Homework (Mini-project)

Objective of this assignment is to implement the basic building blocks of a Deep Learning pipeline on a sample supervised-learning problem in **PyTorch**.

Name: Mhd Karim Janou

Matriculation No.: 03732871

Important: Do not forget to fill the places where you see `### Your code goes here ###`

Task and Setup

In this assignment, we want you to experience doing a mini-project in PyTorch. You are supposed to build the different parts of the pipeline as illustrated in the course notebooks (DataLoading, Model, Loss, Training, Evaluation). For this reason, we suggest the two following supervised-learning setup:

Image Completion:

Assume that we want to build a DL Model that can complete missing pixels in an image. To make the setup a bit simpler, we can assume that the missing pixels always have a fixed location (e.g. center of the image). Additionally, we handle grayscale images only (i.e. our input has 1 channel in this case). We also assume that the input images have a fixed shape (e.g. 200 X 200). To follow this, you will need to resize all Images to this shape before feeding them to the model for training or inference.

Input Format: 1 X 200 X 200 Grayscale Image (1 for the channels, remember: PyTorch in the beginning) Label Format: 1 X 30 X 30 cropped part of the Image (this is what we want to learn and later predict)

Hints:

1. In your DataLoader you have to crop the part that will be the label, and also put 0 in its location in the original image. You do not want to feed the model with the original complete image because the label is part of the input in this case.
2. Take care that the dataset given contains RGB colored images, you would need to convert these to grayscale images (you are free to use any suitable package for that, PIL is also an option).
3. Your model can predict an output of 30 X 30 or the complete Image (200 X 200).
4. Fix the locations of the crop (it is fine if you hardcode them in your code; however, if you allow for setting them, this would be nice).
5. You can reshape the images to any size you prefer, but for simplicity do not use large images (not larger than 300).
6. The missing part cannot be very small, otherwise the task is trivial. Assume that it has to be at least 15% of the corresponding image dimension. For example, if you choose images of 200 X 200, then it has to be 30 X 30 ($30/200 = 15\%$)

Task 1: Data Loading (30 Points)

1. Write code to download and unzip the German Traffic Signs Dataset: (https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/GTSRB-Training_fixed.zip (https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/GTSRB-Training_fixed.zip))
2. Explore and visualize some images
3. Convert some example to grayscale and visualize them
4. Build a custom PyTorch dataset where you implement the required methods `__getitem__` and `__len__`
5. Split the data into train and validation data. Use a reasonable split ratio.
6. Create PyTorch dataloaders for train and validation datasets.

In [1]:

```
%load_ext autoreload
%autoreload 2
%matplotlib inline

### Your code goes here ####
## step 1: download and unzip
from src.dataset import util
import os
import glob
import torch
from torchvision import transforms
import numpy as np
from skimage import io

url = 'https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/GTSRB-T
dataset_dir = os.path.join(util.get_dataset(url), 'GTSRB')
train_dir = os.path.join(dataset_dir, 'Training')
```

```
creating directories...
data dir exists
/home/kareem/PycharmProjects/IBM_Lab/Ex1/data/GTSRB-Training_fixed  ex
ists
Dir is not empty
zip file doesnt exist
```

In [2]:

```
# step 2: visualize images randomly
imgs_path = glob.glob(os.path.join(dataset_dir, f'**/*.ppm'), recursive=True)
print('number of classes', len(os.listdir(train_dir))-1)
print('training size', len(imgs_path))
util.visualize_samples(imgs_path, n_cols=5, n_rows=1)
# step 3: visualize images in grayscale
util.visualize_samples(imgs_path, gray=True, n_cols=5, n_rows=1)
```

```
number of classes 43
training size 26640
```

Note: input size is 256x256x1 and output size 40x40x1

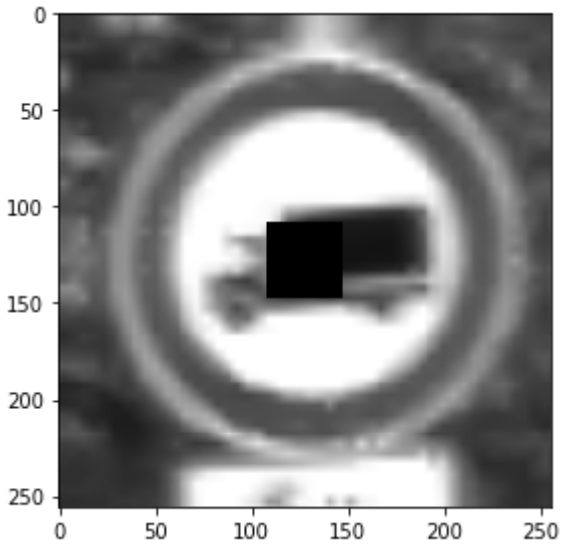
In [3]:

```
# step 4: dataset
from src.dataset.customDataset import custumDataset

dataset = custumDataset(dataset_dir)
img, in_img, target_img = dataset[0]
io.imshow(in_img)

dataset = custumDataset(dataset_dir, transform=transforms.ToTensor())
print(len(dataset))
print(img.shape, in_img.shape, target_img.shape) #full image
```

```
26640
(256, 256) (256, 256) (40, 40)
```



In [4]:

```
# step 5: split
train_size = int(0.7 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(dataset, [train_size, te
print(len(train_dataset))
print(len(val_dataset))
```

```
18648
7992
```

In [5]:

```
# step 6 dataloader
from torch.utils.data import DataLoader
import multiprocessing as mp

batch_size = 32
# note that dataloaders are added to src.Model.Solver.Trainer.
train_loader = DataLoader(train_dataset, batch_size=batch_size, num_workers=mp.cpu_
val_loader = DataLoader(val_dataset, batch_size=batch_size, num_workers=mp.cpu_coun
print('number of batches')
print(len(train_loader))
print(len(val_loader))
```

```
number of batches
583
250
```

Task 2: Model (20 Points)

1. Explore what possible models for the task could be. You do not need to come up with a very complex model, a relatively small CNN would also be fine, just take care of the sizing for the output layer.
2. Build a model class.
3. Test your model with one batch from your dataloader and check the input and output shapes.

The Model

In what follows I will be using an Autoencoder model. The model will take as an input the 256x256x1 with the 40x40 pixels in the middle set to 0. The output will also be 256x256x1 but the loss will only be computed on the blacked out pixels. It uses four convlution and pooling layer for the encoder and 4 upsampling and convlution layers for the decoder. Also Xavier/2 weights initialization was used which helps much in having faster convergence. More possible models will be discussed at the end of the notebook.

Optimization and Visualization

Adam optimizer with MSE loss are used. The Trainer class will manage the training of the model and it supports an early stopping criteria, a schedular, TensorBoard logging(loss, accuracy and images), and saves a checkpoint of the model at the lowest loss value. Retraing the model is also possible (either from the checkpoint or by saving the moddle with the 'inference' flag as False).

The following is a small test for the model.

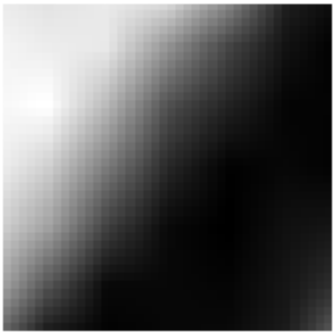
In [6]:

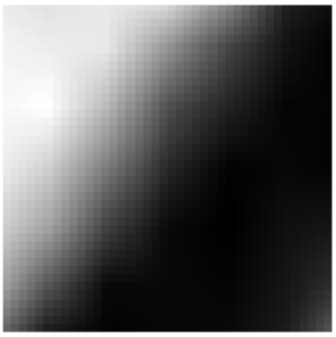
```
### Your code goes here ####
from src.Model.Networks import AutoEncoder
from matplotlib import pyplot as plt
from src.dataset.util import visualize_torch

model_hparams = {
    "filter_channels": 8,
    "filter_size": 3,
    "dropout": 0.02,
    }

AE_model = AutoEncoder(input_channels=1, hparams=model_hparams)
for batch in train_loader:
    print(len(train_loader))
    image, in_img, target_img = batch
    output = AE_model(in_img.float())
    target_center=[128,128]
    target_size=40
    top = int(target_center[0] - (target_size / 2))
    left = int(target_center[1] - (target_size / 2))
    cropped = image[:,:,top:top+target_size,left:left+target_size]
    visualize_torch(cropped, gray=True, n_cols=1, n_rows=1)
    visualize_torch(target_img, gray=True, n_cols=1, n_rows=1)
    print('input shape', in_img.shape, '\noutput shape', output.shape)
    assert output.shape == in_img.shape
    break
```

```
583
input shape torch.Size([32, 1, 256, 256])
output shape torch.Size([32, 1, 256, 256])
```





Task 3: Training (30 Points)

1. Develop Training and Validation code.
2. Choose a suitable loss function.

3. Refactor the code so that it can be easily modified and adapted (use methods, classes, etc...)
4. Make sure to save your trained model when you reach a good score on the validation dataset.
5. Plot the training and validation losses.

Stopping Criteria

The patience flag passed to the Trainer along with the val_epoch controlls the stopping criteria. The number of bad epochs is calculated as following: if the validation epoch loss is larger than the minimumn validation loss a penalty (+1) is counted, and if it is greater an award is counted (-1) as long as the value remains above 0. If the number of bad_epochs = patience then the training will stop. The reason why the validation loss is tracked is because we want our model to generalize on unseen data.

Checkpoint

When checkpoint_dir is assigned to the Trainer, the model with the minimum validation loss will be saved in the given directory.

StepLR Schedular

The schedular will multiply the learning rate by 'gamma' every 'step_size' steps

Hyperparameter Tuning

A small manual hypermarameter tunning was done. Nevertheless, this can be imporved by using some framework such as Optuna and many other.

Note: Run "tensorboard --logdir=runs" to open tensorboard

Note: A model checkpoint is attached. Therefore running the training again is not required. The model checkpoint can be loaded and then used for inference or for further training. Also the model's Tensorboard file is attached.

In [7]:

```
### Your code goes here ####
from torch import nn
from src.Model.Solver import Trainer
from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR

model_hparams = {
    "filter_channels": 32,
    "filter_size": 3,
    "dropout": 0.02,
}
optimizer_hparams = {
    "lr":0.001,
    "weight_decay":1e-5,

}
schedular_params = {
    "step_size": 30,
    "gamma": 0.5,
}

AE_model = AutoEncoder(input_channels=1, hparams=model_hparams)
optimizer = Adam(AE_model.parameters(), **optimizer_hparams)
scheduler = StepLR(optimizer, **schedular_params)

trainer_hparams = {
    "batch_size":32,
    "optimizer":optimizer,
    "criterion": nn.MSELoss(reduction='none'),
    "cuda": torch.cuda.is_available(),
    "scheduler": scheduler,
}

models_dir = os.path.join('data', 'models')
checkpoint_dir = os.path.join(models_dir, 'AEModel')
if not os.path.exists(models_dir):
        os.mkdir(models_dir)

trainer = Trainer(AE_model, train_dataset, val_dataset, **trainer_hparams,
                  patience=5, checkpoint_dir=checkpoint_dir)

train_model = True

if train_model:
    trainer.train(n_epochs=100, val_epoch=1, logs_dir='AEModel')
    print('run \'tensorboard --logdir=runs\' to open tensorboard')
```

```
Validate - loss 0.0015: 100%|██████████| 250/250 [00:20<00:00, 12.0
5it/s]
  0%|          | 0/583 [00:00<?, ?it/s]

Validate Loss: 0.0015. Acc: 0.9295

Epoch Train 42 - loss 0.0013: 100%|██████████| 583/583 [01:58<00:0
0,  4.93it/s]
  0%|          | 0/250 [00:00<?, ?it/s]

Training Loss: 0.0013. Acc: 0.9640
```

```
Validate - loss 0.0015: 100%|██████████| 250/250 [00:20<00:00, 12.1
6it/s]

Validate Loss: 0.0015. Acc: 0.9276
Patience reached.
Total loss 0.0016375216023555566. Total acc 0.35603934391080616
Finished Training
run 'tensorboard --logdir=runs' to open tensorboard
```

Saving and loading the model manually

In [18]:

```
save_model = True
inference = True #if False saves model for retraining and not just inference
model_path = os.path.join(models_dir, 'AEModel_Adam_StepLR_50_2.model')
if save_model:
    trainer.save(model_path, inference=inference)
```

```
Model saved.  data/models/AEModel_Adam_StepLR_50_2.model
```

In [26]:

```
load_model = True
inference = False
load_path = os.path.join(checkpoint_dir, 'checkpoint_best.model')
if load_model:
    trainer.load(load_path, inference=inference)
```

```
Checkpoint loaded.  data/models/AEModel/checkpoint_best.model
```

In [10]:

```
retrain_model = False
if retrain_model:
    trainer.train(n_epochs=3, val_epoch=3, logs_dir='AEModel')
```

Task 4: Evaluation (20 Points)

1. Report some suitable evaluation metrics.
2. Visualize some results from the training data.
3. Visualize some results from the validation data (not used for training).
4. Pick some random images from the internet and test if your model can produce a reasonable prediction for them.
5. Conclude with some comments
6. Give us your feedback about the task (at least a sentence).

Evaluation:

Accuracy

The accuracy is calculated based on 0.005 precision. That is the average loss of each sample is compared with 0 with a tolerance of 0.005 and if it is within the range a point will be rewarded. Then this value is summed for all samples and averages accross batches and then accross the whole training dataset.

Loss

Since it is a regretion problem the MSELoss is also used as a metric to evaluate the model

In [27]:

```
### Your code goes here ####
# Part 1: Evaluation
total_loss, total_accuracy = trainer.evaluate()
print(f'Total loss {total_loss}. Total accuracy {total_accuracy}')
```

```
Validate - loss 0.0014: 100%|██████████| 250/250 [00:20<00:00, 12.14i
t/s]

Validate Loss: 0.0014. Acc: 0.9347
Total loss 0.0014137535423506052. Total accuracy 0.9346666666666666
```

In [31]:

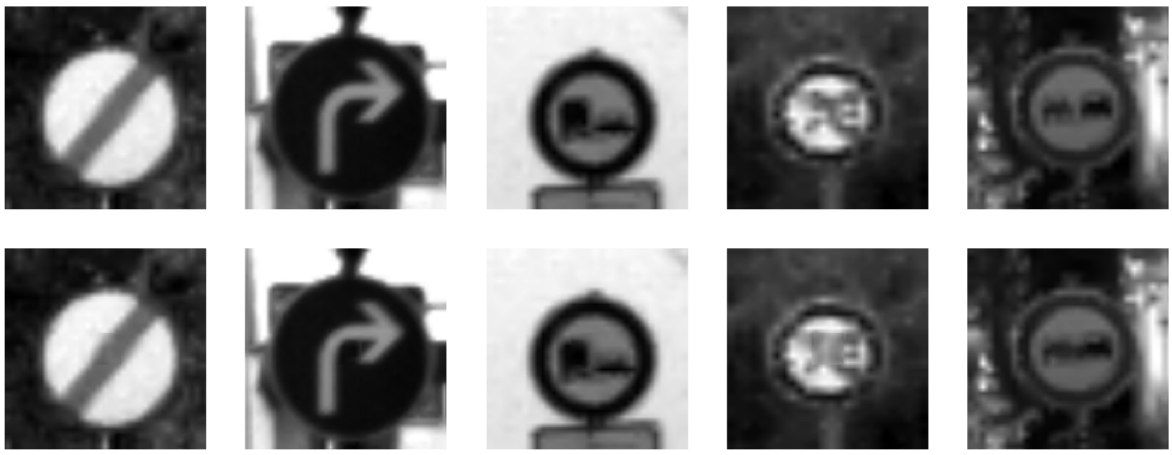
```
# Part 2: visualization from training data.
## note: one raw groundtruth and the following one prediction

print('Visualizing training data')
output_train_images, target_imgs = trainer.inference(loader=trainer.train_loader)
visualize_torch(target_imgs, gray=True).show()
visualize_torch(output_train_images, gray=True).show()
```

```
Visualizing training data

/home/kareem/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.
py:6: UserWarning: Matplotlib is currently using module://ipykernel.py
lab.backend_inline, which is a non-GUI backend, so cannot show the fig
ure.

/home/kareem/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.
py:7: UserWarning: Matplotlib is currently using module://ipykernel.py
lab.backend_inline, which is a non-GUI backend, so cannot show the fig
ure.
  import sys
```

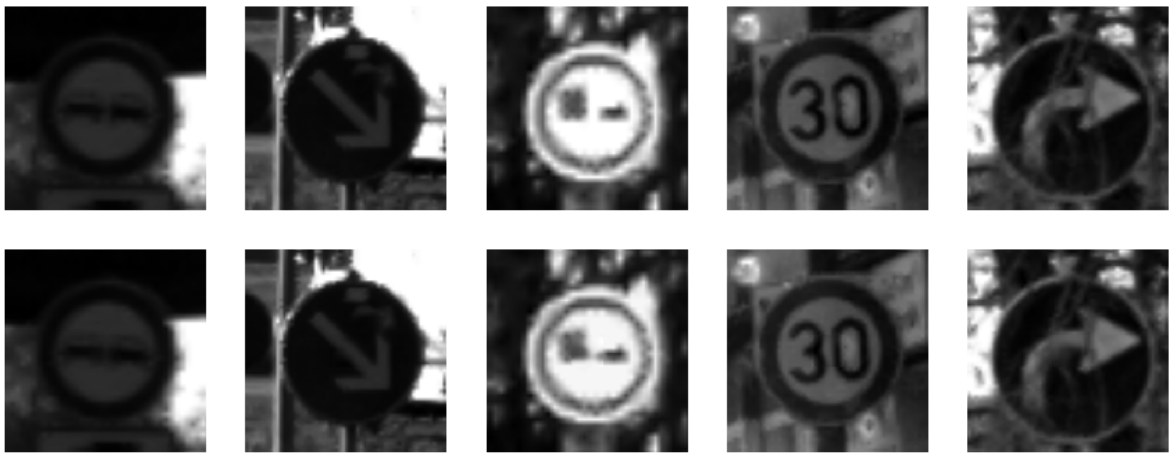


In [29]:

```
# Part 3: visualization from training data.
print('Visualizing validation data')
output_test_images, target_imgs = trainer.inference(loader=trainer.valid_loader)
visualize_torch(target_imgs, gray=True).show()
visualize_torch(output_test_images, gray=True).show()
```

```
Visualizing validation data

/home/kareem/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.
py:4: UserWarning: Matplotlib is currently using module://ipykernel.py
lab.backend_inline, which is a non-GUI backend, so cannot show the fig
ure.
  after removing the cwd from sys.path.
/home/kareem/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.
py:5: UserWarning: Matplotlib is currently using module://ipykernel.py
lab.backend_inline, which is a non-GUI backend, so cannot show the fig
ure.
  """
```



In [32]:

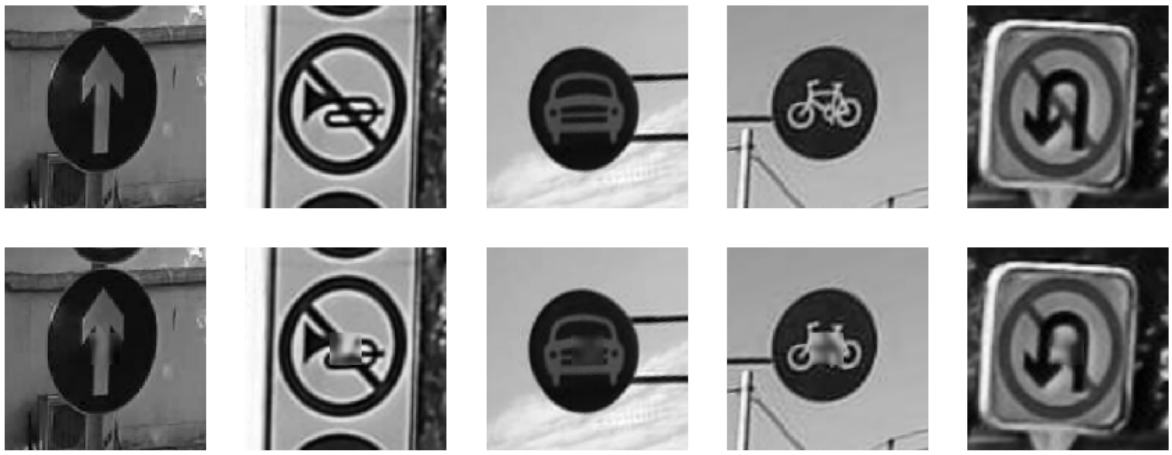
```
# Part 4: visualize images differnet dataset (TSRD)
url = 'http://www.nlpr.ia.ac.cn/pal/trafficdata/TSRD-Test.zip'
dataset_dir = os.path.join(util.get_dataset(url))

tsrd_dataset = custumDataset(dataset_dir, ext='*.png', transform=transforms.ToTenso
tsrd_loader = DataLoader(tsrd_dataset, batch_size=32, num_workers=mp.cpu_count())
output_online_images, target_imgs = trainer.inference(loader=tsrd_loader)
visualize_torch(target_imgs, gray=True).show()
visualize_torch(output_online_images, gray=True).show()
```

```
creating directories...
data dir exists
/home/kareem/PycharmProjects/IBM_Lab/Ex1/data/TSRD-Test  exists
Dir is not empty
zip file doesnt exist

/home/kareem/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.
py:8: UserWarning: Matplotlib is currently using module://ipykernel.py
lab.backend_inline, which is a non-GUI backend, so cannot show the fig
ure.

/home/kareem/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.
py:9: UserWarning: Matplotlib is currently using module://ipykernel.py
lab.backend_inline, which is a non-GUI backend, so cannot show the fig
ure.
  if __name__ == '__main__':
```



Part 5: Comments

Choice of the model:

When talking about generative models, one can immediately think about Autoencoders, Variational Autoencoders (VAEs), or GANs. The difference between Autoencoders and VAEs or GANs is that the later ones predicts parameters for the distribution of the output and then an output is sampled from the distribution. Whereas, Autoencoders directy prident the output which suits our task better since we dont need to learn a distribution but we only car about one ouput.

Improving the model:

Even that the current model gives verry good results, some improvments can be made such as using a UNet model which also an autoencoder but links every level of decoder and encoder together through directly. This always longer training and prevents from overfitting.

Another proposal would be to use a pretrained decoder, fine tune it, and then train the encoder. This will allow the encoder to predict better image features rather than learning from images which are distorted in the middle.

Generaliztion

As is shown in the last cell, the results were not as good as we expects. Collect more images or applying data augmentation could be a way overcome this problem and achieve better generalization.

Part 6: Feedback

I found the task interesting and very suitable for learning. Also I like that no code was given and that we have to start from scratch. The time is always a problem when developing machine learning applications because there is always a place to improve and try new things. Also because the training proccess is time consuming.

Submission Notes:

1. You can surely use external files to organize your code in classes or modules (e.g. `.py` files). However, please make sure that the notebook can run without errors and all the required files are attached in your submission (e.g. .zip file).
2. If you use special libraries or packages, please indicate that clearly and add a `requirements.txt` file to your submission.
3. Do not upload the dataset nor submit it in any way. Please make sure you include the code to download the dataset and work with relative directories so that your work can be reproduced and evaluated.
4. Please do not copy code from someone else, the idea here is that you get a chance to write some code in PyTorch and solve the problem on your own. On the other hand, discuss with your colleagues and support them as much as needed.
5. You can of course reuse the code in all the notebooks of the course.

Evaluation Criteria:

The most important idea we will use for evaluation is that you should get every component of the complete pipeline (1) doing what it is supposed to do and (2) fitting with the other components.

Concrete Examples are:

1. Your code runs and we can reproduce the results.
2. Dataset: Your code downloads and unzips the dataset to a folder (creates the required directory if needed).
3. Dataloading: Your dataset and dataloader produce correct inputs and labels (grayscale images with a rectangle missing at a fixed location, the missing rectangle as label).
4. Model: Your model takes the input and produces a prediction that has the correct shape.
5. Loss: your chosen loss is suitable for the task (e.g. mean squared error or mean absolute error among actual and predicted pixel values).
6. Training: your code for training and validation works without errors and the loss on the training and validation data decreases over the course of training. You do not have to achieve a specific performance.
7. Evaluation: you discuss the results and visualize correctly some original images and the respective predictions (preferably next to each other, but not a must)

Bonus: If you do any of the following ideas, you get a bonus, additionally you get to learn more, which is better than the bonus:

1. Use Tensorboard to visualize your training. (https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html (https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html))
2. Achieve very good performance on the task, where it is very difficult to differentiate the original image from the completed predicted image.
3. Use a learning rate scheduler to improve training. (https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate (https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate))