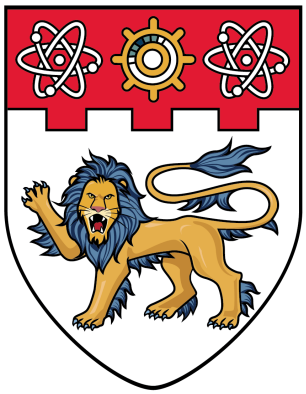

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

AI6121 Project:
Image Translation and UDA

Rao Yixi, G2302775D
Jin Zhixiao, G2303771H
Huang Siwei, G2304149C

School Of Computer Science And Engineering

18/11/2023

Contents

1 Introduction **2**
- 1.1 Image-to-Image (I2I) Translation . . . 2
- 1.2 Unsupervised Domain Adaptation via I2I Translation . . . 2

2 Image-to-Image Translation **3**
- 2.1 Introduction . . . 3
- 2.2 Model Architecture . . . 3
- 2.3 Implementation . . . 4
  - 2.3.1 Dataset & Pre-processing . . . 4
  - 2.3.2 Discriminator Model . . . 5
  - 2.3.3 Generator Model . . . 5
  - 2.3.4 Training . . . 6
- 2.4 Result Discussion . . . 8
  - 2.4.1 Result presentation . . . 8
  - 2.4.2 Major constraints of the CycleGAN . . . 8

3 Unsupervised Domain Adaptation via I2I Translation **12**
- 3.1 Introduction . . . 12
- 3.2 DeepLabV3 . . . 12
  - 3.2.1 Atrous Convolution . . . 12
  - 3.2.2 Atrous Spatial Pyramid Pooling . . . 12
- 3.3 Implementation . . . 13
  - 3.3.1 Dataset & Pre-processing . . . 13
  - 3.3.2 Model . . . 14
  - 3.3.3 Evaluation metrics . . . 15
  - 3.3.4 Testing . . . 15
- 3.4 Settings . . . 16
- 3.5 Result discussion . . . 16

1 Introduction

This project aims to familiarise students with the recent image-to-image translation and unsupervised domain adaptation (UDA) techniques and equip them with hand-on coding experience with deep generative and discriminative networks. It consists of two tasks: 1) Image translation by using CycleGAN or more advanced image translation networks, aiming for a good understanding and practice of the image-to-image translation; 2) Input-space UDA via image translation, which aims to help students have a good understanding and practice of UDA via input-space alignment.

1.1 Image-to-Image (I2I) Translation

For the first task, you are expected to read the paper CycleGAN or other image translation work to have a good understanding of how it works. With that, you need to train an image translation network. You can leverage the open-source codes available on the Internet, and the source and target datasets can be GTA5 (or SYNTHIA) and Cityscapes for semantic segmentation, ICDAR2013 and ICDAR2015 for scene text detection, or other source-target datasets. In the project report, you need to describe your implementation in detail. You are also expected to discuss the major constraints of the image translation network according to your trained model and translated images.

1.2 Unsupervised Domain Adaptation via I2I Translation

For the second task, you are expected to learn and practise UDA via input-space alignment ([1] gives an example). With the image translation model from the first task, you can compare two semantic segmentation models: 1) A Source-only model that is trained with the labelled source data and evaluated over the target data; 2) A domain adaptive semantic segmentation model that is trained with the translated source data and evaluated over the target data. In the project report, you are expected to compare how the two models perform differently and why. You may also explore other translation networks for better UDA performance.

2 Image-to-Image Translation

The image to image translation model we chose is Cycle Generative Adversarial Network (CycleGAN) developed by Zhu et al [2].

2.1 Introduction

For image-to-image translation task, an original image set S and a target image set T are required. For most of the current I2I models, the two image sets used for training need to be paired, meaning that for every image $s \in S$, there must be a corresponding image $t \in T$. However, this kind of paired image set is scarce because (1) Manual production is time-consuming and laborious (2) The target image set required by some vision and graphics tasks does not exist, such as image translation between painters of different styles. The CycleGAN model proposed by Zhu et al. is designed to solve the unpaired image-to-image translation problem.

The CycleGAN model needs to learn a mapping $G : S \rightarrow T$ between an input image and an output image, which can take the special features of the original image set, and then figure out how to transform these features to the target image set, making it difficult to distinguish between the translated image distribution and the target image set distribution. However, this kind of unpaired unidirectional I2I translation will have an under-constrained problem, that is, the translated image distribution $G(s)$ does match the target image set, but it cannot correspond to the original image s in a meaningful way. This is because there are many mappings G that can generate images with the same distribution. Moreover, this unidirectional training also triggers the well-known problem of mode collapse.

Zhu et al. used the property that translation should be "cycle consistent" to define another mapping $F : T \rightarrow S$ to reproject the translated image onto the original image, ensuring that $F(G(s)) = s$ and $G(F(t)) = t$. The model uses two generators and their corresponding discriminators using adversarial loss and defines a novel cycle consistency loss to train the two generators, and discriminators simultaneously. Figure 1 shows the basic design of CycleGAN and the Loss definition.

