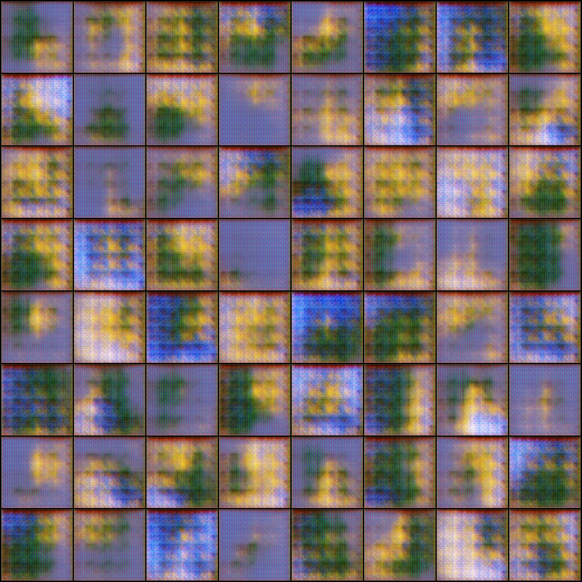
Create new simpsons characters

This document specifies the how I changed the code and how that changed the outputs.

1. I first started with resizing images to 128 x 128. Changed the architecture accordingly, by adding new convolution layers to the generator and the discriminator. Executed for 10 epochs on cpu.

Result – complete noise



A screenshot of a social media post

Description automatically generated

The output file looks like –

[0/10][0/82] Loss\_D: 1.6438 Loss\_G: 13.7977 D(x): 0.5084 D(G(z)): 0.4277 / 0.0000

47 [0/10][50/82] Loss\_D: 27.6330 Loss\_G: 0.0000 D(x): 0.9983 D(G(z)): 1.0000 / 1.0000

48 [1/10][0/82] Loss\_D: 27.6316 Loss\_G: 0.0000 D(x): 0.9994 D(G(z)): 1.0000 / 1.0000

49 [1/10][50/82] Loss\_D: 27.6312 Loss\_G: 0.0000 D(x): 0.9998 D(G(z)): 1.0000 / 1.0000

50 [2/10][0/82] Loss\_D: 27.6311 Loss\_G: 0.0000 D(x): 0.9999 D(G(z)): 1.0000 / 1.0000

51 [2/10][50/82] Loss\_D: 27.6312 Loss\_G: 0.0000 D(x): 0.9998 D(G(z)): 1.0000 / 1.0000

52 [3/10][0/82] Loss\_D: 27.6311 Loss\_G: 0.0000 D(x): 0.9999 D(G(z)): 1.0000 / 1.0000

53 [3/10][50/82] Loss\_D: 27.6311 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

54 [4/10][0/82] Loss\_D: 27.6311 Loss\_G: 0.0000 D(x): 0.9999 D(G(z)): 1.0000 / 1.0000

55 [4/10][50/82] Loss\_D: 27.6311 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

56 [5/10][0/82] Loss\_D: 27.6311 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

57 [5/10][50/82] Loss\_D: 27.6310 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

58 [6/10][0/82] Loss\_D: 27.6311 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

59 [6/10][50/82] Loss\_D: 27.6310 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

60 [7/10][0/82] Loss\_D: 27.6310 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

61 [7/10][50/82] Loss\_D: 27.6310 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

62 [8/10][0/82] Loss\_D: 27.6310 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

63 [8/10][50/82] Loss\_D: 27.6310 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

64 [9/10][0/82] Loss\_D: 27.6310 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

65 [9/10][50/82] Loss\_D: 27.6310 Loss\_G: 0.0000 D(x): 1.0000 D(G(z)): 1.0000 / 1.0000

We can see in this that D(x) and D(G(z)) are going to 1. Could be mode collapse.

1. Went back to trying with 64 x 64 size images as DCGAN is originally meant for.

For 10 epochs –

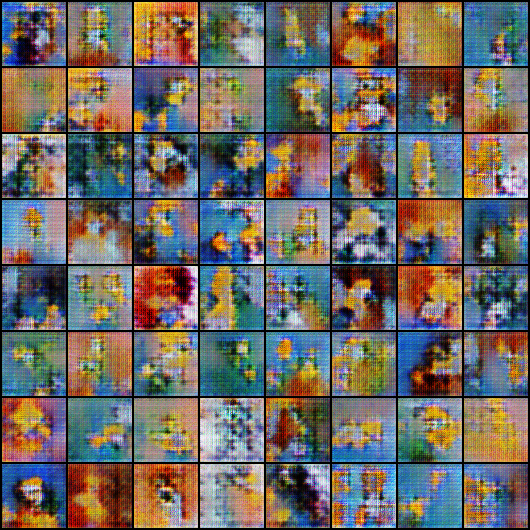
A picture containing many, building, wall

Description automatically generated

Looks like it is creating new images.

The values for losses and D(x) and D(G(z)) look better.

1. Same as 2, but for 25 epochs.

Also, one of the hacks say, ndf should be ngf/4. So, changed ndf = 16 while ngf was same as 64.



Images seem to get better, looks grainy though.

A screenshot of a cell phone

Description automatically generated

Don’t know how to interpret this yet.

1. Trying same for 100 epochs.



A screenshot of a cell phone

Description automatically generated

Discriminator looks good but generator loss is increasing and also varying too much.

Can improve based on this..

1. ndf = 16 while ngf = 64 for 128x128 image size to see if this helps solve the problem of mode\_collapse.

Epochs = 20

A picture containing colorful

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Looks like can be trained more.

1. Continue same for 50 epochs.

A picture containing colorful, indoor, many, wall

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Images generated after changing ndf = 16 are better. But generator loss is still too varied. The variations increase with number of epochs.