



富嶽三十六景 勸風  
快晴

葛原山



【月刊 ZENKEI AI MAGAZINE 2021年3月号】

## 目次

まえがき	2
第1章 当日のイベントの模様	3
第2章 はじめに	4
第3章 コンピュータ会話教室（市來健吾）	5
第4章 東海道5X（ホンダナオ）	10
第5章 東海道五十Xプロジェクト（大島圭佑）	20
執筆者紹介	30
編集後記	31

【月刊 ZENKEI AI MAGAZINE 2021年3月号】

富嶽三景  
甲子年  
春  
澤

まえがき  
河原一色

早いもので、2021年1月からスタートした ZENKEI AI MAGAZINE (ZAM) もこの2021年3月号で3冊目になります。

そろそろ軌道に乗ってきたか？と聞かれると、はい、とは言い切れませんね。今回は ZENKEI AI FORUM の重要なコンセプトの1つである diversity を象徴する、ゲスト・スピーカーによる寄稿が2編あります。

お楽しみください。

2021年5月25日

金沢にて

ZENKEI AI MAGAZINE 編集長

市來健吾

ZAM



## 第1章

# 当日のイベントの模様

3月のZAFはゲストをお二方、通称「チーム・ホンダ」からリーダーのホンダナオさんと、チームの右腕の大島圭佑をお迎えして、昨年2020年8月のZAFで発表してもらった『東海道5X』のその後の発展について発表していただきました！

ZAM本号の内容はこのZAFの内容をベースにして、以下のような構成になっています。

- **【第2章】はじめに（市來健吾）**

当日しゃべった前座の話題「ZAM創刊号、印刷版きました！」を紹介します。

- **【第3章】コンピュータ会話教室（市來健吾）**

突然思いついて新シリーズをスタートさせました。（つづくのかな？）

- **【第4章】東海道5X（ホンダナオ）**

ホンダさんの発表。

- **【第5章】東海道五十Xプロジェクト（大島圭佑）**

大島さんの発表。



## 第2章

### はじめに

当日は、メインの「チーム・ホンダ」による発表を控えて、場をあたためるという本来の意味での「前座」として『ZAM 創刊号の印刷版がきました！』というはなしをしました。

### ZAM 創刊号、印刷版きました！

前回、無事に創刊された『ZAM 1月号』<https://zenkei-ai-forum.github.io/ZAM202101/>ですが、その後自分へのご褒美として10部ほど印刷してみました。印刷を頼んだのは『ちょ古っ都製本工房』<https://www.chokotto.jp/>で、内容は以下の通り：

- B5 30 ページ
- 本文
  - モノクロ印刷（スタンダード）
  - 上質70kg
- 表紙
  - 色上質最厚口・135kg（高品質フルカラー印刷）
  - オプション加工表2・3 モノクロ印刷
- 納期 10営業日コース
- 印刷部数 10冊

税込で2,230円でした。

この印刷版 ZAM 創刊号が先日（3月28日）届きました！



10冊（余丁1部）しか刷りませんでした。ZAF 1月のイベントの関係者、つまり講演者であり執筆者と、ZOOM および YouTube での参加者のみなさまにお礼として配ったら、残り2冊しか余りませんでした。欲しい人、いますか？ 1冊1,500円です！（これで元が取れる！！）……というのは冗談です。今後 ZAF の活動に、例えば『数理クイズ』の賞品にしようかな、と思っています。

2021年3月31日

# Zoomライブ

ZENKEI AI フォーラム

## 第3章

### コンピュータ会話教室（市來健吾）

#### 第1回：for ループ

コンピュータが人間の能力を（画像認識など一部のタスクで）凌駕しつつある2021年の今こそ、みなさんコンピュータと会話してみませんか？コンピュータ会話教室の開幕です！

#### はじめに

「語学学習」は生涯学習の文脈で多くの人々の関心にあります。街の本屋さんには語学コーナーが大きな場所を取ってますし、アマゾンで検索すると膨大な数の本が出てきます。

普通「語学」というと「英会話」ですね。英語ができればとりあえず世界を歩くことはできそうです。将来を見て、また、日本という立地を考えて、やはり今は「中国語」という方もいらっしゃるでしょう。

ここに別の視点、ZENKEI AI FORUM的視点として、近い将来、人間よりも高い能力を獲得するであろう、地球上の種族「コンピュータ」と会話できるって、素敵じゃないですか？

（と、ちょっとキャッチャーなコピーを書いてみました。ちなみに『数理クイズ』を企画したのも背景としては同じ気持ちでした。）

#### 第1回：for ループ

新企画『コンピュータ会話教室』第1回のお題は『for ループ』です。「なんで？」と思った人が大半かなと思います。英語なら "Hello! How are you doing?" とか "This is a pen" なのに、どうしてコンピュータ会話は『for ループ』から始まるのか、ちょっと説明しますね。

#### コンピュータって何？

コンピュータ、英語の computer は日本語だと「電子計算機」ですね（今は誰もこんな風に呼びませんが）。「電卓」ってありますよね。電卓も計算できます。では、電卓とコンピュータの違いは何でしょう？電卓は計算ができますが、逆にいうと、計算しかできません。もっというと、1度に1つの計算しかできません。一

方コンピュータは「プログラム」ができます。

## プログラムとは

さて、それでは「プログラム」って何でしょうか？結論から言うと、これがコンピュータと人間がコミュニケーションするための「言語」です。ということで『コンピュータ会話教室』はプログラム言語を喋れるようになることが目的の会話教室ということになります。『英会話教室』が英語が喋れるようになることが目的のように。少しは親しみを感じてもらえたでしょうか。