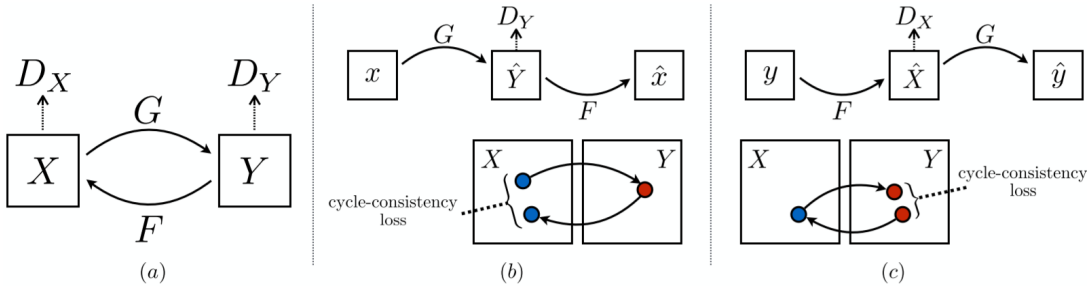

Figure 1: (a) The model contains two mapping functions (generators) and associated adversarial discriminators (b) forward cycle-consistency loss (c) backward cycle-consistency loss.

2.2 Model Architecture

To accomplish I2I, the overall model contains two generators $G : S \rightarrow T$ and $F : T \rightarrow S$ and the corresponding two adversarial discriminators D_S and D_T, which are used to distinguish

between the original image and the translated image. The basic idea is to improve the accuracy of the adversarial discriminators on image sets S or T while improving the image translation ability of the generators. The ideal result is that the discriminators maintain high accuracy on the image sets S or T, but only 50% accuracy in recognizing the fake images translated by the generators.

Adversarial Loss. Adversarial loss is applied to generators and discriminators and is mainly used to ensure that the levels of sets of translations from domain S (T) and domain T (S) are appropriate. Below is the adversarial loss for G and D_T. The other one is similar.

$$\mathscr{L}_{GAN}(G, D_T, S, T) = \mathbb{E}_{t \sim p_{data(t)}}[\log D_T(t)] + \mathbb{E}_{s \sim p_{data(s)}}[\log(1 - D_T(G(s)))] \tag{1}$$

Cycle Consistency Loss. To ensure cycle consistency, a reproject error-like forward cycle consistency loss and backward cycle consistency loss, corresponding to two generators respectively, are used, which are defined as:

$$\mathscr{L}_{cyc}(G, F) = \mathbb{E}_{s \sim p_{data(s)}}[||F(G(s)) - s||_1] + \mathbb{E}_{t \sim p_{data(t)}}[||G(F(t)) - t||_1] \tag{2}$$

Therefore, the full objective function can be defined as:

$$G^*, F^* = \arg\min_{G,F} \max_{D_S, D_T} \mathscr{L}(G, F, D_T, D_S) \tag{3}$$

$$\mathscr{L}(G, F, D_T, D_S) = \mathscr{L}_{GAN}(G, D_T, S, T) + \mathscr{L}_{GAN}(G, D_S, T, S) + \lambda \mathscr{L}_{cyc}(G, F) \tag{4}$$

2.3 Implementation

We have organized the implementation of CycleGAN into four steps. The first step is to define the dataset and pre-processing, the second step is to define the discriminator model, the third step is to define the generator model, and the fourth step is to define the training for CycleGAN. We use Pytorch to implement CycleGAN and refer to the source code of Zhu et al [2] and Aladdin's code reproduction [3]. Here we only present the core codes, please check the source code for detailed implementation.

2.3.1 Dataset & Pre-processing

We use the GTA5 video game street view from [4] as the original dataset S and the real cityscape dataset from [5] as the T dataset, so our goal is to convert the GTA5 video game street view image into a real street view style image. The image augmentation we use is, to resize the image to 180×360, then use horizontal flip on the image with $p = 0.5$, and finally normalize all the image channels to have a mean of 0.5 and standard deviation of 0.5. Finally, convert it to tensor. The detailed implementation is shown below.

```
A.Compose([A.Resize(width=360, height=180),
                    A.HorizontalFlip(p=0.5),
                    A.Normalize(mean=[0.5, 0.5, 0.5],
                                std=[0.5, 0.5, 0.5],
                                max_pixel_value=255),
                    ToTensorV2()])
```

2.3.2 Discriminator Model

The implementation of discriminator architectures follows the paper description of Zhu et al., who use 70 $\times$ 70 PatchGAN as the discriminator model. Define `Ck` as 4 $\times$ 4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2. We use Pytorch's `nn.Module` to define `Ck` block class:

```
self.conv = nn.Sequential(
    nn.Conv2d(
        in_channels,
        out_channels,
        4,
        stride,
        1,
        bias=True,
        padding_mode="reflect"
    ),
    nn.InstanceNorm2d(out_channels),
    nn.LeakyReLU(0.2, inplace=True)
)
```

We then connect four `Ck` blocks in the order of C64-C128-C256-C512 as the discriminator architecture, where the C64 block does not use the InstanceNorm layer. Finally we use a convolution to produce a 1-dimensional output and add the sigmoid function.

