Data Augmentation Using Generative Adversarial Networks (GANs) on the NEU Dataset [1]

Training data is an important part of almost every machine learning project and consequently the performance of classification algorithms would suffer without sufficient amount of training data. However, in many cases, we do not have access to the required amount of training data; leading us to alternative data augmentation methods.

The primary solution would be using data augmentation techniques, such as slightly rotating, flipping, or distorting the original data into new training data. However, these techniques are not always as effective as required.

Generative Adversarial Networks (GANs) is an advanced network that can generate new (synthetic) images from a distribution of (authentic) images. GANs can be applied as a data augmentation network for many problems such as defect detection in industrial production.

1. Generative Adversarial Networks (GANs) [2, 3]

Generative Adversarial Networks, or GANs, are deep learning architecture generative models that can be used to generate plausible images that are very similar to the images in the original dataset.

A generative adversarial network (GAN) is consisted of two neural networks:

1. **The generator network** that takes a random noise vector as input; and produces “synthetic” images as output. The generated images are supposed to look as similar to authentic images as possible.

2. **The discriminator network** that takes generated images as negative training examples along with authentic images as positive examples and learns to distinguish between the “authentic” and “synthetic” images.

In early steps of training, the generator produces weak images, and the discriminator can easily distinguish fake images (figure 1-top).

As training progresses, the generator learns to produce more authentic-looking images and discriminator learns as well (figure 1-middle).

Finally, in a successful training, the generator become so well at producing synthetic images that the discriminator gets worse at determining them leading to losing its accuracy (figure 1-bottom).

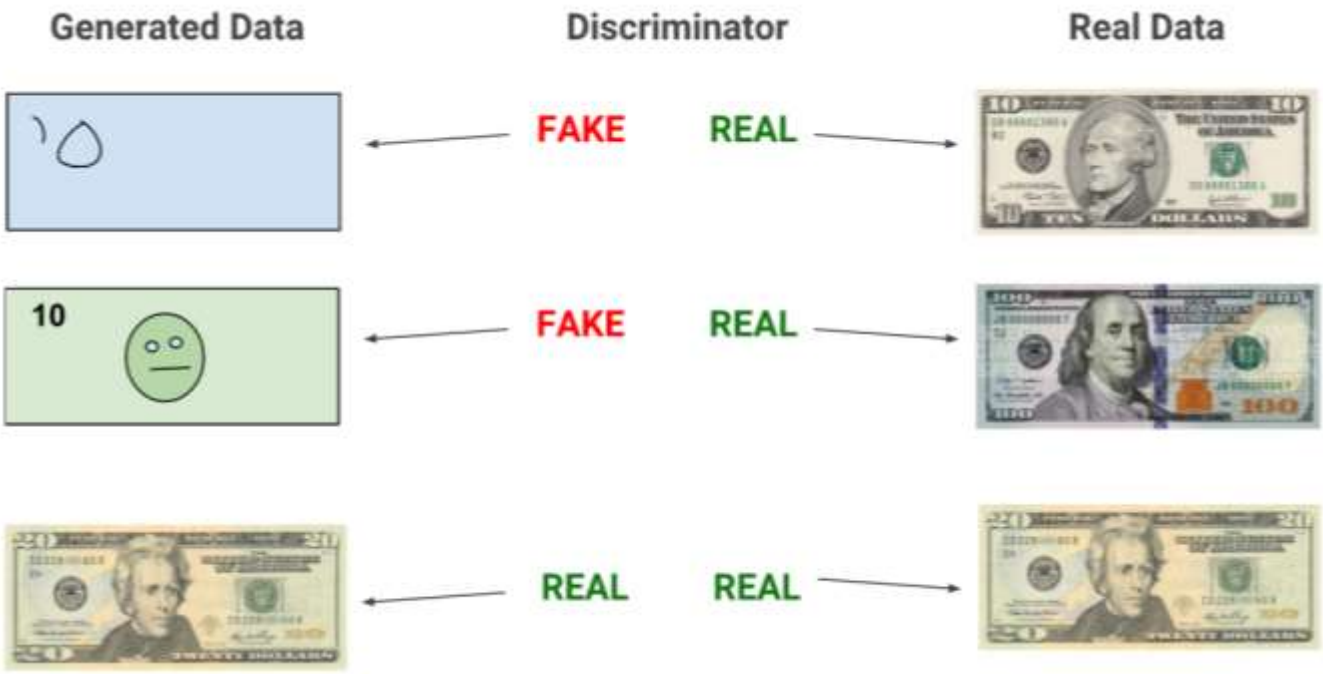

Figure 1: Simplified GAN training process [3]

By training both of these networks at the same time, one giving feedback to the other, we can learn to generate synthetic images (figure 2).

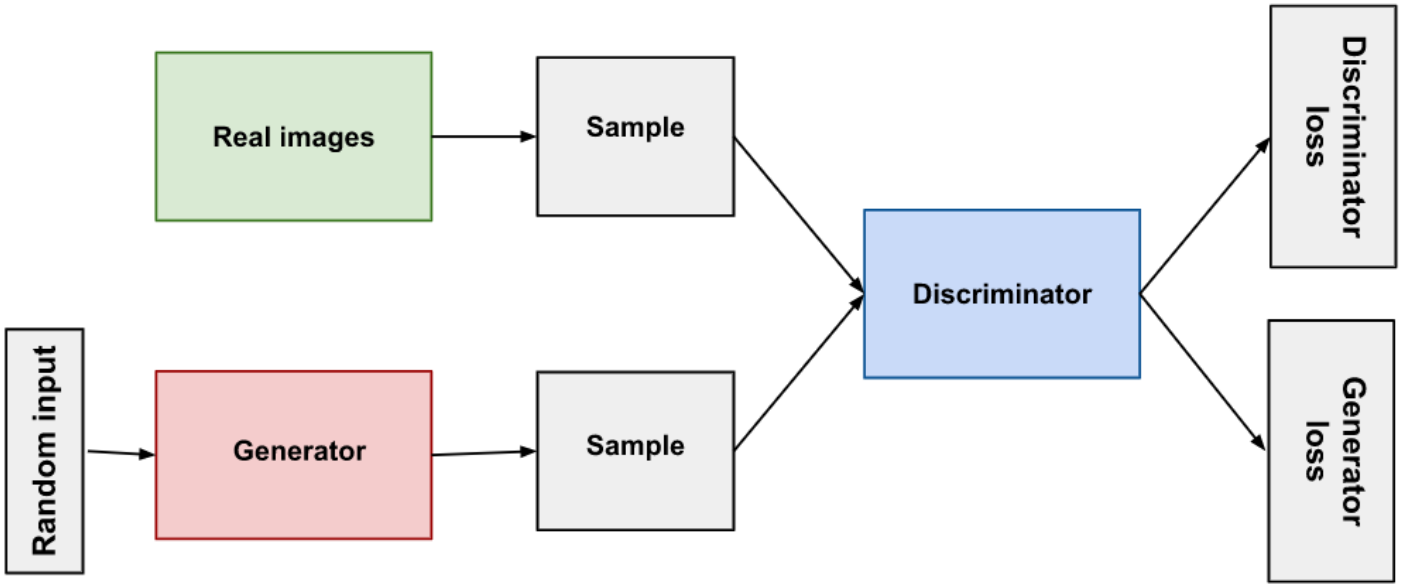

Figure 2: Overall GAN training process [3]

1.1. Deep Convolutional Generative Adversarial Network (DCGAN)

Deep convolutional generative adversarial network (DCGAN) is an extension of the GAN architecture which incorporates deep convolutional neural networks in both the generator and discriminator networks for more stability and better performance.

1.2. The discriminator [3]

The DCGAN discriminator is a classifier network with the goal of distinguishing real data from generated data (figure 3).

The discriminator needs training data consisting positive and negative examples coming from two sources:

- **Authentic (real) data** instances, such as real pictures of steel defects used as positive examples.
- **Synthetic (fake) data** instances created by the generator used as negative examples.

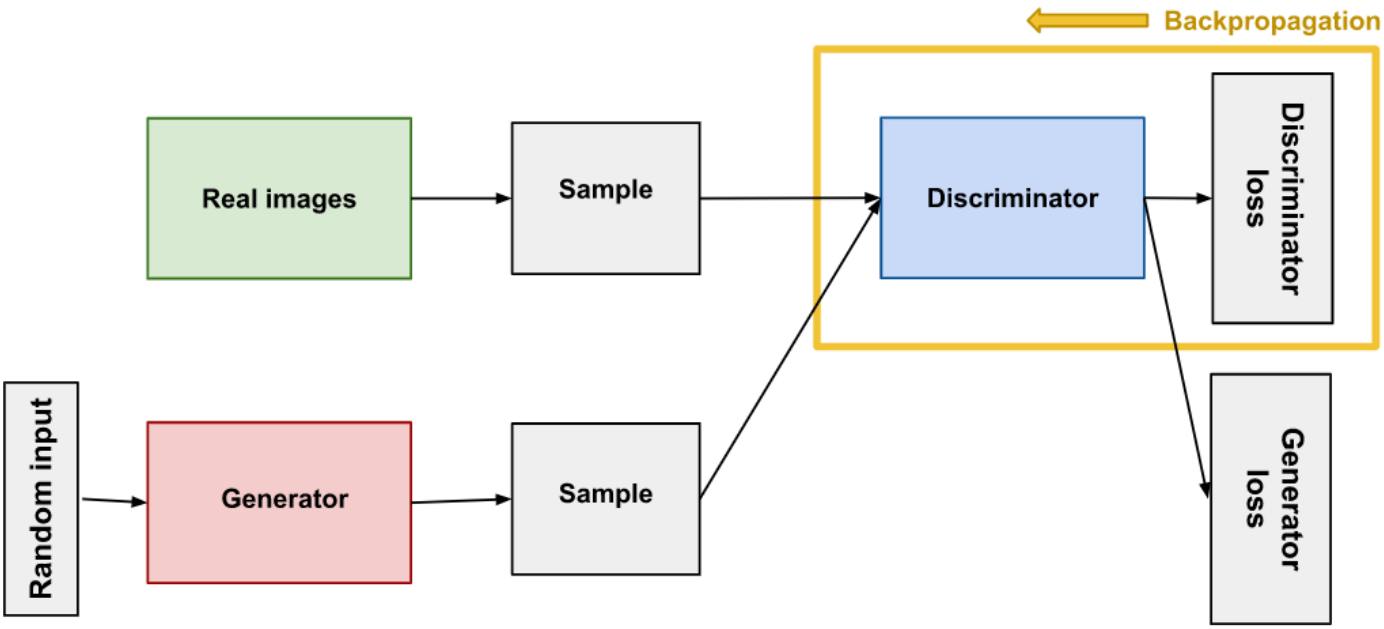

Figure 3: Backpropagation in discriminator training

1.2.1. Training the discriminator [3]

The discriminator is interacting with two loss functions: discriminator loss and generator loss. However, the generator loss is ignored during its training phase, while the discriminator loss is being used:

1. The discriminator takes a mix of real data and fake data from the generator and classified them as “Authentic” or “Synthetic”;
2. Through the discriminator loss, the discriminator would be penalized for misclassification and learn to perform better;
3. The discriminator weights are updated in the end.

1.2.2. The architecture of the discriminator [4]

The DCGAN discriminator can be categorized as a feed-forward neural network with five layers, including an input and an output layer, and three dense layers (figure 4). The discriminator network is merely a classifier processing an image and outputting a probability of the image being authentic or synthetic.

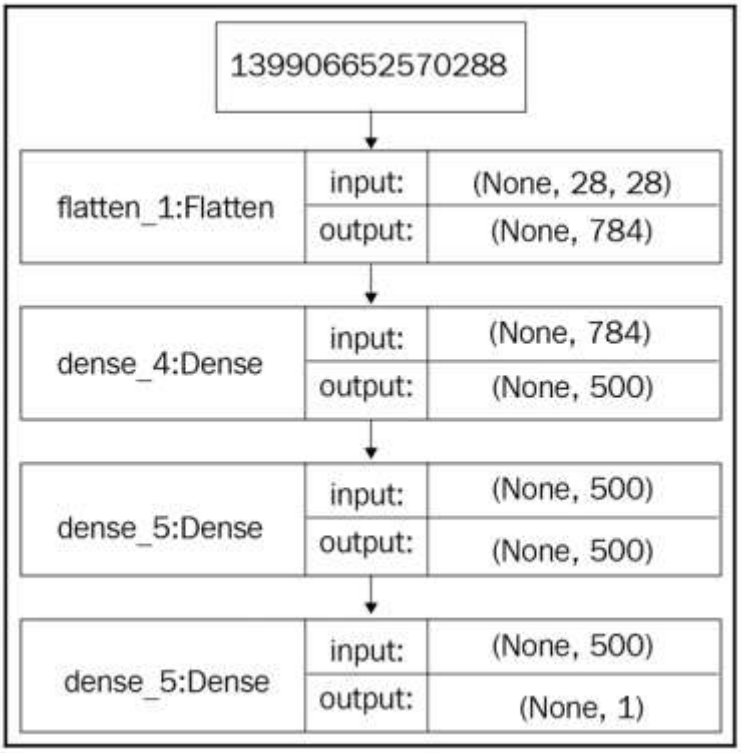

Figure 4: The architecture of the DCGAN discriminator [4]

1.3. The generator [3]

The generator attempts to create synthetic data and improves itself using the feedback from the discriminator and trying to make the discriminator classify its output as authentic.

