DomainAdoptation-ClassificationAndSegmentation

June 7, 2022

0.1 HW 4 - Domain Adaptation

0.1.1 Section 1 - Classification

[1] (a) **Implement a "Gradient Reversal Layer"**. Refer to the paper by Ganin et al. [3]. The main idea is that we want the feature extractor to be domain invariant, which can be enforced by reversing the gradient characteristics emerging from a "Domain Classifer" or an "Adversarial Network". There are multiple ways to do this. One way is to define a "hook" (refer to PyTorch docs) attached to the input of the adversial network which is tasked with inverting the gradient direction. Another way is to define a network module (Check `AdversarialLayer` in `classification/network.py`), with the forward function as the identity and the backward function reversing the gradients. Formally, you have to define a function GRL(`grad`) = -1 * `grad`, where `grad` corresponds to the gradient $\frac{\partial y}{\partial x}$. Here, y is the output of the adversarial network and x is the input. Look out for "IMPLEMENT YOUR CODE HERE" in the code. **[5 points]**

`Paste your implementation of class AdversarialLayer here. Include everything that you think is relevant.`

```
# Implementation of the Gradient Reversal Function
class AdversarialLayer(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, c = 1):
        ctx.c = c
        return x

    @staticmethod
    def backward(ctx, grad_output):
        out = - ctx.c * grad_output
        return out, None
```

Additional Changes: In the training script, define c=1 and replace grl.apply(features) with grl.apply(features, c)

[1] (b) We can also slowly ramp up the gradients from the adversarial network for better optimization. Implement a gradient reversal layer, with the following strategy: GRL(`grad`) = -c × `grad`, $c = \frac{2}{1+e^{\frac{-\alpha i}{\max_i}}} - 1$. Here i is the iteration number and $\max_i$ is the maximum number of iterations. Take $\alpha = 10$. **[3 points]**

`Expand your previous implementation of class AdversarialLayer here, with an added option for the ramp-up strategy.`

```python
# Implementation of the Gradient Reversal Function remains same as the previous implementation
class AdversarialLayer(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, c = 1):
        ctx.c = c
        return x

    @staticmethod
    def backward(ctx, grad_output):
        out = - ctx.c * grad_output
        return out, None
```

Additional Changes: In the training script, define c and replace grl.apply(features) with grl.apply(features, c)

```python
# Applying Ramp Up Strategy
alpha=10
if args.ramp_up:
    c = 2.0/(1 + np.exp(-iter_num*alpha/args.max_iteration)) - 1
else:
    c = 1
...
domain_predicted = discriminator_net(grl.apply(features, c), torch.softmax(logits, dim=1).deta
...
```

[1] (c) Why do you think the ramp-up strategy would help with the optimization? **[3 points]**

`Answer`

The ramp-up strategy allows the model to suppress the noisy signal from the domain classifier to affect the training of the feature extractor during the early stages of training. Once the domain classifier gets better at its task, the gradient reversal parameter gradually increases to ensure training of the feature extractor to be more domain invariant.

The ramp-up strategy allows optimal tradeoff between the training objectives of minimizing the classification loss and maximising the domain (discriminator) classifier loss.

[2] (a) **Implement CDAN architecture**. We provide the class defintion for Domain adversarial neural network (DANN) as `discriminatorDANN` in `classification/network.py`. Inspired by this, implement a Conditional Domain Adversarial Network (CDAN) as class `discriminatorCDAN` in the same file. The network architecture of DANN and CDAN should be kept the same. The only difference between DANN and CDAN is input to the forward function of CDAN. Implement the multilinear conditioning $f \bigotimes g$ defined in equation 4 in [1] and feed the result into the forward pass of the discriminator. Look out for "IMPLEMENT YOUR CODE HERE" in the code. **[5 points]**

`Paste your code for forward function of class discriminatorCDAN here.`

```python
# Forward Function for CDAN discriminator
def forward(self, x, y):
    ## The descriminator is conditioned on both the features(x) and the output logits(y) of th
```

```
# Find the multilinear map of x and y
# print('x.size(), y.size()', x.size(), y.size())
multiLinearMapOut = torch.bmm(y.unsqueeze(2), x.unsqueeze(1))
# print("default", multiLinearMapOut.size())
multiLinearMapOut = multiLinearMapOut.view(-1, y.size(1) * x.size(1))
# print("resized", multiLinearMapOut.size())

f2 = self.fc1(multiLinearMapOut)
f = self.fc2_3(f2)
return f
```

[2] (b) Briefly explain the advantages of using CDAN over DANN. **[3 points]**

`Answer` CDAN adopts the features over both the feature representations(f) and the label classifier predictions(g). The label classifier predictions are found to contain discriminative information which potentially revelas the multi modal structure within the data and can be conditioned on when adapting the features to get more domain invariant features.

On the other hand, DANN adopts the features only over the feature representation which misses the above advantages and might not be enough for domain adaptation. This has the risk of aligning only a fraction of underlying distribution shifts between source and target data.

CDAN also performs Entropy conditioning which leads to equal importance for different examples and classes and leads to better domain adoptation

[3] (a) **Performance comparison**. Run the training script provided in `train.py` and report the adaptation accuracy in a tabular format. We provide numbers for source-only and target-only supervised models. Report the classification accuracy of DANN and CDAN on three transfer tasks, 1) $A \rightarrow D$, 2) $D \rightarrow W$ and, 3) $A \rightarrow D$ on `office-31` dataset for unsupervised domain adaptation. (A - Amazon, D - DSLR and W - Webcam). Please correctly specify the dataset directory `data_dir` and other arguments when running train.py, e.g. `python train.py --data_dir /datasets/cs252d-sp22-a00-public/hw4_data/office31`. Check `opts.py` for input arguments of `train.py`. The accuracy reported in the table is the classification accuracy over all the classes in the target domain. We have provided the evaluation script in `eval.py`. You can simply specify the arguments and run it to get the required accuracy computed. **[15 points]**

Method	A -> W	D -> W	A -> D
Source Only	56.60	96.35	61.85
DANN	84.53	95.09	83.94
CDAN	90.19	98.49	88.35
Target Supervised	99.25	99.25	99.5

[3] (b) What are your observations on the final table? Please briefly explain it. **[5 points]**

`Answer`

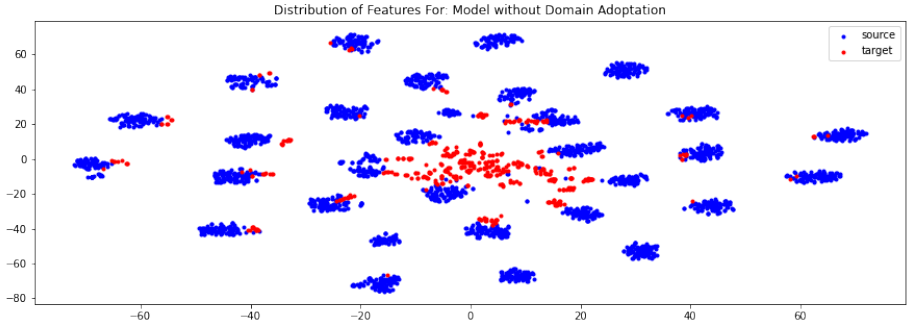
Both DANN and CDAN improve the model performance substantially and are able to model the target dataset distribution effectively. The CDAN shows greater improvement.

The images from DSLR and Webcam datasets are very similar and a model trained on source

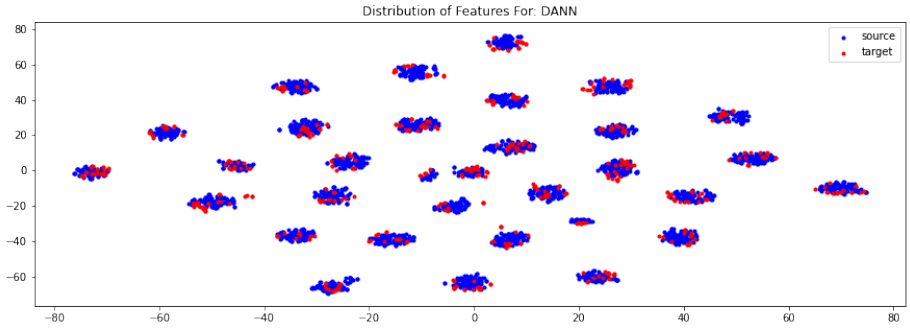
dataset(DSLR) itself performs well on the target dataset(Webcam). Even in this scenario, the CDAN is able to improve model performance. One possible reason could be that the adversarial learning is making the model trained on DSLR to adjust to the different image properties(contrast/brightness/resolution) of the Webcam images.

[4] (a) **tSNE visualization**. Train DANN and CDAN on task $A \rightarrow D$ of `office-31`. Use your implementation of GRL and CDAN with a resnet-50 backbone (–resnet=50). Plot the tSNE visualization of the feature embeddings f extracted by CDAN and DANN on A and D, before and after adaptation. To extract the feature embeddings for case of before adaptation, we provide weights for a pre-trained model in `/datasets/cs252d-sp22-a00-public/hw4_data/SourceOnly_amazon2dslr/best_model.pth.tar`. (A - Amazon, D - DSLR and W - Webcam). Specifically, you need to plot three figures: (1) Before adaptation (trained on source domain A only) (2) DANN adaptation. (3) CDAN adaptation. Each figure should visualize feature embeddings on both of the source domain A (in blue) and target domain D (in red) respectively. See Fig. 3 in [1] for a reference. **[6 points]**

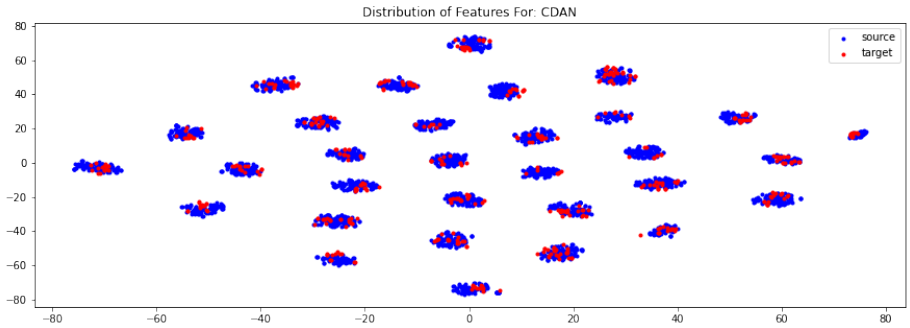
`Paste your plots here.` 1. Feature Embedding for model without Domain Adoptation



2. Feature Embedding for DANN



3. Feature Embedding for CDAN



[4] (b) What are your observations on the plots? Please briefly explain it. **[3 points]**

`Answer`
The feature representations generated by the base model without domain adoptation are very noisy and the source and target data distritbution are very different. This leads to poor classification as expected.

On the other hand, DANN and CDAN generate highly correlated feature representations and both source and target data distributions are very similar. DANN features has a few outliers i.e represnetation which is outside a cluster whereas in CDAN all the feature representations clearly belong to a particular cluster

0.1.2 Section 2 - Segmentation

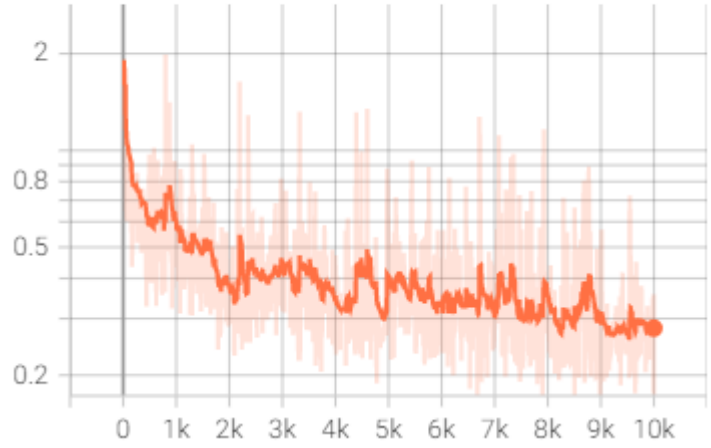
[5] **Implement Adversarial Loss**. You have to implement the multi-level adversarial loss used by Tsai el al. in [2] (Refer to equations 3,4,5). The resnet-based segmentation network generates feature maps at two different resolutions. For each of the resolution, implement the segmentation loss and the adversarial loss. Check the scaffold provided in `segmentation/train_gta2cityscapesmulti.py`. Train the model in the Vanilla-Gan setup (args.gan = 'Vanilla') and use the evaluation script in `evaluate_cityscapes.py`. The datasets are in `/datasets/cs252d-sp22-a00-public/hw4_data`. Look out for "IMPLEMENT THIS" in the train loop. **[12 points]**

`Plot each component of the loss curve (i.e. segmentation loss and adversarial loss at each resolution) here and report the accuacy.`

