Create Bike Images with Wasserstein GAN

OpenCampus Course GAN

Nils Borchert, 02.2023

Image Set

# Properties

- 4510 manually-designed bicycle models
- span all common bicycle styles and feature
- unique models from remote corners
- Image size: 1024×1024, RGB

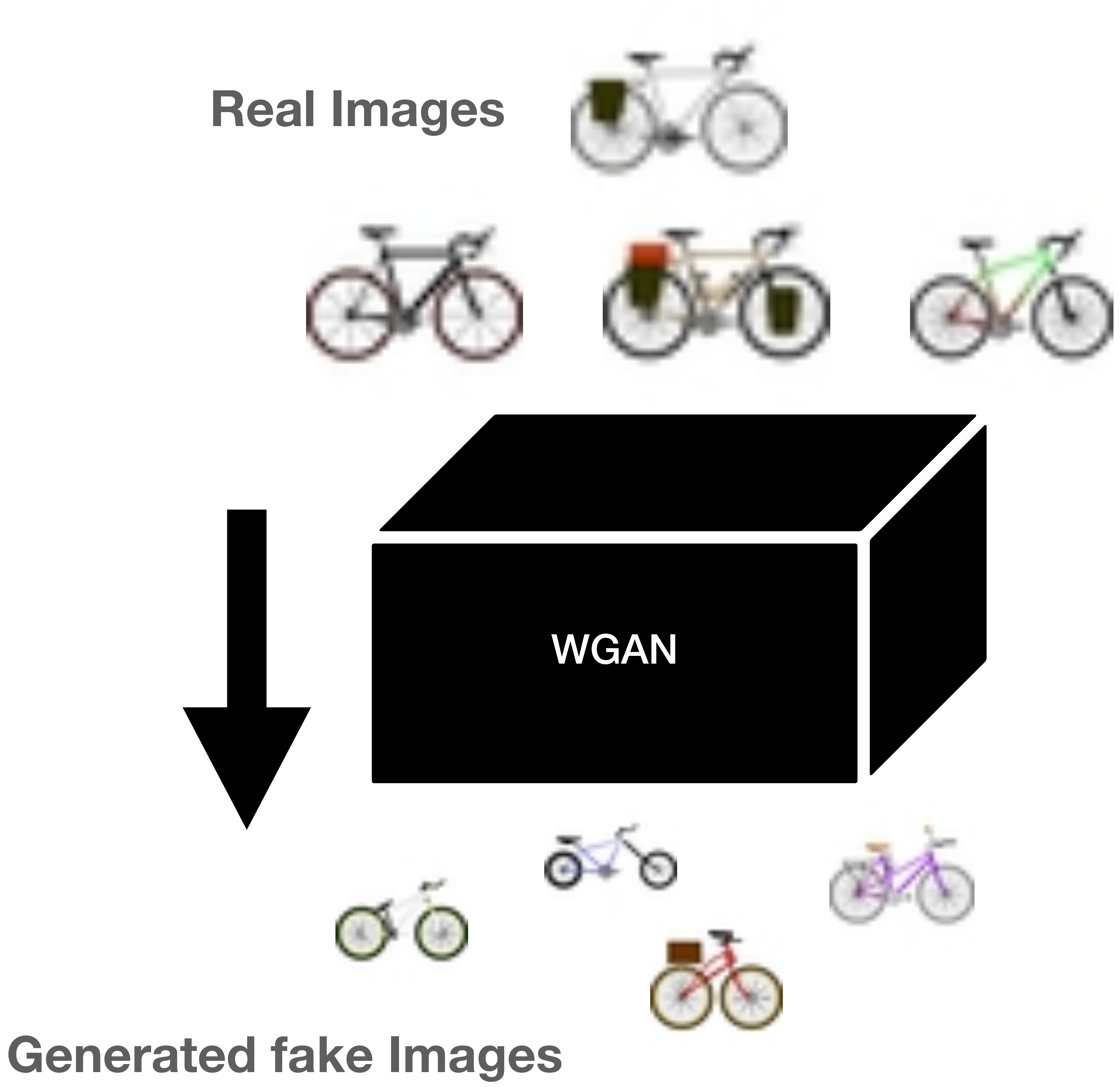
Changes

- Resize Images to 32x32, grayscale

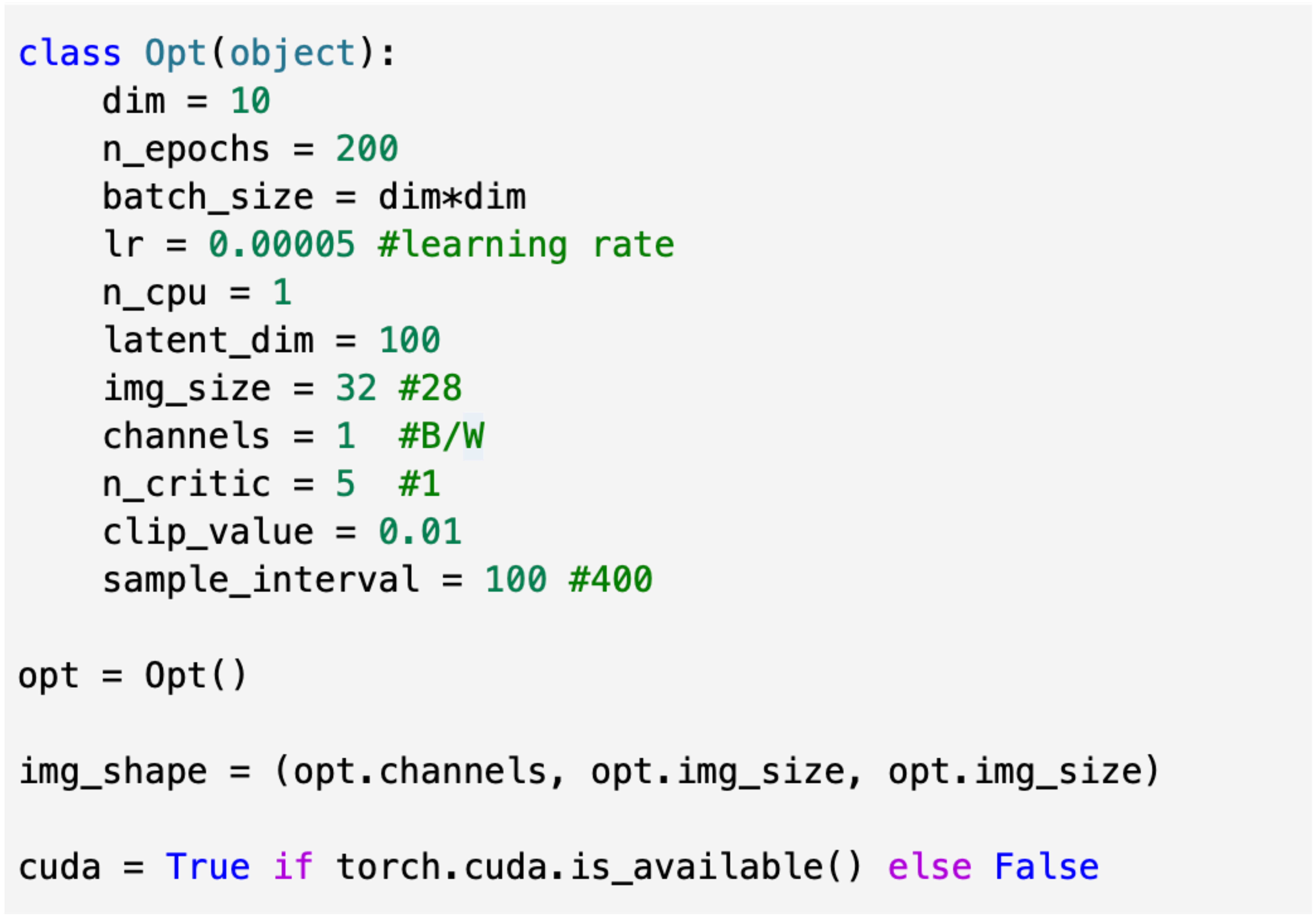
Source: Github Site

Main Goal

- Create New Bike Models with GAN
- Use a Wasserstein GAN for:
    - stability training and
    - prevent mode collapse



Model Properties



```
class Opt(object):
    dim = 10
    n_epochs = 200
    batch_size = dim*dim
    lr = 0.00005 #learning rate
    n_cpu = 1
    latent_dim = 100
    img_size = 32 #28
    channels = 1  #B/W
    n_critic = 5  #1
    clip_value = 0.01
    sample_interval = 100 #400

opt = Opt()

img_shape = (opt.channels, opt.img_size, opt.img_size)

cuda = True if torch.cuda.is_available() else False
```

Generator

```python
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        def block(in_feat, out_feat, normalize=True):
            layers = [nn.Linear(in_feat, out_feat)]
            if normalize:
                layers.append(nn.BatchNorm1d(out_feat, 0.8))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers
        self.model = nn.Sequential(
            *block(opt.latent_dim, 128, normalize=False),
            *block(128, 256),
            *block(256, 512),
            *block(512, 1024),
            nn.Linear(1024, int(np.prod(img_shape))),
            nn.Tanh()
        )

    lef forward(self, z):
        img = self.model(z)
        img = img.view(img.shape[0], *img_shape)
        return img
```