Generator training involves more interactions with the generator than discriminator training requires. The portion of the GAN that trains the generator includes (figure 5):

- Random noise input
- Generator network, which takes the random noise vector as input and outputs synthetic images;
- Discriminator network, which classifies the generated images as “Authentic” or “Synthetic”;
- Discriminator output;
- Generator loss, which guides the generator for producing more “Authentic” looking images;

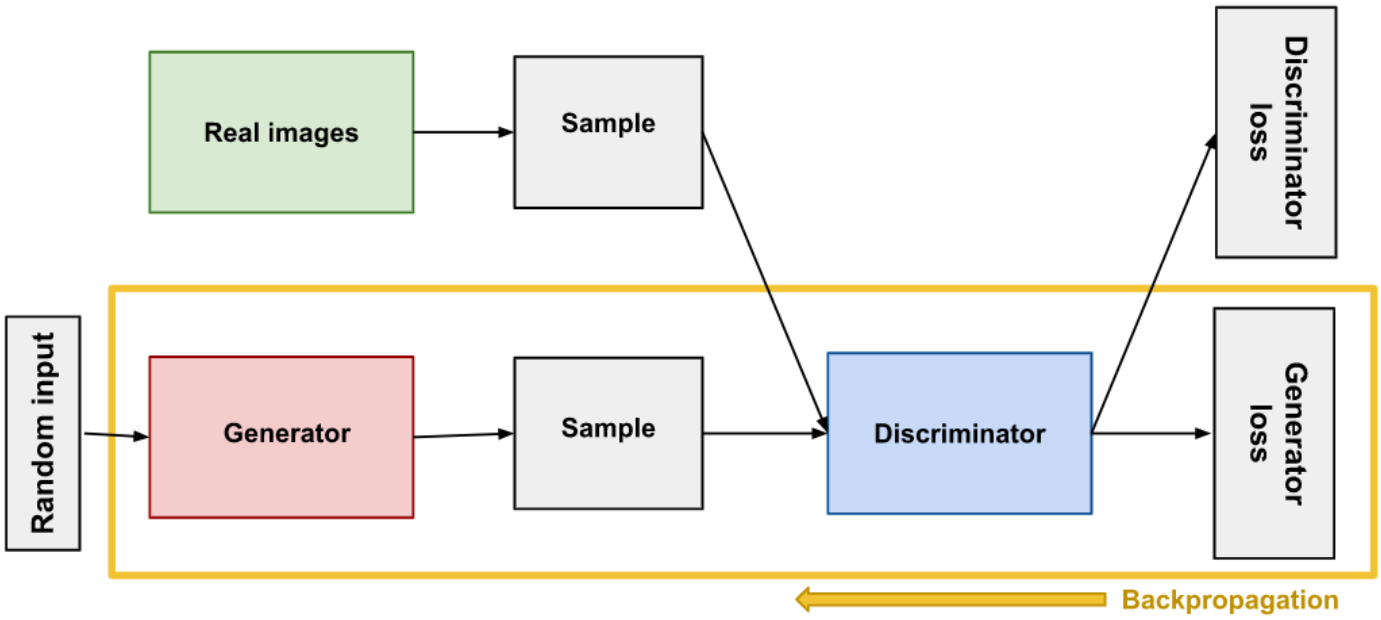

Figure 5: Backpropagation in generator training [3]

1.3.1. Why random Input? [3]

Like any other neural networks, GAN requires some input of some form. With most neural networks, the input is sampled from dataset but in GAN, synthetic images similar to dataset sample are the output. The typical input to GAN is a random noise vector for each output image.

Using random noise as input results in having more divers output images, covering all types of original dataset.

1.3.2. Using the discriminator to train the generator [3]

Training a neural network means altering the weights to minimize the error (loss) of its output. However, the GAN generator is not directly connected to the target loss (the discriminator loss).

This complication can be addressed in a backpropagation architecture. Backpropagation can adjust all weights toward the right direction considering their impacts on the output. Since the effect of a generator weight is correlated with the impact of the discriminator weights, backpropagation begins at the output and feeds backward through the discriminator into the generator.

In order to reduce the complexity of this training process, the weights of the discriminator are frozen. Otherwise, the discriminator will be changing during generator training resulting in a moving target.

The generator training process can be summarized in the following steps:

1. Taking sample random noise as input.
2. Generating output from sampled random noise.
3. Getting discriminator classification results (“Authentic” or “Synthetic”) for generated images.
4. Calculating loss from discriminator classification.
5. Backpropagating through both the discriminator and generator to obtain gradients.
6. Using gradients to change the generator weights as discriminator weights are frozen.

1.3.3. The architecture of the generator [4]

The DCGAN generator network is simply a feed-forward neural network with five layers: an input layer, three hidden layers, and an output layer (figure 6)

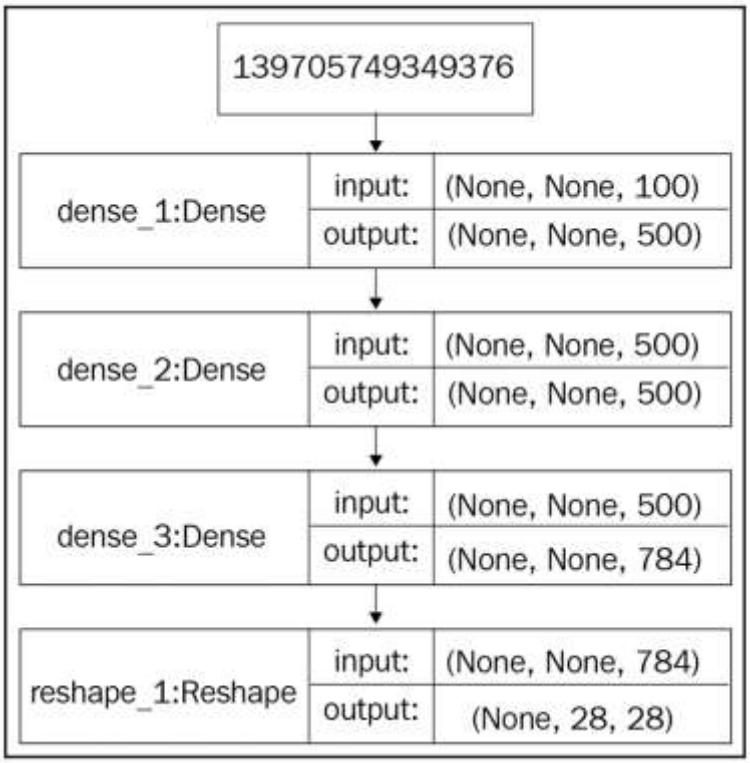

Figure 6: The architecture of the DCGAN generator [4]

1.4. The general training procedure [2]

Having discussed the architecture of DCGAN network, we can review its training process summarized in following steps:

1. Generating a random noise vector;
2. Passing this noise through the generator and having the generated synthetic images as output.
3. Building a trainset consisting of authentic images from training dataset and synthetic images from generator output and labeling them 1 and 0 (as "Authentic" or "Synthetic").
4. Training the discriminator using the "Authentic" & "Synthetic" set and evaluating its performance through the discriminator loss;
5. Training the GAN using a new random noise vector and a corresponding labelled vector of ones (as "Authentic").

2. Dataset: The NEU surface defect database [5, 6]

Surface defect detection is one of the practical problems that can be addressed using classification neural network but the available datasets are very limited: perfect case for data augmentation.

In this project, we have used the NEU surface defect database provided by the Northeastern University. This database is consisted of 1800 grayscale images of scratches on metal surface that occurred during production. These images are categorized into following 6 classes (300 images per each class) (figure 7).

