ДОДАТОК А

Вихідний код для тренування мережі

```
import os
import random
import torch
import torch.backends.cudnn
import torch.utils.data

import utils.binvox_visualization
import utils.data_loaders
import utils.data_transforms
import utils.network_utils

from datetime import datetime as dt
from tensorboardX import SummaryWriter
from time import time

from core.test import test_net
from models.encoder import Encoder
from models.decoder import Decoder
from models.refiner import Refiner
from models.merger import Merger


def train_net(cfg):
    # Enable the inbuilt cudnn auto-tuner to find the best algo-
rithm to use
    torch.backends.cudnn.benchmark = True

    # Set up data augmentation
    IMG_SIZE = cfg.CONST.IMG_H, cfg.CONST.IMG_W
    CROP_SIZE = cfg.CONST.CROP_IMG_H, cfg.CONST.CROP_IMG_W
    train_transforms = utils.data_transforms.Compose([
        utils.data_transforms.RandomCrop(IMG_SIZE, CROP_SIZE),

utils.data_transforms.RandomBackground(cfg.TRAIN.RANDOM_BG_COLOR_
RANGE),
        utils.data_transforms.ColorJitter(cfg.TRAIN.BRIGHTNESS,
cfg.TRAIN.CONTRAST, cfg.TRAIN.SATURATION),
        utils.data_transforms.RandomNoise(cfg.TRAIN.NOISE_STD),
        utils.data_transforms.Normalize(mean=cfg.DATASET.MEAN,
std=cfg.DATASET.STD),
        utils.data_transforms.RandomFlip(),
        utils.data_transforms.RandomPermuteRGB(),
        utils.data_transforms.ToTensor(),
    ])
    val_transforms = utils.data_transforms.Compose([
```

```
        utils.data_transforms.CenterCrop(IMG_SIZE, CROP_SIZE),
utils.data_transforms.RandomBackground(cfg.TEST.RANDOM_BG_COLOR_R
ANGE),
        utils.data_transforms.Normalize(mean=cfg.DATASET.MEAN,
std=cfg.DATASET.STD),
        utils.data_transforms.ToTensor(),
    ])

    # Set up data loader
    train_dataset_loader =
utils.data_loaders.DATASET_LOADER_MAPPING[cfg.DATASET.TRAIN_DATAS
ET](cfg)
    val_dataset_loader =
utils.data_loaders.DATASET_LOADER_MAPPING[cfg.DATASET.TEST_DATASE
T](cfg)
    train_data_loader =
torch.utils.data.DataLoader(dataset=train_dataset_loader.get_data
set(
        utils.data_loaders.DatasetType.TRAIN,
cfg.CONST.N_VIEWS_RENDERING, train_transforms),
batch_size=cfg.CONST.BATCH_SIZE,
num_workers=cfg.TRAIN.NUM_WORKER,
pin_memory=True,
                                                shuffle=True,
drop_last=True)
    val_data_loader =
torch.utils.data.DataLoader(dataset=val_dataset_loader.get_datase
t(
        utils.data_loaders.DatasetType.VAL,
cfg.CONST.N_VIEWS_RENDERING, val_transforms),
                                                batch_size=1,
                                                num_workers=1,
pin_memory=True,
                                                shuffle=False)

    # Set up networks
    encoder = Encoder(cfg)
    decoder = Decoder(cfg)
    refiner = Refiner(cfg)
    merger = Merger(cfg)
    print('[DEBUG] %s Parameters in Encoder: %d.' % (dt.now(),
utils.network_utils.count_parameters(encoder)))
```

```
    print('[DEBUG] %s Parameters in Decoder: %d.' % (dt.now(),
utils.network_utils.count_parameters(decoder)))
    print('[DEBUG] %s Parameters in Refiner: %d.' % (dt.now(),
utils.network_utils.count_parameters(refiner)))
    print('[DEBUG] %s Parameters in Merger: %d.' % (dt.now(),
utils.network_utils.count_parameters(merger)))

    # Initialize weights of networks
    encoder.apply(utils.network_utils.init_weights)
    decoder.apply(utils.network_utils.init_weights)
    refiner.apply(utils.network_utils.init_weights)
    merger.apply(utils.network_utils.init_weights)

    # Set up solver
    if cfg.TRAIN.POLICY == 'adam':
        encoder_solver = torch.optim.Adam(filter(lambda p:
