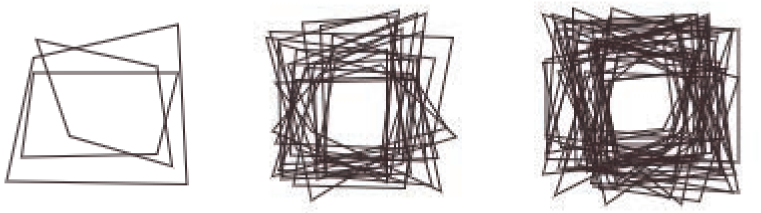
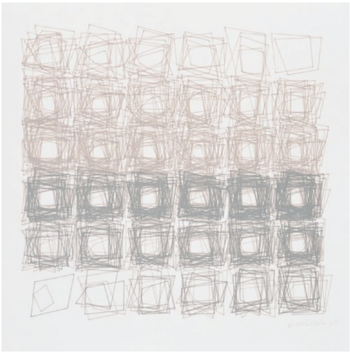


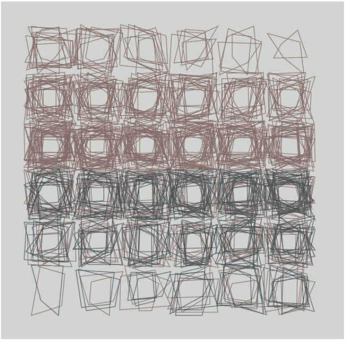
Discovering GANs

algorithmic aesthetics

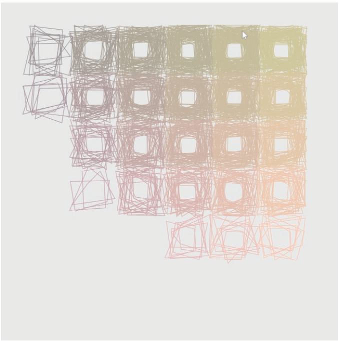
Tristan Debeaune, Wendy Gervais



Original Artwork :
Véra Molnar, *Structure de quadrilatère*, 1985



Static recoding



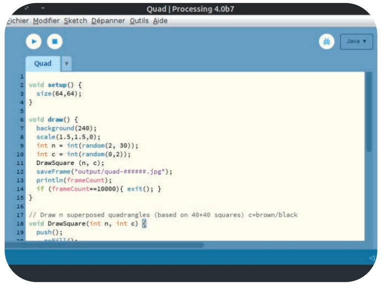
Dynamic recoding

git :
https://gitlab.com/debeaunetristan/projet-recoding
https://gitlab.com/debeaunetristan/gan_project

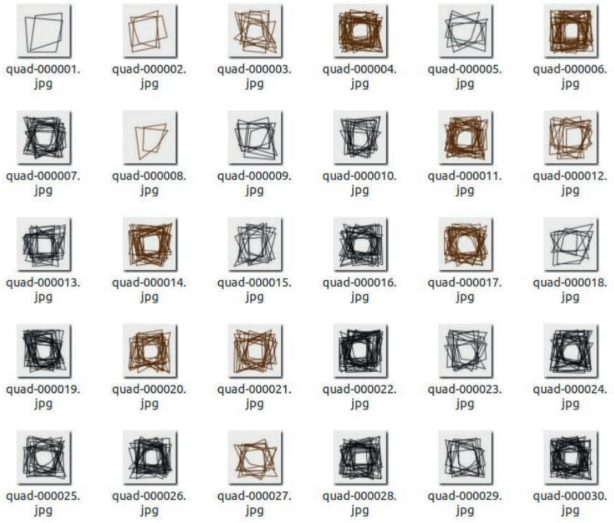
1. Generating the dataset

Pseudo-algorithm

```
void setup()
  create a 64*64 canvas


void draw()

  int n <- random int between 2 and 30
  int c <- random int between 0 and 1

  DrawSquare (n, c)

  save 10 000 times, as «output/quad-######.jpg»


void DrawSquare(int n, int c)
  exactly the same as in recoding project, it draws n randomly
scribbled quads over the same position

but in addition, checks c
  if c=0, draw in brown
  if c=1, draw in black
```



```

void setup() {
  size(64,64);
}

void draw() {
  background(240);
  scale(1.5,1.5,0);
  int n = int(random(2, 30));
  int c = int(random(0,2));
  DrawSquare (n, c);
  saveFrame("output/quad-######.jpg");
  println(frameCount);
  if (frameCount==10000){ exit(); }
}

// Draw n superposed quadrangles (based on 40*40 squares) c=brown/black
void DrawSquare(int n, int c) {
  push();
    noFill();
```

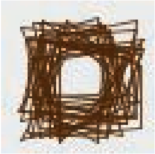


Generated 10 000 with Processing

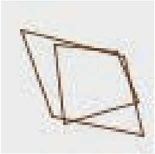
Possibilities and examples

$$N = 2\left(\prod_{i=2}^{30} 27^{(2*4)}\right) \simeq 10^{320} \textit{ possibilities}$$

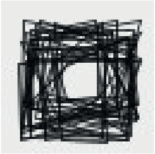
Black or brown * we superpose from 2 to 30 quads * 27 possible values for each x and y of 4 vertices of each quad (they vary by (-12,-12) to (+15,+15) from the original positions of regular square)



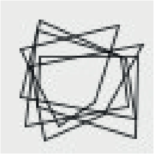
c = 1
n -> 30



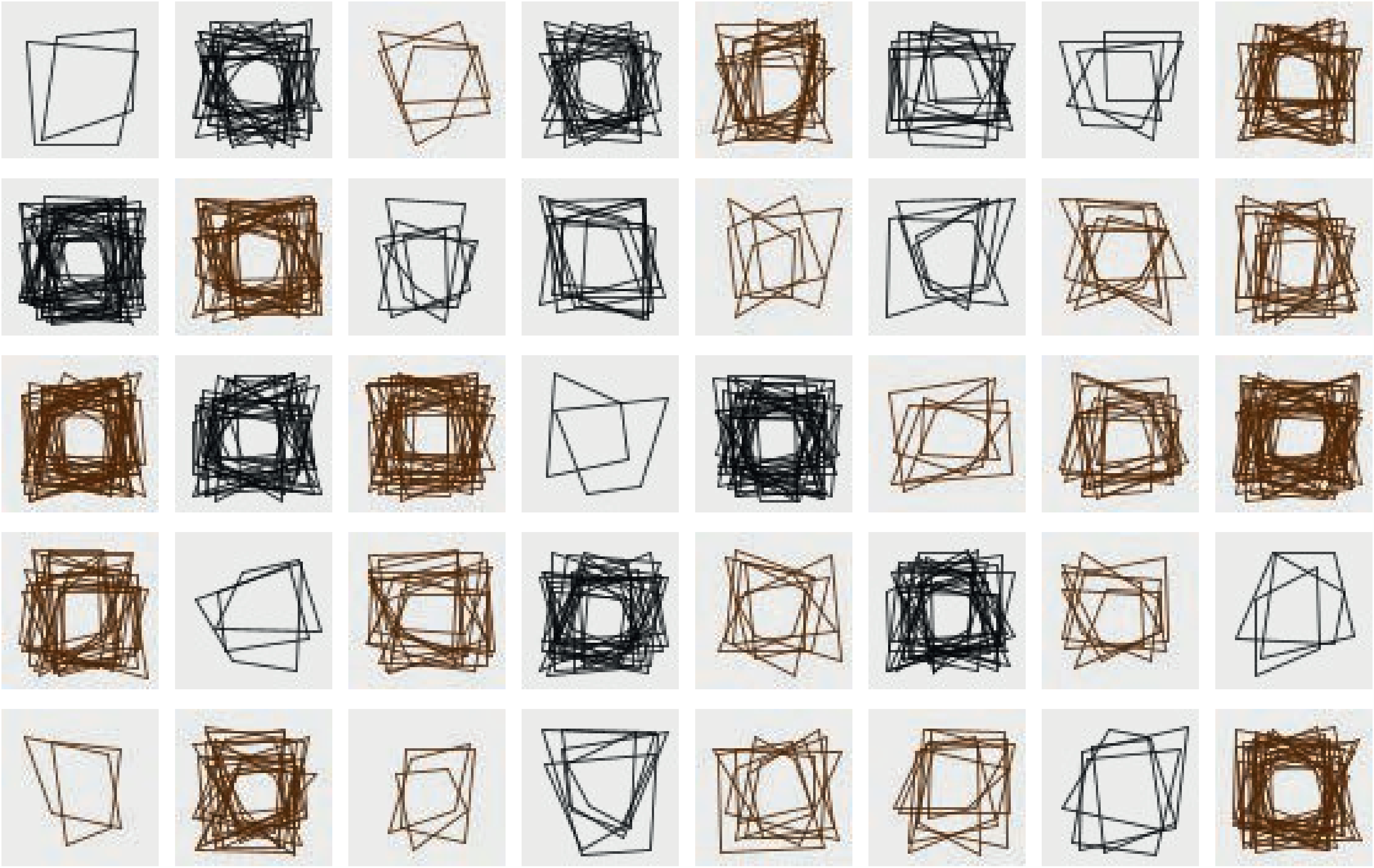
c = 1
n -> 2



c = 0
n -> 30

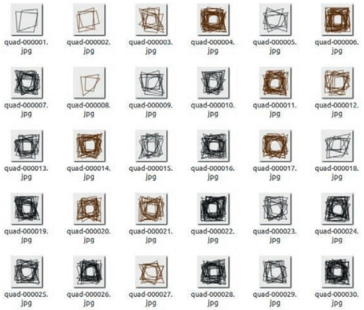


c = 0
n -> 2



see annex 1

2. Training the GAN



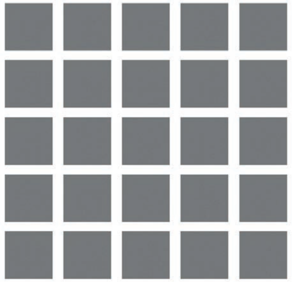
Dataset

output.zip

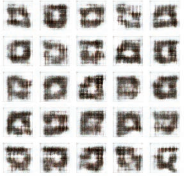
'https://github.com/leogenot/GenerativeDeepDrawing/' + tensorflow

```
EPOCHS = 6500
PRINT_EVERY_N_BATCHES = 100
N_CRITIC = 5
BATCH_SIZE = 64
```