- Class 0: Crazing (Cr)
- Class 1: Inclusion (In)
- Class 2: Patches (Pa)
- Class 3: Pitted Surface (PS)
- Class 4: Rolled-In Scale (RS)
- Class 5: Scratches (Sc)

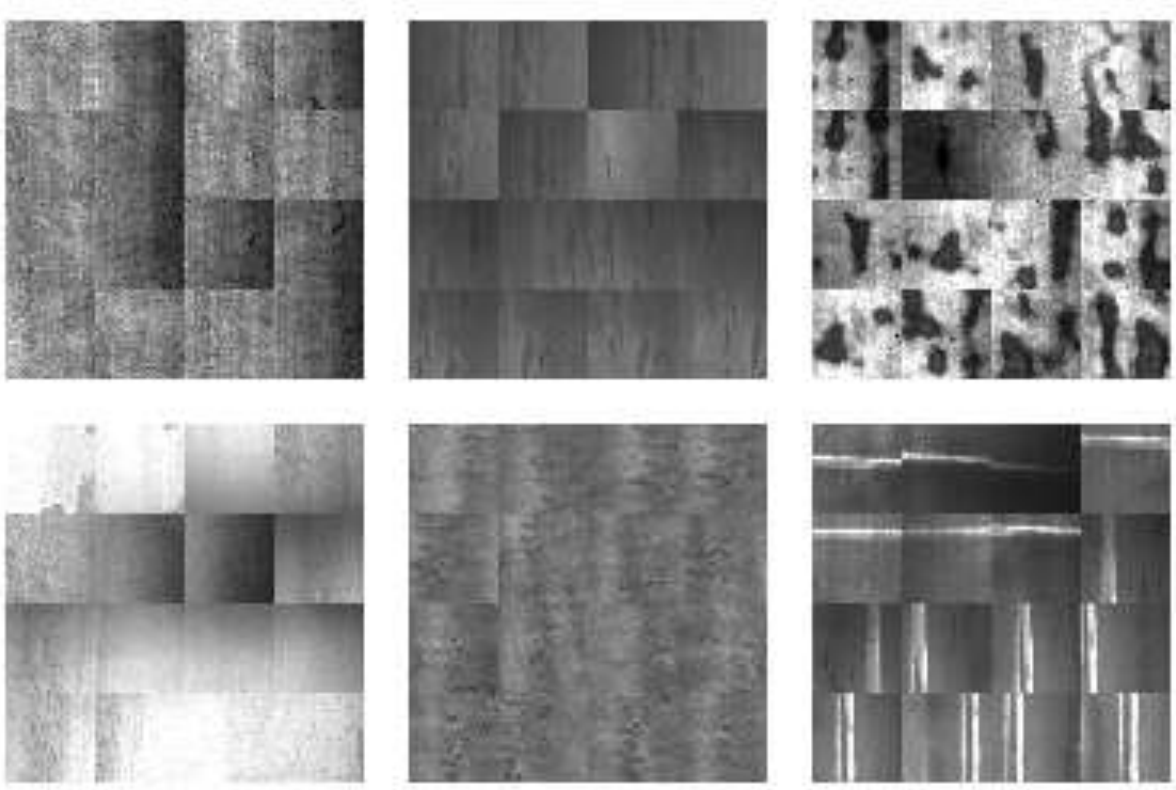

Figure 7: Samples from the NEU dataset (Top row from left: Cr, In, Pa and bottom row from left: PS, RS, SC) [6]

3. Implementing DCGAN on the NEU database

The DCGAN code from [2], originally implemented on MNIST dataset, has been used as the base code for this project. After modification, the code was implemented on the NEU database with following configurations:

- **Input data:**
  - 6 classes x 300 images x 28 x 28 x 1
  - Random noise 256 x 100
- **Number of epochs** = 500
- **Batch size** = 16
- **Discriminator Optimizer** = Adam(lr=0.0002, beta_1=0.5, decay=0.0002 / NUM_EPOCHS)
- **GAN Optimizer** = Adam(lr=0.0002, beta_1=0.5, decay=0.0002 / NUM_EPOCHS)
- **Outputs:**
  - Discriminator and adversarial loss
  - Generated images 256 images x 28 x 28 x 1

4. Results of data augmentation using DCGAN on the NEU database

4.1. Discriminator and adversarial loss during DCGAN training

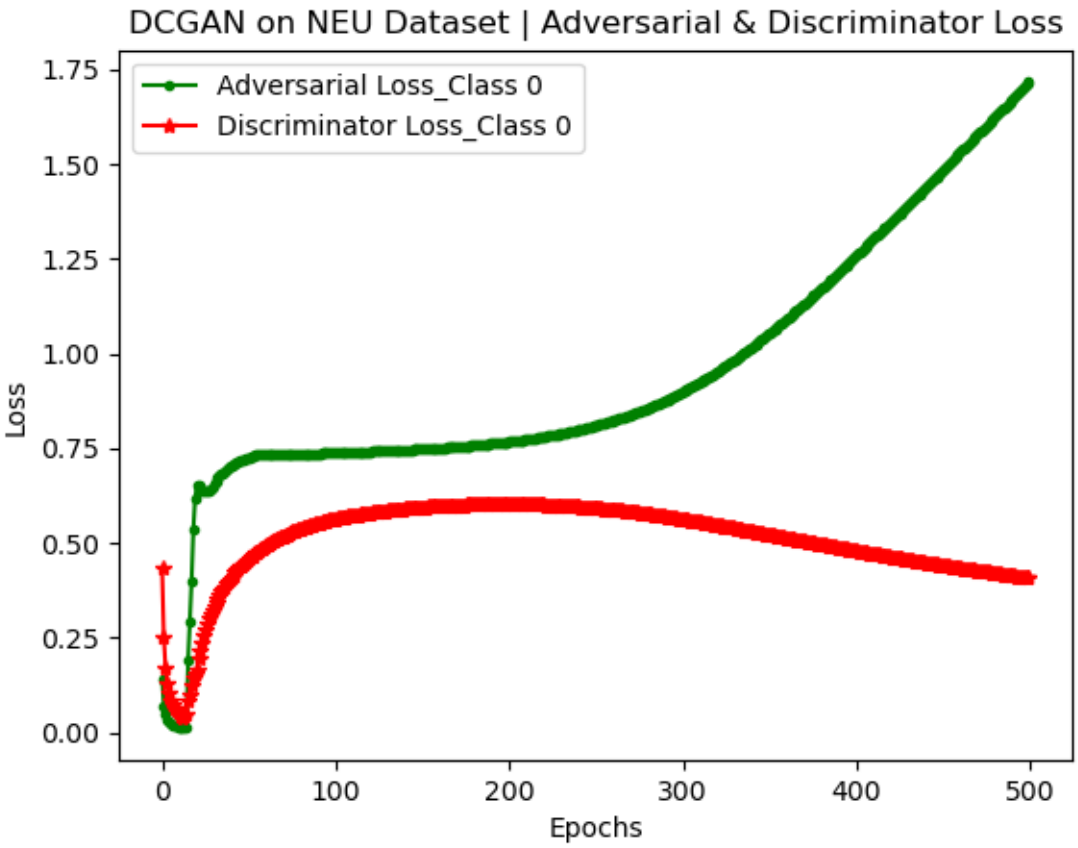

Figure 8: Discriminator and adversarial loss during DCGAN training

Remarks on the results:

- The discriminator and adversarial loss during GAN training are plotted for class 0 in the above figure.
- **The discriminator loss** decreases for a few epochs as it is learning. However, soon after that, it is increasing until epoch 200 due to the fact that generated images are more real-looking and the discriminator is misclassifying more images. After epoch 200, the discriminator loss is decreasing again due to overfitting of generator.
- **The adversarial loss** (overall loss) is increasing at first which shows that the increasing discriminator loss outweighs the decreasing generator loss. Then, it reaches some stability at around epoch 200 which shows the balance between discriminator and generator losses. After epoch 200, the adversarial loss is sharply increasing due to overfitting of generator.

