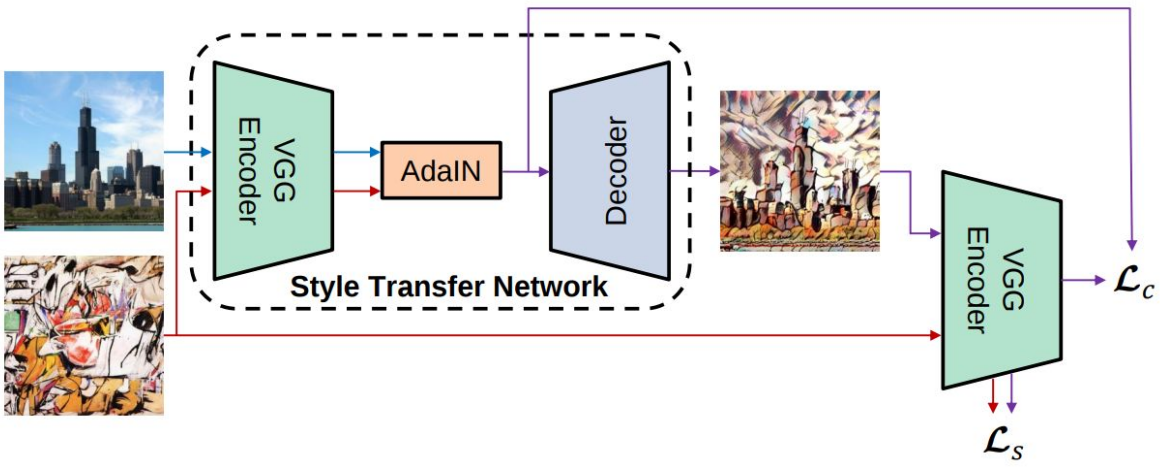
Introduction:

In this report the task was to implement end to end style transfer network. This can be achieved using convolutional neural networks. The essential ingredients of this network are input images. There are two types of input images namely content and style images. Content images contain the feature which need to be preserved. Where as style images contain the desired texture and style that needs to be applied to the final output. In this way, this is a very intuitive and elegant way to actually achieve style transfer.

Instance normalization (IN) is closely related to Batch normalization but, it is different as in case of IN we calculate mean and and std for each channel.

The architecture I follow is a very straightforward one. A pretrained custom encoder(VGG-19) *[1]* is used to extract features from both content and style images. In case of style images intermediate features extracted by the encoder are also saved. The output of of the style and content image is fed to a ADIN operation block. The output is again mixed with content image features with a factor of alpha. The image is finally passed through a decoder to actually generated the stylized image.
[1]



[1] https://github.com/naoto0804/pytorch-AdaIN

Network Architecture

The architecture used is exactly the same as the paper “Arbitrary Style Transfer in Real Time with Adaptive Instance Normalization”.

Encoder:

Standard Pre Trained Vgg19 architecture along with additional layers of ReflectionPad2d was used. [1]

Decoder:

A decoder structure [1] is shown below:

```
self.decoder =  nn.Sequential(
                nn.ReflectionPad2d((1, 1, 1, 1)),
                nn.Conv2d(512, 256, (3, 3)),
                nn.ReLU(),
                nn.Upsample(scale_factor=2, mode='nearest'),
                nn.ReflectionPad2d((1, 1, 1, 1)),
                nn.Conv2d(256, 256, (3, 3)),
                nn.ReLU(),
                nn.ReflectionPad2d((1, 1, 1, 1)),
                nn.Conv2d(256, 256, (3, 3)),
                nn.ReLU(),
                nn.ReflectionPad2d((1, 1, 1, 1)),
                nn.Conv2d(256, 256, (3, 3)),
                nn.ReLU(),
                nn.ReflectionPad2d((1, 1, 1, 1)),
                nn.Conv2d(256, 128, (3, 3)),
                nn.ReLU(),
               nn.Upsample(scale_factor=2, mode='nearest'),
                nn.ReflectionPad2d((1, 1, 1, 1)),
                nn.Conv2d(128, 128, (3, 3)),
                nn.ReLU(),
                nn.ReflectionPad2d((1, 1, 1, 1)),
                nn.Conv2d(128, 64, (3, 3)),
                nn.ReLU(),
                nn.Upsample(scale_factor=2, mode='nearest'),
                nn.ReflectionPad2d((1, 1, 1, 1)),
                nn.Conv2d(64, 64, (3, 3)),
                nn.ReLU(),
                nn.ReflectionPad2d((1, 1, 1, 1)),
                nn.Conv2d(64, 3, (3, 3)),)
```

