[[Towards Data Science](https://towardsdatascience.com/?source=post_page-----39f67f83c7b4--------------------------------)](https://towardsdatascience.com/?source=post_page-----39f67f83c7b4--------------------------------)

[Sign in](https://medium.com/m/signin?operation=login&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fgenerating-modern-arts-using-generative-adversarial-network-gan-on-spell-39f67f83c7b4&source=post_page-----39f67f83c7b4---------------------nav_reg-----------)

[Get started](https://medium.com/m/signin?operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fgenerating-modern-arts-using-generative-adversarial-network-gan-on-spell-39f67f83c7b4&source=post_page-----39f67f83c7b4---------------------nav_reg-----------)

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fcollection%2Ftowards-data-science&operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fgenerating-modern-arts-using-generative-adversarial-network-gan-on-spell-39f67f83c7b4&collection=Towards%20Data%20Science&collectionId=7f60cf5620c9&source=post_page-----39f67f83c7b4---------------------follow_header-----------)

[577K Followers](https://towardsdatascience.com/followers?source=post_page-----39f67f83c7b4--------------------------------)

·

[Editors' Picks](https://towardsdatascience.com/tagged/editors-pick?source=post_page-----39f67f83c7b4--------------------------------)[Features](https://towardsdatascience.com/tagged/tds-features?source=post_page-----39f67f83c7b4--------------------------------)[Deep Dives](https://towardsdatascience.com/tagged/deep-dives?source=post_page-----39f67f83c7b4--------------------------------)[Grow](https://towardsdatascience.com/how-to-get-the-most-out-of-towards-data-science-3bf37f75a345?source=post_page-----39f67f83c7b4--------------------------------)[Contribute](https://towardsdatascience.com/questions-96667b06af5?source=post_page-----39f67f83c7b4--------------------------------)

[About](https://towardsdatascience.com/about?source=post_page-----39f67f83c7b4--------------------------------)



You have **1** free member-only story left this month.

[Sign up for Medium and get an extra one](https://medium.com/m/signin?operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fgenerating-modern-arts-using-generative-adversarial-network-gan-on-spell-39f67f83c7b4&source=-----39f67f83c7b4---------------------metered_view_2-----------)

**Fun GAN**

**Generating Modern Art using  
Generative Adversarial Network(GAN) on Spell**

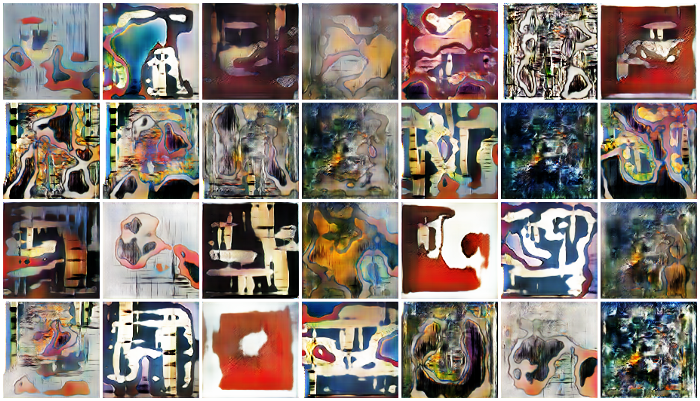
**Creating a Generative Adversarial Network to generate modern art and training it on GPU of Spell platform**

[Anish Shrestha](https://medium.com/@sthacruz?source=post_page-----39f67f83c7b4--------------------------------)

[Anish Shrestha](https://medium.com/@sthacruz?source=post_page-----39f67f83c7b4--------------------------------)

[Nov 13, 2019·17 min read](https://towardsdatascience.com/generating-modern-arts-using-generative-adversarial-network-gan-on-spell-39f67f83c7b4?source=post_page-----39f67f83c7b4--------------------------------)

https://miro.medium.com/max/60/0*1prOPM-kWOJ2tT21?q=20



Arts Generated By Our GAN

**Prerequisites**

You need to have a good understanding of:

1. Image processing
2. Python Programming Language
3. Numpy — Scientific Computing Library
4. Keras — Deep Learning Library

And some basic knowledge of:

1. Generative Adversarial Network (GAN)

**Dataset Used**

Image data used in this project has been collected from [**WikiArts.org**](https://www.wikiart.org/).

**Introduction**

In this tutorial, we are going to look at the step by step process to create a Generative Adversarial Network to generate Modern Art and write a code for that using Python and Keras together.

After that, for training the model, we are going to use a powerful GPU Instance of Spell platform. Everything will be explained along the way and links will be provided for further readings.

Let’s get started!

**Exploring Dataset**

Before getting started, let’s look at our image dataset.

WikiArt has a huge collection of modern art with various different styles. For our particular project, we are going to use images of [**Cubism**](https://www.wikiart.org/en/paintings-by-style/cubism?select=featured#!#filterName:featured,viewType:masonry) Style.



You can know more about the art styles and Modern Art from [WikiArt.org](https://www.wikiart.org/).

**Downloading Images**

You can either download images that you like from WikiArt or head to this repository by [**cs-chan**](https://github.com/cs-chan/ArtGAN/tree/master/WikiArt%20Dataset) to download the 26GB of WikiArt images.

Since it has a collection of all the different types, we are only going to pick **cubism** and store them in folder named dataset.

https://miro.medium.com/max/60/0*-xytA99_61uWlUg_?q=20

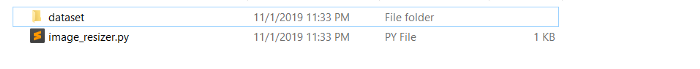


**Processing Image Data**

Images in our dataset are of different sizes, to feed them into our Generative Adversarial Neural Network we are going to resize all our images to 128X128.

Before starting, create a python file at the root directory where your dataset folder is located.

https://miro.medium.com/max/60/0*7VIiCX8zmcstVMC1?q=20



Now let’s write a small python script to select all the images from the folder and resize them to 128X128 and save them on **cubism\_data.npy** file.

*## image\_resizer.py*  
*# Importing required libraries*  
**import** os  
**import** numpy **as** np  
**from** PIL **import** Image  
  
*# Defining an image size and image channel*  
*# We are going to resize all our images to 128X128 size and since our images are colored images*  
*# We are setting our image channels to 3 (RGB)*  
  
IMAGE\_SIZE = 128  
IMAGE\_CHANNELS = 3  
IMAGE\_DIR = 'dataset/'  
  
*# Defining image dir path. Change this if you have different directory*  
images\_path = IMAGE\_DIR   
  
training\_data = []  
  
*# Iterating over the images inside the directory and resizing them using*  
*# Pillow's resize method.*  
print('resizing...')  
  
**for** filename **in** os.listdir(images\_path):  
 path = os.path.join(images\_path, filename)  
 image = Image.open(path).resize((IMAGE\_SIZE, IMAGE\_SIZE), Image.ANTIALIAS)  
  
 training\_data.append(np.asarray(image))  
  
training\_data = np.reshape(  
 training\_data, (-1, IMAGE\_SIZE, IMAGE\_SIZE, IMAGE\_CHANNELS))  
training\_data = training\_data / 127.5 - 1  
  
print('saving file...')  
np.save('cubism\_data.npy', training\_data)

Let’s break it down.

In our code block above, in the first few lines, we have imported all the required libraries to perform the resizing operation.

**import** os  
**import** numpy **as** np  
**from** PIL **import** Image  
  
IMAGE\_SIZE = 128  
IMAGE\_CHANNELS = 3  
IMAGE\_DIR = 'dataset/'  
  
images\_path = IMAGE\_DIR

Here, we are using Pillow to resize all images to our desired size and appending them on a list as numpy array.

training\_data = []**for** filename **in** os.listdir(images\_path):  
 path = os.path.join(images\_path, filename)  
 image = Image.open(path).resize((IMAGE\_SIZE, IMAGE\_SIZE), Image.ANTIALIAS)  
training\_data.append(np.asarray(image))

After that, we are using **numpy** to reshape the array in a suitable format and normalizing data.

training\_data = np.reshape(  
training\_data, (-1, IMAGE\_SIZE, IMAGE\_SIZE, IMAGE\_CHANNELS))  
training\_data = training\_data / 127.5–1

After normalization, we are saving our image array in **npy** binary file so that we don’t have to go through all the images every time.

np.save(‘cubism\_data.npy’, training\_data)

That’s it for processing our image data.

**Creating GAN**

Now it’s time for the most exciting part of our project, from here on we are going to write our code for Generative Adversarial Network (GAN).

We are going to use **Keras — A Deep Learning Library** to create our GAN.

Before starting let’s briefly understand what is GAN and it’s structure.

**What is GAN?**

Generative adversarial networks (GANs) are an exciting recent innovation in machine learning. It was first introduced by Ian Godfellow in his paper [**Generative Adversarial Networks**](https://arxiv.org/abs/1406.2661).

GANs are **generative** models: after given some training data, they can create new data instances that look like your training data. For example, GANs can create images that look like photographs of human faces, even though the faces don’t belong to any real person.

For a great example of GAN you can visit [**https://www.thispersondoesnotexist.com/**](https://www.thispersondoesnotexist.com/) which was created by **Nvidia**. It generates a high quality image of a person who does not even exist.

Sounds interesting right?

**How does it work?**

Let’s understand it’s structure and how it works.

https://miro.medium.com/max/60/0*cWzeYvhtN48sjqGN?q=20

GAN composes of two types of models: Generative Model and a Discriminative Model.

Generative Models are responsible for generating different kinds of noise data whereas discriminative models are responsible to discriminate whether the given data is real or fake.

https://miro.medium.com/max/60/0*IZtx-_pohl2NzGMP?q=20

Generative models constantly trains itself to fool discriminative models by generating fake noise data and discriminative models trains itself from the training set to classify either the data is from dataset or not and not to be fooled by generative models.

https://miro.medium.com/max/60/0*NdsdJRXUhpa45UJH?q=20

GAN structure [source](https://skymind.ai/images/wiki/GANs.png)

**Loss Function**

Discriminator in GAN uses a cross entropy loss, since discriminators job is to classify; cross entropy loss is the best one out there.

https://miro.medium.com/max/60/0*w4ITupD2-EPV8pgh?q=20

This formula represents the cross entropy loss between p: the true distribution and q: the estimated distribution.