4.2. Generated images through DCGAN training

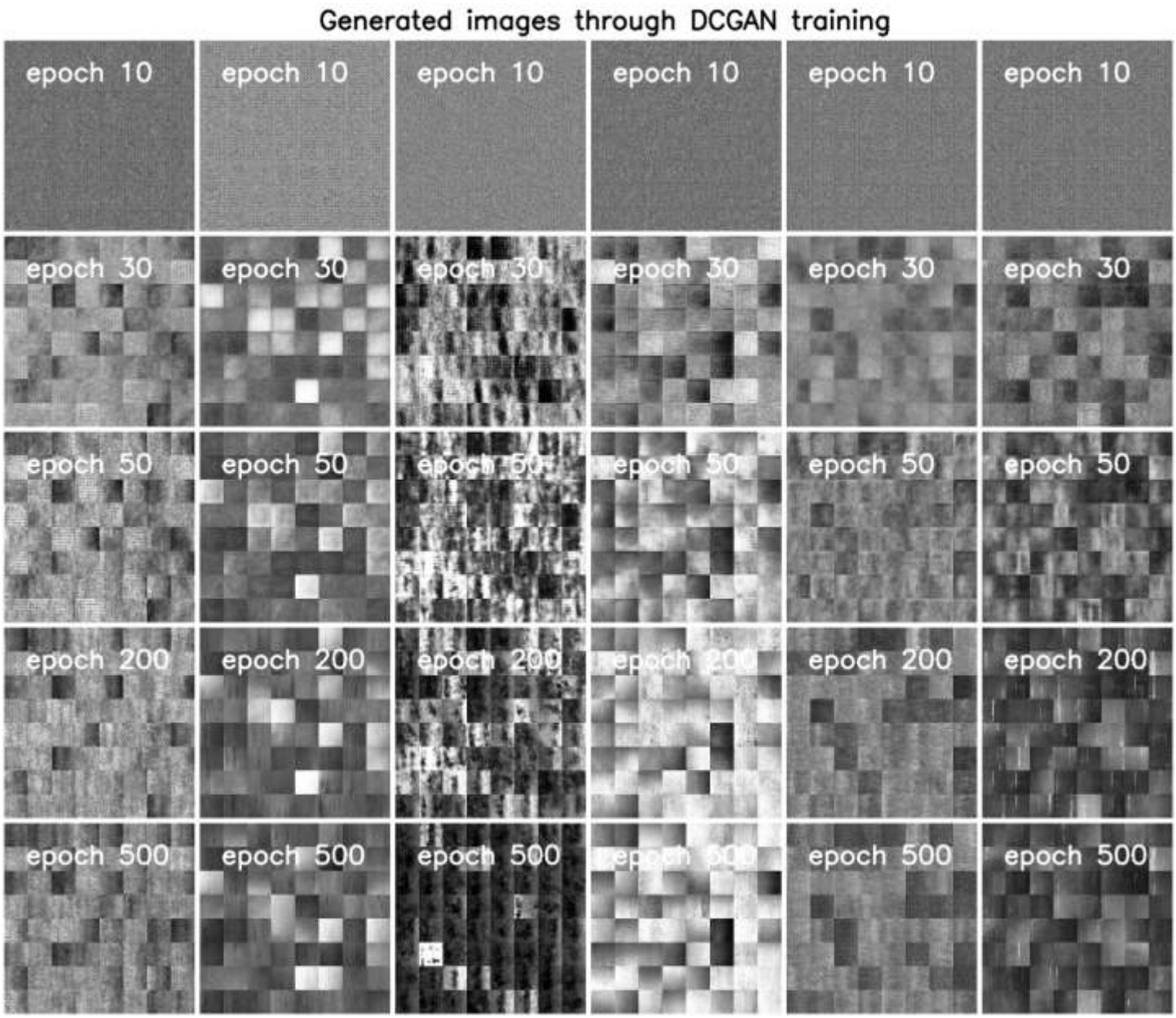

Figure 9: Generated images through DCGAN training

Remarks on the results:

- For all classes, the first generated images are just noisy grey images without any pattern but soon after epoch 20, the patterns of original classes are showing up in generations.
- From epoch 50, the generated images are not changing significantly but some important details are improving up to epoch 200 (best performance).
- After epoch 200, the generator is overfitting and its outputs are drifting away from original images in some small details.

4.3. Dataset images vs. generated images

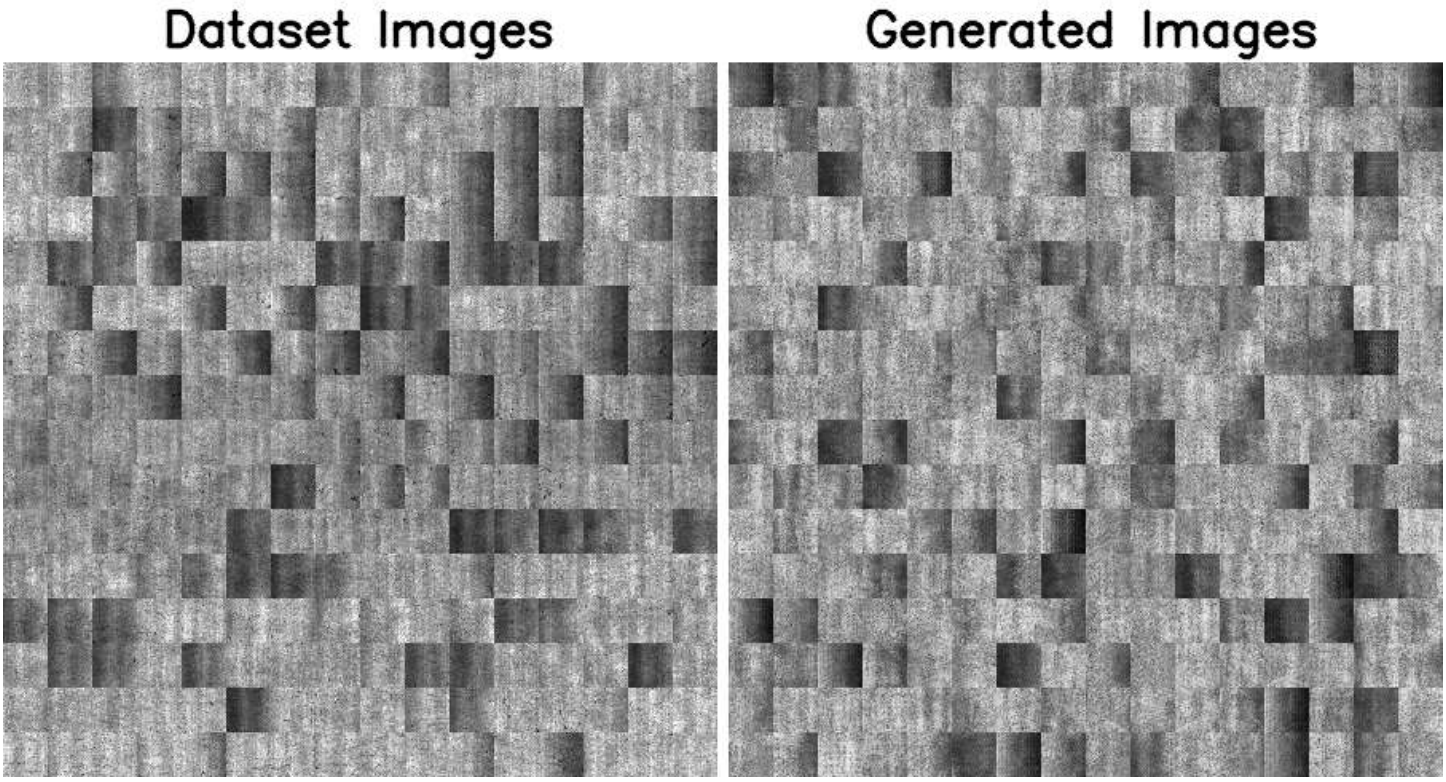

Figure 10: Dataset images vs generated images for class 0

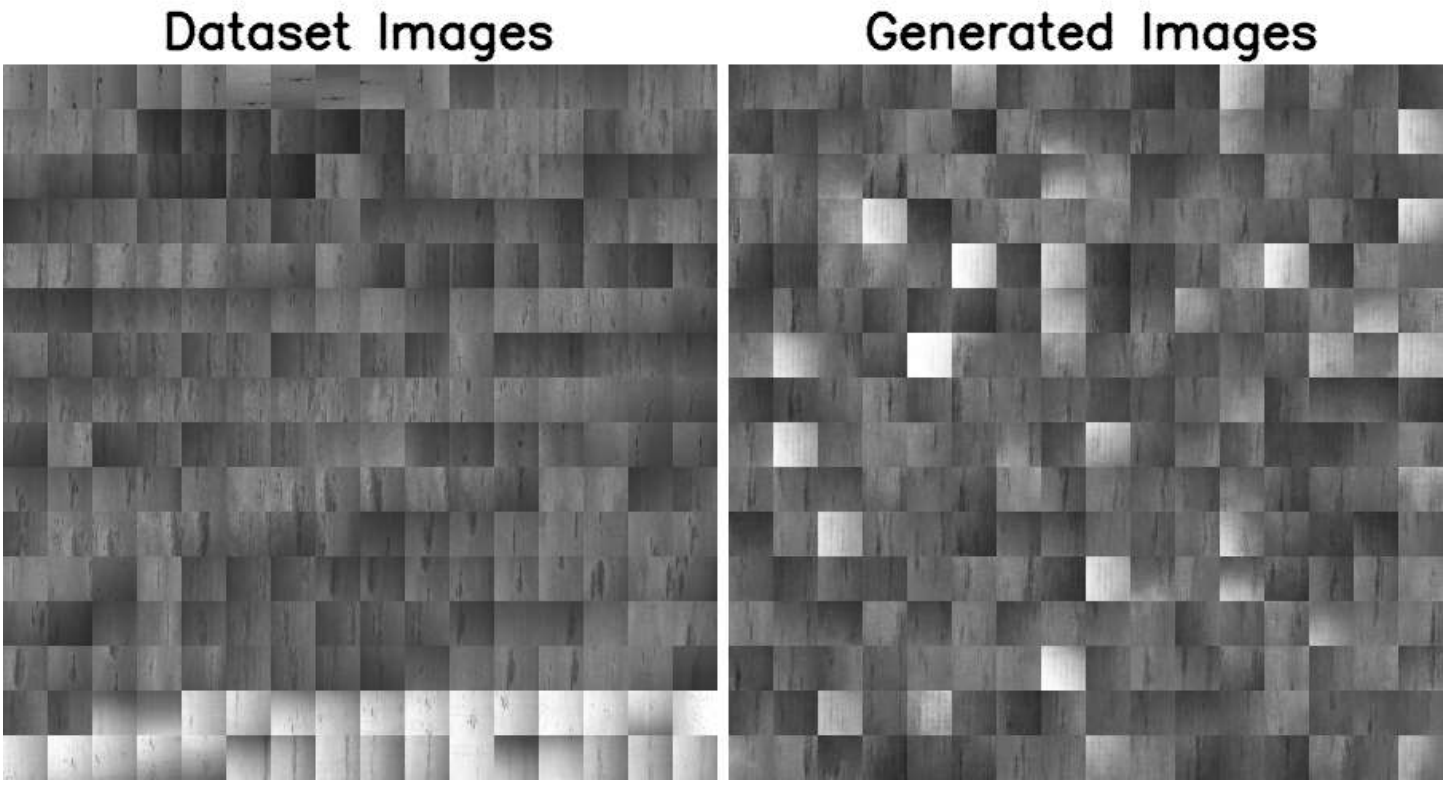

Figure 11: Dataset images vs generated images for class 1

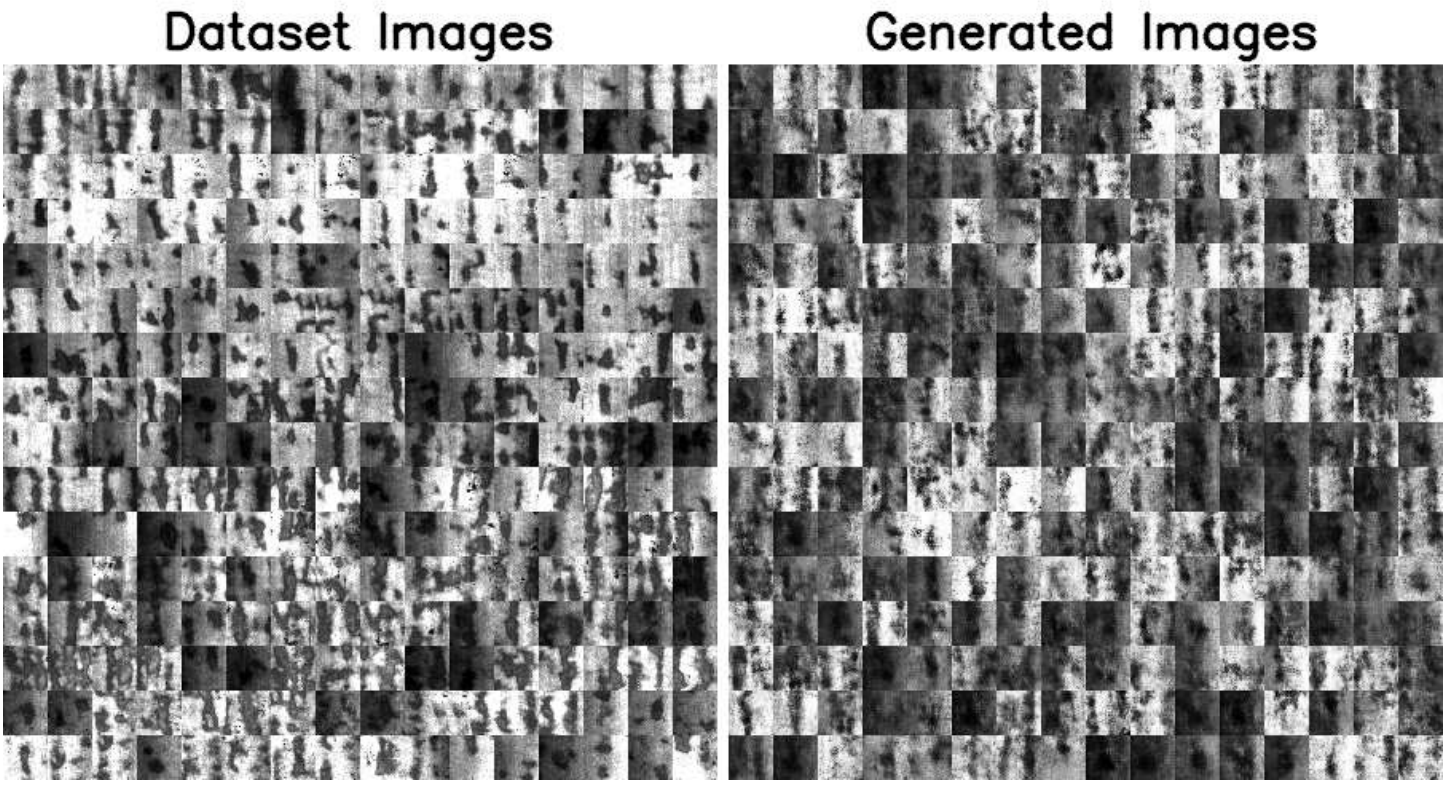

Figure 12: Dataset images vs generated images for class 2

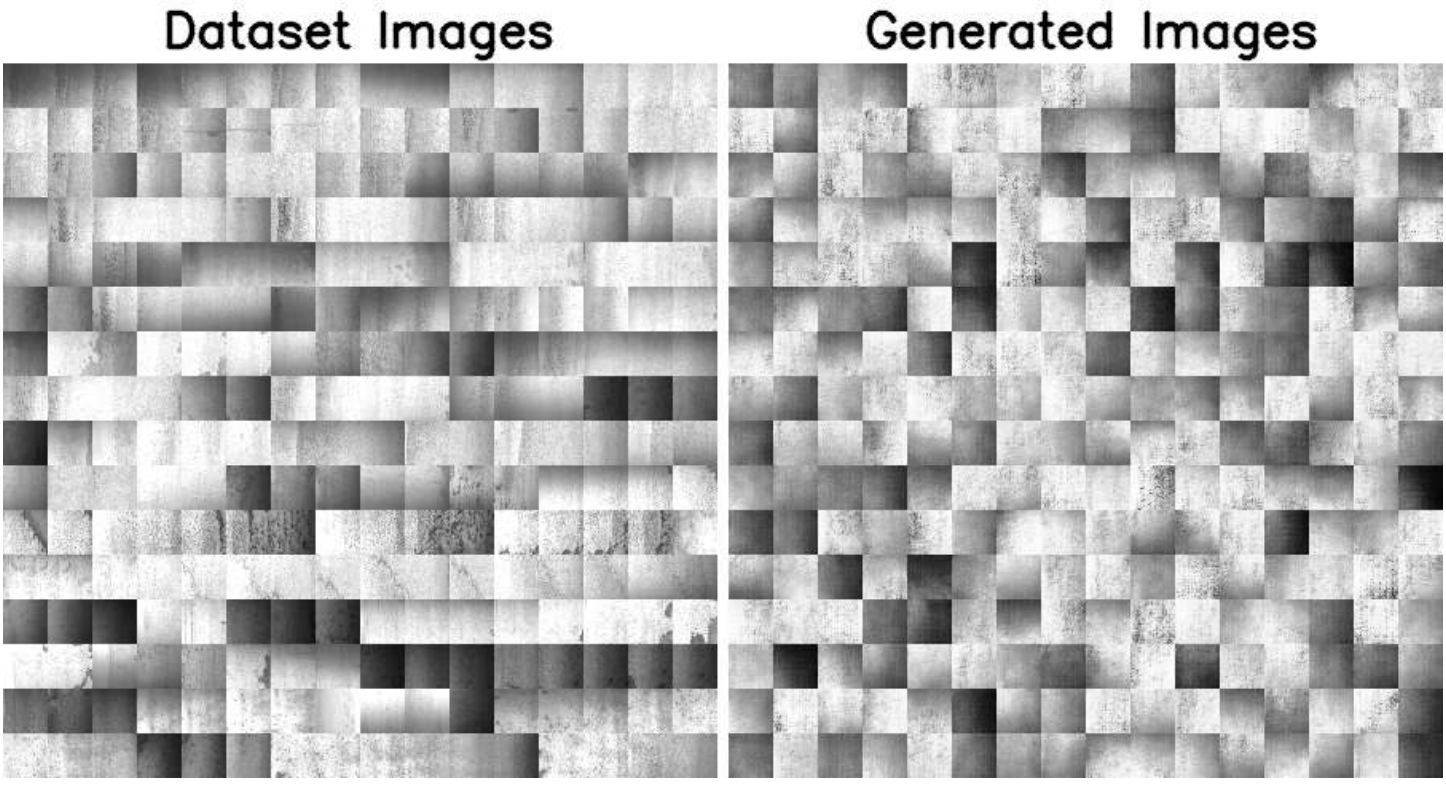

Figure 13: Dataset images vs generated images for class 3

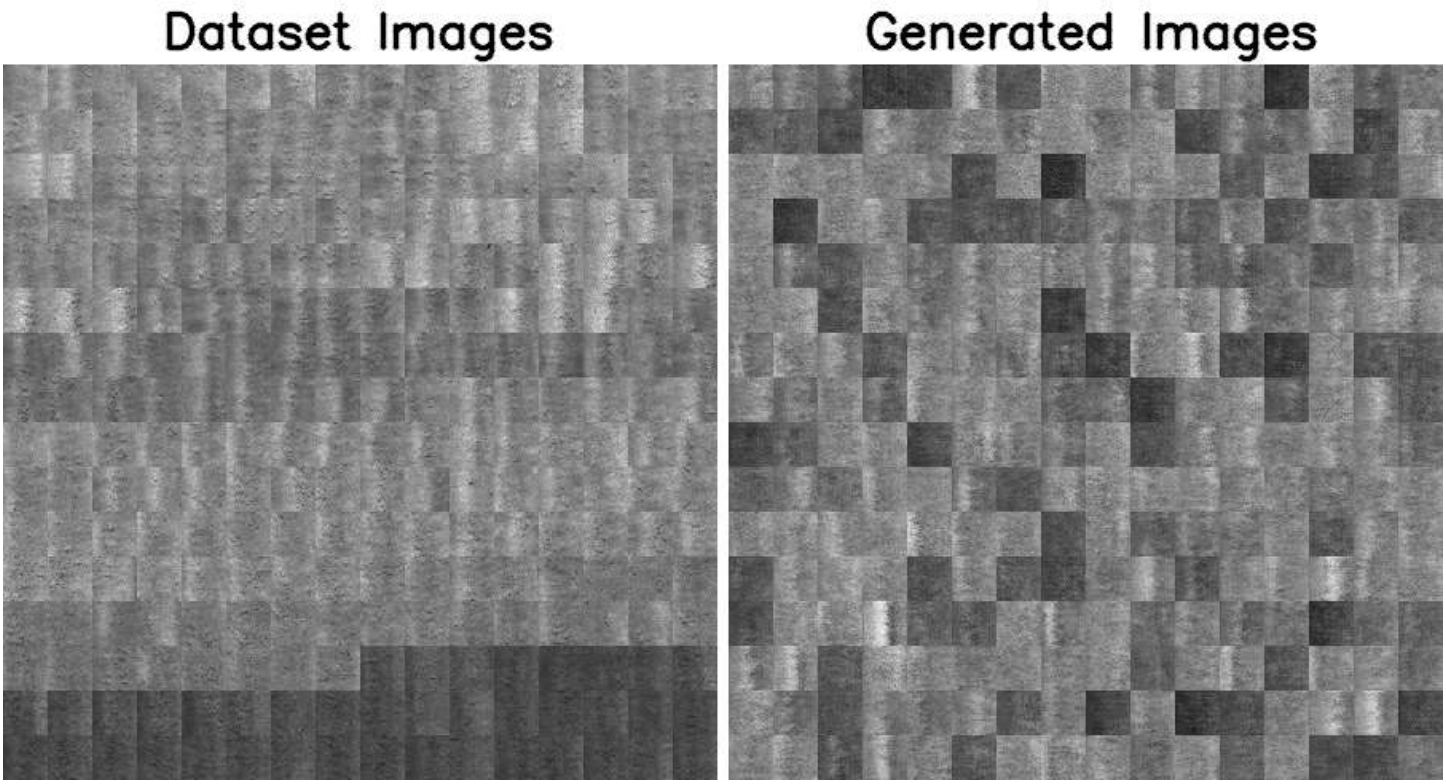

Figure 14: Dataset images vs generated images for class 4

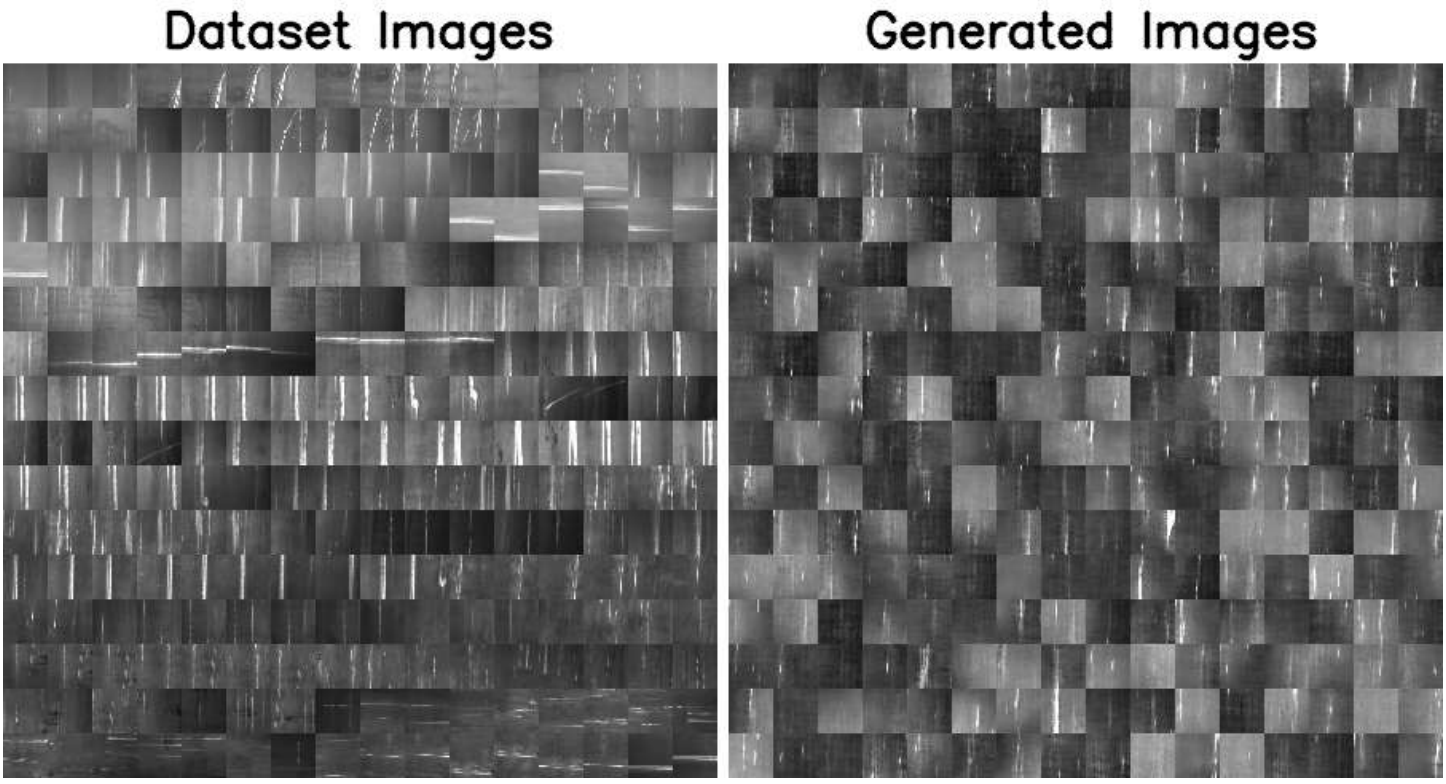

Figure 15: Dataset images vs generated images for class 5

Remarks on the results:

- For all classes except class 5, the generated images are almost identical to original images.