とはいっても、プログラムは人と人のコミュニケーションの言語ではなく、人とコンピュータのコミュニケーションのための言語で、人の方が少しコンピュータに寄り添ったものです。つまり基本的にはプログラムは、コンピュータに対する指示文書です。

電卓は、1つの指示をその場で受け取って(ボタンを叩いて)、その場で答えます(7セグのLEDで結果を表示して)。コンピュータは、指示の書かれた文書(プログラム)を受け取って、その指示に沿って計算し、その答えを返します。

電卓	コンピュータ
「1 + 3 =」と電卓を叩くと 「4」と表示する	print(1+3) というプログラムを実行すると 「4」と表示する

## プログラムのパワー

これだけ見るとプログラムの凄さが分かりませんね。しかしプログラムには沢山の指示を書き込むことができて、それを受け取ったコンピュータはその指示を全ていっぺんに計算する

ことができます。「仕事の効率化」というやつですね！

電卓	コンピュータ
「1 + 3 =」と電卓を叩くと 「4」と表示する	以下のようなプログラムを実行すると 以下の答えが表示される
「2 + 4 =」と電卓を叩くと 「6」と表示する	print(1+3) 4
「3 + 5 =」と電卓を叩くと 「8」と表示する	print(2+4) 6
「4 + 6 =」と電卓を叩くと 「10」と表示する	print(3+5) 8 print(4+6) 10

すごい、すごい、めでたし、めでたし。しかし私たちが、いつもいつも「1+3」とか「4+6」が計算したいわけではありません。しかしプログラムは、そこに書かれた指示を一般化できます。つまりプログラムには「変数」を使うことができます。

1+3, 2+4, 3+5, 4+6 が計算したい時	10+5, 11+6, 12+7, 13+8 が計算したい時
以下のプログラムを実行すると 以下の答えが表示される print(x+y) print(x+1+y+1) print(x+2+y+2) print(x+3+y+3)	以下のプログラムを実行すると 以下の答えが表示される print(x+y) print(x+1+y+1) print(x+2+y+2) print(x+3+y+3)

ここで注目するポイントは、プログラムは全く同じということです。いちいちプログラムを書き換えることなく、いろんな足し算が計算できます！すごい、すごい、めでたし、めでたし。

しかし4行も「print」とか打ち込みたくないですよね。人は強欲なもので、便利になってもすぐに慣れてしまいます。しかしこのことは(今の文脈においては)全然悪いことではありません。効率化は正義、不精はプログラマーの美德です。

## for ループ

お待たせしました。ここで出てくるのが『forループ』です。個人的には「プログラムの本質は『forループ』だ！」というくらいに思ってます。(もう1つのプログラムの本質はサブルーチン、あるいは関数ですね。)『forループ』とは、繰り返し処理を簡単に書く構文、イディオムですね。(語学っぽい！)

for ループを使わないで書いた場合	C 言語で for ループを使って書いた場合	Python で for ループを使って書いた場合
<pre>print(x+y) print(x+1+y+1) print(x+2+y+2) print(x+3+y+3)</pre>	<pre>for (int i = 0; i &lt; 4; i++) {     print(x+i + y+i) }</pre>	<pre>for i in range(4):     print(x+i + y+i)</pre>

## C 言語編

C 言語の for ループは（条件など具体的に書かれているので）分かりやすいですね。

```
for (int i = 0; i < 4; i++) {
    // 繰り返し処理
}
```

ここで for につづくカッコの中の、セミコロン（；）で区切られた3つの部分の意味を説明します。最初の int i = 0 は「繰り返し処理」に入る前に一度だけ呼ばれる処理です。今の場合、変数 i を準備して、値 0 を代入しています。2つ目の部分 i < 4 は「繰り返し処理」が継続する条件です。つまり変数 i の値が 4 よりも小さい限り処理が繰り返されることになります。最後の部分 i ++ は各「繰り返し処理」が終わった後に実行される処理です。今の場合「繰り返し処理」が終わった後に、その都度変数 i の値を 1 ずつ増やしていきます。

具体的な処理の流れを見ていきましょう。「繰り返し処理」が変数 i の値を表示する処理、つまり printf(i) を実行する場合を考えます。

```
for (int i = 0; i < 4; i++) {
    printf(i);
}
```

- まず変数 i に 0 が代入される
- 繰り返し処理が実行され 0 と表示される
- i ++ が実行され i が 1 つ増え 1 となる
- i は条件を満たすので、繰り返し処理は継続する（今 i は 1 が入っている）
- 繰り返し処理が実行され 1 と表示される
- i ++ が実行され i が 1 つ増え 2 となる

- i は条件を満たすので、繰り返し処理は継続する（今 i は 2 が入っている）

- 繰り返し処理が実行され 2 と表示される

- i ++ が実行され i が 1 つ増え 3 となる

- i は条件を満たすので、繰り返し処理は継続する（今 i は 3 が入っている）

- 繰り返し処理が実行され 3 と表示される

- i ++ が実行され i が 1 つ増え 4 となる

- i は条件を満たさないので、繰り返し終了

以上がこの for ループの具体的な処理の流れです。

## Python 編

ZAF の公式プログラム言語 Python で先の C 言語の for ループを書き換えると以下のようになります。

```
for i in range(4):
    print(i)
```

慣れるまでは、これをイディオムとして丸暗記してもいいですが（語学教室っぽいですね！）ここでは簡単にその中身を説明しましょう。

range(4) はイテレータ (iterator) と呼ばれるものです。このイテレータは for ループを抽象的に扱えるようにするためのオブジェクトで、「順番に値を出してくれるオブジェクト」くらいのイメージです。具体的には配列 (Array) をそのまま思い浮かべてもいいでしょう（実際、イテレータが導入される前の Python では range(4) は [0, 1, 2, 3] と言う配列でした。）この Python の for ループの構文

```
for (変数) in (イテレータ):
    # 繰り返し処理
```