2.3.3 Generator Model

The Generator architecture used by Zhu et al. consists of a convolution block and a residual block, where the convolution block is divided into downsampling and upsampling blocks. In the following, we use `nn.Module` define convolution block as:

```
# Downsampling layers or upsampling layers
self.conv = nn.Sequential(
    nn.Conv2d(in_channels,
              out_channels,
              padding_mode="reflect",
              **kwargs)
              if down
              else
              nn.ConvTranspose2d(in_channels,
                                 out_channels,
                                 **kwargs),
    nn.InstanceNorm2d(out_channels),
    nn.ReLU(inplace=True) if use_act else nn.Identity()
)
```

Based on this convolution block class, we can define three types of convolution blocks. The first is the `c7s1-k` block, which uses a 7 $\times$ 7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1. The second is the downsampling block `dk`, which uses a 3 $\times$ 3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. The third is the upsampling block `uk`, which is a 3 $\times$ 3 fractional-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride $\frac{1}{2}$.

For residual block, it is two 3×3 convolutional layers with the same number of filters on both layer. We defined it as the residual block (below) class using `nn.Module`. The `forward` function is `return x +self.block(x)`.

```
self.block = nn.Sequential(
    ConvBlock(channels,
              channels,
              kernel_size=3,
              padding=1),
    ConvBlock(channels,
              channels,
              use_act=False,
              kernel_size=3,
              padding=1)
)
```

We build the generator network as: "c7s1-64,d128,d256,R256,R256,R256,R256,R256,R256, R256, R256, R256, R256, R256, R256, R256, R256, u128 u64,c7s1-3", and followed by a tanh activation function.

```
# c7s1-64
x = self.initial(x)
# d128,d256
for layer in self.down_blocks:
    x = layer(x)
# R256,R256,R256,R256,R256,R256,R256,R256,R256
x = self.res_blocks(x)
# u128 u64
for layer in self.up_blocks:
    x = layer(x)
# c7s1-3
return torch.tanh(self.last(x))
```

2.3.4 Training

When implementing training, we adopted Zhu et al.'s training details and their recommended settings, such as training parameter settings:

- Use a batch size of 2 to prevent out of memory.
- A total of 200 epochs are trained, of which the learning rate is kept constant for 100 epochs, and the decay learning rate strategy is used for the other 100 epochs. Here we use the cosine annealing learning rate strategy.
- Set the initial learning rate to 0.0002
- Set cycle consistent control factor $\lambda = 10$
- Use Adam solver

At the same time, we adopt their recommended improvement techniques, such as using least-squares loss to replace the negative log likelihood objective in $\mathscr{L}_{GAN}$. This loss is more stable during training and generates higher quality results. Therefore, the new goal will be:

- When training G or F: minimize

$$\mathbb{E}_{s\sim p_{data(s)}}[(D_T(G(s))-1)^2]+$$

$$\mathbb{E}_{t\sim p_{data(t)}}[(D_S(F(t))-1)^2]+$$

$$\mathscr{L}_{cyc}(G,F)$$

- When training D_S or D_T: minimise

$$\mathbb{E}_{s\sim p_{data(s)}}[(D_S(s)-1)^2]+\mathbb{E}_{t\sim p_{data(t)}}[D_S(F(t))^2]+$$

$$\mathbb{E}_{t\sim p_{data(t)}}[(D_T(t)-1)^2]+\mathbb{E}_{s\sim p_{data(s)}}[D_T(G(s))^2]$$

Following the above training setting, for each batch images (source, target), we first train the discriminators D_T and D_S. We can obtain two discriminators' result by using corresponding generators.

```
# Patch: DT(T) and DT(GT(S))
DT_real = disc_T(target)
DT_fake = disc_T(gen_T(source))

# Patch: DS(S) and DS(GS(T))
DS_real = disc_S(source)
DS_fake = disc_S(gen_S(target))
```

And then we can calculate the $\mathscr{L}_{GAN}$ losses with respect to discriminators D_T and D_S.

```
# L_GAN(GT,DT,S,T) = minimize Et[(DT(t) - 1)^2] + Es[DT(GT(s))^2]
DT_real_loss = mse(DT_real, torch.ones_like(DT_real))
DT_fake_loss = mse(DT_fake, torch.zeros_like(DT_fake))
# L_GAN(GT,DT,S,T)
DT_loss = DT_real_loss + DT_fake_loss

# L_GAN(GS,DS,S,T) = minimize Es[(DS(s) - 1)^2] + Et[DS(GS(t))^2]
DS_real_loss = mse(DS_real, torch.ones_like(DS_real))
DS_fake_loss = mse(DS_fake, torch.zeros_like(DS_fake))
# L_GAN(GS,DS,S,T)
DS_loss = DS_real_loss + DS_fake_loss
```

Finally, we put it together to get the total adversarial loss, and then train the model with the defined optimizer to opitmize discriminators.

```
# put it togethor: L_GAN(GT,DT,S,T) + L_GAN(GS,DS,S,T)
D_loss = (DT_loss + DS_loss) / 2
```

Next, we need to train the generators G_t and G_S. In the full objective function, the two generator models appear both in adversarial loss and cycle consistency loss. To compute the adversarial loss in terms of generators, the results of the discriminators are first computed for the forward and backward translated images.

```
# Patch: DT(GT(S)) and DS(GS(T))
DT_fake = disc_T(gen_T(source))
DS_fake = disc_S(gen_S(target))
```

Then we need to compute the adversarial loss with respect to generators.

```
# minimize Et[(DT(t) - 1)^2] and Es[(DS(s) - 1)^2]
loss_GT = mse(DT_fake, torch.ones_like(DT_fake))
loss_GS = mse(DS_fake, torch.ones_like(DS_fake))
```