p.requires_grad, encoder.parameters()),

lr=cfg.TRAIN.ENCODER_LEARNING_RATE,
                                          betas=cfg.TRAIN.BETAS)
        decoder_solver = torch.optim.Adam(decoder.parameters(),

lr=cfg.TRAIN.DECODER_LEARNING_RATE,
                                          betas=cfg.TRAIN.BETAS)
        refiner_solver = torch.optim.Adam(refiner.parameters(),

lr=cfg.TRAIN.REFINER_LEARNING_RATE,
                                          betas=cfg.TRAIN.BETAS)
        merger_solver = torch.optim.Adam(merger.parameters(),
lr=cfg.TRAIN.MERGER_LEARNING_RATE, betas=cfg.TRAIN.BETAS)
    elif cfg.TRAIN.POLICY == 'sgd':
        encoder_solver = torch.optim.SGD(filter(lambda p:
p.requires_grad, encoder.parameters()),

lr=cfg.TRAIN.ENCODER_LEARNING_RATE,
                                         momen-
tum=cfg.TRAIN.MOMENTUM)
        decoder_solver = torch.optim.SGD(decoder.parameters(),

lr=cfg.TRAIN.DECODER_LEARNING_RATE,
                                         momen-
tum=cfg.TRAIN.MOMENTUM)
        refiner_solver = torch.optim.SGD(refiner.parameters(),

lr=cfg.TRAIN.REFINER_LEARNING_RATE,
                                         momen-
tum=cfg.TRAIN.MOMENTUM)
        merger_solver = torch.optim.SGD(merger.parameters(),
```

```
lr=cfg.TRAIN.MERGER_LEARNING_RATE,
                                        momen-
tum=cfg.TRAIN.MOMENTUM)
    else:
        raise Exception('[FATAL] %s Unknown optimizer %s.' %
(dt.now(), cfg.TRAIN.POLICY))

    # Set up learning rate scheduler to decay learning rates dy-
namically
    encoder_lr_scheduler =
torch.optim.lr_scheduler.MultiStepLR(encoder_solver,

milestones=cfg.TRAIN.ENCODER_LR_MILESTONES,

gamma=cfg.TRAIN.GAMMA)
    decoder_lr_scheduler =
torch.optim.lr_scheduler.MultiStepLR(decoder_solver,

milestones=cfg.TRAIN.DECODER_LR_MILESTONES,

gamma=cfg.TRAIN.GAMMA)
    refiner_lr_scheduler =
torch.optim.lr_scheduler.MultiStepLR(refiner_solver,

milestones=cfg.TRAIN.REFINER_LR_MILESTONES,

gamma=cfg.TRAIN.GAMMA)
    merger_lr_scheduler =
torch.optim.lr_scheduler.MultiStepLR(merger_solver,

milestones=cfg.TRAIN.MERGER_LR_MILESTONES,

gamma=cfg.TRAIN.GAMMA)

    if torch.cuda.is_available():
        encoder = torch.nn.DataParallel(encoder).cuda()
        decoder = torch.nn.DataParallel(decoder).cuda()
        refiner = torch.nn.DataParallel(refiner).cuda()
        merger = torch.nn.DataParallel(merger).cuda()

    # Set up loss functions
    bce_loss = torch.nn.BCELoss()

    # Load pretrained model if exists
    init_epoch = 0
    best_iou = -1
    best_epoch = -1
    if 'WEIGHTS' in cfg.CONST and cfg.TRAIN.RESUME_TRAIN:
```

```
        print('[INFO] %s Recovering from %s ...' % (dt.now(),
cfg.CONST.WEIGHTS))
        checkpoint = torch.load(cfg.CONST.WEIGHTS)
        init_epoch = checkpoint['epoch_idx']
        best_iou = checkpoint['best_iou']
        best_epoch = checkpoint['best_epoch']

        encoder.load_state_dict(checkpoint['encoder_state_dict'])
        decoder.load_state_dict(checkpoint['decoder_state_dict'])
        if cfg.NETWORK.USE_REFINER:
            refin-
er.load_state_dict(checkpoint['refiner_state_dict'])
        if cfg.NETWORK.USE_MERGER:
            mer-
ger.load_state_dict(checkpoint['merger_state_dict'])

        print('[INFO] %s Recover complete. Current epoch #%d,
Best IoU = %.4f at epoch #%d.' %
              (dt.now(), init_epoch, best_iou, best_epoch))

    # Summary writer for TensorBoard
    output_dir = os.path.join(cfg.DIR.OUT_PATH, '%s',
dt.now().isoformat())
    log_dir = output_dir % 'logs'
    ckpt_dir = output_dir % 'checkpoints'
    train_writer = SummaryWriter(os.path.join(log_dir, 'train'))