- The generated images for class 5 are showing similar details but not exactly the same as the database images.

4.4. DCGAN on the NEU dataset: Summary

The DCGAN network was successfully implemented on the NEU database with following results:

- Although training was continued until epoch 500, best performance (in terms of overall lost and quality of generated images) was obtained at epoch 200;
- The adversarial and discriminator loss were changing as expected during training, reaching to steady balance between epochs 100 to 200. After epoch 200, the discriminator loss decreased while the adversarial loss increased indicating the overfitting of the DCGAN;
- For most classes the generated images are almost identical to the original dataset.

References

[1] https://dida.do/blog/data-augmentation-with-gans-for-defect-detection

[2] Rosebrock, A. (2017). Deep Learning for Computer Vision with Python: Practitioner Bundle. PyImageSearch.

[3] https://developers.google.com/machine-learning/gan/discriminator

[4] Ahirwar, K. (2019). Generative Adversarial Networks Projects: Build Next-generation Generative Models Using TensorFlow and Keras. Packt Publishing Ltd.

[5] http://faculty.neu.edu.cn/yunhyan/NEU_surface_defect_database.html (Link not working)

[6] K. Song and Y. Yan, “A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects,” Applied Surface Science, vol. 285, pp. 858-864, Nov. 2013. (paper)