To compute the cycle consistency loss, which only concludes the generators, we first need to get the reprojection of the translated images.

```
# GS(GT(S)) -> S'
cycle_source = gen_S(gen_T(source))
# GT(GS(T)) -> T'
cycle_target = gen_T(gen_S(target))
```

And then calculate the cycle consistency loss according to the definition shown above. Finally, minimising the cycle consistency loss by using the defined optimizer to opitmize the generators model.

```
# Lcyc(GS,GT) = Es[|GS(GT(s)) - s|] + Et[|GT(GS(t)) - t|]
cycle_source_loss = l1(source, cycle_source)
cycle_target_loss = l1(target, cycle_target)

# add all togethor: L(GS,GT,DS,DT) = LGAN(GT,DT,X,Y) + LGAN(GS,DS,X,Y) +
                                     Lcyc(GS,GT)
G_loss = loss_GT + loss_GS +
         lambda * (cycle_source_loss + cycle_target_loss)
```

2.4 Result Discussion

2.4.1 Result presentation

We randomly sampled 3000 images from GTA5 image set and 3000 images from cityscape image set for training and continued to randomly sample 1000 images from the remaining GTA5 image set for testing. Figure 2 are the results of the test which contains the testing results from the training set and the results from the test set. We can observe that our CycleGAN recognizes color and texture variations excellently across different image sets and successfully converts video game cityscapes to real-world style cityscapes. It performs better on the training set than on the test set, which conforms to our expectations. Note that the converted image on the test set seems to be slightly blurry compared to the image on the training set, and there may be some color distortion.

Figure 3 shows CycleGAN's byproduct generator F, which converts real-world cityscape into video game style.

2.4.2 Major constraints of the CycleGAN

Wrong recognization of objects. In Figure 4, we find that CycleGAN has a constraint that recognises the wrong class of objects. For example, it may convert the clouds in the source image into trees in the translated image, or in Figure 4 a, it converts the clouds in the source image into buildings in the translated image, and it may also interpret the cloud above the tree in the source image as a portion of the tree, and then generate a huge tree, such as in Figure 4 c. The reason for this phenomenon may be that CycleGAN is better at achieving color

Figure 2: Our CycleGAN model evaluation results on training set (left) and testing set (right).

Figure 3: The CycleGAN model's byproduct: backward translation results.

and texture changes rather than complex geometric changes, especially clouds and trees with irregular shapes. This limitation was also mentioned by Zhu et al. in [2].

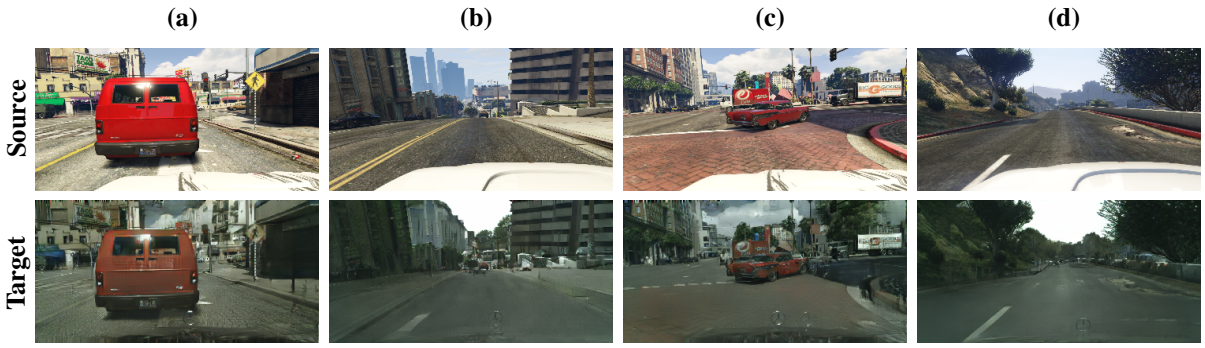

Figure 4: Failure cases of the CycleGAN: Wrong recognization of the class of the object.

Generation of artifact. In Figure 5 we find that CycleGAN may generate some artifacts that do not exist in the real world, such as in the translated images in examples a, b, and c in Figure 5. A strange dark green patch is generated in the sky. Besides, in example b in Figure 5, in the right part of the image, the image which was originally a sunny sky was translated into an image with a dark cloud artifact. I think the reason for this phenomenon may be the poor performance of the learned discriminators, which may mislead the generator, and thus the image with artifacts may be mistakenly recognized as the correct one when performing reprojection during the training process. As a result, balancing the training speed of discirminators and generators and balancing the adversarial loss and the cycle consistency loss is also a constraint for CycleGAN.

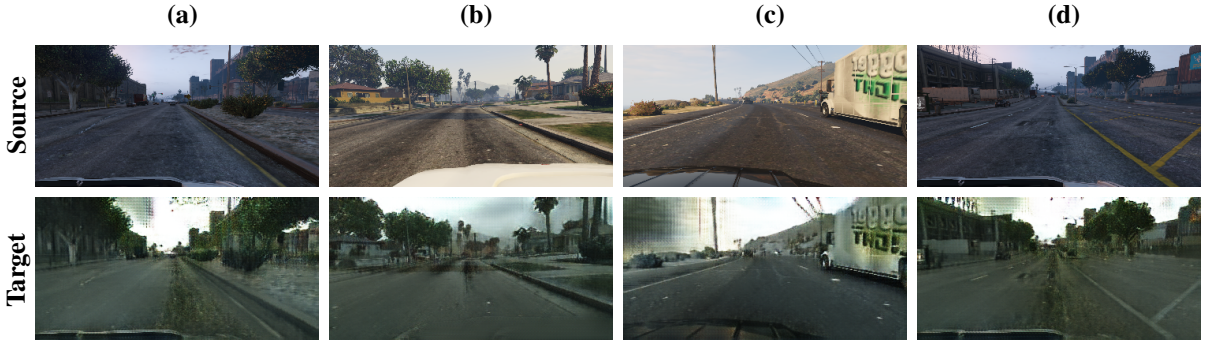

Figure 5: Failure cases of the CycleGAN: Wrong generation of artifact.