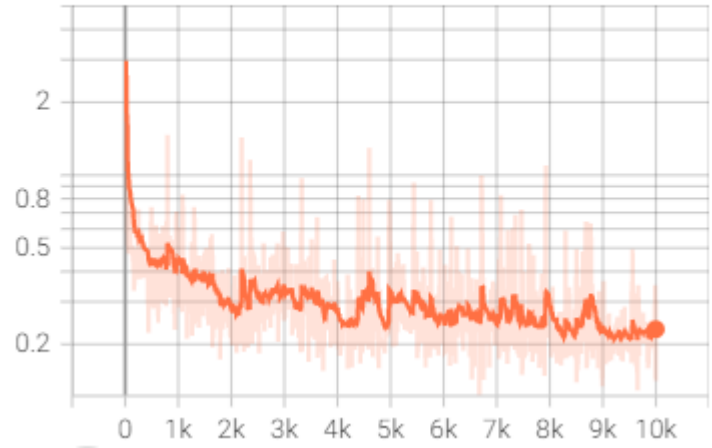
0.1.3 Loss Curves

The x-axis denotes number of iterations the model is trained for and the y-axis denotes the loss.
1. Plots for Segmentation Losses

seg_loss_1
tag: Loss/seg_loss_1

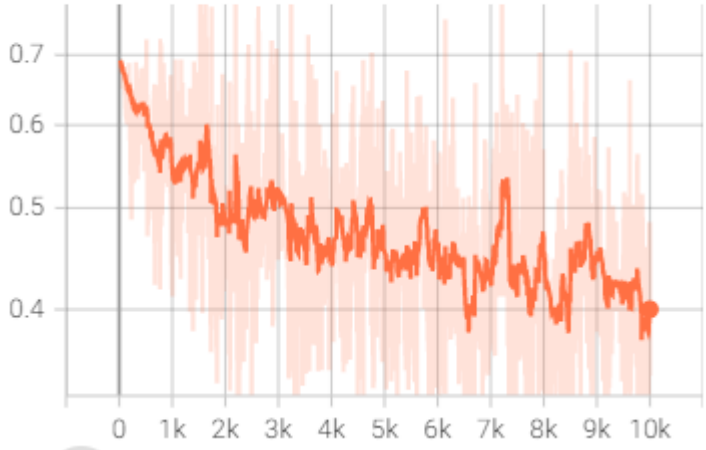


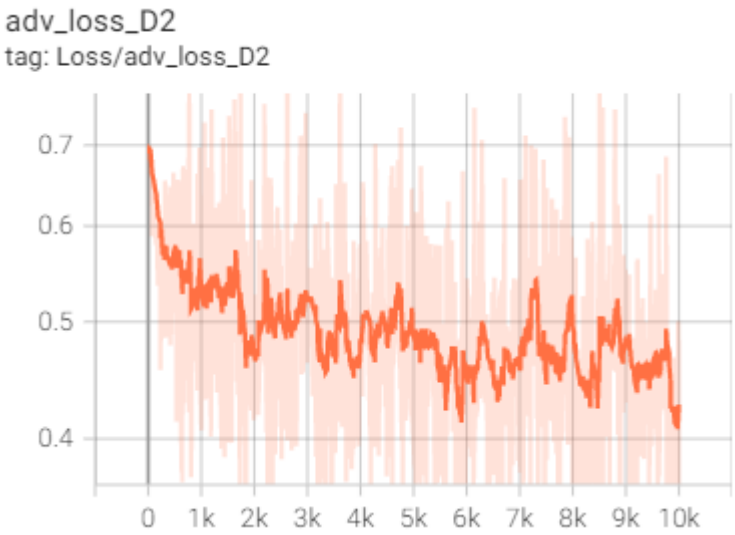
seg_loss_2
tag: Loss/seg_loss_2



2. Plots for Adverserial Losses

adv_loss_D1
tag: Loss/adv_loss_D1





Class	IoU
road	69.12
sidewalk	21.29
building	73.87
wall	20.69
fence	12.34
pole	18.78
light	15.81
sign	14.14
vegetation	78.73
terrain	17.65
sky	73.69
person	48.42
rider	21.83
car	77.05
truck	33.58
bus	8.95
train	1.86
motocycle	18.92
bicycle	8.75
mIoU	33.45

Reported IoU scores

0.1.4 References

1. Conditional Adversarial Domain Adaptation
2. Learning to Adapt Structured Output Space for Semantic Segmentation
3. Unsupervised Domain Adaptation by Backpropagation

Open Source Code References 1. https://github.com/wasidennis/AdaptSegNet/ 2. https://github.com/thuml/CDAN/

[ ]: