/s: 103.5 | loss: 0.21808 | time e
lapsed: 04h54m09s | time left: 00h21m18s
epoch  40 | lr 0.000070 |lr_p 0.000071 | batch   9840 | examples/s:  84.9 | loss: 0.30779 | time e
lapsed: 04h55m29s | time left: 00h20m02s
Training
epoch  41 | lr 0.000066 |lr_p 0.000068 | batch    436 | examples/s:  93.3 | loss: 0.48145 | time e
lapsed: 04h56m50s | time left: 00h18m46s
epoch  41 | lr 0.000066 |lr_p 0.000068 | batch   2436 | examples/s: 100.3 | loss: 0.67462 | time e
lapsed: 04h58m09s | time left: 00h17m30s
epoch  41 | lr 0.000066 |lr_p 0.000068 | batch   4436 | examples/s: 103.6 | loss: 0.62339 | time e
lapsed: 04h59m28s | time left: 00h16m14s
epoch  41 | lr 0.000066 |lr_p 0.000068 | batch   6436 | examples/s: 102.3 | loss: 0.49116 | time e
lapsed: 05h00m47s | time left: 00h14m58s
epoch  41 | lr 0.000066 |lr_p 0.000068 | batch   8436 | examples/s: 115.6 | loss: 0.33865 | time e
lapsed: 05h02m00s | time left: 00h13m42s
epoch  41 | lr 0.000066 |lr_p 0.000068 | batch  10436 | examples/s: 114.7 | loss: 0.25293 | time e
lapsed: 05h03m10s | time left: 00h12m25s
Training
epoch  42 | lr 0.000062 |lr_p 0.000064 | batch   1032 | examples/s: 116.0 | loss: 0.40150 | time e
lapsed: 05h04m24s | time left: 00h11m09s
epoch  42 | lr 0.000062 |lr_p 0.000064 | batch   3032 | examples/s: 112.3 | loss: 0.54956 | time e
lapsed: 05h05m37s | time left: 00h09m53s
epoch  42 | lr 0.000062 |lr_p 0.000064 | batch   5032 | examples/s: 105.6 | loss: 0.55137 | time e
lapsed: 05h06m50s | time left: 00h08m37s
epoch  42 | lr 0.000062 |lr_p 0.000064 | batch   7032 | examples/s: 114.4 | loss: 0.29713 | time e
lapsed: 05h08m02s | time left: 00h07m21s
epoch  42 | lr 0.000062 |lr_p 0.000064 | batch   9032 | examples/s: 114.0 | loss: 0.45576 | time e
lapsed: 05h09m14s | time left: 00h06m04s
epoch  42 | lr 0.000062 |lr_p 0.000064 | batch  11032 | examples/s: 112.6 | loss: 0.66483 | time e
lapsed: 05h10m26s | time left: 00h04m48s
Training
epoch  43 | lr 0.000058 |lr_p 0.000060 | batch   1628 | examples/s: 113.4 | loss: 0.50200 | time e
lapsed: 05h11m41s | time left: 00h03m32s
epoch  43 | lr 0.000058 |lr_p 0.000060 | batch   3628 | examples/s: 102.4 | loss: 0.68082 | time e
lapsed: 05h12m53s | time left: 00h02m16s
epoch  43 | lr 0.000058 |lr_p 0.000060 | batch   5628 | examples/s: 108.0 | loss: 0.57775 | time e
lapsed: 05h14m05s | time left: 00h01m00s
epoch  43 | lr 0.000058 |lr_p 0.000060 | batch   7628 | examples/s: 113.8 | loss: 0.31813 | time e
lapsed: 05h15m17s | time left: -1h59m45s
epoch  43 | lr 0.000058 |lr_p 0.000060 | batch   9628 | examples/s: 115.1 | loss: 0.35753 | time e
lapsed: 05h16m28s | time left: -1h58m29s
Training
epoch  44 | lr 0.000054 |lr_p 0.000056 | batch    224 | examples/s: 106.0 | loss: 0.64282 | time e
lapsed: 05h17m43s | time left: -1h57m13s
epoch  44 | lr 0.000054 |lr_p 0.000056 | batch   2224 | examples/s: 113.4 | loss: 0.40013 | time e
lapsed: 05h18m56s | time left: -1h55m57s
epoch  44 | lr 0.000054 |lr_p 0.000056 | batch   4224 | examples/s: 113.2 | loss: 0.71669 | time e
lapsed: 05h20m09s | time left: -1h54m42s
epoch  44 | lr 0.000054 |lr_p 0.000056 | batch   6224 | examples/s: 110.8 | loss: 0.43887 | time e
lapsed: 05h21m21s | time left: -1h53m26s
epoch  44 | lr 0.000054 |lr_p 0.000056 | batch   8224 | examples/s: 114.6 | loss: 0.67291 | time e
lapsed: 05h22m33s | time left: -1h52m10s
epoch  44 | lr 0.000054 |lr_p 0.000056 | batch  10224 | examples/s: 113.1 | loss: 0.46929 | time e
lapsed: 05h23m45s | time left: -1h50m54s
Training
epoch  45 | lr 0.000050 |lr_p 0.000052 | batch    820 | examples/s: 101.8 | loss: 0.32622 | time e
lapsed: 05h25m00s | time left: -1h49m38s
epoch  45 | lr 0.000050 |lr_p 0.000052 | batch   2820 | examples/s: 105.3 | loss: 0.63897 | time e
lapsed: 05h26m13s | time left: -1h48m23s
epoch  45 | lr 0.000050 |lr_p 0.000052 | batch   4820 | examples/s: 113.9 | loss: 0.39574 | time e
lapsed: 05h27m25s | time left: -1h47m07s
epoch  45 | lr 0.000050 |lr_p 0.000052 | batch   6820 | examples/s: 117.0 | loss: 0.33405 | time e
lapsed: 05h28m37s | time left: -1h45m51s
epoch  45 | lr 0.000050 |lr_p 0.000052 | batch   8820 | examples/s: 110.3 | loss: 0.86956 | time e
lapsed: 05h29m49s | time left: -1h44m35s
epoch  45 | lr 0.000050 |lr_p 0.000052 | batch  10820 | examples/s: 110.9 | loss: 0.64461 | time e
```

```
lapsed: 05h31m01s | time left: -1h43m20s
Training
epoch  46 | lr 0.000045 |lr_p 0.000048 | batch   1416 | examples/s: 116.4 | loss: 0.38747 | time e
lapsed: 05h32m16s | time left: -1h42m04s
epoch  46 | lr 0.000045 |lr_p 0.000048 | batch   3416 | examples/s: 106.4 | loss: 0.24114 | time e
lapsed: 05h33m28s | time left: -1h40m48s
epoch  46 | lr 0.000045 |lr_p 0.000048 | batch   5416 | examples/s: 113.2 | loss: 0.68167 | time e
lapsed: 05h34m41s | time left: -1h39m33s
epoch  46 | lr 0.000045 |lr_p 0.000048 | batch   7416 | examples/s: 117.3 | loss: 0.28477 | time e
lapsed: 05h35m53s | time left: -1h38m17s
epoch  46 | lr 0.000045 |lr_p 0.000048 | batch   9416 | examples/s: 113.6 | loss: 0.43869 | time e
lapsed: 05h37m05s | time left: -1h37m02s
Training
epoch  47 | lr 0.000041 |lr_p 0.000044 | batch     12 | examples/s: 103.2 | loss: 0.27357 | time e
lapsed: 05h38m20s | time left: -1h35m46s
epoch  47 | lr 0.000041 |lr_p 0.000044 | batch   2012 | examples/s: 109.0 | loss: 0.32599 | time e
lapsed: 05h39m33s | time left: -1h34m31s
epoch  47 | lr 0.000041 |lr_p 0.000044 | batch   4012 | examples/s: 115.9 | loss: 0.54442 | time e
lapsed: 05h40m45s | time left: -1h33m15s
epoch  47 | lr 0.000041 |lr_p 0.000044 | batch   6012 | examples/s: 116.0 | loss: 0.41749 | time e
lapsed: 05h41m57s | time left: -1h32m00s
epoch  47 | lr 0.000041 |lr_p 0.000044 | batch   8012 | examples/s: 115.4 | loss: 0.56326 | time e
lapsed: 05h43m09s | time left: -1h30m44s
epoch  47 | lr 0.000041 |lr_p 0.000044 | batch  10012 | examples/s: 109.6 | loss: 0.22976 | time e
lapsed: 05h44m21s | time left: -1h29m29s
Training
epoch  48 | lr 0.000037 |lr_p 0.000040 | batch    608 | examples/s: 116.7 | loss: 0.30489 | time e
lapsed: 05h45m35s | time left: -1h28m13s
epoch  48 | lr 0.000037 |lr_p 0.000040 | batch   2608 | examples/s: 112.8 | loss: 0.28081 | time e
lapsed: 05h46m48s | time left: -1h26m58s
epoch  48 | lr 0.000037 |lr_p 0.000040 | batch   4608 | examples/s: 112.4 | loss: 0.30178 | time e
lapsed: 05h47m59s | time left: -1h25m43s
epoch  48 | lr 0.000037 |lr_p 0.000040 | batch   6608 | examples/s: 100.4 | loss: 0.49426 | time e
lapsed: 05h49m12s | time left: -1h24m27s
epoch  48 | lr 0.000037 |lr_p 0.000040 | batch   8608 | examples/s: 114.6 | loss: 0.27966 | time e
lapsed: 05h50m23s | time left: -1h23m12s
epoch  48 | lr 0.000037 |lr_p 0.000040 | batch  10608 | examples/s: 111.1 | loss: 0.22168 | time e
lapsed: 05h51m35s | time left: -1h21m57s
Training
epoch  49 | lr 0.000033 |lr_p 0.000036 | batch   1204 | examples/s: 104.9 | loss: 0.47220 | time e
lapsed: 05h52m49s | time left: -1h20m42s
epoch  49 | lr 0.000033 |lr_p 0.000036 | batch   3204 | examples/s: 109.3 | loss: 0.79936 | time e
lapsed: 05h54m02s | time left: -1h19m26s
epoch  49 | lr 0.000033 |lr_p 0.000036 | batch   5204 | examples/s: 111.9 | loss: 0.21000 | time e
lapsed: 05h55m14s | time left: -1h18m11s
epoch  49 | lr 0.000033 |lr_p 0.000036 | batch   7204 | examples/s: 106.9 | loss: 0.38305 | time e
lapsed: 05h56m26s | time left: -1h16m56s
epoch  49 | lr 0.000033 |lr_p 0.000036 | batch   9204 | examples/s: 107.0 | loss: 0.31274 | time e
lapsed: 05h57m38s | time left: -1h15m41s
epoch  49 | lr 0.000033 |lr_p 0.000036 | batch  11204 | examples/s: 113.4 | loss: 0.47945 | time e
lapsed: 05h58m50s | time left: -1h14m26s
```

In [ ]:

```
# pip install 'git+https://github.com/saadnaeem-dev/pytorch-linear-warmup-cosine-annealing-warm-res
```

In [ ]:

```
import os
os.getcwd()
```

Out[ ]:

```
'/mnt/workspace/sunqiao/mymono'
```

In [ ]:

```
pwd
```

Out[ ]:

```
'/mnt/workspace/sunqiao/mymono'
```