Vision and Learning Lab
Prof. Yu-Chiang Frank Wang

Deep Learning for Computer Vision

Homework 3

Submission Date: 05. December 2019

Chiara Pullem · A08323104 · chiara.pullem@gmx.de

Problem 1: GAN

Part 1

For implementing GAN, the archtitecture of the pytorch DCGAN tutorial was used. The generator consists of transposed convolutional layers combined with batch normalization and ReLU activation. For the discriminator, convolutional layers and batch normalization is implemented as well as LeakyReLU for activation. The detailed structure is given in the following.

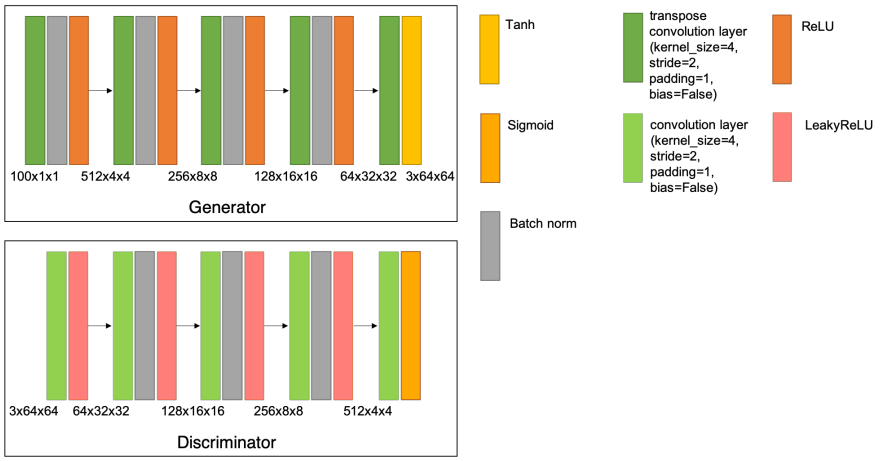


Figure 1: Architecture of GAN

The real images were normalized with a mean $= [0.5, 0.5, 0.5]$ and a standard derivation std $= [0.5, 0.5, 0.5]$. The noise serving as input for the generator was sampled using gaussian distribution. During training the tips given in the slides as well as in the github blog post are applied. Mini batches of real and fake data were used for training and the loss function was modified by training the generator with flipped labels.

Part 2

In the following, 32 randomly sampled images are plotted.

Figure 2: Faces generated by GAN

Part 3

As already mentioned in the slides, GAN models are not so easy to train. The main problem was to balance training of discriminator and generator. The first step was to adapt the learning rate. A slow decrease resulted in a more stable training.

For further balancing, the training of the generator was slightly adapted. The discriminator is trained on two mini batches, one batch with real data and one batch with fake data, within each iteration through the data loader. Aiming to improve the training, the training code of the generator was adapted to ensure that it is also trained on two mini batches instead of one. This was realized by sampling noise of two time the batch size.

These adaptions lead to a more stable training compared to the first runs. Nevertheless, the losses had quite high variances and converged only around a generator loss of 1.5.

Problem 2: ACGAN

Part 1

In order to implement ACGAN, the structure of the previous problem was used and extended. An overview on the architecture is given in the following plot.

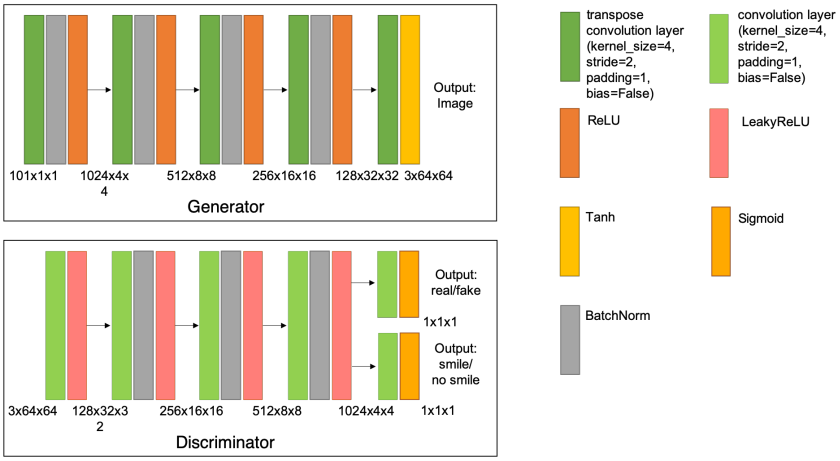


Figure 3: Architecture of ACGAN

The main difference between the two GAN implementations is that labels are used for training ACGAN aiming to generate images depending on the label. To do that, the discriminator does not only need to distinguish if the input is true or fake, a further classification of the image class is needed. In order to realize that, two convolutional layers each combined with sigmoid are used to predict real/fake as well as the class. In the given case, only two classes, smiling and no smiling, needed to be classified. Therefore, the application of sigmoid was feasible.

Part 2

The following ten images showcase how the generator transforms the noise input to two different faces given a certain label.

Figure 4: Faces generated by ACGAN

Part 3

Implementing ACGAN hold similar challenges as already described in problem 1. The main difficulty was balancing both, discriminator and generator. During training, the discriminator's loss went down very fast. However, the generator's loss continued to decrease over time. Nevertheless, it didn't converge probably which can also be seen in the plot above.

The generator was able to learn the generations of pictures. However, it was not able to distinguish between the different classes yet. Adjusting the learning rate as well as changing the training in order to put less emphasis on the discriminator improved the training a little bit. Still, further improvement would be needed.

Problem 3: DANN

Part 1, 2, and 3

Problem 3 was about implementing DANN. The following table summarises the results of the network trained on either target or source data or both as well as the respective results on the two different datasets. Normally, the row trained on source is supposed to be the lower bound. In the case of SVHN → MNIST-M, the accuracy lays above the one of DANN. A short elaboration on the reasons is done in part 6.

	SVHN → MNIST-M	**MNIST-M → SVHN**
Trained on source	52.99%	25.17%
Adaption (DANN)	45.35%	41.42%
Trained on target	98.11%	93.65%

Part 4

In order to visualize the output of the feature extractor (latent space), the test data was mapped to the 2D-space using t-SNE. The results are visualized in the following for (a) the different digit classes and (b) the different domains. The first two images belong to the model SVHN → MNIST-M and the second two images belong to the model MNIST-M → SVHN respectively.

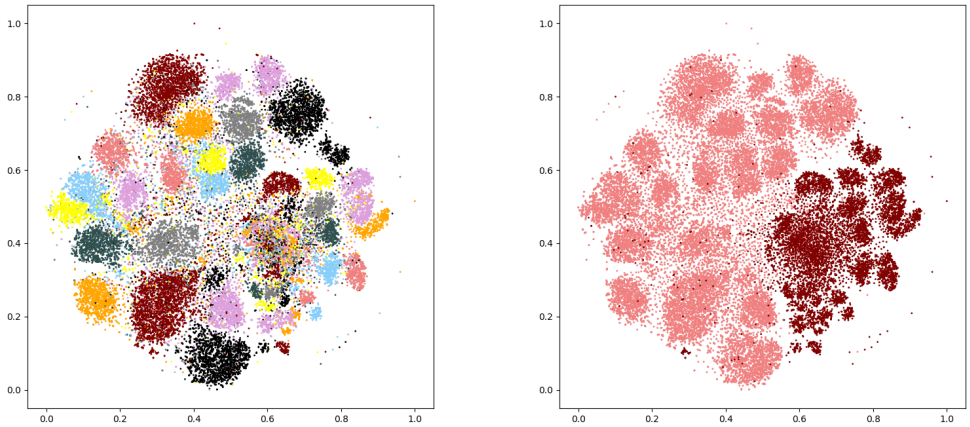


Figure 5: tSNE for SVHN → MNIST-M

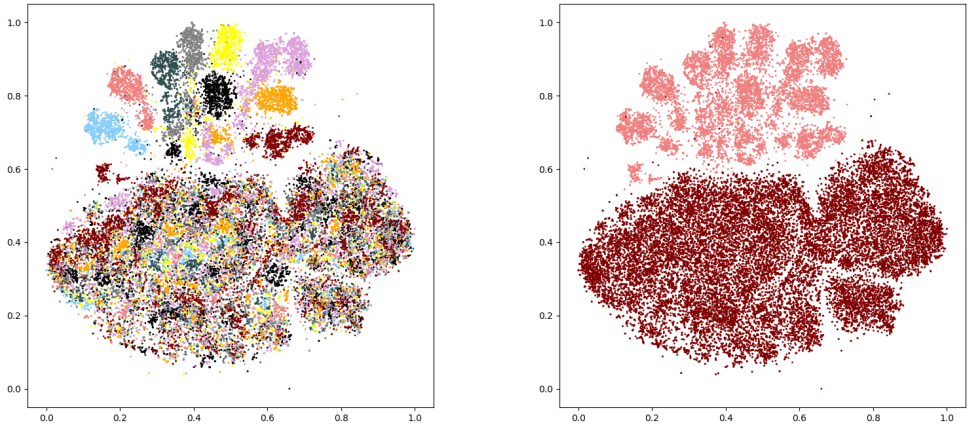
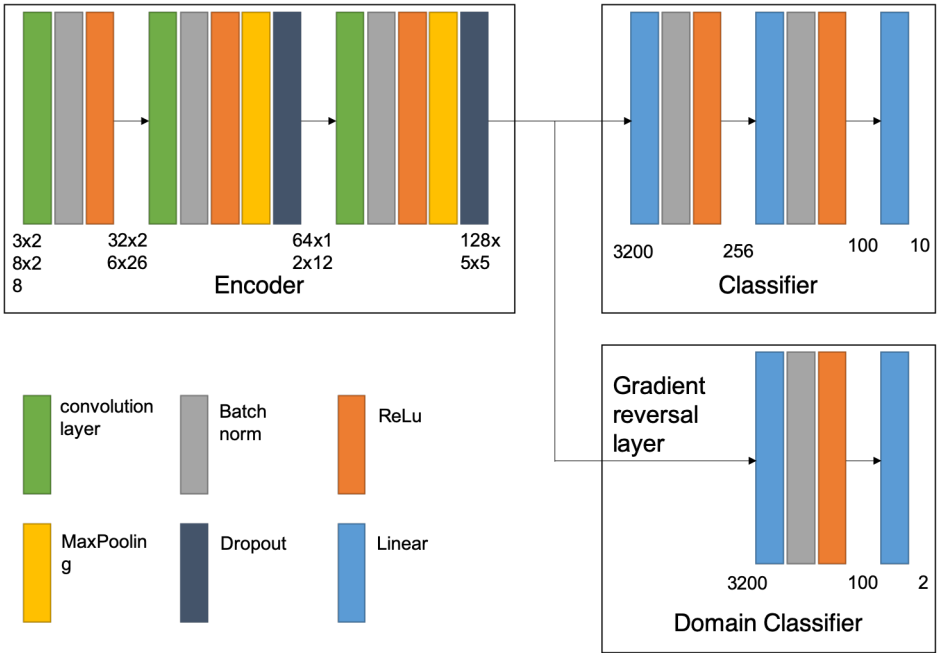


Figure 6: tSNE for MNIST-M → SVHN

Part 5

The architecture of the paper presented in the lecture was used as orientation. The detailed architecture is visualized in the following. The feature extractor consists of 3 convolutional layers combined with maxpooling and batch normalization. The classifier consists of 3 fully connected layer while the domain classifier only has 2 fully connected layers. Between the feature extractor and the domain classifier, a gradient reverse layer was added. The layer ensures that the loss of the domain classifier is maximized by multiplying it with -1 before backpropagation. This is important in order to ensure that the layer does not learn domain specific characteristics.



Part 6

Training the network was more or less straight forward since the implementation of the paper combined with some fine tuning of the learning rate resulted in before reported accuracies. As

shortly mentioned before, the model trained on source data received a higher accuracy than the DANN even though it is supposed to serve as a lower bound. This showcased that training DANN is not a very stable process and can lead to results as given in the table above.

During training a convolutional layer was added to the feature extractor. This didn't effected the learning much but resulted in less variations of the loss and allowed a more smooth training. Due to time limitations of the homework, further improvements of the model could not be implemented.

Problem 4: UDA

Part 1

In order to improve the results of DANN given in problem 3, ADDA was implemented following the paper given in the homework. The results of the models are displayed in the following table.

	SVHN → MNIST-M	**MNIST-M → SVHN**
Adaption (ADDA)	45.75%	43.05%

Part 2

Similar to problem 3, the test data is mapped to the 2D-space applying t-SNE. In the following, the (a) different digit classes and the (b) domains are visualized for each of the two improved UDA models.

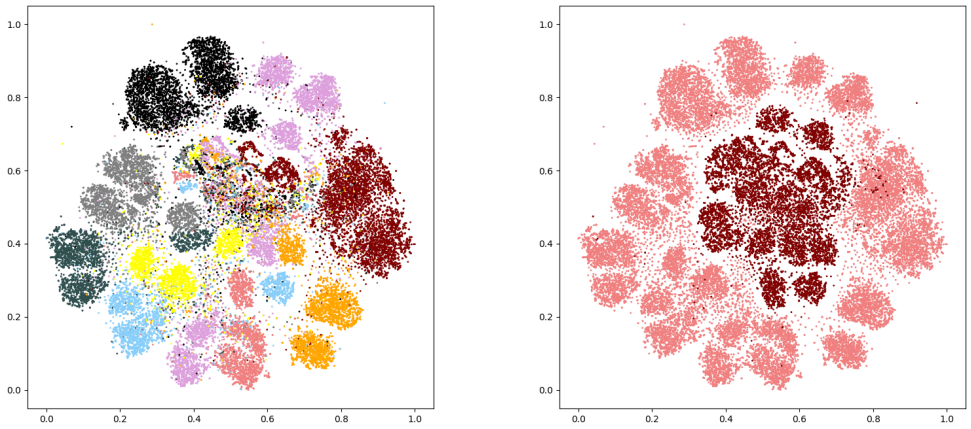


Figure 7: tSNE for SVHN → MNIST-M

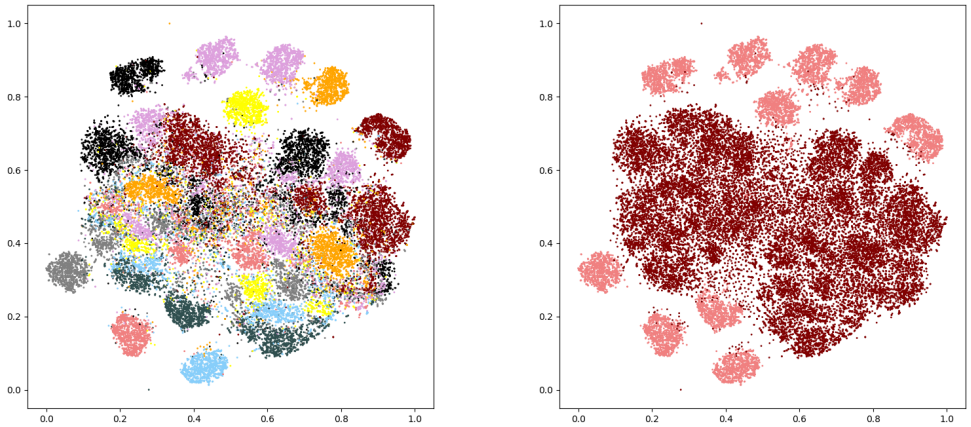
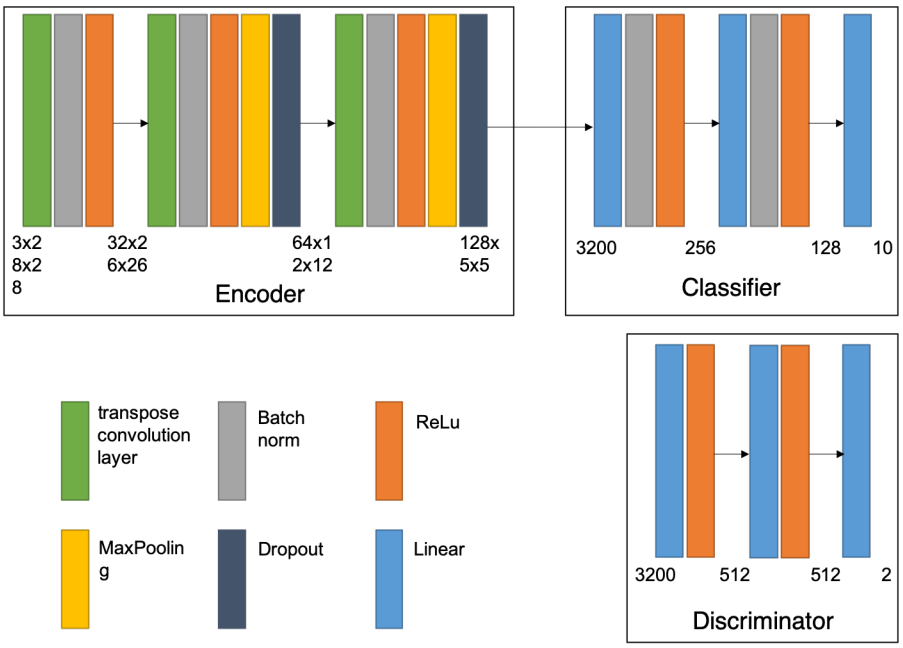


Figure 8: tSNE for MNIST-M → SVHN

Part 3

The detailed architecture is visualized in the following. The feature extractor was used from before implemented DANN and consists of 3 convolutional layers combined with maxpooling and batch normalization. The classifier consists of 3 fully connected layers and was only slightly adapted compared to DANN. For the discriminator, 3 fully connected layers are used with one being a hidden layer.



Part 4

Training ADDA was not really stable in the beginning and very often resulted in staying at a certain accuracy level. It seemed like the network was not learning. In this case, it helped to train the network slowly with a low learning rat and a higher number of iterations. In addition, training MNIST-M $\rightarrow$ SVHN was easier and resulted in an improvement by decreasing the learning rate. For SVHN $\rightarrow$ MNIST-M, more iterations as well as more adjustments to the network were needed such as changing the structure of the convolutional input and output of the discriminator.

Collaboration

In order to solve the homework, I mainly used the Pytorch documentation, Git examples and stackoverflow. The detailed list can be found in the following. Further, I discussed the homework with Clarissa Hamann (A08922111).

1 https://github.com/ddtm/caffe/tree/grl

2 https://github.com/fungtion/DANN/blob/master

3 https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1

4 https://github.com/jvanvugt/pytorch-domain-adaptation

5 https://github.com/aabbas90/ADDA-PyTorch

6 https://github.com/Carl0520/ADDA-pytorch

7 https://machinelearningmastery.com/how-to-develop-an-auxiliary-classifier-gan-ac-gan-from-scratch-with-keras/