の具体的な処理の流れは、

- 「イテレータ」から順番に要素を取り出して
- 「変数」に代入して
- 「繰り返し処理」を行います。

この Python で出てきたイテレータは結構奥が深くて、興味深いです。わたしは元々 C 言語でプログラムを学んできて、長い間この素朴で原始的な C の for ループで十分だと思ってきました。しかし Python を本格的に使い始めて、いろいろ使っていくうちに、少し考え方を改めて、イテレータによって for ループが一段階、抽象化（一般化）されたと感じています。この辺りの感覚を知りたい方は、是非、Python のエキスパートである Ned Batchelder さんのすごく印象的な 30 分の講演ビデオをご覧になってください。きっと眼から鱗が落ちる体験をすると思います。<https://youtu.be/EnSu9hHGq5o>

**Better: iterate cells**

```
for cell in spreadsheet.cells():
    value = cell.get_value()
    do_something(value)

    if this_is_my_value(value):
        break
```

bit.ly/pyiter

Loop like a native: while, for, iterators, generators

97,776 views • Mar 22, 2013

1K 2.3K 20 SHARE SAVE ...

## ベクトル

実は今日あえて『for ループ』を取り上げたのは（温めていた『コンピュータ会話教室』という企画の他に）もう 1 つの理由があります！

みなさん配列 (Array) はご存知ですよね？ 上で少し言及しましたが、たくさんの数字を格納する入れ物みたいなものです。ところでベクトル (vector) は知っていますか？ 高校の数学の授業を思い出せば「あの矢印ね」となります（よね）。2 次元空間（平面）のベクトル  $v$  は 2 つ

の成分  $(x, y)$  で表される矢印に対応し、3 次元空間のベクトル  $w$  は 3 つの成分  $(x, y, z)$  で表される矢印に対応します。今は数学の授業ではなく『コンピュータ会話教室』なので、それぞれ長さが 2 とか 3 の配列という認識で十分です。

## 行列

では次に行列 (matrix) はご存知ですか？ 高校でベクトルと一緒に習ったはずです。「1 次変換」とか「ベクトルの回転」とか覚えてますか？ 2 次元空間（平面）の行列  $A$  は  $2 \times 2$  個の要素（数字）を持つ量です。その 4 つの成分は  $(A_{xx}, A_{xy}, A_{yx}, A_{yy})$  です。原点を中心に角度  $\theta$  の回転を表す行列は以下のように書けます。

$$A = \begin{bmatrix} A_{xx} & A_{xy} \\ A_{yx} & A_{yy} \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

この辺で頭痛がしてきた人がいるかもしれませんね。「あれ、マイナスは右上の  $\sin$  だったっけ？」とか、記憶が怪しくなってきます。数学は丸暗記するものではなく、考えると分かるものなので、 $\sin$  と  $\cos$  の並びや、その符号に自信が持てなくなっても心配は要りません！

この行列が本当に回転行列なのかを確認するために、行列  $A$  とベクトル  $x$  の積の定義を確認しておきましょう。

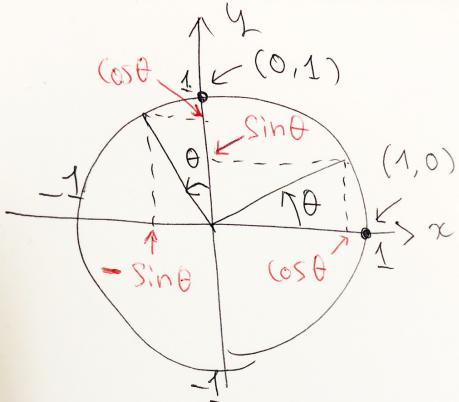
$$A \cdot x = \begin{bmatrix} A_{xx} & A_{xy} \\ A_{yx} & A_{yy} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A_{xx}x + A_{xy}y \\ A_{yx}x + A_{yy}y \end{bmatrix}$$

この式に先の回転行列と、ベクトルの成分  $x = (x, y)$  を代入すると、以下のようになりますね。

$$A \cdot x = \begin{bmatrix} \cos \theta x - \sin \theta y \\ \sin \theta x + \cos \theta y \end{bmatrix}$$

ここで  $x$  に簡単な場合を考えて、つまり  $x = (1, 0)$  の時（右向きベクトル）の場合と

$x = (0, 1)$  の時（上向きベクトル）の場合の具体的な結果をみてみよう。上の結果から $x = (1, 0)$  の時は  $(\cos \theta, \sin \theta)$  というベクトル、 $x = (0, 1)$  の時は  $(-\sin \theta, \cos \theta)$  というベクトルがそれぞれ得られます。つまり、それぞれ実際に  $\theta$  回転していることが確認できました！



## プログラム

ではここでこの行列とベクトルの積をプログラムにしてみましょう。ベクトル  $\text{vec}$  を配列で、行列  $A$  を 2 次元配列で書いて

```
vec = [x, y]
A = [[Ax, Ay], [Bx, By]]
```

行列とベクトルの積をプログラムで書くと

```
def mat_vec_prod(A, vec):
    y = [0, 0]
    for i in range(2):
        for j in range(2):
            y[i] += A[i][j] * vec[j]
    return y
```

となります。上で書いた式と比べて確認してみてくださいね。

## 行列と行列の積のプログラム

次に行列と行列の積を考えてみましょう。これは今みた行列とベクトルの積のシンプルな拡

張です。2つの  $2 \times 2$  行列  $A$  と  $B$  の行列積の計算は

```
def mat_mat_prod(A, B):
    C = [[0, 0], [0, 0]]
    for i in range(2):
        for j in range(2):
            for k in range(2):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

となります。（ここでは、こういうものだと思ってください。）上のプログラムに出てくる `range(2)` の 2 は今考えている空間が 2 次元だからです。任意の次元  $n$  に対する行列積は