    val_writer = SummaryWriter(os.path.join(log_dir, 'test'))

    # Training loop
    for epoch_idx in range(init_epoch, cfg.TRAIN.NUM_EPOCHES):
        # Tick / tock
        epoch_start_time = time()

        # Batch average meterics
        batch_time = utils.network_utils.AverageMeter()
        data_time = utils.network_utils.AverageMeter()
        encoder_losses = utils.network_utils.AverageMeter()
        refiner_losses = utils.network_utils.AverageMeter()

        # switch models to training mode
        encoder.train()
        decoder.train()
        merger.train()
        refiner.train()

        batch_end_time = time()
        n_batches = len(train_data_loader)
        for batch_idx, (taxonomy_names, sample_names, render-
ing_images,
```

```
                        ground_truth_volumes) in enumer-
ate(train_data_loader):
            # Measure data time
            data_time.update(time() - batch_end_time)

            # Get data from data loader
            rendering_images =
utils.network_utils.var_or_cuda(rendering_images)
            ground_truth_volumes =
utils.network_utils.var_or_cuda(ground_truth_volumes)

            # Train the encoder, decoder, refiner, and merger
            image_features = encoder(rendering_images)
            raw_features, generated_volumes = decod-
er(image_features)

            if cfg.NETWORK.USE_MERGER and epoch_idx >=
cfg.TRAIN.EPOCH_START_USE_MERGER:
                generated_volumes = merger(raw_features, generat-
ed_volumes)
            else:
                generated_volumes = torch.mean(generated_volumes,
dim=1)
            encoder_loss = bce_loss(generated_volumes,
ground_truth_volumes) * 10

            if cfg.NETWORK.USE_REFINER and epoch_idx >=
cfg.TRAIN.EPOCH_START_USE_REFINER:
                generated_volumes = refiner(generated_volumes)
                refiner_loss = bce_loss(generated_volumes,
ground_truth_volumes) * 10
            else:
                refiner_loss = encoder_loss

            # Gradient decent
            encoder.zero_grad()
            decoder.zero_grad()
            refiner.zero_grad()
            merger.zero_grad()

            if cfg.NETWORK.USE_REFINER and epoch_idx >=
cfg.TRAIN.EPOCH_START_USE_REFINER:
                encoder_loss.backward(retain_graph=True)
                refiner_loss.backward()
            else:
                encoder_loss.backward()

            encoder_solver.step()
            decoder_solver.step()
```

```
            refiner_solver.step()
            merger_solver.step()

            # Append loss to average metrics
            encoder_losses.update(encoder_loss.item())
            refiner_losses.update(refiner_loss.item())
            # Append loss to TensorBoard
            n_itr = epoch_idx * n_batches + batch_idx
            train_writer.add_scalar('EncoderDecoder/BatchLoss',
encoder_loss.item(), n_itr)
            train_writer.add_scalar('Refiner/BatchLoss', refin-
er_loss.item(), n_itr)

            # Tick / tock
            batch_time.update(time() - batch_end_time)
            batch_end_time = time()
            print(
                '[INFO] %s [Epoch %d/%d][Batch %d/%d] BatchTime =
%.3f (s) DataTime = %.3f (s) EDLoss = %.4f RLoss = %.4f'
                % (dt.now(), epoch_idx + 1,
cfg.TRAIN.NUM_EPOCHES, batch_idx + 1, n_batches, batch_time.val,
                   data_time.val, encoder_loss.item(), refin-
er_loss.item()))

        # Append epoch loss to TensorBoard
        train_writer.add_scalar('EncoderDecoder/EpochLoss', en-
coder_losses.avg, epoch_idx + 1)
        train_writer.add_scalar('Refiner/EpochLoss', refin-
er_losses.avg, epoch_idx + 1)

        # Adjust learning rate
        encoder_lr_scheduler.step()
        decoder_lr_scheduler.step()
        refiner_lr_scheduler.step()
        merger_lr_scheduler.step()

        # Tick / tock
        epoch_end_time = time()
        print('[INFO] %s Epoch [%d/%d] EpochTime = %.3f (s) ED-
Loss = %.4f RLoss = %.4f' %
              (dt.now(), epoch_idx + 1, cfg.TRAIN.NUM_EPOCHES,
epoch_end_time - epoch_start_time, encoder_losses.avg,
               refiner_losses.avg))

        # Update Rendering Views
        if cfg.TRAIN.UPDATE_N_VIEWS_RENDERING:
            n_views_rendering = random.randint(1,
cfg.CONST.N_VIEWS_RENDERING)
```

```
train_data_loader.dataset.set_n_views_rendering(n_views_rendering
)
            print('[INFO] %s Epoch [%d/%d] Update #RenderingViews
to %d' %
                  (dt.now(), epoch_idx + 2,
cfg.TRAIN.NUM_EPOCHES, n_views_rendering))

        # Validate the training models
        iou = test_net(cfg, epoch_idx + 1, output_dir,
val_data_loader, val_writer, encoder, decoder, refiner, merger)

        # Save weights to file
        if (epoch_idx + 1) % cfg.TRAIN.SAVE_FREQ == 0:
            if not os.path.exists(ckpt_dir):
                os.makedirs(ckpt_dir)

            utils.network_utils.save_checkpoints(cfg,
os.path.join(ckpt_dir, 'ckpt-epoch-%04d.pth' % (epoch_idx + 1)),
                                                 epoch_idx + 1,
encoder, encoder_solver, decoder, decoder_solver,
                                                 refiner, refin-
er_solver, merger, merger_solver, best_iou, best_epoch)
        if iou > best_iou:
            if not os.path.exists(ckpt_dir):
                os.makedirs(ckpt_dir)

            best_iou = iou
            best_epoch = epoch_idx + 1
            utils.network_utils.save_checkpoints(cfg,
os.path.join(ckpt_dir, 'best-ckpt.pth'), epoch_idx + 1, encoder,
                                                 encoder_solver,
decoder, decoder_solver, refiner, refiner_solver,
                                                 merger, mer-
ger_solver, best_iou, best_epoch)

    # Close SummaryWriter for TensorBoard
    train_writer.close()
    val_writer.close()


import json
import numpy as np
import os
import torch
import torch.backends.cudnn
import torch.utils.data

import utils.binvox_visualization
```

```
import utils.data_loaders
import utils.data_transforms
import utils.network_utils

from datetime import datetime as dt

from models.encoder import Encoder
from models.decoder import Decoder
from models.refiner import Refiner
from models.merger import Merger


def test_net(cfg,
             epoch_idx=-1,
             output_dir=None,
             test_data_loader=None,
             test_writer=None,
             encoder=None,
             decoder=None,
             refiner=None,
             merger=None):
    # Enable the inbuilt cudnn auto-tuner to find the best algo-
rithm to use
    torch.backends.cudnn.benchmark = True

    # Load taxonomies of dataset
    taxonomies = []
    with
open(cfg.DATASETS[cfg.DATASET.TEST_DATASET.upper()].TAXONOMY_FILE
_PATH, encoding='utf-8') as file:
        taxonomies = json.loads(file.read())
    taxonomies = {t['taxonomy_id']: t for t in taxonomies}

    # Set up data loader
    if test_data_loader is None:
        # Set up data augmentation
        IMG_SIZE = cfg.CONST.IMG_H, cfg.CONST.IMG_W
        CROP_SIZE = cfg.CONST.CROP_IMG_H, cfg.CONST.CROP_IMG_W
        test_transforms = utils.data_transforms.Compose([
            utils.data_transforms.CenterCrop(IMG_SIZE,
CROP_SIZE),

utils.data_transforms.RandomBackground(cfg.TEST.RANDOM_BG_COLOR_R
ANGE),

utils.data_transforms.Normalize(mean=cfg.DATASET.MEAN,
std=cfg.DATASET.STD),
            utils.data_transforms.ToTensor(),
        ])
```

```
        dataset_loader =
utils.data_loaders.DATASET_LOADER_MAPPING[cfg.DATASET.TEST_DATASE
T](cfg)
        test_data_loader =
torch.utils.data.DataLoader(dataset=dataset_loader.get_dataset(
            utils.data_loaders.DatasetType.TEST,
cfg.CONST.N_VIEWS_RENDERING, test_transforms),

batch_size=1,

num_workers=1,

pin_memory=True,
                                                        shuf-
fle=False)

    # Set up networks
    if decoder is None or encoder is None:
        encoder = Encoder(cfg)
        decoder = Decoder(cfg)
        refiner = Refiner(cfg)
        merger = Merger(cfg)

        if torch.cuda.is_available():
            encoder = torch.nn.DataParallel(encoder).cuda()
            decoder = torch.nn.DataParallel(decoder).cuda()
            refiner = torch.nn.DataParallel(refiner).cuda()
            merger = torch.nn.DataParallel(merger).cuda()

        print('[INFO] %s Loading weights from %s ...' %
(dt.now(), cfg.CONST.WEIGHTS))
        checkpoint = torch.load(cfg.CONST.WEIGHTS)
        epoch_idx = checkpoint['epoch_idx']
        encoder.load_state_dict(checkpoint['encoder_state_dict'])
        decoder.load_state_dict(checkpoint['decoder_state_dict'])

        if cfg.NETWORK.USE_REFINER:
            refin-
er.load_state_dict(checkpoint['refiner_state_dict'])
        if cfg.NETWORK.USE_MERGER:
            mer-
ger.load_state_dict(checkpoint['merger_state_dict'])

    # Set up loss functions
    bce_loss = torch.nn.BCELoss()

    # Testing loop
    n_samples = len(test_data_loader)
```

```
    test_iou = dict()
    encoder_losses = utils.network_utils.AverageMeter()
    refiner_losses = utils.network_utils.AverageMeter()

    # Switch models to evaluation mode
    encoder.eval()
    decoder.eval()
    refiner.eval()
    merger.eval()

    for sample_idx, (taxonomy_id, sample_name, rendering_images,
ground_truth_volume) in enumerate(test_data_loader):
        taxonomy_id = taxonomy_id[0] if isin-
stance(taxonomy_id[0], str) else taxonomy_id[0].item()
        sample_name = sample_name[0]

        with torch.no_grad():
            # Get data from data loader
            rendering_images =
utils.network_utils.var_or_cuda(rendering_images)
            ground_truth_volume =
utils.network_utils.var_or_cuda(ground_truth_volume)

            # Test the encoder, decoder, refiner and merger
            image_features = encoder(rendering_images)
            raw_features, generated_volume = decod-
er(image_features)

            if cfg.NETWORK.USE_MERGER and epoch_idx >=
cfg.TRAIN.EPOCH_START_USE_MERGER:
                generated_volume = merger(raw_features, generat-
ed_volume)
            else:
                generated_volume = torch.mean(generated_volume,
dim=1)
            encoder_loss = bce_loss(generated_volume,
ground_truth_volume) * 10

            if cfg.NETWORK.USE_REFINER and epoch_idx >=
cfg.TRAIN.EPOCH_START_USE_REFINER:
                generated_volume = refiner(generated_volume)
                refiner_loss = bce_loss(generated_volume,
ground_truth_volume) * 10
            else:
                refiner_loss = encoder_loss

            # Append loss and accuracy to average metrics
            encoder_losses.update(encoder_loss.item())
            refiner_losses.update(refiner_loss.item())
```

```
            # IoU per sample
            sample_iou = []
            for th in cfg.TEST.VOXEL_THRESH:
                _volume = torch.ge(generated_volume, th).float()
                intersection =
torch.sum(_volume.mul(ground_truth_volume)).float()
                union =
torch.sum(torch.ge(_volume.add(ground_truth_volume), 1)).float()
                sample_iou.append((intersection / union).item())

            # IoU per taxonomy
            if taxonomy_id not in test_iou:
                test_iou[taxonomy_id] = {'n_samples': 0, 'iou':
[]}
            test_iou[taxonomy_id]['n_samples'] += 1
            test_iou[taxonomy_id]['iou'].append(sample_iou)

            # Append generated volumes to TensorBoard
            if output_dir and sample_idx < 3:
                img_dir = output_dir % 'images'
                # Volume Visualization
                gv = generated_volume.cpu().numpy()
                rendering_views =
utils.binvox_visualization.get_volume_views(gv,
os.path.join(img_dir, 'test'),

epoch_idx)
                test_writer.add_image('Test Sample#%02d/Volume
Reconstructed' % sample_idx, rendering_views, epoch_idx)
                gtv = ground_truth_volume.cpu().numpy()
                rendering_views =
utils.binvox_visualization.get_volume_views(gtv,
os.path.join(img_dir, 'test'),

epoch_idx)
                test_writer.add_image('Test Sample#%02d/Volume
GroundTruth' % sample_idx, rendering_views, epoch_idx)

            # Print sample loss and IoU
            print('[INFO] %s Test[%d/%d] Taxonomy = %s Sample =
%s EDLoss = %.4f RLoss = %.4f IoU = %s' %
                  (dt.now(), sample_idx + 1, n_samples, taxono-
my_id, sample_name, encoder_loss.item(),
                   refiner_loss.item(), ['%.4f' % si for si in
sample_iou]))

    # Output testing results
    mean_iou = []
```

```
    for taxonomy_id in test_iou:
        test_iou[taxonomy_id]['iou'] =
np.mean(test_iou[taxonomy_id]['iou'], axis=0)
        mean_iou.append(test_iou[taxonomy_id]['iou'] *
test_iou[taxonomy_id]['n_samples'])
    mean_iou = np.sum(mean_iou, axis=0) / n_samples

    # Print header
    print('============================ TEST RESULTS
============================')
    print('Taxonomy', end='\t')
    print('#Sample', end='\t')
    print('Baseline', end='\t')
    for th in cfg.TEST.VOXEL_THRESH:
        print('t=%.2f' % th, end='\t')
    print()
    # Print body
    for taxonomy_id in test_iou:
        print('%s' % taxono-
mies[taxonomy_id]['taxonomy_name'].ljust(8), end='\t')
        print('%d' % test_iou[taxonomy_id]['n_samples'],
end='\t')
        if 'baseline' in taxonomies[taxonomy_id]:
            print('%.4f' % taxono-
mies[taxonomy_id]['baseline']['%d-view' %
cfg.CONST.N_VIEWS_RENDERING], end='\t\t')
        else:
            print('N/a', end='\t\t')

        for ti in test_iou[taxonomy_id]['iou']:
            print('%.4f' % ti, end='\t')
        print()
    # Print mean IoU for each threshold
    print('Overall ', end='\t\t\t\t')
    for mi in mean_iou:
        print('%.4f' % mi, end='\t')
    print('\n')

    # Add testing results to TensorBoard
    max_iou = np.max(mean_iou)
    if test_writer is not None:
        test_writer.add_scalar('EncoderDecoder/EpochLoss', encod-
er_losses.avg, epoch_idx)
        test_writer.add_scalar('Refiner/EpochLoss', refin-
er_losses.avg, epoch_idx)
        test_writer.add_scalar('Refiner/IoU', max_iou, epoch_idx)

    return max_iou
```

ДОДАТОК Б

Відомість атестаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	88 с.
	Інші документи	
2.	Презентаційні матеріали	20 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Божченко О.В.			Дослідження нейронних мереж в задачі тривимірної реконструкції
Перевірив		Магдаліна І.В.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец
СШІм-19-1	122
ХНУРЕ кафедра ШІ	