Loss

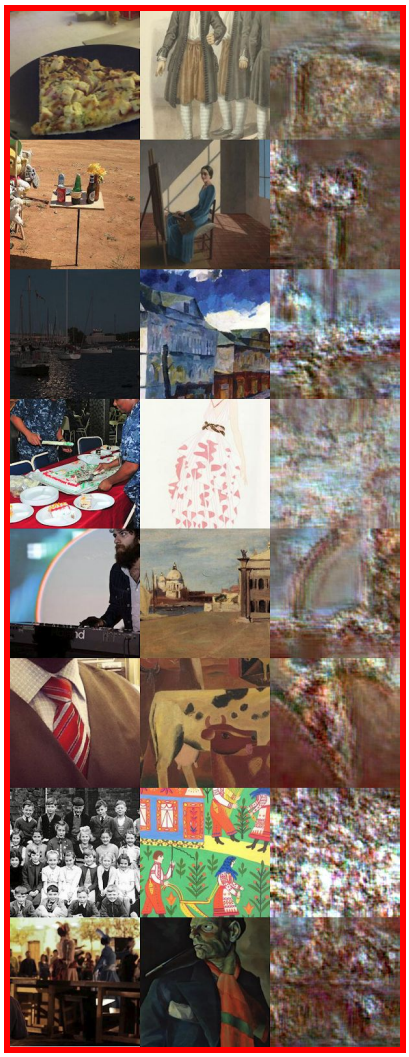
The final loss for training is shown below:

Loss = loss_content + loss_style* wstyle

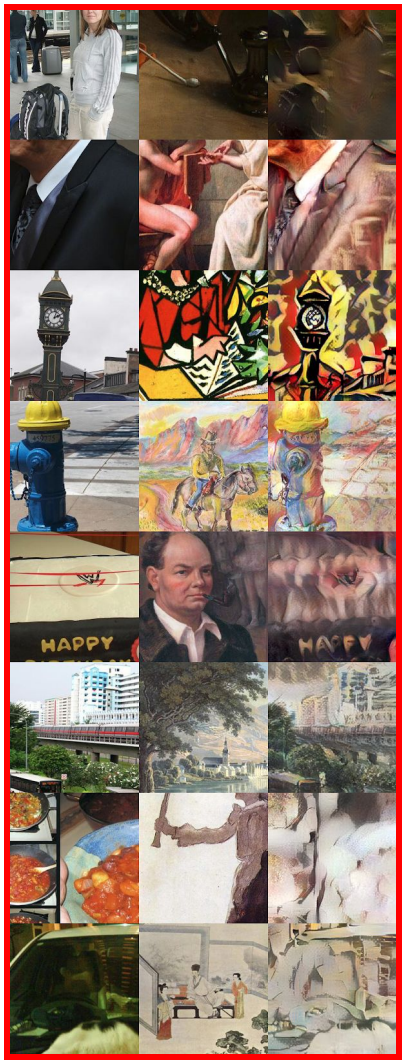
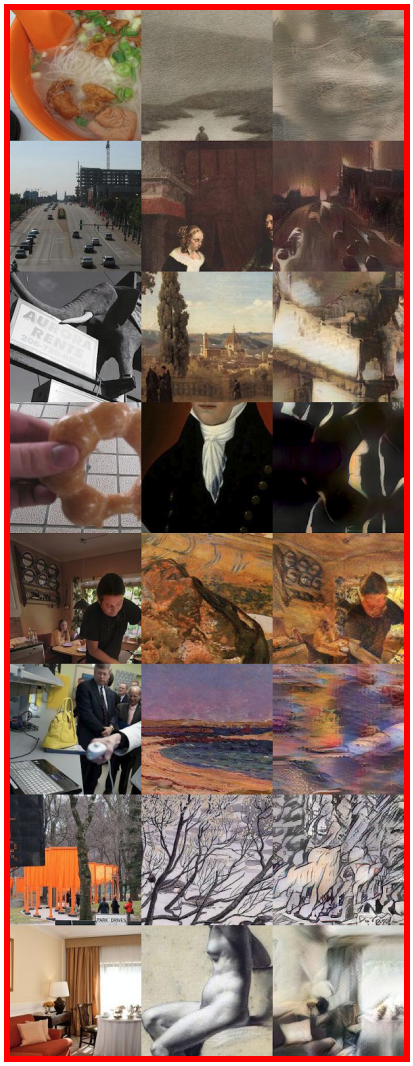
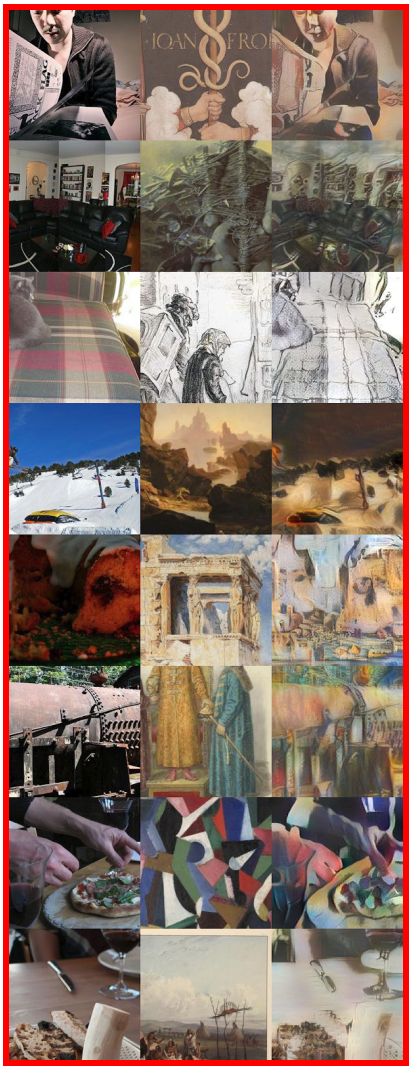
Wstyle was first set to 10.0 but then changed to 1.0. I show results with both configurations. I saw very interesting properties of training with different wstyle

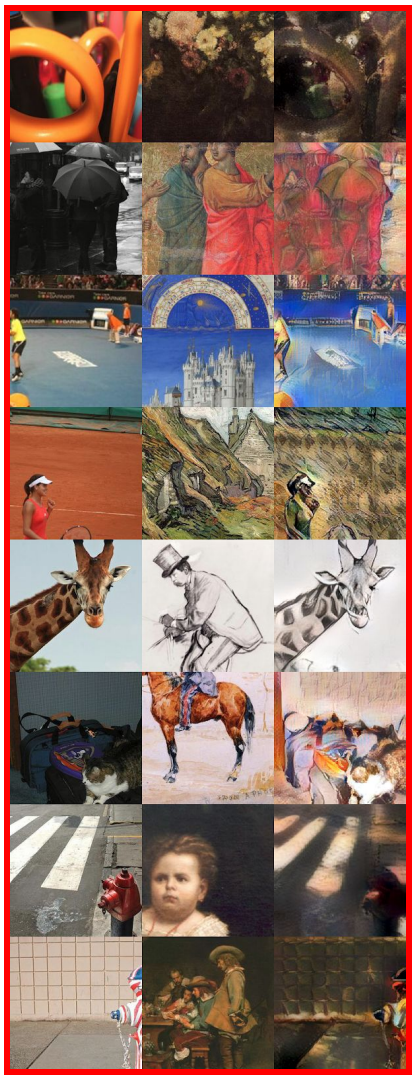
Intermediate Training Results wstyle =10.0

The results are shown below as training progresses:
Clearly in the first few epochs the image is not very clear. I have not mentioned exact epoch number here but the results are in accending order.

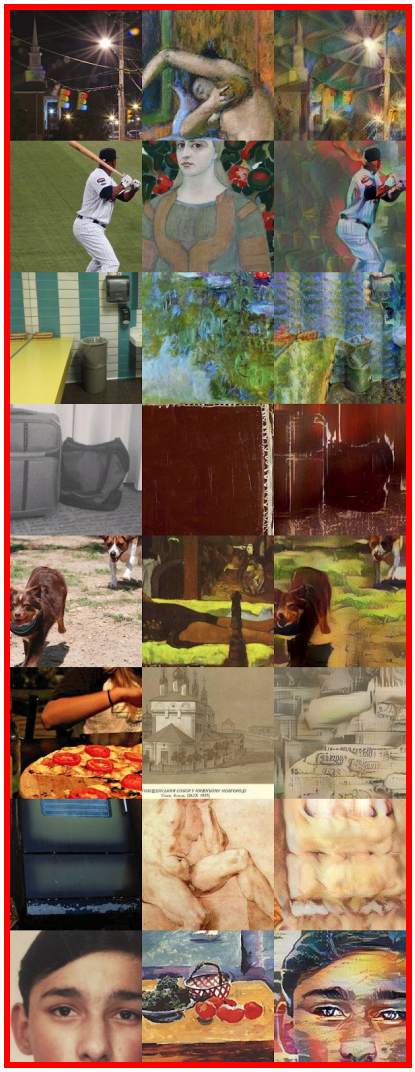


The situation starts improving soon as style transfer starts occurring.


HAPPY





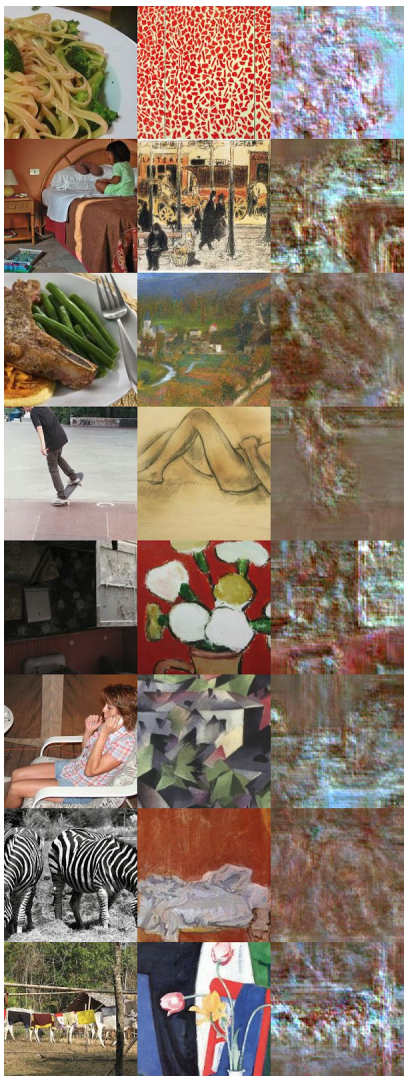

www.krajicek.net




E SYSTEMS

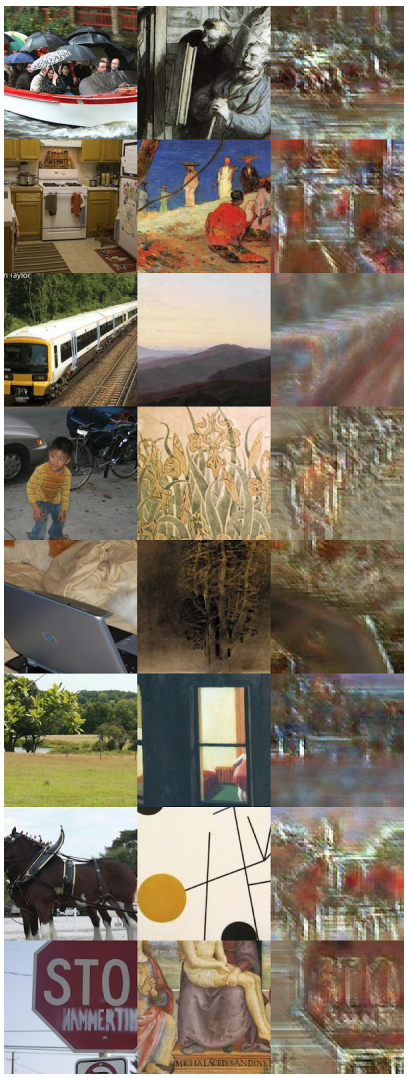
Intermediate Training Results wstyle =1.0

The training progress with wstyle 1.0 is shown in below given figures:



Epoch : 3

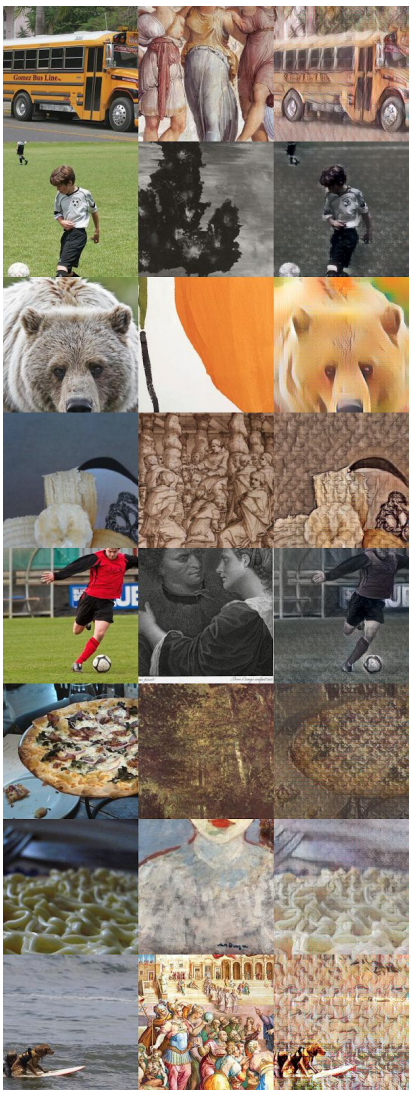
Epoch : 4



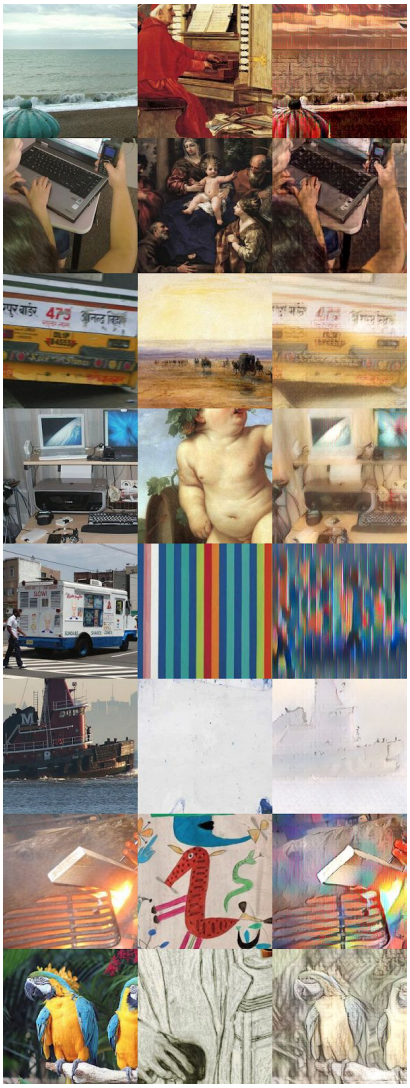
Epoch : 6

Epoch: 22 Epoch: 23 Epoch: 24

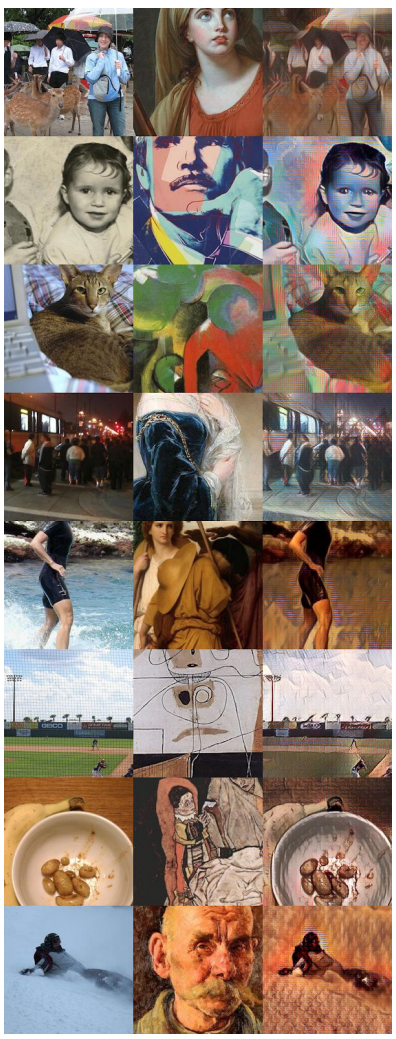
Epoch: 60



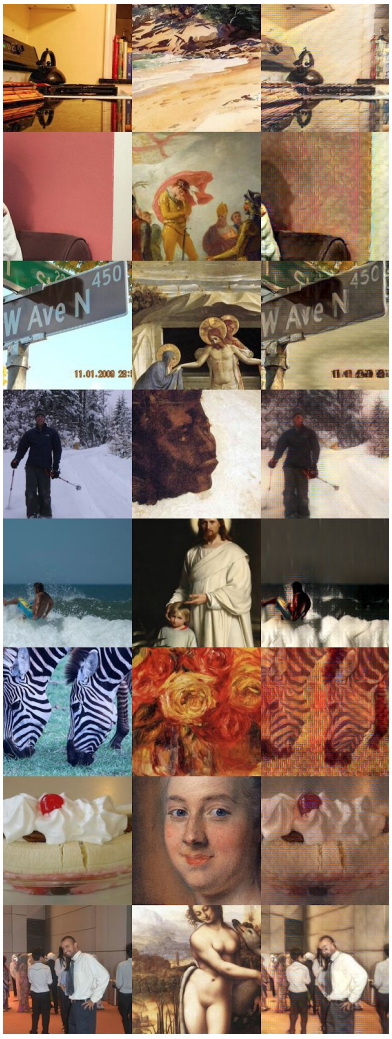
Epoch: 450



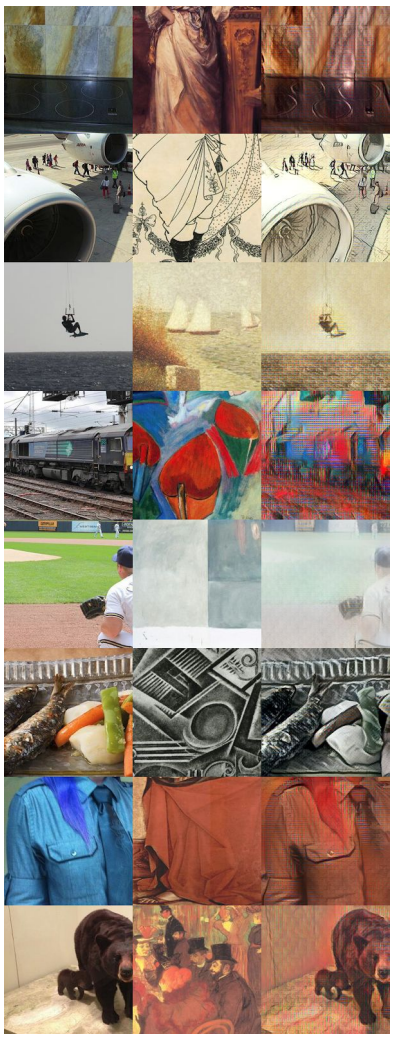
Epoch: 500



Epoch: 800



Epoch: 850



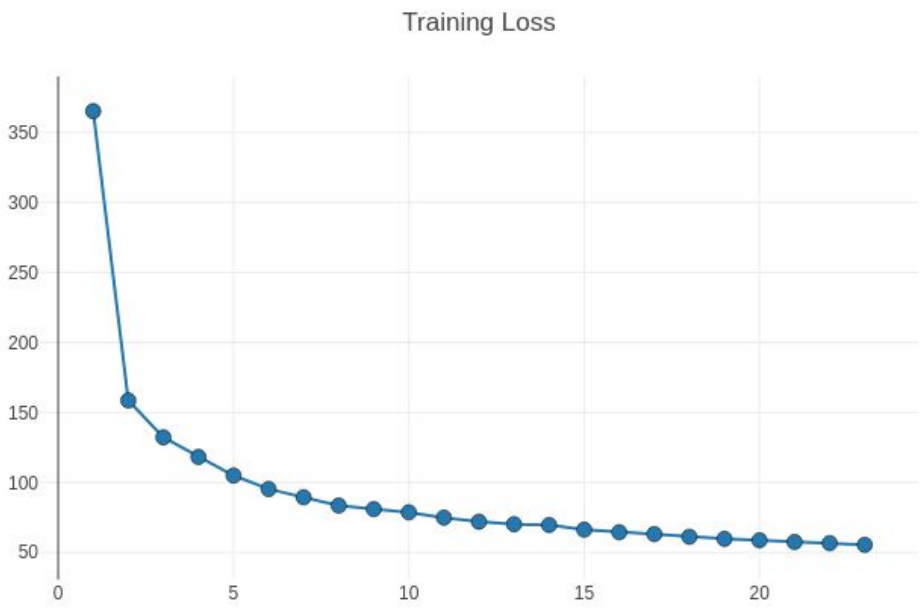
Epoch: 1000

Training Loss Curves wstyle =10.0

The curves for training are shown below at different zoom level:

Training Loss Curves wstyle =1.0

The curves for training are shown below at different zoom level:



ADIN Layer, Style and Content Loss

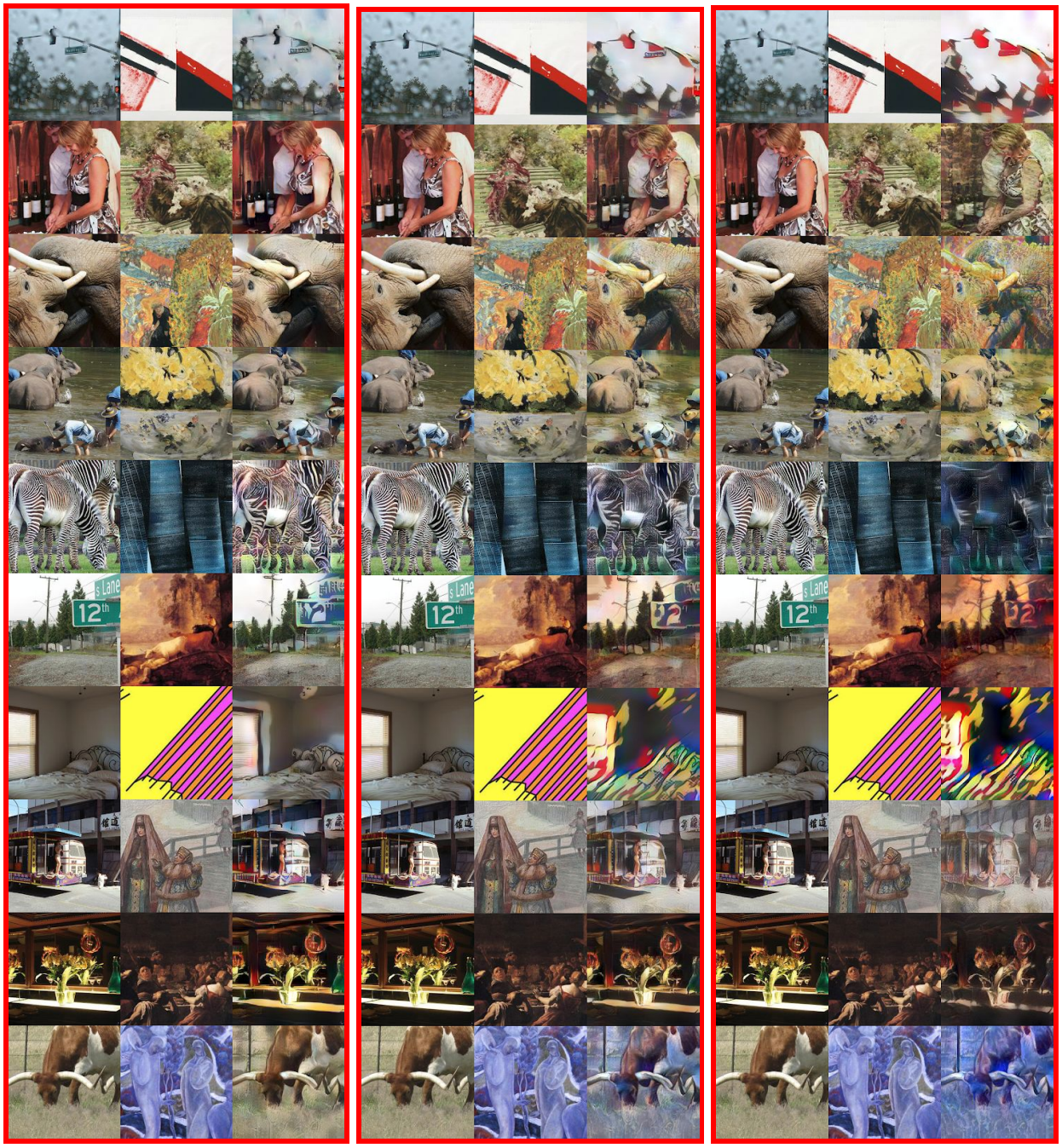
The losses and ADIN layer were implemented correctly as shown below:

```
def AdaINLayer(self, x, y):
  Bx, Cx, Hx, Wx = x.shape
  By, Cy, Hy, Wy = y.shape

  assert Bx == By
  assert Cx == Cy

  """
    Define your AdaIN layer in here

    output : the result of AdaIN operation
  """
  s = x.size()

  mean_content = mean(x)
  std_content = std(x)

  mean_style = mean(y)
  std_style = std(y)

  "ADIN Layer Formula"
  content_norm = (x - mean_content.expand(s)) / std_content.expand(s)
  output= std_style.expand(s) * content_norm  + mean_style.expand(s)
  return output

def loss_content(self, x, y):
  assert (x.size() == y.size())
  assert (y.requires_grad is False)
  return self.mse_loss(x, y)


def loss_style(self, x, y):
  assert (x.size() == y.size())
  assert (y.requires_grad is False)

  mean_x = mean(x)
  std_x = std(x)

  mean_y = mean(y)
  std_y = std(y)

  loss = self.mse_loss(mean_x, mean_y) + self.mse_loss(std_x, std_y)

  return loss
```

Test Results (Output behaviour with various alpha values)

wstyle = 10.0

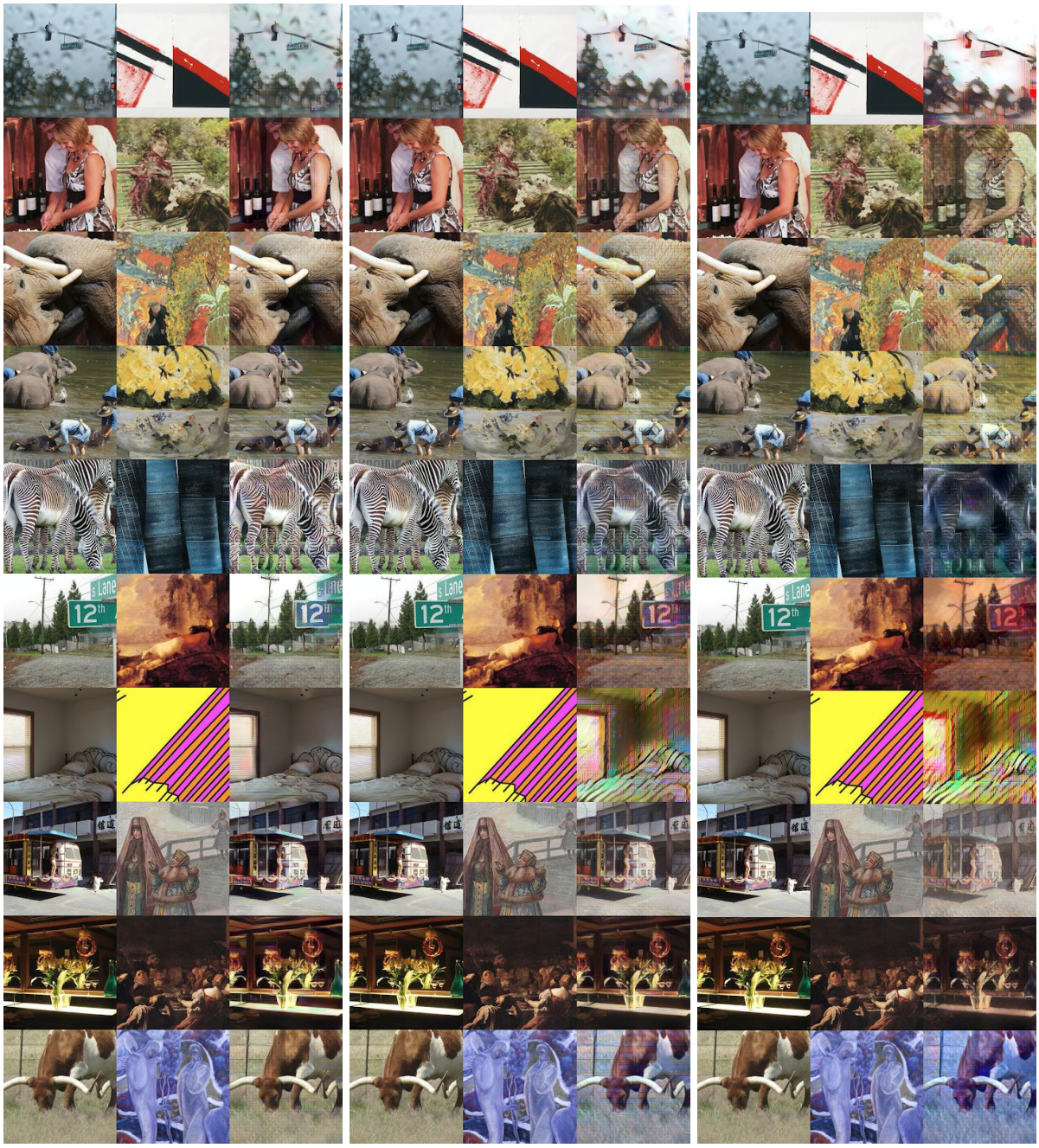
Alpha : 0.0 **Alpha : 0.5** **Alpha : 1.0**



Test Results (Output behaviour with various alpha values)

wstyle = 1.0

Alpha : 0.0 **Alpha : 0.5** **Alpha : 1.0**

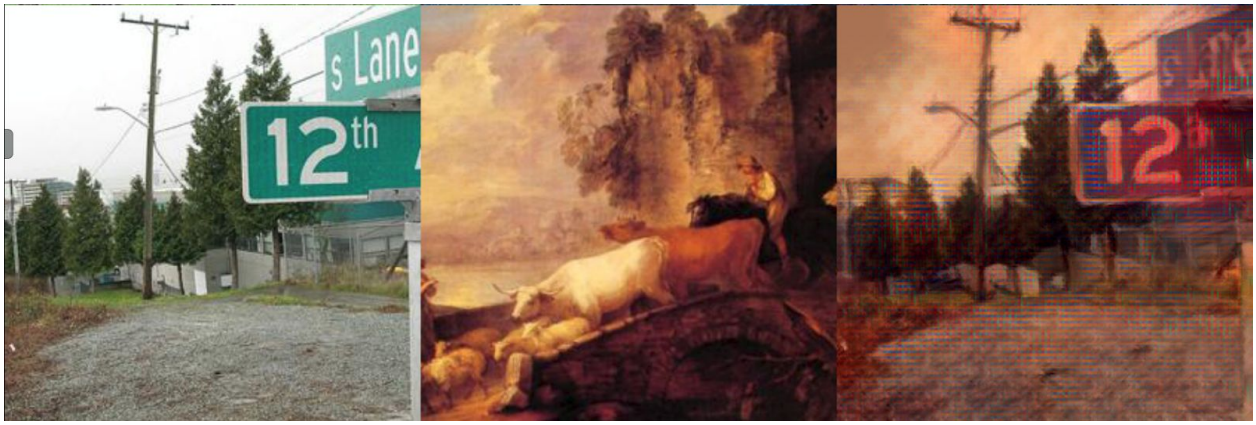


Discussion and Conclusion

Therefore I implemented end-to-end style transfer network. Then I trained my network to visualize intermediate results i.e evolution of results. The output behaviour changes with alpha. By changing it we can control how much style is applied to the output image. In this report I have shown results for alpha = 0.0, 0.5 and 1.0. In addition I have shown the results for wstyle 10.0 and 1.0.

wstyle= 1.0 vs wstyle = 10.0

I first trained using **wstyle = 10** but i observed that some contents are lost. So I trained using **w style = 1** and as i expected contents were preserved but some style application was lost. In particular the following figure explains the trade off. Notice how the number “12” on the board is skewed in case of wstyle 10. However in case of wstyle 1.0 the number is preserved, however smoothness is lost a bit. We can choose a middle value according to our desire. I like wstyle 10 as I prefer style preservation. Maybe additional technique can be applied to preserve content and apply max style



Wstyle = 1.0

Wstyle = 10.0

How to run the code and Visualize Results:

The code structure is very simple. As i followed the skeleton code. The only thing added is the network architecture and ADIN block of code. Run train.py to train the network. Run test.py to test the network. Visualize results on Visdom. In addition all results which are mentioned in the report have been provided in the folder of “Result Images” wstyle 10 and wstyle 1.0 both are given in results images. Pretrained models can be found in “snapshots” folder. I have given wstyle = 10.0 as default as they look good visually.