Perform badly at night and dusk scenes. In Figure 6, we observe that CycleGAN performs poorly in night and dusk scenes. In the night scene, it translates the dark sky into the hue that does not exist in real-world cityscapes. In the dusk scene, it translates the orange sky into a sunny sky, failing to accurately capture the temporal information of the original image. The reason for this phenomenon may be the fact that the training set does not contain such images, or there are a few pairs of such training images compared to daytime images. The generalizability of CycleGAN is also very weak compared to other models using the aligned paired training method.

The fuzzy images and Color distortion. In the above Figures 4, 5, and 6, we can see that the images generated on the test set have less clarity compared to the source images and there

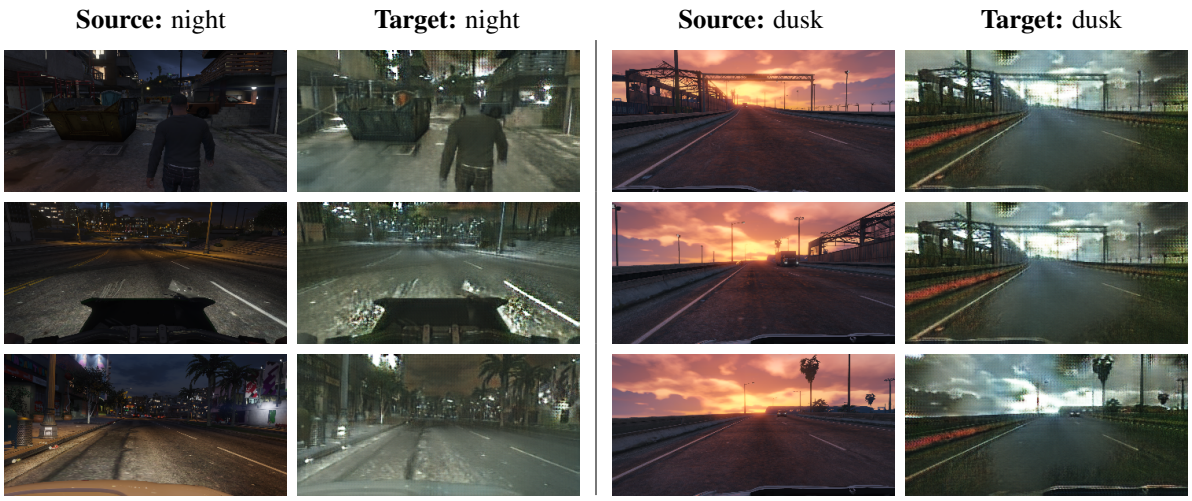

Figure 6: Failure cases of the CycleGAN: Performs poorly in night and dusk scenes.

will be some color distortion. The reason for this phenomenon is that we used too few images in the training set or too few epochs. Thus we can conclude that although CycleGAN can learn mappings without paired images, it still needs a lot of data to achieve good results in some cases, especially when the target area changes greatly.

Intrinsic constraints of CycleGAN. The mapping generated by CycleGAN is a kind of deterministic mapping because cycle consistency enforces G, and F to be inverses of each other [6]. When dealing with complex cross-domain datasets, this deterministic mapping causes CycleGAN to learn an arbitrary one-to-one mapping instead of capturing the true, structured conditional distribution more faithfully [6]. At the same time, CycleGAN still does not eliminate the mode collapse problem. It still exists that the generator tends to generate similar or identical outputs while ignoring some subtle differences in the inputs. This may lead to a lack of diversity in the generated results.

3 Unsupervised Domain Adaptation via I2I Translation

3.1 Introduction

In the application of computer vision model, the ability of domain shift is seen as an important evaluation index. Because supervised models are usually trained under pairs of training image set and corresponding labels. However, there exists plenty of wild images with different distribution and patterns from training set. That's say these models may only perform well in limited domain. And if we want to obtain a model with enough versatility, we need to include enough images and label them for the training. That's a big cost. So researchers are trying to find a way to implement one model with limited data and adapt it to other domains.

The idea of Cycada [1] is to apply unsupervised domain adaption (UDA) to images semantic segmentation to improve its versatility. They first use image-to-image translation mode to implement domain shift on training set. Then they train two image semantic segmentation model. One is on original training domain and label, another is on target domain with original label. The second model have better performance on semantic segmentation task in target domain. By mapping source domain to target domain, we find a way to adapt model to other tasks without additional labeling.

In the first task we implement the unpaired stylish image translation between GTA5 set and Cityscape set. In the second task we utilize the translation results to train the semantic segmentation model, and compare its performance on target domain with the model trained on source image and label.

3.2 DeepLabV3

The model we use for semantic segmentation is DeepLabV3 [7]. The author of DeepLabV3 wants to solve two challenges in application of DCNN to the field of semantic segmentation. The first one is the reduced feature resolution caused by consecutive pooling operations or convolution striding. This invariance to local image transformation may impede dense prediction tasks, where detailed spatial information is desired. Another is the poor performance in detecting multiple objects at different scales. To solve them, the paper proposes Atrous Spatial Pyramid Pooling (ASPP) in DeepLabV3.

3.2.1 Atrous Convolution

Atrous convolution is a technique widely applied in semantic segmentation, also known as dilated convolution. It can help extract denser feature maps by removing the downsampling operations from the last few layers and upsampling the corresponding filter kernels, equivalent to inserting holes between filter weights. Atrous convoltion allows us to effectively enlarge the field of view of filters to incorporate multi-scale context [8] without expanding computation.

3.2.2 Atrous Spatial Pyramid Pooling

Atrous Spatial Pyramid Pooling is Spatial Pyramid Pooling(SPP) combined with atrous convolution. SPP was first proposed to handle the issue that convolution network can only get fix-sized images [9]. It proved to be an effective way to resample features at different scales for accurately and efficiently classifying regions of an arbitrary scale. In DeepLabV3, author

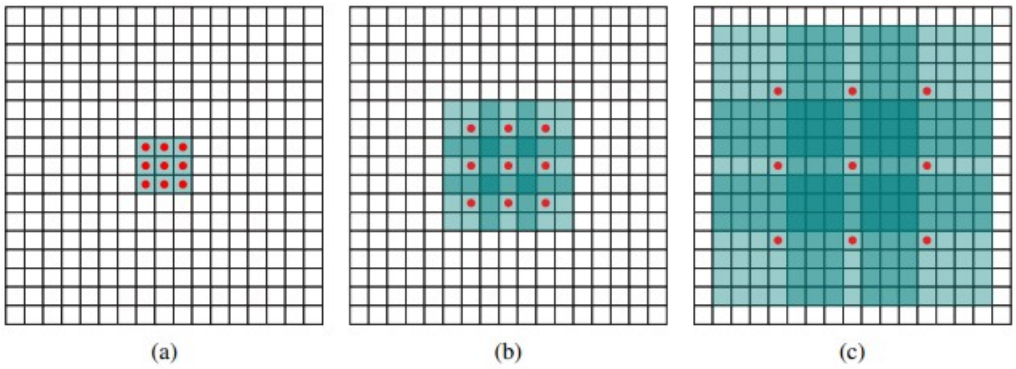

Figure 7: Systematic atrous supports exponential expansion of the receptive field without loss of resolution or coverage

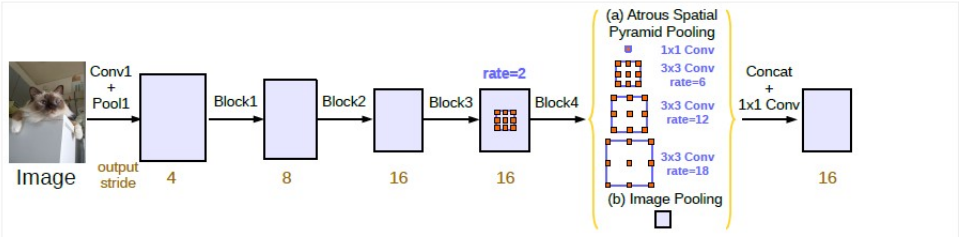

Figure 8: Parallel modules with ASPP, augmented with image-level features.

explore atrous convolution as a context module and tool for spatial pyramid pooling, and add batch normalize to the model. To solve the filter degeneration problem and incorporate global context information to the model, image-level features are adopted, as shown in Figure 8.

3.3 Implementation

Our implementation of DeepLab V3 is largely based on [10], and we make some adaptations in order to train the model on our data set. As code for testing is not included in [10], we also implement a script for model testing. The pseudo code for the overall procedure of training, validation and testing is shown in Algorithm 1, and the details will be discussed in following sections.

3.3.1 Dataset & Pre-processing

The source-only model use original GTA5 data set from [4] for training and validation, and use cityscape data set from [5] for testing. On the other hand, the domain adaptive semantic segmentation model is trained on translated cityscape-style GTA5 images obtained from the first task, and also evaluated on cityscape data set.

As GTA5 and cityscape data set share the same label space, we follow the pre-processing procedure for cityscape in [10]. We define three dataloaders to load data sets: train_loader, val_loader and test_loader. For training images and labels, we perform random flip, crop and gaussian blur, and normalize with given parameters. For validation and testing sets, we adjust

Algorithm 1 Overall procedure for Semantic Segmentation

Objective: Train a semantic segmentation model and evaluate performance
Input: Training set $train\_set$, validation set val_set, testing set $test\_set$
Parameter: Parameter sets, including training epoch e.

Load $train\_set$, val_set, $test\_set$ and perform pre-processing.
Initialize the DeepLab model M.
for $i = 1, 2, ..., e$ **do**
    Train M on $train\_set$.
    Validate M on val_set, evaluate with evaluation metrics.
    if M is the best model in validation so far **then**
        Save parameters of M.
    end if
end for
Load the best model M.
Evaluate M on $test\_set$ with evaluation metrics.
Save processed images.

the size of images and labels and perform normalization with the same parameters. The pre-processing of input images can better augment its feature.

3.3.2 Model

The DeepLab model consists of three main parts: a pre-trained backbone network, the Atrous Spatial Pyramid Pooling (ASPP) module, and a decoder. The input sample is first put into the backbone network for feature extraction to obtain a feature map and a low-level feature map from the first layer of the backbone network. The ASPP module uses atrous convolution method to perform convolution on the feature map to obtain different context information. Afterwards, the decoder combines the processed feature map and the low-level feature map in order to achieve better semantic segmentation accuracy. Finally, bilinear interpolation is performed on the feature map to make it the same size of the input image. We compute the cross entropy loss between model output and labels, and optimize with stochastic gradient decent method. For backbone networks, we choose pre-trained ResNet101. According to [11], with the idea of residual blocks, ResNet can build more network layers without performance loss, thus extracting more information from images. The main part of the definition of the DeepLab model is shown below. The parameter num_classes defines the dimension of model output, and can be adjusted according to target label spaces. In both GTA5 and cityscapes data set, the number of classes is 19.

```
class DeepLab(nn.Module):
    def __init__(self, backbone, output_stride=16, num_classes):
        super(DeepLab, self).__init__()
        BatchNorm = nn.BatchNorm2d
        self.backbone = build_backbone(backbone, output_stride, BatchNorm)
        self.aspp = build_aspp(backbone, output_stride, BatchNorm)
        self.decoder = build_decoder(num_classes, backbone, BatchNorm)
    def forward(self, input):
        x, low_level_feat = self.backbone(input)
        x = self.aspp(x)
```

```
        x = self.decoder(x, low_level_feat)
        x = F.interpolate(x, size=input.size()[2:], mode='bilinear',
                                                    align_corners=True)

        return x
```

3.3.3 Evaluation metrics

As for model evaluation, we adopt four widely-used evaluation metrics in semantic segmentation to obtain a comprehensive result.

- Pixel Accuracy (PA): the most simple metric, which is the proportion of right predicted pixels.

$$PA = \frac{\sum_{i=0}^{k} p_{ii}}{\sum_{i=0}^{k} \sum_{j=0}^{k} p_{ij}}$$

- Mean Pixel Accuracy (MPA): an improved version of PA. First compute the proportion of right predicted pixels of each category, and take average.

$$MPA = \frac{1}{k+1} \sum_{i=0}^{k} \frac{p_{ii}}{\sum_{i=0}^{k} \sum_{j=0}^{k} p_{ij}}$$

- Mean Intersection over Union (MIoU): a standard metric for semantic segmentation. Compute the intersection over union(IoU) of ground truth and predicted result for each category and take average.

$$MIoU = \frac{1}{k+1} \sum_{i=0}^{k} \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k} p_{ji} - p_{ii}}$$

- Frequency Weighted Intersection over Union (FWIoU): a revised version of MIoU. Each category is assigned a weight based on its appearance frequency and computer the weighted average of IoU.

$$FWMIoU = \frac{1}{\sum_{i=0}^{k} \sum_{j=0}^{k} p_{ij}} \sum_{i=0}^{k} \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k} p_{ji} - p_{ii}}$$

MIoU is the most popular evaluation metrics in semantic segmentation, but in some data sets, the distribution of target categories may be imbalanced, so we may need to consider FWIoU at the same time. In every evaluation step, we generate the confusion matrix of target and prediction, and use it to compute the four metrics above.

3.3.4 Testing

After training and validation, we have obtained the best model. In testing, we load the saved model parameters and evaluate on testing set with aforementioned evaluation metrics to obtain quantitative results. We also visualize output images for qualitative analysis and comparison.

3.4 Settings

In the training and validation, we define our own data set. The parameter settings mainly follows the suggested setting in [10], and we make some changes according to our data set and time and GPU limitations. The details are as follows:

- To train the source-only model, we use 2000 images from GTA5 data set, 1700 for training and 300 for validation. For the domain adaptive model, the training and validation set are translated version of the 2000 GTA5 images, obtained from the CycleGAN from Task 1. Both testing set are 316 original images from cityscape data set.
- We compute the cross entropy loss between model output and labels, and use stochastic gradient descent optimizer.
- The suggested training epoch for cityscape-style data sets is 200, but it is hard to implement due to our time and GPU limitations. In some tentative training, we find that on our relatively small data set, the model converges quickly. The train and test loss remain stable after around 10 epochs for the source-only model, so we set training epoch at 15. However, it takes several more epochs for the domain adaptive model to converge, so we set training epoch at 20.
- After each epoch we perform a quick validation, and save checkpoints if the model outperforms previous best in MIoU.
- The learning rate is initially set to 0.01, and decay with poly learning rate strategy, and the power of decay is set to 0.9.
- The random crop size for training images is set to 513, which means training images are cropped into 513×513 patches.

3.5 Result discussion

Here we present some testing results to compare the source-only model and the domain adaptive model with aforementioned semantic segmentation metrics.

As is shown in Table 1, there is a significant performance drop on validation and testing set. This is due to the domain difference between the training, validation and testing data set. In our experiment setting, training set and validation set are from the same data set, thus they share the same domain and similar features. The categories that appear in training set take on similar feature distribution in validation set, so the model can make a correct detection and segmentation. However, the testing set is from another data set, and is from a greatly different domain. Take GTA5 and cityscape data set as an example. The two data sets share the same label space, and they are both images of city scenes. However GTA5 data set is synthetic images taken from videogames, while cityscape images are taken from the real world. Differences including illumination changes, viewpoint changes and so on can greatly change the feature distribution of images. A category in GTA5 may look greatly different from the same category in cityscape. As a result, the information the model learned from GTA5 transfer poorly to cityscape data set, leading to a significant performance loss.

