Cryptopunk and Meebit alike NFT Art generator

written by:Udit Sangule (uditsangule@gmail.com)

Introduction:

CryptoPunks are 24x24 pixel art images, generated algorithmically,whereas the Meebits are 3D voxels characters. Each and every punk has their own profile page that shows their attributes as well as their ownership/for-sale status. Once the characeter are generated ,each character then are registered on Etherium blockchain.

We can generate these unique characters alogrithmically by applying two approaches:

1. Computer Vision Naive Method.
2. Machine Learning Generative Neural Networks.

Before Discussing these methods , some important concepts should to be understood before applying these approach.

Concept #1 Deciding Attributes and Archetypes:

The basic building block of an NFT art are the face/(head in Meebits) or say basic Archetype being generated.To make these archetype unique or different we add attributes or features to these faces and we get enumerous combinations out of it. We will name it *selector_database* as it will contain all the possible list we are adding into images.

For eg – Punks contain:

- Types : ['Male', 'Female', 'Zombie', 'Ape', 'Alien']
- Attributes : ['Hat', 'Hair', 'Eyes', 'Blemishes', 'Nose', 'Ears', 'Mouth', 'Beard', 'Neck']

Concept #2 Deciding Key bits and calculations:

So the Attributes and types are decided, follow these steps:

1. Add Contents of each attributes and types in sorted order. (I have written a python script to create this csv data from README given by cryptopunks in Appendix).

*Table 1: selector_database_snapshort (*contains few base attributes)*

type	Hat	Hair	Eyes	Blemishes	Nose	Ears
basic/male1.png	basic/m/bandana.png	basic/f/blondebob.png	basic/m/3dglasses.png	basic/m/mole.png	basic/m/clownnose.png	basic/m/earring.png
basic/male2.png	basic/f/bandana.png	basic/f/blondeshort.png	basic/f/3dglasses.png	basic/f/mole.png	basic/f/clownnose.png	basic/f/earring.png
basic/male3.png	basic/m/beanie.png	basic/m/clownhairgreen.png	basic/m/bigshades.png	basic/m/rosycheeks.png		
basic/male4.png	basic/m/cap.png	basic/f/clownhairgreen.png	basic/f/bigshades.png	basic/f/rosycheeks.png		
basic/female1.png	basic/f/cap.png	basic/m/crazyhair.png	basic/m/classicshades.png	basic/m/spots.png		
basic/female2.png	basic/m/capforward.png	basic/f/crazyhair.png	basic/f/classicshades.png	basic/f/spots.png		
basic/female3.png	basic/m/cowboyhat.png	basic/f/darkhair.png	basic/m/eyemask.png			
basic/female4.png	basic/m/dorag.png	basic/m/frumpyhair.png	basic/f/eyemask.png			
basic/zombie.png	basic/m/fedora.png	basic/f/frumpyhair.png	basic/m/eyepatch.png			
basic/ape.png	basic/m/headband.png	basic/f/halfshaved.png	basic/f/eyepatch.png			
basic/alien.png	basic/f/headband.png	basic/m/messyhair.png	basic/m/hornedrimglasses.png			

2. Take a count on all availabe attributes and type then assign length of bits required to assign each attribute feature selected.

   Note: you need to take max bits and also some room for future content introduction, the extra_selector_bits (lets take +2 extra for each attribute).

   $$selector\_bits = math.ceil(\log_2(total\_count\_per\_attribute)) + (extra\_selector\_bits)$$

   $$max\_bits \mathrel{+}= selector\_bits$$

   For punks:

   Type contains unique files :11 with min bits_req=4
   Hat contains unique files :19 with min bits_req=5
   Hair contains unique files :35 with min bits_req=6
   Eyes contains unique files :27 with min bits_req=5
   Blemishes contains unique files :6 with min bits_req=3
   Nose contains unique files :2 with min bits_req=1
   Ears contains unique files :2 with min bits_req=1
   Mouth contains unique files :14 with min bits_req=4
   Beard contains unique files :12 with min bits_req=4
   Neck contains unique files :5 with min bits_req=3

   So,

   36 bits needed for image key generation.

You can assign more bits to add more attribute list in future. So lets take key bits to be:

Generator_key = 64 bits (8 extra + (36 required + (2 *10 extra_selector_bits))

3. Assign the bit locations for each attributes in key with reference to future bits. Key_bits (in order) =

Extra	Type	Hat	Hair	Eyes	Belmish	Nose	Ears	Mouth	Beard	Neck
8	6	7	8	7	5	3	3	6	6	5

The generator_key has been designed to generate each image a unique pixel combination. While generating an image we assign a key to it. So that no new image will get same generator key.Some combinations will be discarded as it doesnt contains value.

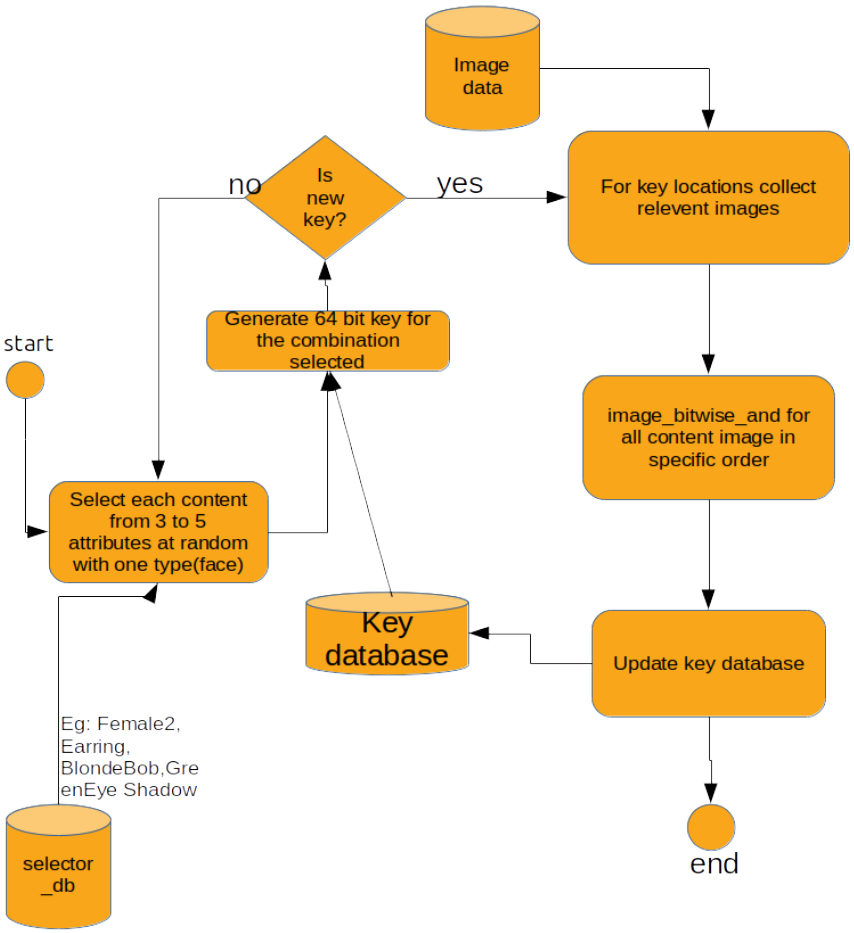
With the current available data we can generate almost **~ 68 Billion** (2^{36}) images. And if we have sufficient attributes to fill all bits, we can generate **2 x 10^{19}** unique images.

Now, Lets take a look at the methods to generate !

Method #1 Compute Vision:

This method require images data of all the types and attributes in same image-resolution (28x28) pixels like cyberpunks made available on github(link).

Flow chart:



Algoritm:

1. Select one type from selector database and one from random attributes (such as: Hat , Hair , Eyes , Nose) upto 7 attributes. (Note: sequential selection is also accepted , instead of random)
2. Generate corresponding key for the selected combination with Generate_key Algorithm.
    1. Check if key is already taken ? i.e if such combination is already generated.
        1. If yes , then search again for the combination until a new unique key combination is generated.
        2. If No, we are good to proceed for the generation.
3. For each attribute selected , take the corresponding 24x24 image out of image dataset and perform:
    1. Add the image layers into a black image (np.zeros(24,24)) in to order:
        1. type > attr1 > attr2 > attr3 > and so on....
4. update the key_database with registered key , so that current image couldnt be generated again in future.
5. Save the image (punk1.png) and attributes (punk_attr.csv) into system.
6. Go to step no 1 for next generation.

Conclusion :

We can see this is a generic approach which is doesnt need any intelligence in generation. The main component here is the key for each image , which we have registered into database.

This algorithm will generate all the possible images, However this algorithm will be slowed down later beacause most of the keys will be registerd into database and it will find difficult to get any new key in future. This mimic the unavailabity of NFT image in Future and thus, the Crypto-currency.

Method #2 Machine Learning :

This approch again takes dataset given by cryptopunks with *punks.csv* and *punks.png*. You can also creating your own dataset via above CV method. We are taking Generative Adversarial Networks for generations of images out of given attributes and type fed as a list.

punks.png

punks.csv

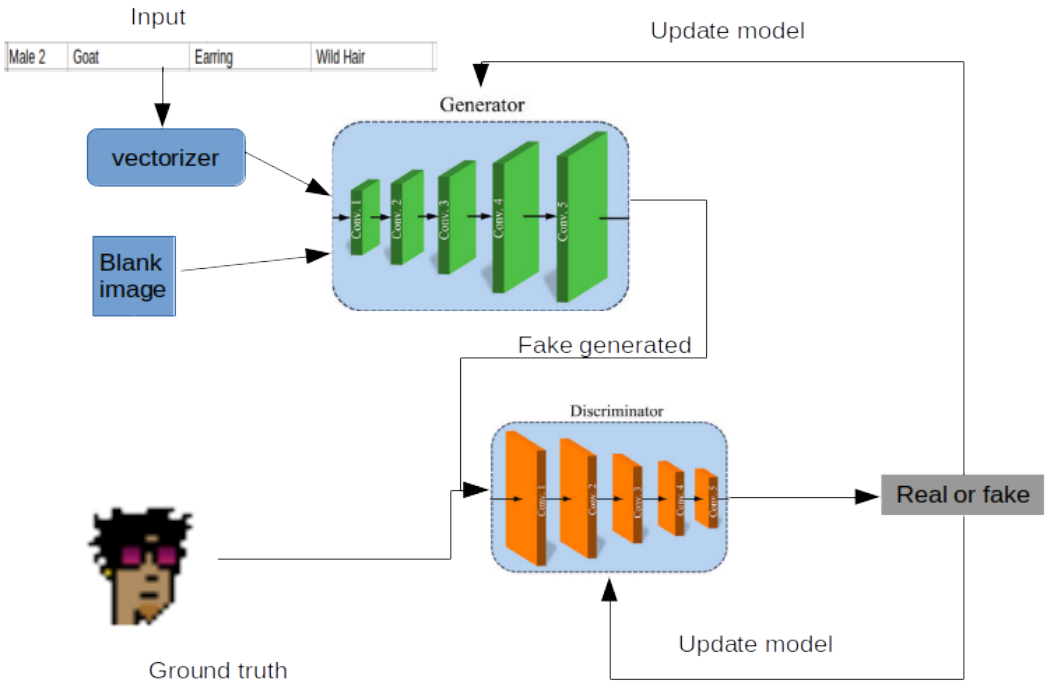
	A	B	C	D	E
1	type	attribute1	attribute2	attribute3	attribute4
2	Female 2	Earring	Blonde Bob	Green Eye Shadow	
3	Male 1	Smile	Mohawk		
4	Female 3	Wild Hair			
5	Male 1	Wild Hair	Pipe	Nerd Glasses	
6	Male 2	Goat	Earring	Wild Hair	Big Shades
7	Female 2	Earring	Half Shaved	Purple Eye Shadow	
8	Male 2	Do-rag			
9	Female 2	Spots	Wild White Hair	Clown Eyes Blue	
10	Male 1	Luxurious Beard	Messy Hair		
11	Male 2	Big Beard	Police Cap	Clown Nose	
12	Female 1	Mohawk	Blue Eye Shadow		
13	Female 2	Black Lipstick	Straight Hair Dark	Clown Eyes Green	
14	Female 1	Purple Lipstick	Blonde Short		
15	Female 3	Black Lipstick	Straight Hair Blonde	Big Shades	
16	Female 1	Hot Lipstick	Pilot Helmet	Pipe	
17	Male 4	Luxurious Beard	Wild Hair	Regular Shades	
18	Male 2	Earring	Stringy Hair	Small Shades	
19	Male 3	Frown	Mohawk		
20	Male 2	Muttonchops	Eye Mask		
21	Female 1	Hot Lipstick	Bandana	Horned Rim Glasses	

Consider a typical Text-to-Image GAN network in which a Generator takes a vector input and generates a image , and a Discriminator which compares a Fake and Orignal image and then it corrects the generator which will yeild a better image in next iteration. After Few iteration , the generator will produce almost real image and a discriminator will not be able to tell the diffence between them , which is our required final state.

Training network

1. Training neural netwok requires a preprocessing of input attributes list to a fixed size vector. We can analyze the input csv and convert it into fixed size vector (size=32) by applying OneHot Encoding or any encoding techniques.
2. Consider the inputs:
    1. Generator Input : attributes vector and blank image (24x24) are fed into a generator .
    2. Discriminator Input : ground truth image cutout from grid (punks.png) and generated image from generator .
3. For each row in csv , iterate it for atleast 100 epochs to get the desired data.
4. When the final state is achieved the generator is used to generate images.

Architecture:



Conclusion:

We can see Computer Vision Approach which is doesnt need any intelligence in generation. The main component here is the key for each image , which we have registered into database.

Machine Learning Algorithm depends purely on dataset we are providing for Training , so it is highly recomended to provide a good dataset which contains atleast 3 items from all the all attribute list. Also it may generate two look alike image but with a minor pixel change within. Hence the key model is not applicabe here directly. But by using key generation via pixel value in image may work here . For eg : 24x24x3 image can be assign as a key bits so that we dont generate a similiar pixel image in future.

Both the Algoritm will work for Generation of Meebit or Punks alike NFT art . However , for meebits we can first convert the 3D voxel to 2D image and then apply these algorithms. Or directly using 3D neural Networks can also work by applying these algoritms.

These algorithm will generate all the possible images, However this algorithm will be slowed down later beacause most of the keys will be registerd into database and it will find difficult to get any new key in future. This mimic the unavailabity of NFT image in Future and thus, the Crypto-currency.

References:

- Official Git Repository : https://github.com/cryptopunksnotdead
- CyberPunks official site : https://www.larvalabs.com/cryptopunks
- meebits official site : https://meebits.larvalabs.com/
- Text to Image Gans research paper : http://proceedings.mlr.press/v48/reed16.pdf