(p) and (q) are the of m dimensions where m is the number of classes.

https://miro.medium.com/max/60/0*a0s18Q1a596HEHId?q=20

In GAN, discriminator is a binary classifier. It needs to classify either the data is real or fake. Which means m = 2. The true distribution is one hot vector consisting of only 2 terms.

For n number of samples, we can sum over the losses.

This above shown equation is of binary cross entropy loss, where y can take two values 0 and 1.

GAN’s have a latent vector z, image G(z) is magically generated out of it. We apply the discriminator function D with real image x and the generated image G(z).

The intention of the loss function is to push the predictions of the real image towards 1 and the fake images to 0. We do so by log probability term.

https://miro.medium.com/max/60/0*WFN6GA_JM5_h1q1k?q=20

**Note**: **~** sign means: is distributed as and **Ex** here means expectations: since we don’t know how samples are fed into the discriminator, we are representing them as expectations rather than the sum.

If we observe the joint loss function we are maximizing the discriminator term, which means log of D(x) should inch closer to zero, and log D(G(z)) should be closer to 1. Here generator is trying to make D(G(z)) inch closer to 1 while discriminator is trying to do the opposite.

**Code for GAN**

Now without any delay let’s write our GAN.

We are going to name our file **art\_gan.py** and store it in the root directory. This file will contain all the hyperparameters and functions for our generator and discriminator.

Let’s write some code:

**from** keras.layers **import** Input, Reshape, Dropout, Dense, Flatten, BatchNormalization, Activation, ZeroPadding2D  
**from** keras.layers.advanced\_activations **import** LeakyReLU  
**from** keras.layers.convolutional **import** UpSampling2D, Conv2D  
**from** keras.models **import** Sequential, Model, load\_model  
**from** keras.optimizers **import** Adam  
**import** numpy **as** np  
**from** PIL **import** Image  
**import** os

Here we are importing all the required libraries and helper functions for creating our GAN.

All the imports are self-explanatory. Here we are importing a bunch of keras layers for creating our models.

Now let’s define some parameters:

*# Preview image Frame  
PREVIEW\_ROWS = 4  
PREVIEW\_COLS = 7  
PREVIEW\_MARGIN = 4  
SAVE\_FREQ = 100# Size vector to generate images from  
NOISE\_SIZE = 100# Configuration  
EPOCHS = 10000 # number of iterations  
BATCH\_SIZE = 32GENERATE\_RES = 3  
IMAGE\_SIZE = 128 # rows/cols*IMAGE\_CHANNELS = 3

Here in the first few lines, we have defined Image frame size and padding to save our generated images.

**NOISE\_SIZE** here is a latent dimension size to generate our images.

**EPOCHS** is a number of iterations: it defines how many times we want to iterate over our training images and **BATCH\_SIZE** is a number of images to feed in every iteration.

**IMAGE\_SIZE** is our image size which we resized earlier to 128X128 and **IMAGE\_CHANNELS** is a number of channel in our images; which is 3.

*Note: Images should always be of square size*

Let's load our npy data file which we’ve created earlier.

training\_data = np.load(‘cubism\_data.npy’)

To load the npy file we are using numpy’s load function and passing file path as a parameter.

Since we have our data file in the root directory we no additional path parameters were required. If you have stored your data somewhere else, you can use the following code to load data:

training\_data = np.load(os.path.join(‘dirname’, ‘filename.npy’))

That’s it for loading our training data.

Now we can create our Generator and Discriminator functions.

Let’s see that in code:

**def** **build\_discriminator**(image\_shape): model = Sequential() model.add(Conv2D(32, kernel\_size=3, strides=2,  
 input\_shape=image\_shape, padding=”same”))  
 model.add(LeakyReLU(alpha=0.2))  
 model.add(Dropout(0.25)) model.add(Conv2D(64, kernel\_size=3, strides=2, padding=”same”))  
 model.add(ZeroPadding2D(padding=((0, 1), (0, 1))))  
 model.add(BatchNormalization(momentum=0.8))  
 model.add(LeakyReLU(alpha=0.2))  
 model.add(Dropout(0.25)) model.add(Conv2D(128, kernel\_size=3, strides=2, padding=”same”))  
 model.add(BatchNormalization(momentum=0.8))  
 model.add(LeakyReLU(alpha=0.2))  
 model.add(Dropout(0.25)) model.add(Conv2D(256, kernel\_size=3, strides=1, padding=”same”))  
 model.add(BatchNormalization(momentum=0.8))  
 model.add(LeakyReLU(alpha=0.2)) model.add(Dropout(0.25))  
 model.add(Conv2D(512, kernel\_size=3, strides=1, padding=”same”))  
 model.add(BatchNormalization(momentum=0.8))  
 model.add(LeakyReLU(alpha=0.2)) model.add(Dropout(0.25))  
 model.add(Flatten())  
 model.add(Dense(1, activation=’sigmoid’)) input\_image = Input(shape=image\_shape)  
 validity = model(input\_image)  
 **return** Model(input\_image, validity)

breaking it down:

If you have some knowledge of keras than the code is self-explanatory.