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=1024, bias=True)
    (12): Tanh()
```

Discriminator

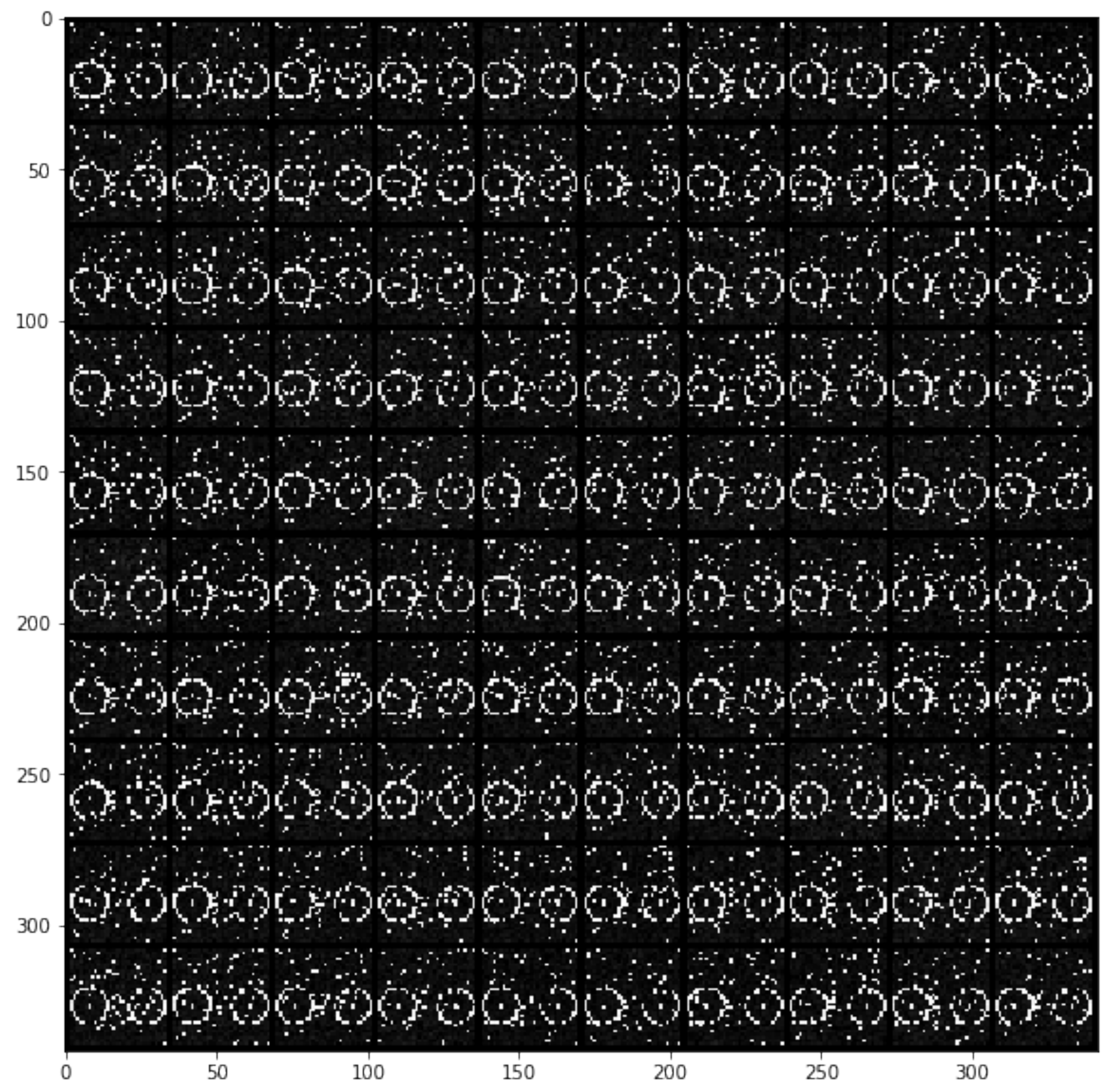
```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(opt.img_size ** 2, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
        )

    def forward(self, img):
        img_flat = img.view(img.shape[0], -1)
        validity = self.model(img_flat)
        return validity
```

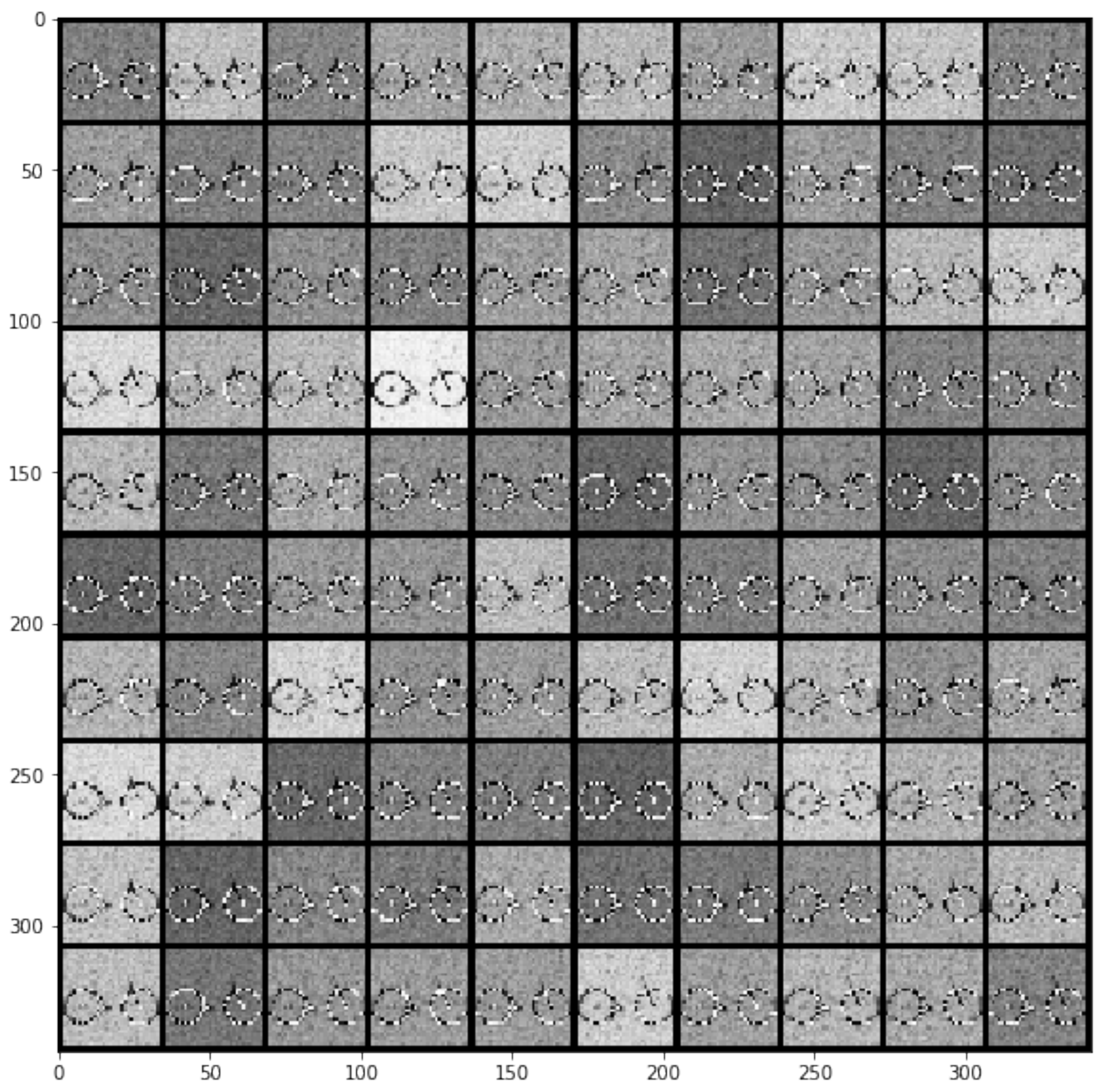
```
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