As is shown in Table 2, domain adaptive model outperforms source-only model and achieves higher scores in all four evaluation metrics. Based on the analysis above, the domain difference

	PA	MPA	MIoU	FWMIoU
val	0.913	0.533	0.462	0.850
test	0.226	0.078	0.015	0.224

Table 1: Comparison of validation and testing results

	PA	MPA	MIoU	FWMIoU
source-only	0.226	0.078	0.015	0.224
domain adaptive	0.272	0.084	0.018	0.270

Table 2: Results on test set with ResNet101 backbone.

between the training data set and the testing data set greatly contributes to the performance drop. But with the I2I translation model trained from the first task, we have learned a mapping function that can convert GTA5 images into a cityscape style. After the translation, the image looks more 'cityscape', and the source domain is closer to the target domain. The image features are more consistent between the source domain and target domain, so the semantic segmentation model can better transfer what it has learned from the training set to the testing set. As a result, domain adaptive model achieves better performance than the source-only model.

Figure 9 shows some examples of test output for comparison. The domain adaptive model produces a better result than the source-only model. We have noticed that the domain adaptive model especially does better in segmenting 'road' and 'sidewalk'. In GTA5 images, the roads appears more coarse with many holes, while in cityscape images, the roads look more smooth and plain. This performance improvement on certain categories further proves the effectiveness of domain adaptation.

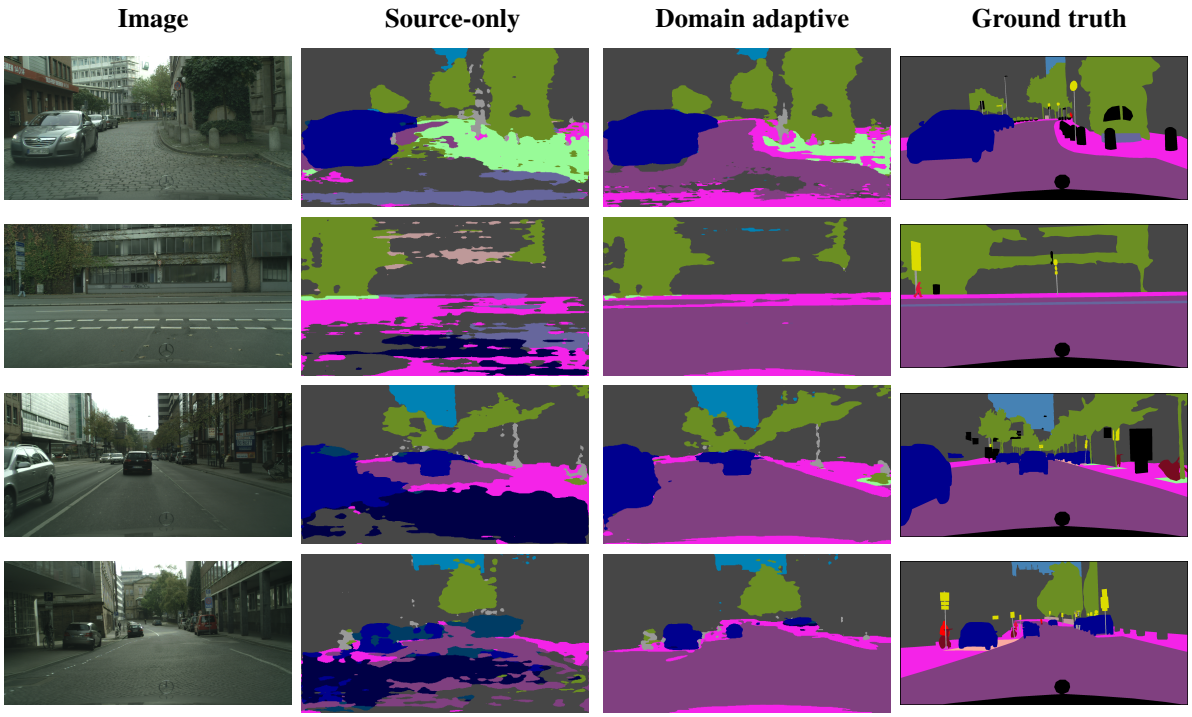

Figure 9: Visualization results on test set.

References

[1] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," in *International conference on machine learning*. Pmlr, 2018, pp. 1989–1998.

[2] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[3] A. Persson, "A clean, simple and readable implementation of cyclegan in pytorch," https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/GANs/CycleGAN, accessed: 2023-11-10.

[4] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," 2016.

[5] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[6] A. Almahairi, S. Rajeshwar, A. Sordoni, P. Bachman, and A. Courville, "Augmented cyclegan: Learning many-to-many mappings from unpaired data," in *International conference on machine learning*. PMLR, 2018, pp. 195–204.

[7] L. C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017.

[8] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[10] jfzhang95, "pytorch-deeplab-xception," https://github.com/jfzhang95/pytorch-deeplab-xception/tree/master, accessed: 2023-11-10.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE*, 2016.