Import libraries

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
from torch.nn import Sequential, ReLU, Module, Dropout, Sigmoid, Linear, BatchNorm2d
from torch.optim import Adam
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from cv2 import PSNR
from SSIM_PIL import compare_ssim
from PIL import Image

import warnings
warnings.filterwarnings("ignore")


if torch.cuda.is_available():
    device = torch.device('cuda:0')
else:
    device = torch.device('cpu')
```

Load data

```
seed = 42   # for reproducibility
torch.manual_seed(seed)     # set seed for torch
torch.backends.cudnn.benchmark = False
torch.backends.cudnn.deterministic = True
batch_size = 512
epochs = 40
learning_rate = 1e-4
```

Creating train, val & test loaders

```
svhn = datasets.SVHN(root='E:/torchvision/datasets', download=False, transform=transfo
# downloaded the SVHN dataset

# using train_test_split to split the dataset into train, test and validation sets

train_data, test_data = train_test_split(svhn, train_size=0.8, random_state=42)
train_data, val_data = train_test_split(train_data, test_size=0.125, random_state=42)

# creating dataloaders for train, test and validation sets
# Dataloaders will create batches of data with the specified batch size and shuffle the

train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=True)


# Checking the shape of the input

print("Train data")
print(len(train_data))   # length of the train data
for batch in train_loader:
    images, labels = batch
    print(images.shape)     # shape of the images
    print(labels.shape)     # shape of the labels
    break

print("\nValidation data")
print(len(val_data))
for batch in val_loader:
    images, labels = batch
    print(images.shape)
    print(labels.shape)
    break

print("\nTest data")
print(len(test_data))
for batch in test_loader:
    images, labels = batch
    print(images.shape)
    print(labels.shape)
    break
```

```
Train data
51279
torch.Size([512, 3, 32, 32])
torch.Size([512])

Validation data
7326
torch.Size([512, 3, 32, 32])
torch.Size([512])

Test data
14652
torch.Size([512, 3, 32, 32])
torch.Size([512])
```

Undercomplete model architecture

```
class UnderCompleteAutoencoder(nn.Module):
    def __init__(self,input_dim):
        super(UnderCompleteAutoencoder, self).__init__()

        # Encoder layers
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, int(input_dim*0.75)),
            nn.ReLU(),
            nn.Linear(int(input_dim*0.75), int(input_dim*0.5)),
            nn.ReLU(),
            nn.Linear(int(input_dim*0.5), int(input_dim*0.25)),
            nn.ReLU()
        )

        # Decoder layers
        self.decoder = nn.Sequential(
            nn.Linear(int(input_dim*0.25), int(input_dim*0.5)),
            nn.ReLU(),
            nn.Linear(int(input_dim*0.5), int(input_dim*0.75)),
            nn.ReLU(),
            nn.Linear(int(input_dim*0.75), input_dim),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x

# defining the model, optimizer and loss function

# we shall select the device to be cuda (gpu) if available else cpu
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# We initialise the model by creating an object of the class defined above
model = UnderCompleteAutoencoder(input_dim=3*32*32).to(device)