Epoch Samples

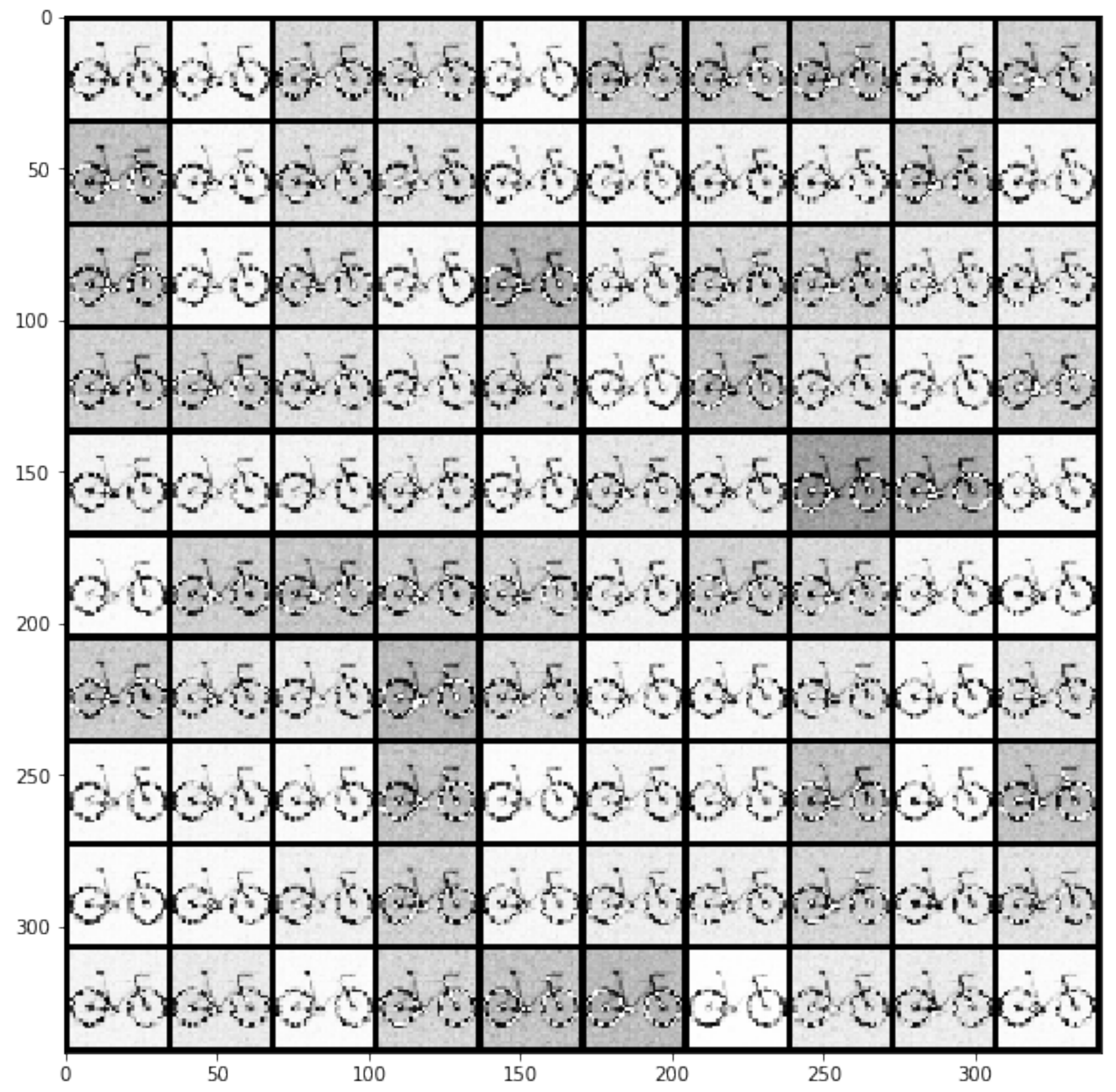
1



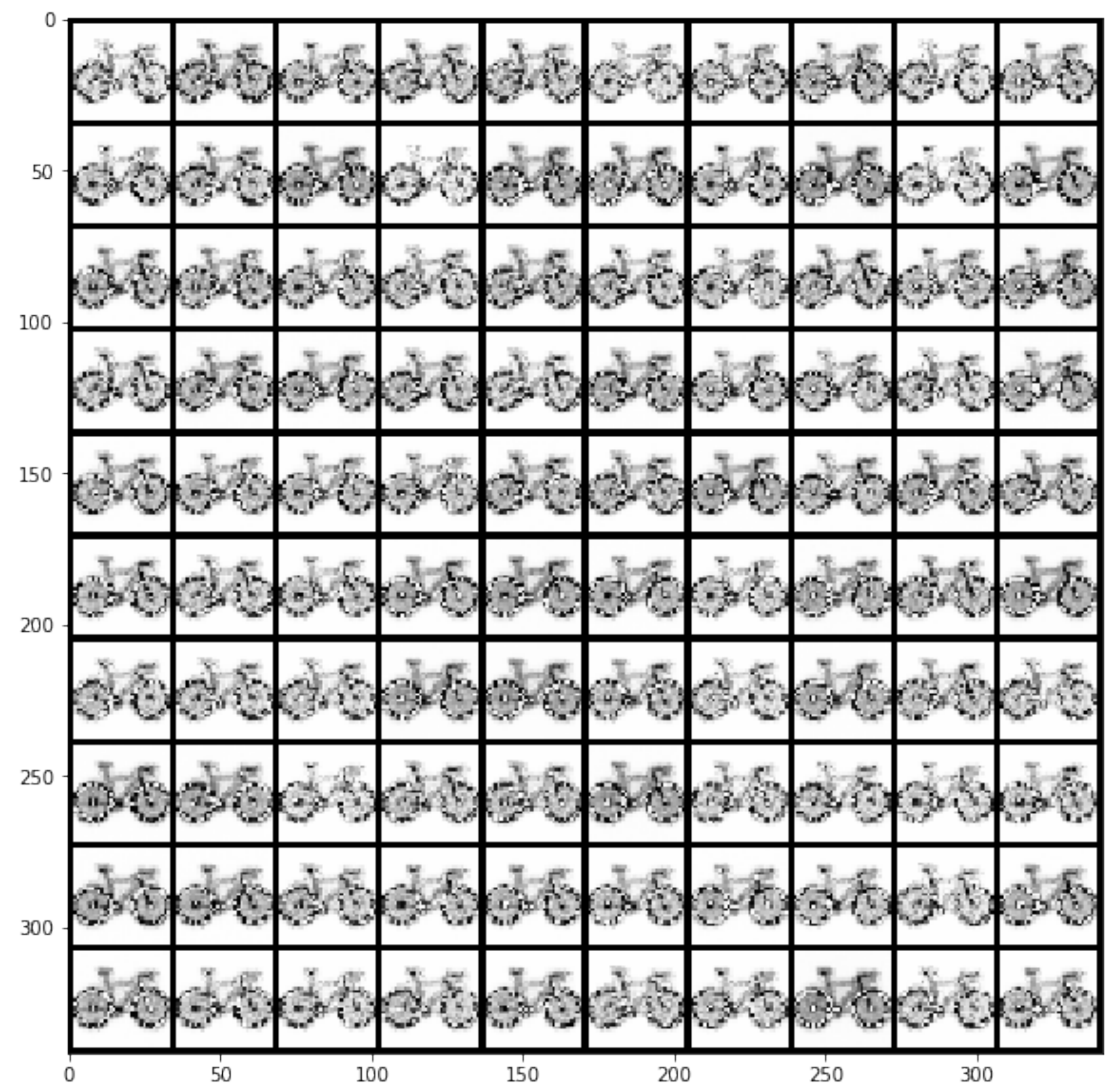
5



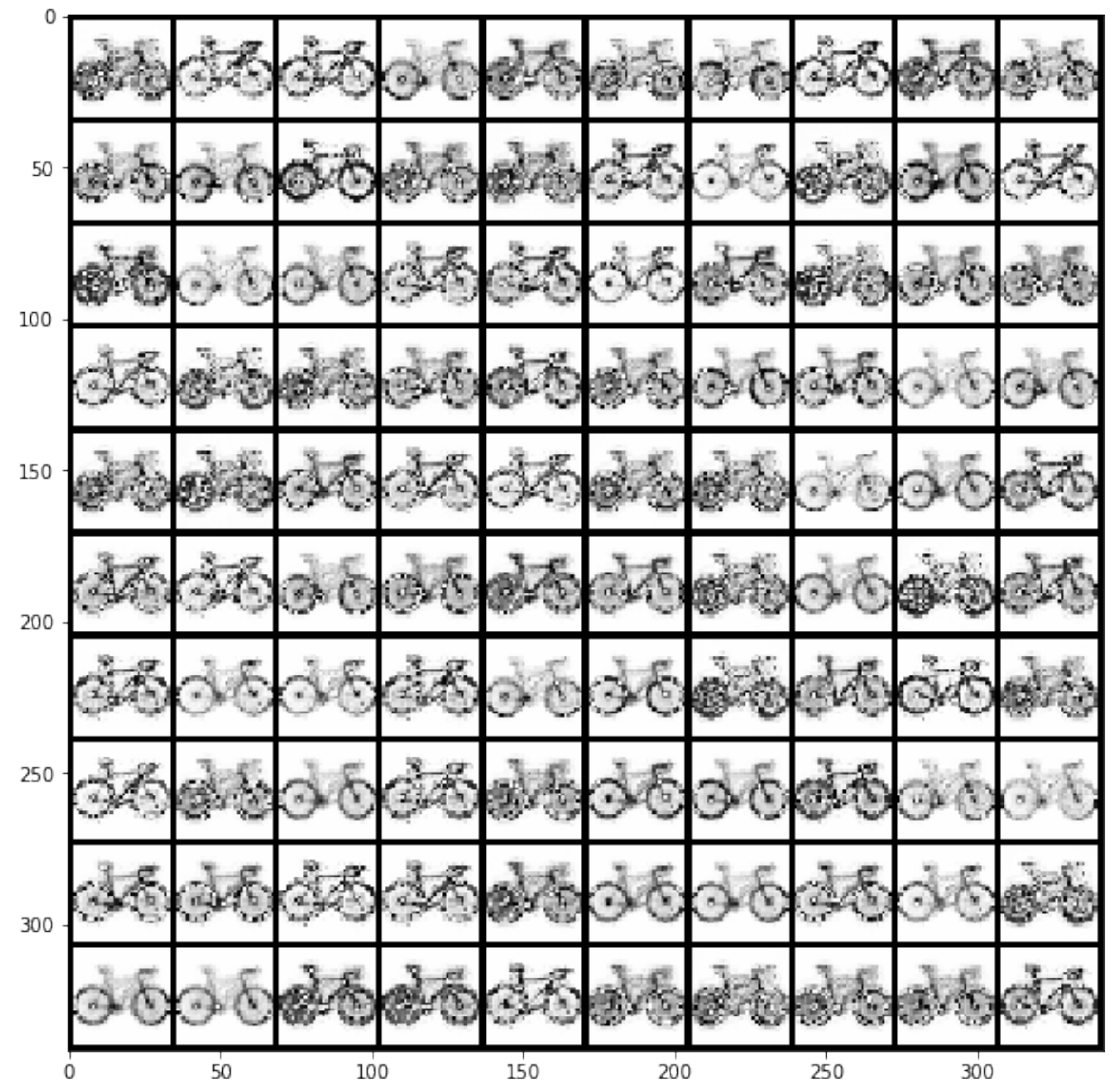
50



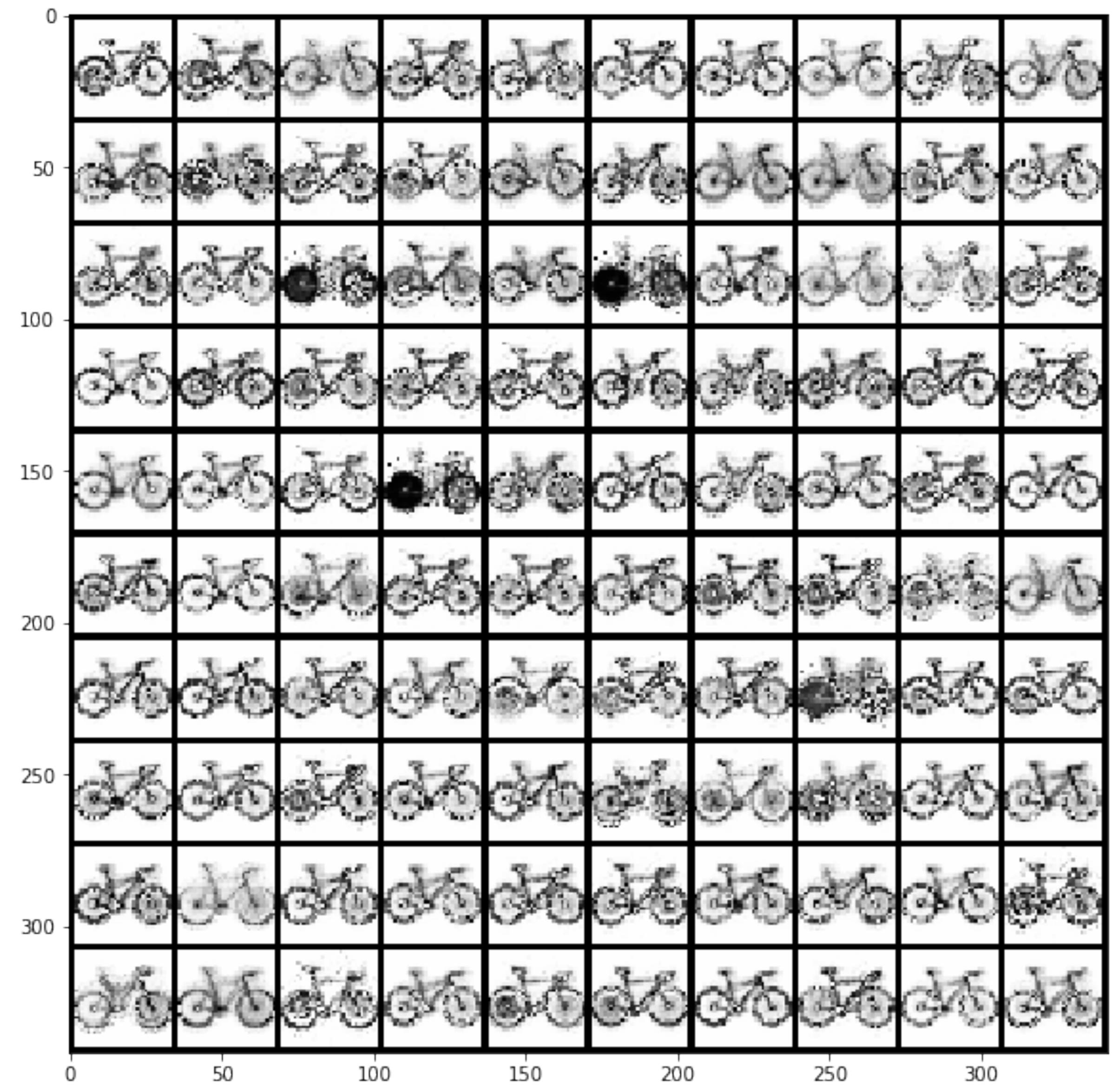
100



150

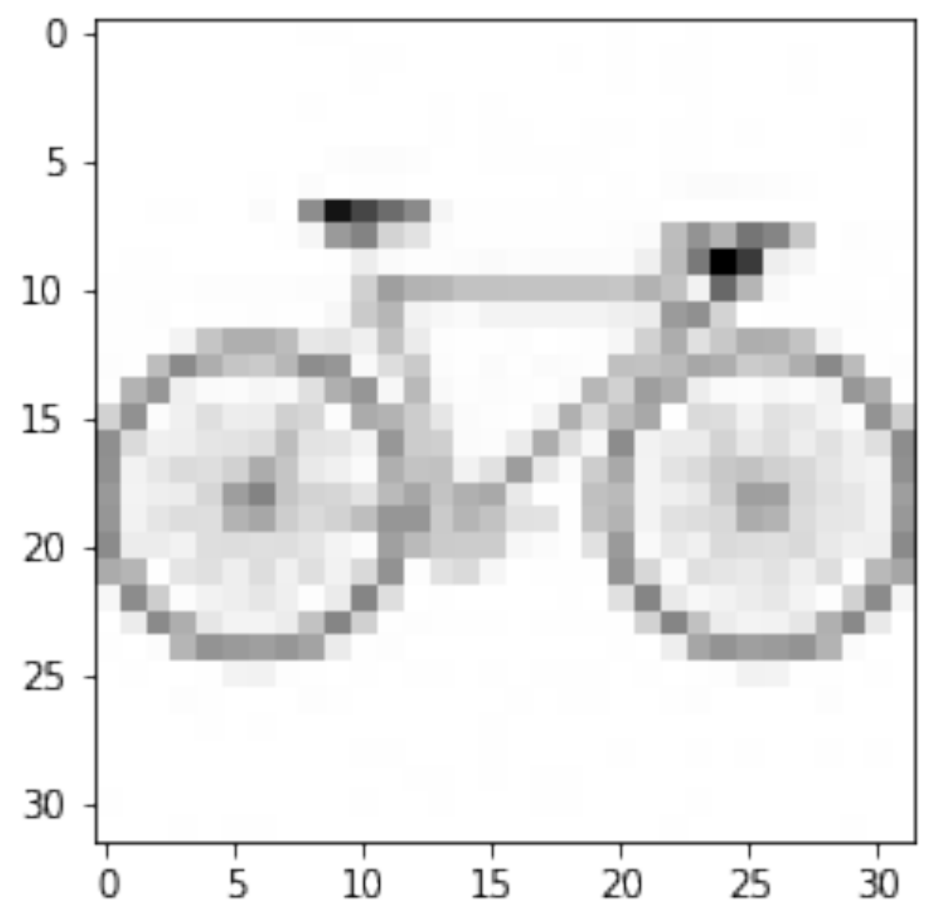


200



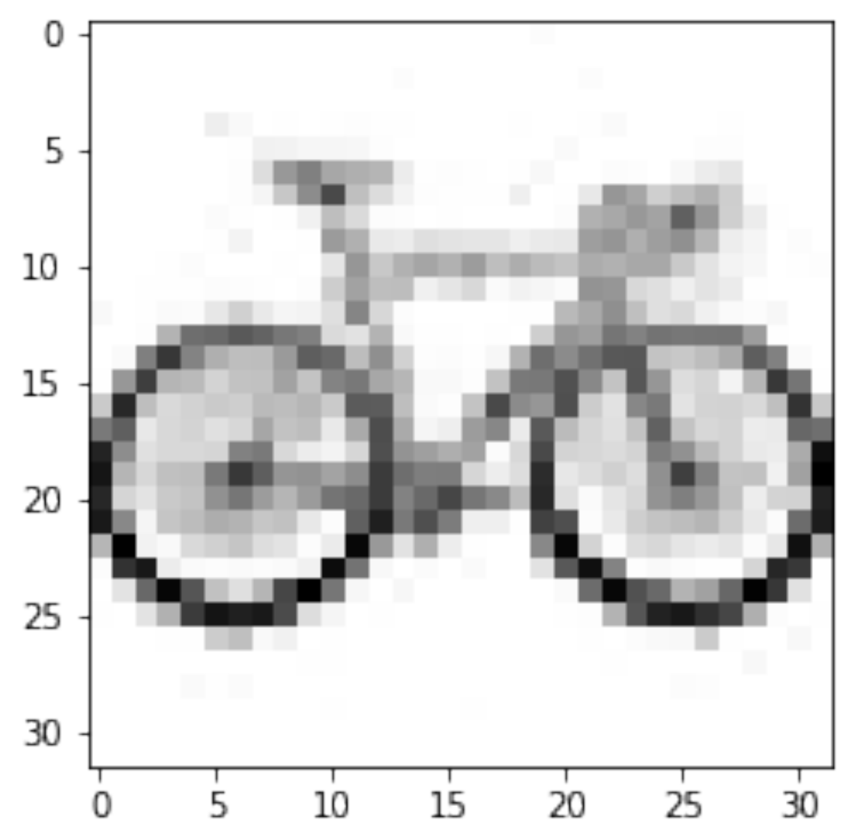
Conclusion

REAL



VS.

FAKE



-> The Bike WGAN produces good looking bike images after 200 iteration