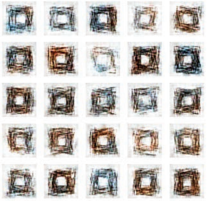
gan.train



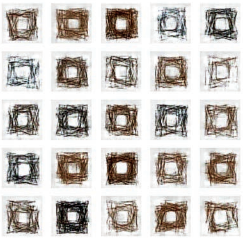
0



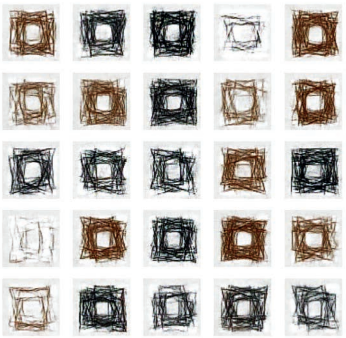
100



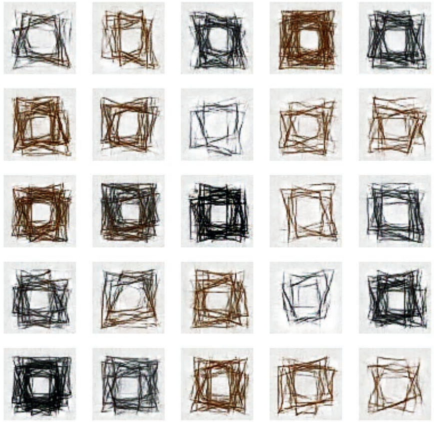
300



1000

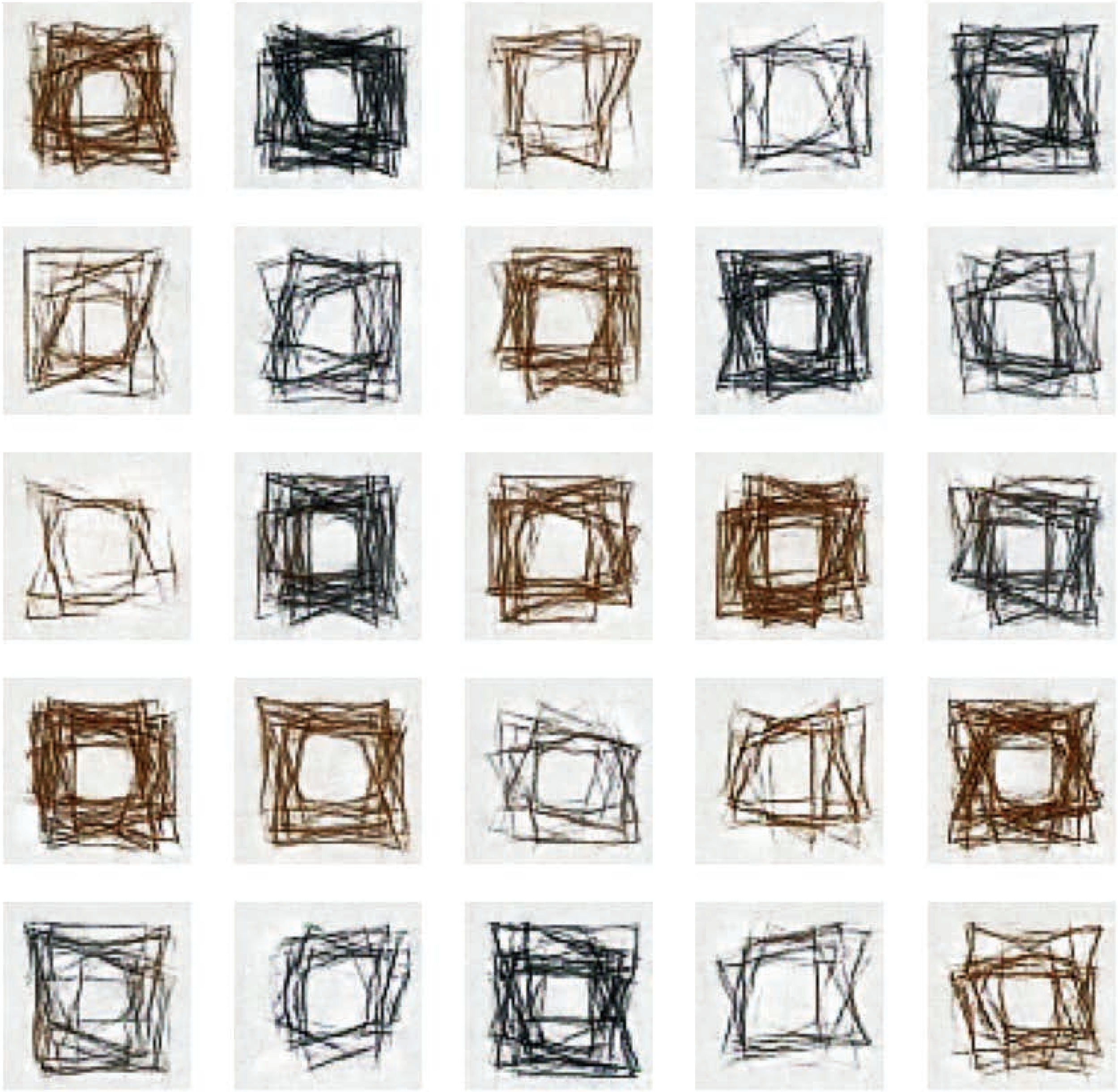


2500



5000

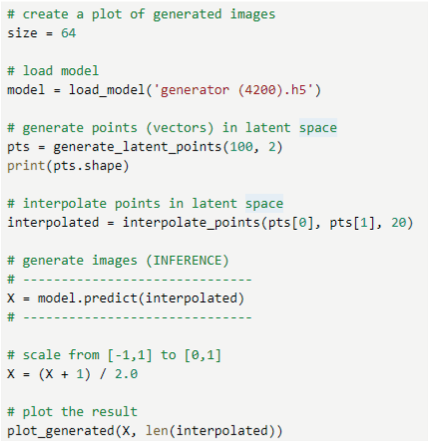
see annex 2



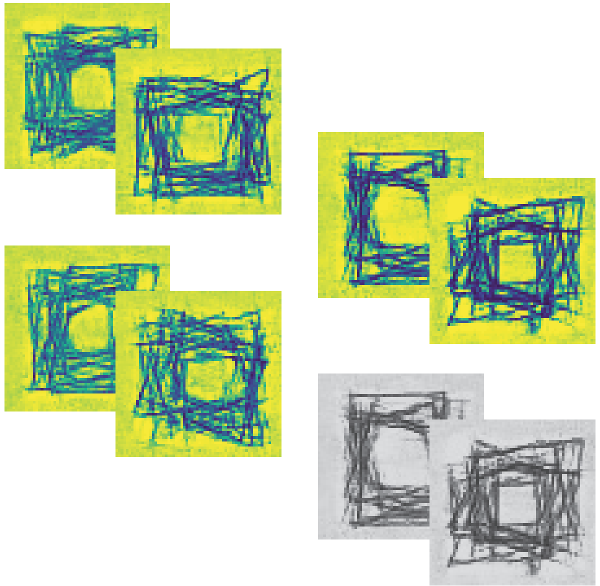
6500 + generator.h5

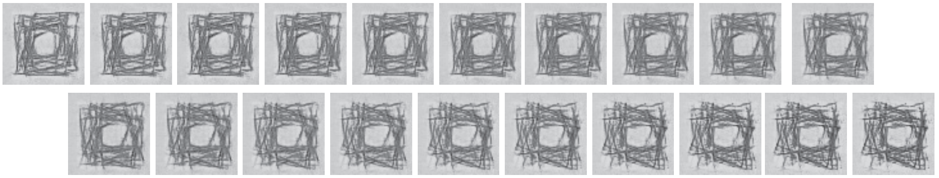
3. Exploring the latent space

generator.h5

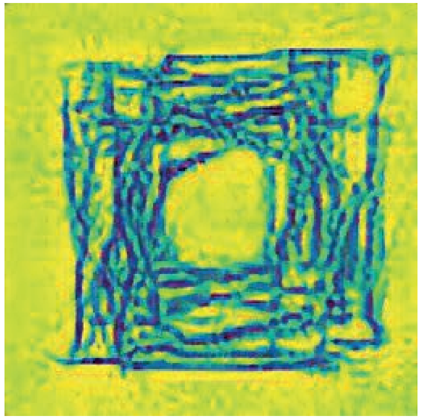


```
# create a plot of generated images
size = 64

# load model
model = load_model('generator (4200).h5')