# We define the optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Here we define the loss function
criterion = nn.MSELoss()
```

Without noise

Training undercomplete model

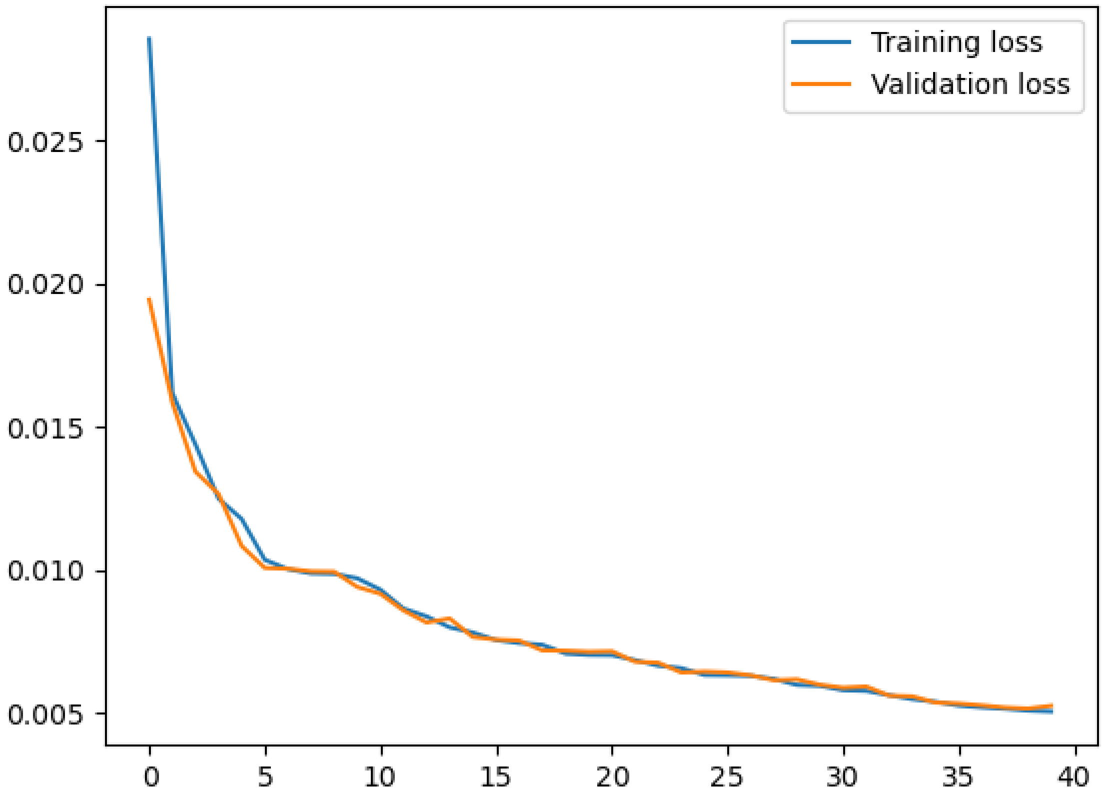
```
# In this section, we shall use the original images as input and pass them through the
# We shall then compare the original images with the reconstructed images using PSNR ar

train_losses = []
val_losses = []

for epoch in range(epochs):
    running_loss = 0
    for images, _ in train_loader:

        images = images.reshape(-1, 3072).to(device)        # reshaping the images to 
        optimizer.zero_grad()                               # setting the gradients to
        outputs = model(images)                             # passing the images throug
        train_loss = criterion(outputs, images)             # calculating the loss
        train_loss.backward()                               # backpropagating the loss
        optimizer.step()                                    # updating the weights
        running_loss += train_loss.item()

    training_loss = running_loss/len(train_loader)

    with torch.no_grad():                                   # we do not need to calcul
        val_running_loss = 0
        for images, _ in val_loader:
            images = images.reshape(-1, 3072).to(device)
            outputs = model(images)
            val_loss = criterion(outputs, images)
            val_running_loss += val_loss.item()

    validation_loss = val_running_loss/len(val_loader)

    train_losses.append(training_loss)
    val_losses.append(validation_loss)

    print("Epoch : ",epoch+1,"/",epochs,"Training loss = ",round(training_loss,6), "Va
```

```
Epoch :  1 / 40 Training loss =  0.028543 Validation loss =  0.019423
Epoch :  2 / 40 Training loss =  0.01616 Validation loss =  0.015841
Epoch :  3 / 40 Training loss =  0.01438 Validation loss =  0.013424
Epoch :  4 / 40 Training loss =  0.012481 Validation loss =  0.012643
Epoch :  5 / 40 Training loss =  0.011769 Validation loss =  0.010826
Epoch :  6 / 40 Training loss =  0.010341 Validation loss =  0.010057
Epoch :  7 / 40 Training loss =  0.010014 Validation loss =  0.010029
Epoch :  8 / 40 Training loss =  0.009867 Validation loss =  0.009936
Epoch :  9 / 40 Training loss =  0.00985 Validation loss =  0.009925
Epoch :  10 / 40 Training loss =  0.009694 Validation loss =  0.009393
Epoch :  11 / 40 Training loss =  0.009297 Validation loss =  0.009153
Epoch :  12 / 40 Training loss =  0.008634 Validation loss =  0.00857
Epoch :  13 / 40 Training loss =  0.008361 Validation loss =  0.008147
Epoch :  14 / 40 Training loss =  0.007979 Validation loss =  0.008287
Epoch :  15 / 40 Training loss =  0.007796 Validation loss =  0.007643
Epoch :  16 / 40 Training loss =  0.007534 Validation loss =  0.00755
Epoch :  17 / 40 Training loss =  0.007428 Validation loss =  0.007517
```

```
Epoch :  18 / 40 Training loss =  0.007369 Validation loss =  0.007174
Epoch :  19 / 40 Training loss =  0.007059 Validation loss =  0.007159
Epoch :  20 / 40 Training loss =  0.007014 Validation loss =  0.007125
Epoch :  21 / 40 Training loss =  0.007007 Validation loss =  0.00714
Epoch :  22 / 40 Training loss =  0.006831 Validation loss =  0.006776
Epoch :  23 / 40 Training loss =  0.006636 Validation loss =  0.00674
Epoch :  24 / 40 Training loss =  0.006551 Validation loss =  0.006399
Epoch :  25 / 40 Training loss =  0.006311 Validation loss =  0.006431
Epoch :  26 / 40 Training loss =  0.006302 Validation loss =  0.0064
Epoch :  27 / 40 Training loss =  0.006282 Validation loss =  0.0063
Epoch :  28 / 40 Training loss =  0.006175 Validation loss =  0.006128
Epoch :  29 / 40 Training loss =  0.005965 Validation loss =  0.006158
Epoch :  30 / 40 Training loss =  0.005932 Validation loss =  0.00597
Epoch :  31 / 40 Training loss =  0.00578 Validation loss =  0.005868
Epoch :  32 / 40 Training loss =  0.005756 Validation loss =  0.005908
Epoch :  33 / 40 Training loss =  0.005603 Validation loss =  0.005587
Epoch :  34 / 40 Training loss =  0.005465 Validation loss =  0.005564
Epoch :  35 / 40 Training loss =  0.005374 Validation loss =  0.005347
Epoch :  36 / 40 Training loss =  0.005242 Validation loss =  0.005313
Epoch :  37 / 40 Training loss =  0.005177 Validation loss =  0.00525
Epoch :  38 / 40 Training loss =  0.00513 Validation loss =  0.005173
Epoch :  39 / 40 Training loss =  0.005066 Validation loss =  0.005137
Epoch :  40 / 40 Training loss =  0.005036 Validation loss =  0.005233
```

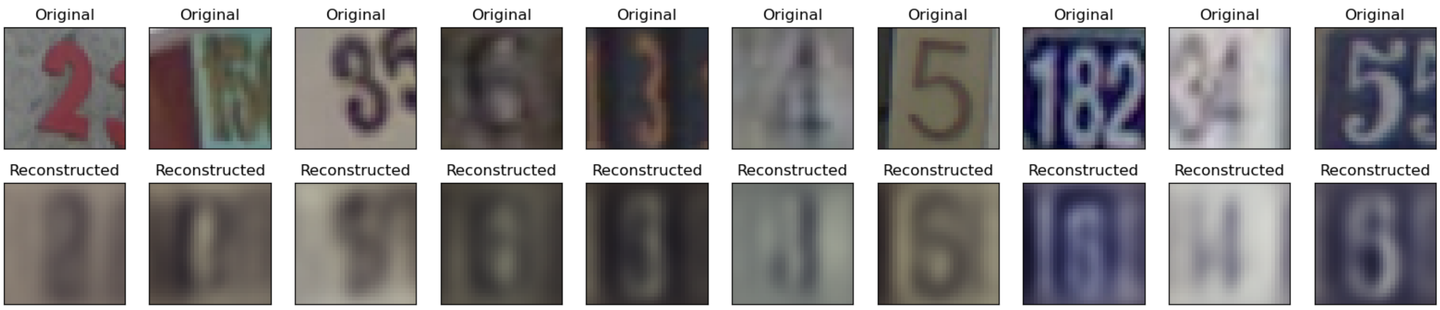
```python
# We shall save the best model based on the validation loss for testing in future.
import pickle
pickle.dump(model, open("undercomplete_autoencoder.pkl", "wb"))
```

```python
plt.plot(train_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.show()
```



Testing of undercomplete model

```
# In testing, we shall use the best model saved above and pass the test images through
# We shall then compare the original images with the reconstructed images using PSNR ar


test_examples = None
with torch.no_grad():                                                    # we do not need to
    for batch_features in test_loader:                                   # iterating through
        batch_features = batch_features[0].to(device)                    # batch_features a
        test_examples = batch_features.view(-1,3072)                     # reshaping the ima
        reconstruction = model(test_examples).view(-1,3,32,32)           # passing the image
        test_examples = test_examples.view(-1,3,32,32)                   # reshaping the ima


with torch.no_grad():
    number = 10
    plt.figure(figsize=(20, 4))

    for index in range(number):

        # display original
        ax = plt.subplot(2, number, index + 1)
        plt.imshow(test_examples[index].permute(1,2,0).cpu().numpy(), cmap='gray')
        plt.gray()
        plt.title("Original")
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        # display reconstruction
        ax = plt.subplot(2, number, index + 1 + number)
        plt.imshow(reconstruction[index].permute(1,2,0).cpu().numpy(), cmap='gray')
        plt.gray()
        plt.title("Reconstructed")
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

    plt.show()
```



Reconstruction accuracy

```
# We shall now calculate the PSNR and SSIM scores for the reconstructed images

'''PSNR'''
# PSNR stands for Peak Signal to Noise Ratio. It is a metric to measure the quality of
# Higher the PSNR, better the quality of the reconstructed image.
''' PSNR = 20 * log10(MAXp) - 10 * log10(MSE) '''
# where MAXp = maximum possible pixel value of the image
# MSE = mean squared error between the original and reconstructed images

'''SSIM'''
# SSIM stands for Structural Similarity Index. It is a metric to measure the similarity
# Higher the SSIM, better the quality of the reconstructed image.
''' SSIM = (2*mean_x*mean_y + c1)*(2*cov_xy + c2) / (mean_x^2 + mean_y^2 + c1)*(var_x +
# where mean_x = mean of the original image,
# mean_y = mean of the reconstructed image
# cov_xy = covariance of the original and reconstructed images,
# var_x = variance of the original image
# var_y = variance of the reconstructed image.


PSNR_values = []

for i in range(len(test_examples)):
    PSNR_values.append(PSNR(test_examples[i].permute(1,2,0).cpu().numpy(), reconstruct

print("Average PSNR value (reconstruction accuracy) = ",np.mean(PSNR_values))
```

```
Average PSNR value (reconstruction accuracy) =  72.40515116171142
```

```
SSIM_values = []

for i in range(len(test_examples)):

    img = test_examples[i].permute(1,2,0).cpu().numpy()
    reconstructed = reconstruction[i].permute(1,2,0).cpu().numpy()

    original = Image.fromarray((img*255).astype(np.uint8))
    reconstructed = Image.fromarray((reconstructed * 255).astype(np.uint8))

    ssim_val = compare_ssim(original, reconstructed)
    SSIM_values.append(ssim_val)

print("Average SSIM value (reconstruction accuracy) = ",np.mean(SSIM_values))
```

```
Average SSIM value (reconstruction accuracy) =  0.7501918689813465
```

With Noise Introduced

```
# In this section, we shall first induce the original images with random Gaussian nois
# Then we shall use the noisy images as input and pass them through the model to get tl
# We shall then decode the encoded images to get the reconstructed images.

# It is expected that the encoding and decoding task, removes the noise from the image
```

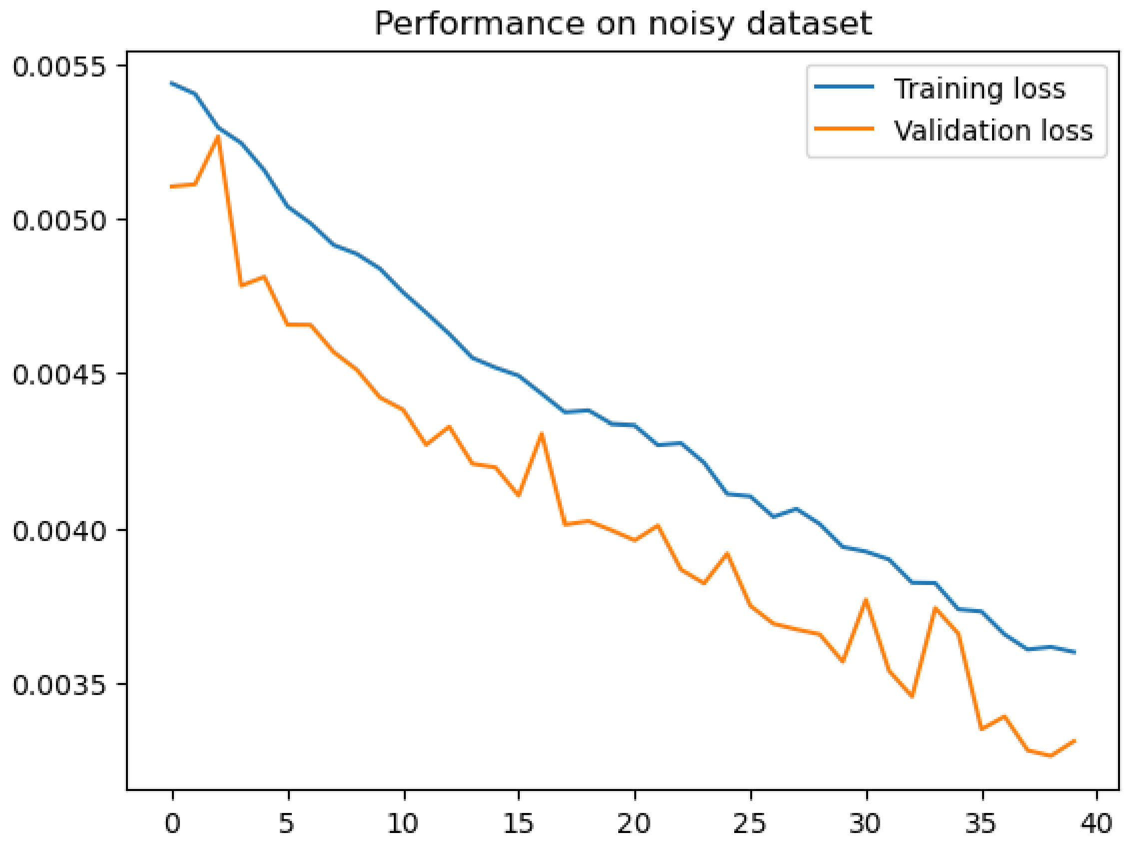
Training

```
train_losses = []
val_losses = []

for epoch in range(epochs):
    running_loss = 0
    for images, _ in train_loader:

        images = images.reshape(-1, 3072).to(device)
        shape = images.shape

        error = 0.02*torch.randn(shape).to(device)                # gaussian noise
        images = images + error                                   # adding the noise to 

        optimizer.zero_grad()
        outputs = model(images)
        train_loss = criterion(outputs, images)
        train_loss.backward()
        optimizer.step()
        running_loss += train_loss.item()

    training_loss = running_loss/len(train_loader)

    with torch.no_grad():
        val_running_loss = 0
        for images, _ in val_loader:

            images = images.reshape(-1, 3072).to(device)
            shape = images.shape

            error = 0.02*torch.randn(shape).to(device)            # gaussian noise
            images = images + error                               # adding the noise to 

            outputs = model(images)
            val_loss = criterion(outputs, images)
            val_running_loss += val_loss.item()

    validation_loss = val_running_loss/len(val_loader)

    train_losses.append(training_loss)
    val_losses.append(validation_loss)

    print("Epoch : ",epoch+1,"/",epochs,"Training loss = ",round(training_loss,6), "Va
```

```
Epoch :  1 / 40 Training loss =  0.003556 Validation loss =  0.003606
Epoch :  2 / 40 Training loss =  0.003531 Validation loss =  0.003609
Epoch :  3 / 40 Training loss =  0.003481 Validation loss =  0.004221
Epoch :  4 / 40 Training loss =  0.003483 Validation loss =  0.003496
Epoch :  5 / 40 Training loss =  0.003433 Validation loss =  0.003502
Epoch :  6 / 40 Training loss =  0.00343 Validation loss =  0.003486
Epoch :  7 / 40 Training loss =  0.003331 Validation loss =  0.003902
Epoch :  8 / 40 Training loss =  0.003348 Validation loss =  0.003482
Epoch :  9 / 40 Training loss =  0.00329 Validation loss =  0.003697
Epoch :  10 / 40 Training loss =  0.003288 Validation loss =  0.003381
Epoch :  11 / 40 Training loss =  0.003215 Validation loss =  0.003283
Epoch :  12 / 40 Training loss =  0.003233 Validation loss =  0.00326
```

```
Epoch :  13 / 40 Training loss =  0.003199 Validation loss =  0.003257
Epoch :  14 / 40 Training loss =  0.003182 Validation loss =  0.003404
Epoch :  15 / 40 Training loss =  0.003206 Validation loss =  0.003271
Epoch :  16 / 40 Training loss =  0.003197 Validation loss =  0.003347
Epoch :  17 / 40 Training loss =  0.003165 Validation loss =  0.00325
Epoch :  18 / 40 Training loss =  0.003134 Validation loss =  0.003166
Epoch :  19 / 40 Training loss =  0.00315 Validation loss =  0.003122
Epoch :  20 / 40 Training loss =  0.003121 Validation loss =  0.003116
Epoch :  21 / 40 Training loss =  0.003079 Validation loss =  0.003102
Epoch :  22 / 40 Training loss =  0.00306 Validation loss =  0.003198
Epoch :  23 / 40 Training loss =  0.003059 Validation loss =  0.003209
Epoch :  24 / 40 Training loss =  0.002957 Validation loss =  0.003115
Epoch :  25 / 40 Training loss =  0.002973 Validation loss =  0.003002
Epoch :  26 / 40 Training loss =  0.002994 Validation loss =  0.002996
Epoch :  27 / 40 Training loss =  0.002983 Validation loss =  0.003027
Epoch :  28 / 40 Training loss =  0.002873 Validation loss =  0.003014
Epoch :  29 / 40 Training loss =  0.002953 Validation loss =  0.003059
Epoch :  30 / 40 Training loss =  0.002897 Validation loss =  0.002953
Epoch :  31 / 40 Training loss =  0.002837 Validation loss =  0.002909
Epoch :  32 / 40 Training loss =  0.003011 Validation loss =  0.002874
Epoch :  33 / 40 Training loss =  0.002807 Validation loss =  0.002922
Epoch :  34 / 40 Training loss =  0.002801 Validation loss =  0.00284
Epoch :  35 / 40 Training loss =  0.00286 Validation loss =  0.002861
Epoch :  36 / 40 Training loss =  0.002793 Validation loss =  0.002884
Epoch :  37 / 40 Training loss =  0.00284 Validation loss =  0.00287
Epoch :  38 / 40 Training loss =  0.002738 Validation loss =  0.002787
Epoch :  39 / 40 Training loss =  0.002757 Validation loss =  0.002792
Epoch :  40 / 40 Training loss =  0.002761 Validation loss =  0.002751
```

```
plt.plot(train_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.title('Performance on noisy dataset')
plt.legend()
plt.show()
```



Testing

```
test_examples = None
original_examples = None

with torch.no_grad():
    for batch_features,_ in test_loader:

        shape = batch_features.shape
        error = 0.01*torch.randn(shape).to(device)

        original_examples = batch_features.to(device)
        batch_features = original_examples + error

        original_examples = original_examples.view(-1,3,32,32)
        test_examples = batch_features.view(-1,3072)

        reconstruction = model(test_examples).view(-1,3,32,32)
        test_examples = test_examples.view(-1,3,32,32)
        break

with torch.no_grad():
    number = 10
    plt.figure(figsize=(20, 6))
```