In general, we are defining a function which takes **image\_shape** as a parameter.

Inside that function, we are initializing a Sequential model from keras which helps us in creating linear stacks of layers.

model = Sequential()

After that, we are appending a bunch of layers in sequential model.

Our first layer is a convolutional layer of 32 shape having kernel\_size of 3 and our stride value is 2 with padding same. Since it is a first layer it holds input\_shape.

To understand what is going on here, you can refer to keras official documentation page.

But in simple language, here we are defining a convolutional layer which has a filter of size 3X3 and that filter strides over our image data. We have padding of same which means, no additional paddings are added. It remains the same as the original.

model.add(Conv2D(32, kernel\_size=3, strides=2,  
 input\_shape=image\_shape, padding=”same”))model.add(LeakyReLU(alpha=0.2))

After that, we are adding a LeakyRelu layer which is an activation function.

Similarly in other block of layers are added in a sequential model with some dropouts and batch normalization to prevent overfitting.

The last layer of our model is a Fully connected layer with an activation function sigmoid.

Since our discriminator’s job is to classify whether the given image is fake or not, it is a binary classification task and sigmoid is an activation that squeezes every value to values between 0 and 1.

model.add(Flatten())  
model.add(Dense(1, activation=’sigmoid’))

Now after initializing our discriminator model let’s create a generative model as well.

**def** **build\_generator**(noise\_size, channels):  
 model = Sequential()  
 model.add(Dense(4 \* 4 \* 256, activation=”relu”, input\_dim=noise\_size))  
 model.add(Reshape((4, 4, 256))) model.add(UpSampling2D())  
 model.add(Conv2D(256, kernel\_size=3, padding=”same”))  
 model.add(BatchNormalization(momentum=0.8))  
 model.add(Activation(“relu”)) model.add(UpSampling2D())  
 model.add(Conv2D(256, kernel\_size=3, padding=”same”))  
 model.add(BatchNormalization(momentum=0.8))  
 model.add(Activation(“relu”)) **for** i **in** range(GENERATE\_RES):  
 model.add(UpSampling2D())  
 model.add(Conv2D(256, kernel\_size=3, padding=”same”))  
 model.add(BatchNormalization(momentum=0.8))  
 model.add(Activation(“relu”)) model.summary()  
 model.add(Conv2D(channels, kernel\_size=3, padding=”same”))  
 model.add(Activation(“tanh”)) input = Input(shape=(noise\_size,))  
 generated\_image = model(input)  
   
 **return** Model(input, generated\_image)

Breaking it down:

Here we have defined a function that takes noise\_size and channels as parameters.

Inside the function, we have again initialized a sequential model.

Since our generator model has to generate images from noise vector, our first layer is a fully connected dense layer of size 4096 (4 \* 4 \* 256) which takes noise\_size as a parameter.

model.add(Dense(4 \* 4 \* 256, activation=”relu”, input\_dim=noise\_size))

*Note: We have defined its size to be of 4096 to for resizing it in 4X4X256 shaped layer.*

After that, we are using Reshape layer to reshape our fully connected layer in the shape of 4X4X256.

model.add(Reshape((4, 4, 256)))

Layer blocks after this are just a Convolutional layer with batch normalizations and activation function relu.

Just to see and understand what it looks like, let’s look at the model summary:

https://miro.medium.com/max/38/0*TGPVC1OWza25W4J5?q=20

From the shape of 4X4 it is extended up to the size of 128X128 which is our training\_data shape.

Our generator model takes noise as an input and outputs an image.

After initializing both the generator and discriminator model, let’s write a helper function to save the image after some iteration.

**def** save\_images(cnt, noise):  
 image\_array = np.full((  
 PREVIEW\_MARGIN + (PREVIEW\_ROWS \* (IMAGE\_SIZE + PREVIEW\_MARGIN)),  
 PREVIEW\_MARGIN + (PREVIEW\_COLS \* (IMAGE\_SIZE + PREVIEW\_MARGIN)), 3),  
 255, dtype=np.uint8)generated\_images = generator.predict(noise)generated\_images = 0.5 \* generated\_images + 0.5image\_count = 0  
 **for** row in range(PREVIEW\_ROWS):  
 **for** col in range(PREVIEW\_COLS):  
 r = row \* (IMAGE\_SIZE + PREVIEW\_MARGIN) + PREVIEW\_MARGIN  
 c = col \* (IMAGE\_SIZE + PREVIEW\_MARGIN) + PREVIEW\_MARGIN  
 image\_array[r:r + IMAGE\_SIZE, c:c +  
 IMAGE\_SIZE] = generated\_images[image\_count] \* 255  
 image\_count += 1output\_path = 'output'  
 if not os.path.exists(output\_path):  
 os.makedirs(output\_path)filename = os.path.join(output\_path, f"trained-{cnt}.png")  
 im = Image.fromarray(image\_array)  
 im.save(filename)

Our save\_images function takes to count and noise as an input.

Inside the function, it generates frames from the parameters we’ve defined above and stores our generated image array which are generated from the noise input.

After that, it saves it as an image.

Now, it’s time for us to compile the models and train them.

Let’s write a block of code for that as well:

image\_shape = (IMAGE\_SIZE, IMAGE\_SIZE, IMAGE\_CHANNELS)optimizer = Adam(1.5e-4, 0.5)discriminator = build\_discriminator(image\_shape)  
discriminator.compile(loss=”binary\_crossentropy”,  
optimizer=optimizer, metrics=[“accuracy”])  
generator = build\_generator(NOISE\_SIZE, IMAGE\_CHANNELS)random\_input = Input(shape=(NOISE\_SIZE,))generated\_image = generator(random\_input)discriminator.trainable = **False**validity = discriminator(generated\_image)combined = Model(random\_input, validity)  
combined.compile(loss=”binary\_crossentropy”,  
optimizer=optimizer, metrics=[“accuracy”])y\_real = np.ones((BATCH\_SIZE, 1))  
y\_fake = np.zeros((BATCH\_SIZE, 1))fixed\_noise = np.random.normal(0, 1, (PREVIEW\_ROWS \* PREVIEW\_COLS, NOISE\_SIZE))cnt = 1  
**for** epoch **in** range(EPOCHS):  
 idx = np.random.randint(0, training\_data.shape[0], BATCH\_SIZE)  
 x\_real = training\_data[idx]  
   
 noise= np.random.normal(0, 1, (BATCH\_SIZE, NOISE\_SIZE))  
 x\_fake = generator.predict(noise)  
   
 discriminator\_metric\_real = discriminator.train\_on\_batch(x\_real, y\_real)discriminator\_metric\_generated = discriminator.train\_on\_batch(  
 x\_fake, y\_fake)  
   
discriminator\_metric = 0.5 \* np.add(discriminator\_metric\_real, discriminator\_metric\_generated)generator\_metric = combined.train\_on\_batch(noise, y\_real)**if** epoch % SAVE\_FREQ == 0:  
 save\_images(cnt, fixed\_noise)  
 cnt += 1  
   
 print(f”{epoch} epoch, Discriminator accuracy: {100\* discriminator\_metric[1]}, Generator accuracy: {100 \* generator\_metric[1]}”)

Breaking it down:

Here in the first few lines, we have defined our input shape: which is 128X128X3 (image\_size, image\_size, image\_channel).

After that, we are using Adam as our optimizer.

*Note: All the parameters has been sourced from the paper [1].*

image\_shape = (IMAGE\_SIZE, IMAGE\_SIZE, IMAGE\_CHANNELS)optimizer = Adam(1.5e-4, 0.5)discriminator = build\_discriminator(image\_shape)  
discriminator.compile(loss=”binary\_crossentropy”,  
optimizer=optimizer, metrics=[“accuracy”])

After initializing the optimizer, we are calling our build\_discriminator function and passing the image shape then compiling it with a loss function and an optimizer.

Since it is a classification model, we are using accuracy as its performance metric.

Similarly, in the next line, we are calling our build\_generator function and passing our random\_input noise vector as its input.

generator = build\_generator(NOISE\_SIZE, IMAGE\_CHANNELS)random\_input = Input(shape=(NOISE\_SIZE,))generated\_image = generator(random\_input)

It returns a generated image as it’s output.

Now, one important part of GAN is we should prevent our discriminator from training.

discriminator.trainable = **False**validity = discriminator(generated\_image)combined = Model(random\_input, validity)  
combined.compile(loss=”binary\_crossentropy”,  
optimizer=optimizer, metrics=[“accuracy”])

Since we are only training generators here, we do not want to adjust the weights of the discriminator.

This is what really an “Adversarial” in Adversarial Network means.

If we do not set this, the generator will get its weight adjusted so it gets better at fooling the discriminator and it also adjusts the weights of the discriminator to make it better at being fooled.

We don’t want this. So, we have to train them separately and fight against each other.

We are then compiling the generative model with loss function and optimizer.

After that, we are defining two vectors as y\_real and y\_fake.

y\_real = np.ones((BATCH\_SIZE, 1))  
y\_fake = np.zeros((BATCH\_SIZE, 1))fixed\_noise = np.random.normal(0, 1, (PREVIEW\_ROWS \* PREVIEW\_COLS, NOISE\_SIZE))

These vectors are composed of random 0’s and 1’s values.

After that we are creating a fixed\_noise: this will result in generating images that are saved later on which we can see it getting better on every iteration.

After that, we are going to iterate over our training data with the range of epochs we’ve defined.

cnt = 1  
**for** epoch **in** range(EPOCHS):  
 idx = np.random.randint(0, training\_data.shape[0], BATCH\_SIZE)  
 x\_real = training\_data[idx]  
   
 noise= np.random.normal(0, 1, (BATCH\_SIZE, NOISE\_SIZE))  
 x\_fake = generator.predict(noise)  
   
 discriminator\_metric\_real = discriminator.train\_on\_batch(x\_real, y\_real) discriminator\_metric\_generated = discriminator.train\_on\_batch(  
 x\_fake, y\_fake)  
   
 discriminator\_metric = 0.5 \* np.add(discriminator\_metric\_real, discriminator\_metric\_generated)  
 generator\_metric = combined.train\_on\_batch(noisse, y\_real)  
 **if** epoch % SAVE\_FREQ == 0:  
 save\_images(cnt, fixed\_noise)  
 cnt += 1  
   
 print(f”{epoch} epoch, Discriminator accuracy: {100\* discriminator\_metric[1]}, Generator accuracy: {100 \* generator\_metric[1]}”)

During the iteration process, we are taking a sample from a real image and putting that on x\_real. After that, we are defining a noise vector and passing that to our generator model to generate a fake image in x\_fake.

Then we are training our discriminator model in both real and fake images separately.

discriminator\_metric\_real = discriminator.train\_on\_batch(x\_real, y\_real)discriminator\_metric\_generated = discriminator.train\_on\_batch(  
 x\_fake, y\_fake)

Some research has shown that training them separately can get us some better results.

After training, we are taking the metric from both models and taking the average.

discriminator\_metric = 0.5 \* np.add(discriminator\_metric\_real, discriminator\_metric\_generated)

This way we get the metric for the discriminator model, now for the generator model, we are training it on our noise vector and y\_real: which is a vector of 1’s.

Here we are trying to train the generator. Overtime generator will get better from these inputs and the discriminator will not be able to discriminate whether the input is fake or real.

One thing to note here, our combined model is based on the generator model linked directly to the discriminator model. Here our Input is what the generator wants as an input: which is noise and output is what the discriminator gives us.

Now in the end we have an if statement which checks for our checkpoint.

**if** epoch % SAVE\_FREQ == 0:  
 save\_images(cnt, fixed\_noise)  
 cnt += 1  
   
 print(f”{epoch} epoch, Discriminator accuracy: {100\* discriminator\_metric[1]}, Generator accuracy: {100 \* generator\_metric[1]}”)

If it reaches the checkpoint then it saves the current iteration noise and prints the current accuracy of generator and discriminator.

This is all for the coding part to create GAN, but we are not finished yet.

We have just written code for it, now we have to actually train those models and see the output how it performs.

**Training GAN**

Training GAN in a normal laptop is kind of impossible since it requires high computation power.

Normal laptops with normal CPUs cannot handle such huge tasks, so we are going to use [**Spell**](https://web.spell.run/refer/anyesh): Which is the fastest and most powerful end-to-end platform for Machine Learning and Deep Learning.

**Why Spell?**

https://miro.medium.com/max/60/0*78pyUstpRlfgMyoQ.png?q=20

The spell is a powerful platform for building and managing machine learning projects. The spell takes care of infrastructure, making machine learning projects easier to start, faster to get results, more organized, and safer than managing infrastructure on your own.

In every signup with Spell, you can get 10$ free credit!

In simple words, we are going to upload our data file in the spell platform and let it handle all our training task. Spell runs our task in their Powerful GPUs so that we don’t have to worry about anything. We can monitor our logs from their Web GUI and all our outputs are saved safely as well.

**Training Process**

There are few things to cover before running our project at [Spell](https://spell.run/).

First off, we must create our account on Spell. There is good and easy [documentation](https://spell.run/docs/) to get started on their official page.

After account creation, we can install Spell CLI using pip install:

pip install spell

This installs all the power of spell into our laptop. We can either use the Web GUI or we can easily log in to the spell server and execute commands from our cmd or bash.

To upload our project on Spell, we are going to use command-line tools.

Let’s open the command line terminal in the root directory of our project folder and login to the server by using spell login command:

spell login

After a successful login, now we can upload our training data file and run our code in the Spell server:

Spell upload “filename”

https://miro.medium.com/max/60/0*bZOaWNYVA8gKEHkU?q=20

After that our training\_data will be uploaded to the server.

*Note: Before running code in the server, code has been pushed to the github.*

Now we are ready to execute our code in the Spell server.

In the command line let’s run the following command:

Spell run python art\_gan.py -t V100 -m uploads/art\_gan/cubism\_data.npy

The command above runs our code in the Spell server with the Machine type V100 which is a GPU machine. The last argument there mounts the dataset directory so that it can be accessed by our code.

Now that code is executed successfully you can see the logs on your console. If you want to monitor in GUI then you can log in to the Web GUI of Spell and see under Runs Section.

https://miro.medium.com/max/60/0*KcHHE6mAGHLxHmnr?q=20

As you can see, it holds all the information about our recent run.

https://miro.medium.com/max/60/0*EHHElwcQWATC6Yss?q=20

As we have written code for; in every 100th iteration our generated image is saved in the output directory and log with accuracy metric are printed.

https://miro.medium.com/max/60/0*qvvsvEZCJePKR-K7?q=20

You can view them in logs section.

https://miro.medium.com/max/60/0*fcBliro5ZjrSbrsw?q=20

Awesome isn’t it? You can do your other works while it trains and saves the output for you.

**Output**

Now after the training completes Spell automatically saves our output in the resources/runs directory.

https://miro.medium.com/max/60/0*H6Sh3jIK1o8RvQvj?q=20

After that, we can download the outputs from the runs of Spell to our local machine by using the following command:

spell cp [OPTIONS] SOURCE\_PATH [LOCAL\_DIR]

For this project it will be:

spell cp runs/44

You just have to enter the runs/<number of your run> to download the content of that runs.

That’s it!! Now you have outputs of the GAN trained on Spell’s GPU machine in your local machine. You can now visualize how your GAN performed from that output images.

**Conclusion**

GANs are an exciting and rapidly changing field, delivering on the promise of generative models in their ability to generate realistic examples across a range of problem domains. It is not an easy task to understand GAN or any Machine Learning and Deep Learning field overnight. It needs patience and a lot of practice plus understanding.

In previous days it was not possible for Aspiring ML enthusiast likes us to perform repetitive practice to see what went how. But now, a platform like **Spell** helps us to provide a system to run and manage our projects so that we can run and test our models.

What we have created is just a simple representation of how GAN can be created and what GAN can do. There are still more advanced tweaks yet to perform.

To take it further you can tweak the parameters and see how differently it generates the images.

There are still a lot of things one can research.

For any queries and discussion, you can join the Spell community from here: <https://chat.spell.ml/>

**References**

[1] **Generative Adversarial Network**, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, 2014

[2] **A Gentle Introduction to Generative Adversarial Networks (GANs),** Jason Brownlee, 2019 [Online] <https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/>

[3] **A Beginner’s Guide to Generative Adversarial Networks (GANs),** Chris, 2019 [Online] <https://skymind.ai/wiki/generative-adversarial-network-gan>

**Sign up for The Variable**

**By Towards Data Science**

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](https://medium.com/towards-data-science/newsletters/the-variable?source=newsletter_v3_promo--------------------------newsletter_v3_promo-----------)

* [Machine Learning](https://towardsdatascience.com/tagged/machine-learning)
* [Deep Learning](https://towardsdatascience.com/tagged/deep-learning)
* [Programming](https://towardsdatascience.com/tagged/programming)
* [Gpu Computing](https://towardsdatascience.com/tagged/gpu-computing)
* [Generative Adversarial](https://towardsdatascience.com/tagged/generative-adversarial)

[**More from Towards Data Science**](https://towardsdatascience.com/?source=follow_footer-----39f67f83c7b4--------------------------------)

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fcollection%2Ftowards-data-science%2F39f67f83c7b4&operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fgenerating-modern-arts-using-generative-adversarial-network-gan-on-spell-39f67f83c7b4&collection=Towards%20Data%20Science&collectionId=7f60cf5620c9&source=follow_footer-----39f67f83c7b4---------------------follow_footer-----------)

Your home for data science. A Medium publication sharing concepts, ideas and codes.

[Read more from Towards Data Science](https://towardsdatascience.com/?source=follow_footer-----39f67f83c7b4--------------------------------)

**More From Medium**

[**Machine Learning: What it is and Why it Matters**](https://medium.com/@vertikaguptaeduonix/machine-learning-what-it-is-and-why-it-matters-39f6bc613e84?source=post_internal_links---------0----------------------------)

[Vertika Gupta](https://medium.com/@vertikaguptaeduonix?source=post_internal_links---------0----------------------------)

[[https://miro.medium.com/max/60/1*aZ6VB_mqZfp4xpPNLnyhkg.png?q=20](https://medium.com/@vertikaguptaeduonix/machine-learning-what-it-is-and-why-it-matters-39f6bc613e84?source=post_internal_links---------0----------------------------)](https://medium.com/@vertikaguptaeduonix/machine-learning-what-it-is-and-why-it-matters-39f6bc613e84?source=post_internal_links---------0----------------------------)

**[How SMOTE Oversampling Improved the Accuracy on Imbalanced Classification with Python (Part I)](https://medium.com/@xue.susan.chen/how-smote-oversampling-improved-the-accuracy-on-imbalanced-classification-with-python-part-i-39fdad247d3b?source=post_internal_links---------1----------------------------)**

[Xue Susan Chen](https://medium.com/@xue.susan.chen?source=post_internal_links---------1----------------------------)

[[https://miro.medium.com/max/60/1*hn4v1tCaJy7cWMyb0bpNpQ.png?q=20](https://medium.com/@xue.susan.chen/how-smote-oversampling-improved-the-accuracy-on-imbalanced-classification-with-python-part-i-39fdad247d3b?source=post_internal_links---------1----------------------------)](https://medium.com/@xue.susan.chen/how-smote-oversampling-improved-the-accuracy-on-imbalanced-classification-with-python-part-i-39fdad247d3b?source=post_internal_links---------1----------------------------)

[**How to choose a machine learning consulting firm**](https://medium.com/swlh/how-to-choose-a-machine-learning-consulting-firm-39fec446f5fc?source=post_internal_links---------2----------------------------)

[Markus Schmitt](https://medium.com/@DataRevenue?source=post_internal_links---------2----------------------------) in [The Startup](https://medium.com/swlh?source=post_internal_links---------2----------------------------)

[[https://miro.medium.com/max/60/1*3_EIaHV8m_oIsrIku7fdkw.png?q=20](https://medium.com/swlh/how-to-choose-a-machine-learning-consulting-firm-39fec446f5fc?source=post_internal_links---------2----------------------------)](https://medium.com/swlh/how-to-choose-a-machine-learning-consulting-firm-39fec446f5fc?source=post_internal_links---------2----------------------------)

[**Optimized a promotion campaign by using Uplift Models and Machine Learning**](https://medium.com/@lumeilin301/optimized-a-promotional-campaign-by-using-uplift-models-and-machine-learning-3a01cf80eec1?source=post_internal_links---------3----------------------------)

[Lumei Digital](https://medium.com/@lumeilin301?source=post_internal_links---------3----------------------------)

[[https://miro.medium.com/max/60/1*7LaeZihVRsLoa1uiUTGv5w.png?q=20](https://medium.com/@lumeilin301/optimized-a-promotional-campaign-by-using-uplift-models-and-machine-learning-3a01cf80eec1?source=post_internal_links---------3----------------------------)](https://medium.com/@lumeilin301/optimized-a-promotional-campaign-by-using-uplift-models-and-machine-learning-3a01cf80eec1?source=post_internal_links---------3----------------------------)

[**Build your own conversational chatbot in minutes - Part one: RASA-NLU**](https://medium.com/analytics-vidhya/build-your-own-conversational-chatbot-in-minutes-part-one-rasa-nlu-13df39601032?source=post_internal_links---------4----------------------------)

[Vaishnavi K B](https://medium.com/@vaishnavikodai?source=post_internal_links---------4----------------------------) in [Analytics Vidhya](https://medium.com/analytics-vidhya?source=post_internal_links---------4----------------------------)

[[https://miro.medium.com/max/60/1*xf_WKdnysNSxa2UIlPBcpg.jpeg?q=20](https://medium.com/analytics-vidhya/build-your-own-conversational-chatbot-in-minutes-part-one-rasa-nlu-13df39601032?source=post_internal_links---------4----------------------------)](https://medium.com/analytics-vidhya/build-your-own-conversational-chatbot-in-minutes-part-one-rasa-nlu-13df39601032?source=post_internal_links---------4----------------------------)

[**Free Coursera Guided Projects Courses on Machine Learning**](https://exploreroftruth.medium.com/free-coursera-guided-projects-courses-on-machine-learning-13de36b50434?source=post_internal_links---------5----------------------------)

[Explorer of truth](https://exploreroftruth.medium.com/?source=post_internal_links---------5----------------------------)

[[https://miro.medium.com/max/60/1*iBQZeP9OoyZWhmZBE5iDIg.png?q=20](https://exploreroftruth.medium.com/free-coursera-guided-projects-courses-on-machine-learning-13de36b50434?source=post_internal_links---------5----------------------------)](https://exploreroftruth.medium.com/free-coursera-guided-projects-courses-on-machine-learning-13de36b50434?source=post_internal_links---------5----------------------------)

[**Producing sounds for pictures with Convolutional Neural Network**](https://medium.com/@michael.nitsenko/producing-sounds-for-pictures-with-convolutional-neural-network-13da92198a1a?source=post_internal_links---------6----------------------------)

[Michael Nitsenko](https://medium.com/@michael.nitsenko?source=post_internal_links---------6----------------------------)

[[https://miro.medium.com/max/60/1*aQPfrIr8we4_tvfObBaIbw.png?q=20](https://medium.com/@michael.nitsenko/producing-sounds-for-pictures-with-convolutional-neural-network-13da92198a1a?source=post_internal_links---------6----------------------------)](https://medium.com/@michael.nitsenko/producing-sounds-for-pictures-with-convolutional-neural-network-13da92198a1a?source=post_internal_links---------6----------------------------)

[**Convolution, Padding, Stride, and Pooling in CNN**](https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26?source=post_internal_links---------7----------------------------)

[Abhishek Kumar Pandey](https://medium.com/@abhisekumarpnd?source=post_internal_links---------7----------------------------) in [Analytics Vidhya](https://medium.com/analytics-vidhya?source=post_internal_links---------7----------------------------)

[[https://miro.medium.com/freeze/max/60/1*ZCjPUFrB6eHPRi4eyP6aaA.gif?q=20](https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26?source=post_internal_links---------7----------------------------)](https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26?source=post_internal_links---------7----------------------------)

[About](https://medium.com/about?autoplay=1&source=post_page-----39f67f83c7b4--------------------------------)

[Write](https://medium.com/new-story?source=post_page-----39f67f83c7b4--------------------------------)

[Help](https://help.medium.com/hc/en-us?source=post_page-----39f67f83c7b4--------------------------------)

[Legal](https://policy.medium.com/medium-terms-of-service-9db0094a1e0f?source=post_page-----39f67f83c7b4--------------------------------)

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](https://policy.medium.com/medium-privacy-policy-f03bf92035c9), including cookie policy.