```
def mat_mat_prod(A, B):
    C = [[0]*n for _ in range(2)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

となります。（ $n$  次元行列  $C$  の初期化はこう言うものだと思ってください。）ちなみに、そこで出てくる `range(2)` は  $C$  が行列だから、つまりインデックスが 2 つだからです。

## 突然の『数理クイズ』

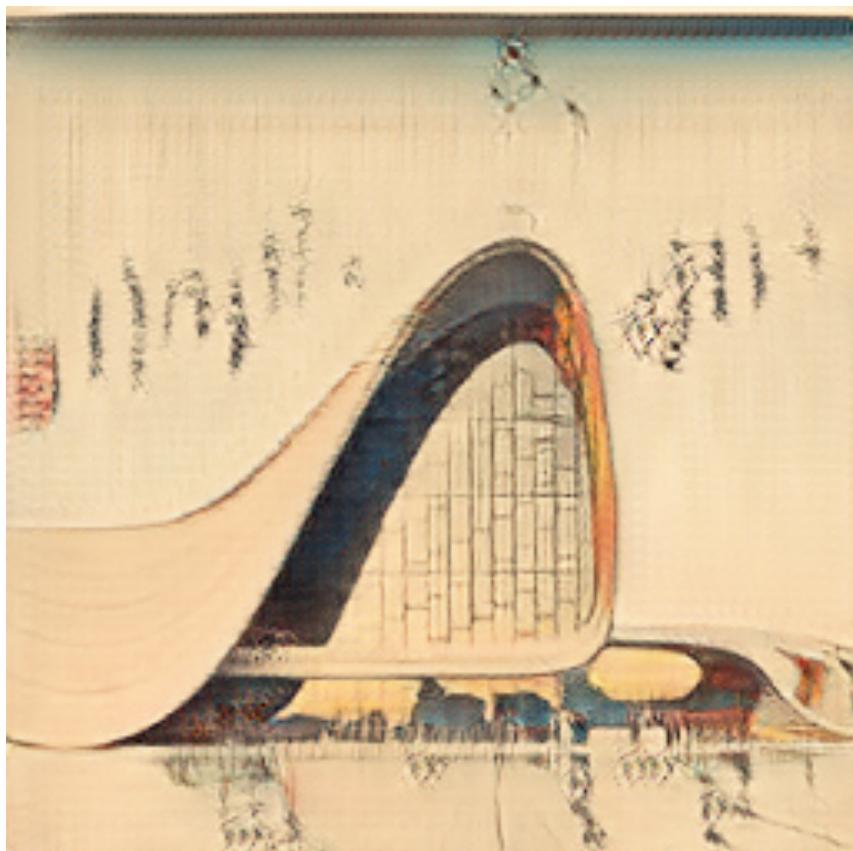
さて、ここからが本題（えっ？）今月の数理クイズです！  $n$  が大きい時、行列と行列の積の計算に必要な計算量はどうなるでしょう？ 上に書いた通り、素朴に実装すると計算量は  $O(n^3)$  となります。クイズは

$n$  が大きい時  $O(n^3)$  より効率的に行列積を計算できるでしょうか？

です。みなさんの回答、お待ちしています！

# 第4章

## 東海道5x



ホンダナオ

April 2021

# 「東海道54次を作ろう」

2020年のある日、市來さんに軽く誘われた。

そうだな、それは楽しそうだ。楽しいことをしたかった私は二つ返事でお受けした。

さあ、どうする。

”東海道五十三次を学習させて、54番目の浮世絵を作ろう”

とりあえず東海道五十三次をおさらいしてみた。



東海道五十三次：東海道（江戸ー京都）にある53の宿場

Wikiによれば東海道に大阪を含めた延長線があり、それを含めると五十七次あることが判明。大津、伏見、淀、枚方、守口がそう。

「じゃあ、54じゃなくて、5xにしよう」と決め、職場にいるエンジニアの大島に連絡を入れた。

大島は私より息子に歳が近い。若いので焼肉と好奇心の二つで動く。彼もまた二つ返事だった。最近導入したTeamsで話を詰めて、やり方をざっくり確認。

## 「5xの作り方」

1. Google mapで東海道をオンライントリップで「良い景色」をキャプチャー
2. キャプチャーしたイメージから線画をおこす
3. Pix2pix する

「これでできるやろ」「完璧」お気楽に始めた。が、割と最初からつまづいた。Googleストリートビューで東海道をうろついてあまり「良い景色」を発見できなかつたからである。

はて、良い景色ってなんだろう？初心に戻って東海道五十三次をじっくり見てみた。



原宿：東海道五十三次の13番目の宿場 現在の静岡県沼津市



掛川宿：26番目の宿場 現在の静岡県掛川市



現在の掛川某所

良い景色って？

- ・遠景と近景がある（奥行き感がある）
- ・山か川、海がある（広さを感じる）

- ・アシメトリーな構図
- ・相似形を配置して、絵の中にリズムがある。

どうやらそういうことらしい。というか、旅情、旅情や… それが全くない。

アカン。

ストリートビューじゃ無理な気がしてきた。時間の無駄っぽい。

焦った私は、DMMの河西さんに頼み込んで黒崎Baseを予約した。



## 黒崎BASE

時間も押してきたのでここで一気に仕上げようと開発合宿することにした。

が、LT会とBBQと折り紙に夢中になってしまい、あまり進捗せず。なんてこった。

参加者の一人がCycle GANで卒業論文を書いていた。そう、あのウマをシマウマにするやつである。彼に話をもらうつもりだったが、彼は野良猫2匹とじやれて疲れて眠ってしまった。（普段から早寝早起きらしい）

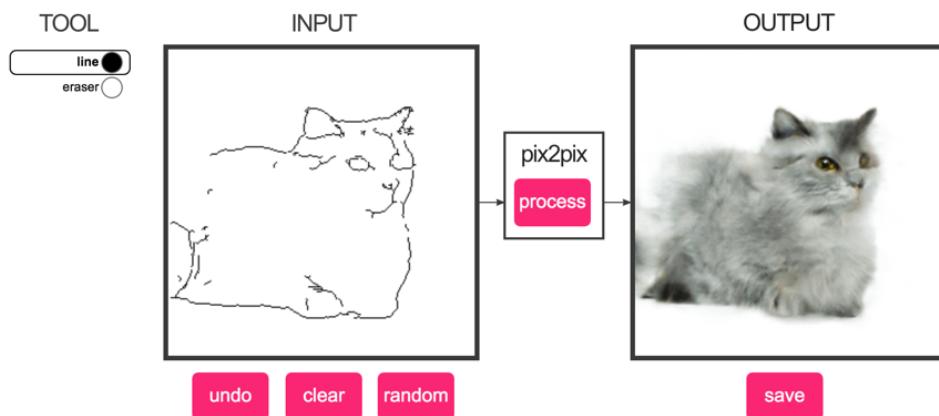
別の参加者、金沢工業大学の中沢先生曰く「犬もシマシマになる！」とシマシマの犬を見せてくれた。おお、本当だ。凄い！ペア画像でなくていいというのは本当だつたんだ！

更に調子に乗って「人もシマシマになる！」と見せてくれた。

ここでは肖像権の兼合いもありお見せできないが、見事に学内の偉いオジサンがシマシマ人間なっていた。なんというシマシマっぷり。まるでウルトラ怪獣ダダではないか。

Pix2pix案はあっさり却下となり、Cycle GANでクルクルしようぜ！と決まった。

## edges2cats



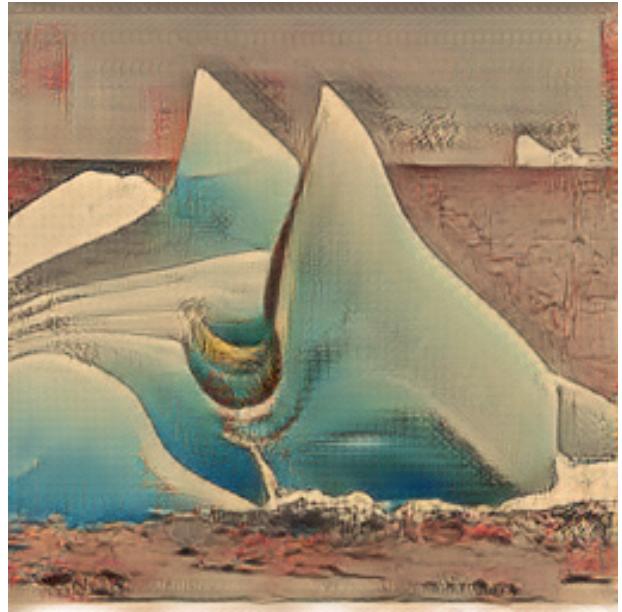
pix2pix：輪郭の線画から絵画を画像生成できる



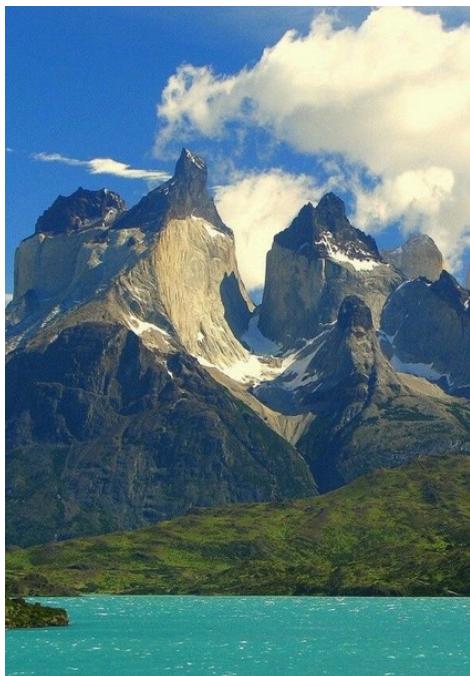
horse → zebra

CycleGAN : pix2pixみたいに輪郭がぴったりあってるようなペア画像ではなくても柔軟に変換可能

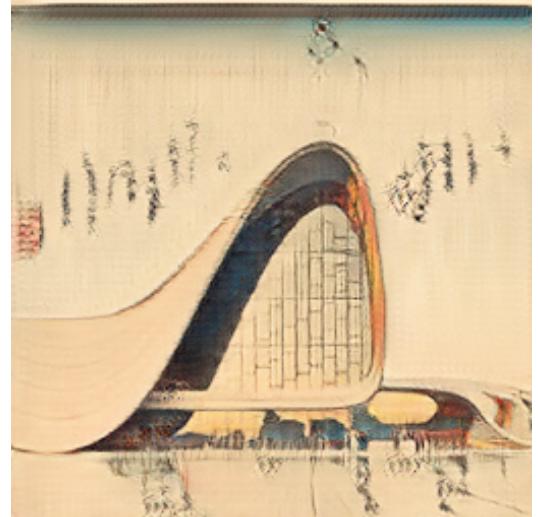
## Cycle GANでクルクルしてみた



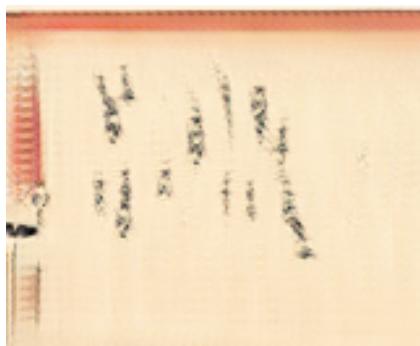
瑪瑙みたいな氷山の写真を浮世絵風に



パタゴニアの山体崩壊も浮世絵風に



ザハ・ハディドの見事な建築も浮世絵風に



技術的詳細は次項の大島のテキストに任せるとして、  
写真をどんどん浮世絵風にするのが大変面白い。  
単純に面白い。

非常に面白いのが、空の余白に文字が浮かび上がった  
ところ。拡大しても読めない。全然読めない。  
偽文字爆誕である。

実は私は偽文字が大好きなので、これには大変興奮。21世紀のアタナシウス・キルヒャーがここにいるのだから。

## DLの徒花

アタナシウス・キルヒャーは17世紀のドイツ出身のイエズス会司祭で、当時のヨーロッパの中において、あらゆる知識に接するポジションにあり、学会の権威でもあつたのですが、晩年はあのデカルトにかなりミソクソ批判されたオジサンである。



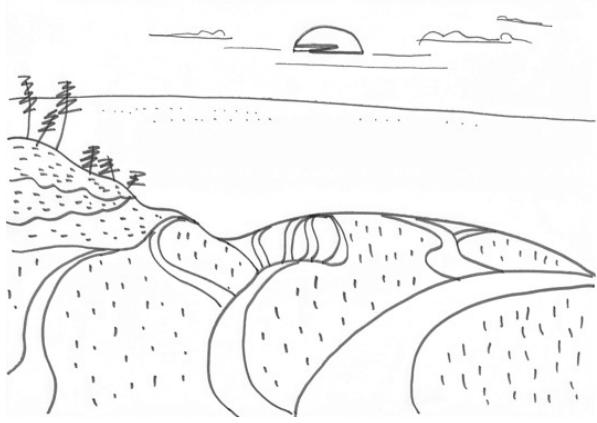
確かに彼の業績は勘違いの塊で、漢字もその中の一つですが、当時行きたくて行けなかった東方に対する情熱がこじれまくった末の不可読漢字を見るに、ビバ！！想像力！！と万歳したくなります。

死んだら会いたい人のトップに立つキルヒャーがDLで復活してしまったのです。これに興奮しないわけがない。さすがディープラーニング！！！

興奮のあまり、このプロジェクトの本旨であった「描いた線画を浮世絵風にして新しい東海道5xを作る」を忘れてしまっていた。

は！線画描かなきや。慌てて描いた絵を渡したが、全く、浮世絵風にはならなかつた。失敗である。

ダメな理由は多分、絵としての粒度が荒い、これにつくるのかもしれない。佐伯さんが手伝って描いてくれた山の絵は鉛筆で丁寧に書き込んだ部分がちゃんと彩色されており、ちょっとした昭和の画家が描いた風景画の様になっていた。



浮世絵化に失敗したテキトー過ぎる能登の風景



と、ということで次回は

- ・絵の粒度を調整して再度 5X に挑戦
- ・アウトプットが荒いので超解像化して綺麗にしてみる

要するにクオリティをあげていく、という方向で進めていきますので、今後も宜しくお願いします。

続く！

## 第5章 東海道五十X プロジェクト(大島圭佑)

### はじめに

筆者がZAFにおいて発表するのは今回で3度目になります。1度目は2019年の夏に「AI開発に必要なもの」という題で、2度目は2019年の冬に「TOEIC500点台の僕が1日10本以上英語論文を読めるようになるまで」という題で発表させていただきました。この2度の発表では「AIの開発において大事なのは気合と根性」というお話をした気がします(正直な話をするときAIという言葉はあまり好きではないので自分は基本的に人工知能と言っています)。この発表を聞いた方の中には「AIの開発って想像以上に地味で地道だな」と感じた方もいるかもしれません。発表では主に学習データ作成を頑張ったお話をしましたが、実はそれ以外でもたくさんの方業を試しています。



AIエンジニアと呼ばれる人の中にはアカデミックな知識を持って実世界の現状の課題解決であったり、現在世にあるDeepLearningのモデルアーキテクチャの改善手法などを考えたりしている人たちがいます。彼らは筆者からすると雲の上の存在で、平たく言うと「超賢い人が超タフなことをやっている」というイメージでいます(本当のところはよくわからないですが)。かたや筆者はというと、基本的な理論を抑えているつもりではいますが超賢いわけではないので最後の最後は力業に頼りがちです。さすがに雲の上の存在と張り合うつもりはありませんが、ある程度の頭脳、理論の理解の差は力業で埋められるのがDeepLearningのいいところでもあります。真面目に試行錯誤をすれば精度を上げられるため日本の技術者に向いている技術と言われたりもします。要はやる気があれば決してハードルが高い技術ではないわけです。先の2回の発表ではそのこともあって気合と根性が大事だというお話をしました(発表当時はそこまで真面目に考えていたわけではありませんが)。

と、散々言ったわけですが今回は珍しく力業ではなく、「いろいろ試してみました」というある意味前回前々回と比べると一番ZAFらしい発表となりました。特に今回はZAFへの発表を見据えた開発をしているので、前回までの発表と比べると技術的には少し難しいかもしれませんが面白い発表になっていると思います。

# 1. 東海道五十 X プロジェクト

## 1. 1. 東海道五十 X プロジェクトとは？

平たく言うと「AI を使って東海道五十三次に続く存在しない浮世絵を作ろう」というプロジェクトになります。東海道五十三次については名前だけでも読者はなんとなくでもご存じだとは思いますので割愛します。おそらく Nao Honda 氏が説明してくれていると思いますので。



## 1. 2. 取り組みのきっかけ

Nao 氏が紹介してくれているとは思いますが、今回の取り組みのきっかけを筆者目線で簡単に説明いたします。筆者が初めて今回の取り組みとなる「東海道五十 X プロジェクト」について聞かされたのは去年の夏ごろ（現在は 2021 年 4 月）だったと記憶しています。そのお話を聞いたときはやろうやろうという話にはなったのですが、個人的に忙しかったために当時はあまり手を付けられず（すみません）。先月の頭に ZAF で発表するというお話を Nao 氏から聞いて取り組みを再開したわけです。

## 1. 3. 開発の方針

AI で浮世絵を作るというお話を聞いたとき、筆者の頭には以下の 2 通りのやり方が浮かんでいました。

1. ゼロから浮世絵を生成する
2. 写真、絵などを浮世絵に変換する

Nao 氏は「GAN で浮世絵作ろう」とだけ言っていましたが、どうやら彼女の中ではゼロから浮世絵を作る気だったようです。ただ線画から浮世絵を作ることもできるということを話したところそちらの方が琴線に触れたようで、めでたく（？）2 の「写真、絵などを浮世絵に変換する」方法で開発を進めることになりました。

技術はスタイル変換系の DeepLearning の技術である CycleGAN を試しています。選定した理由は「よく聞くから」という適當なものなのですが、そこはご愛嬌ということで。ただ、試してみたところ期待以上に「いい感じ」にしてくれるので最初に触ってみるにはちょうどいい技術ではないでしょうか。逆に言うと適当にやってもそれなりにいい絵を出力してくれる所以よく聞くのでしょう。ここではまず CycleGAN について説明していきます。基本的に中身に関しては知らなくても問題ないので興味がない方は飛ばしちゃって大丈夫です。

## 2. CycleGAN について

### 2. 1. GAN

CycleGAN は、主に画像生成等に使われる GAN と呼ばれる技術の一つです。GAN をご存じの方にとっては耳タコかもしれません、ご存じない方のために少しだけ説明します。

#### 2. 1. 1. GAN の大枠

GAN とは敵対的生成ネットワークと呼ばれるもので、以下の二つのモデルがお互いに敵対し、競い合うことで学習を進めていく仕組みとなっています。

- Generator : 学習画像をもとに架空の画像を生成する生成器
- Discriminator : Generator が生成した画像か本物の画像かを見破る画像認識器

流れとしては基本的に以下の通りです。

- ① Generator に値を入力し、生成画像を出力（多くの場合入力は乱数ベクトル）。
- ② ①の架空画像、学習画像（本物画像）Discriminator に入力、本物か生成画像を識別。
- ③ Generator は生成画像を Discriminator に本物画像と誤認させるように、Discriminator は生成画像を本物画像と見分けられるようにそれぞれ学習、①へ。

この一連の流れを繰り返すことで人でも見分けがつかないような完成度の高い架空の画像を生成することができます。

#### 2. 1. 2. ルパン VS 銭形警部

先ほどの説明ではいまいちイメージがつかなかった方のためにもう少しだけ。GAN の基本的な仕組みはアニメ「ルパン三世」に出てくる主人公ルパンと銭形警部の関係性で説明できます。（※ルパン三世をご存じない方はスルーしてください。数分時間を無駄にすること

になります。)

ルパンは一流の怪盗でありながら変装の名手であり、本人と寸分たがわぬ変装でたいでいの警察官を巻いてしまいます。しかしながら銭形警部はそんなルパンの変装をわずかな違和感で見破る術に長けており、彼だけはほぼ毎回変装を見破ってしまいます。これは最初から銭形警部がルパンの変装を見破られたわけではなく、何度も何度もルパンを追い詰めた経験の賜物とみることができます。そしてルパンもまた、そんな銭形警部の目を欺くために変装の腕を上げていっているに違いありません。

このときルパンを Generator、銭形警部を Discriminator とみることができます。まさにルパン (Generator) は銭形警部 (Discriminator) を欺くように変装し、銭形警部 (Discriminator) はルパン (Generator) の変装を見破れるように目を光らせるわけです。そうやって、ルパンと銭形警部の欺き、見破るいたちごっこは私たちのような第三者（ルパン三世で言うなら普通の警官でしょうか）から見ると何がどうなっているかわからないようなレベルの高い変装（生成画像）に見えるわけです。

## 2. 1. 2. Mode Collapse

発表でもお話ししたので、GAN 特有の問題として Mode Collapse について簡単に書いておきます。Mode Collapse とは Generator がある特定の画像を出力し続ける、いわば学習が崩壊してしまった状態を指します。これは GAN が二つのモデルを同時に学習させ、しかも両モデルの実力の均衡を保ちながら学習を進めないとうまく学習が進まないという繊細な問題から来ています。例えば 0~9 の 10 種類の手書き数字を生成しようとした場合、「1」の画像は縦棒が一本で最も単純な形をしています。もしも Generator 様が「1 は簡単に出力できて、Discriminator を簡単に欺けるから 1 だけ出力してればいいや！」となるとそこで学習が止まっちゃうわけです。GAN の技術の進歩はこの Mode Collapse との闘いでもあります。