# generate points (vectors) in latent space
pts = generate_latent_points(100, 2)
print(pts.shape)

# interpolate points in latent space
interpolated = interpolate_points(pts[0], pts[1], 20)

# generate images (INFERENCE)
# ------------------------------
X = model.predict(interpolated)
# ------------------------------

# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0

# plot the result
plot_generated(X, len(interpolated))
```

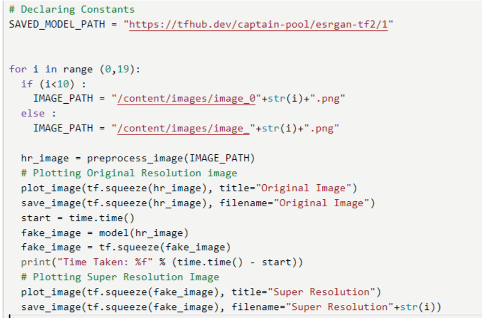




see annex 3



Super Resolution
64x64 -> 256x256



```
# Declaring Constants
SAVED_MODEL_PATH = "https://tfhub.dev/captain-pool/esrgan-tf2/1"


for i in range (0,19):
  if (i<10) :
    IMAGE_PATH = "/content/images/image_0"+str(i)+".png"
  else :
    IMAGE_PATH = "/content/images/image_"+str(i)+".png"

  hr_image = preprocess_image(IMAGE_PATH)
  # Plotting Original Resolution image
  plot_image(tf.squeeze(hr_image), title="Original Image")
  save_image(tf.squeeze(hr_image), filename="Original Image")
  start = time.time()
  fake_image = model(hr_image)
  fake_image = tf.squeeze(fake_image)
  print("Time Taken: %f" % (time.time() - start))
  # Plotting Super Resolution Image
  plot_image(tf.squeeze(fake_image), title="Super Resolution")
  save_image(tf.squeeze(fake_image), filename="Super Resolution"+str(i))
```

see annex 4