## 2. 2. CycleGAN

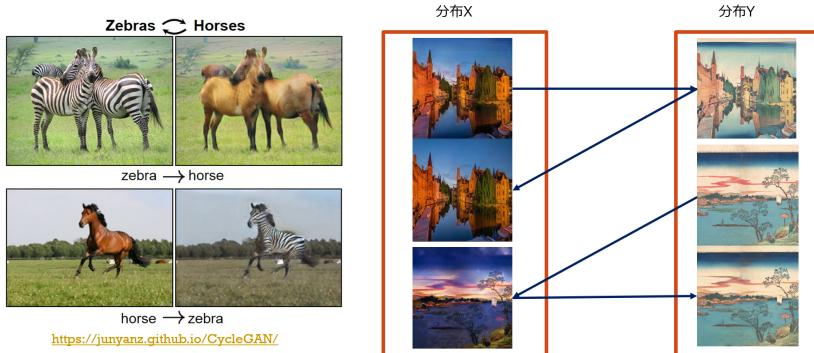
GAN についてはこの辺にして、CycleGAN について説明します。CycleGAN は GAN の仕組みを使った画像のスタイル変換技術です。スタイル変換には CycleGAN が出た時点での Pix2Pix をはじめとした GAN 系の技術がすでにありましたが、CycleGAN は学習画像に対する画像が必要ないという大きな利点がありました。要するにそれ以前は線画と写真の変換を学習させる場合写真と、その写真をなぞったような線画が必要だったわけです。

### 2. 2. 1. CycleGAN の技術概要

CycleGAN の学習方法は、ある分布 X の画像から分布 Y の画像へ変換する場合、、、と説明するとよくわからなくなるので、写真と浮世絵の変換を考えてみます。通常のスタイル変換技術では写真を浮世絵に変換する際は写真と浮世絵の組を用意し、写真→浮世絵の変換の学習を行うという素直なアプローチをとります。一方 CycleGAN では写真→浮世絵の変換と同時に浮世絵→写真の変換を学習させます。この操作は Mode Collapse を防ぐ働きを

します。Mode Collapse は Generator がどんな入力に対しても同じような画像を出す現象を引き起こしました。CycleGAN では写真→浮世絵→写真という変換を行い、変換前の写真と変換後の写真の差がなくなるように学習させます。これにより写真→浮世絵の変換において、写真によって別の画像を出力させる制約を加えるわけです。この時写真→浮世絵→写真の変換と同時に浮世絵→写真→浮世絵の変換も学習しますので、写真→浮世絵の変換器(Generator)と浮世絵→写真の変換器が同時に作られることになります。少し乱暴な言い方ですがこの枠組みでは写真と、それを描きなおした浮世絵の組が必要ないことがわかります。雰囲気だけでも理解していただけたでしょうか。なおこの CycleGAN にも問題があって、写真→浮世絵の GAN と浮世絵→写真の GAN が必要になるので単純に 2 組の GAN となりリソースが通常の 2 倍必要になります。。。

やったこと : CycleGANでスタイル変換



### 3. 開発、検証

Nao 氏は線画から浮世絵を作りたいと言っていましたが、線画を持ってないのでちょっと順を追って検証を進めることにします。学習画像は以下のようないものを使っていきます。

今回使う学習画像 : 歌川広重



今回使う学習画像 : 情景画像



#### 3. 1. 情景画像→浮世絵

線画は持っていないので、まずは情景画像(画像処理の世界では風景などの写真を慣習的に

情景画像と呼びます) を浮世絵に変換するという学習をしてみました。これは予想以上にうまくいっているように見えます。

#### 検証1：浮世絵to情景画像 & 情景画像to浮世絵



#### 3. 2. 情景画像→歌川広重

情景画像→浮世絵が予想以上にうまくいっていましたが、いきなり線画から！ではなく少しづつ検証を進めます。ここでは浮世絵を歌川広重(東海道五十三次の作者)が描いたものに限定して学習してみました。これも期待通りの出来です。次はいよいよ線画から変換を試してみます。

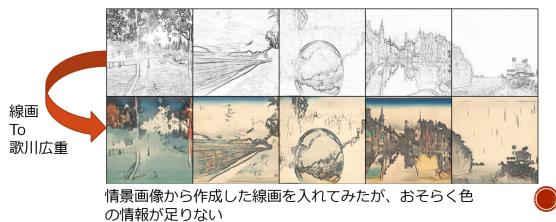
#### 検証2：歌川広重to情景画像 & 情景画像to歌川



#### 3. 3. 線画→歌川広重

いよいよ線画から歌川広重の画像を作ってみます。この時線画を持っていないので情景画像から線画を生成しました。筆者はこの学習に用いた線画を事前に Nao 氏に見せて「このくらいの線画ならすぐ描けたりしますか?」と聞いたところ「すぐ描けるんじゃない?」という回答が。つまり、この学習がうまくいけば大方の開発は終了というわけです。結果は御覧の通り。

#### 検証3：線画to歌川 & 歌川to線画



決して悪くはないものの、やはり線画だけでは色の情報が足りないようです。最初から分かっていたことではあります。

#### 3. 4. 線画→歌川広重（生成画像）

次に試したのは 4. 2 で試した、情景画像→歌川広重で生成した歌川広重画像を学習データに加える手法です。期待はしていたものの、ちょっと筆者の思惑とは違う画像が output されています。やっぱり線画から自然な色を付けさせるのは難しいようです。市來さんはじめ何人かの方からは「これはこれで味があっていい」という風に言っていただきました。確かに少し陰鬱な雰囲気が漂っていてこれはこれで雰囲気があるように見えます。ただ筆者からするとこれは歌川広重ではなく、東海道五十 X ではありません！

また、写真→歌川広重じゃダメなの？というお話をされました。そこはやはり線画にこだわりたいです。何しろ Nao 氏が線画からがいいと言っているので（彼女が言うには自分で描いた絵が浮世絵になるのがいいらしいです）。

今回の技術開発はここまで来て時間切れとなりましたので、次回発表する機会があればそれまでに何かしら進展させようと思っています。

#### 検証4：線画to歌川（フェイク） & 歌川（フェイク） to線画



## 4. 実践

線画から浮世絵の開発は志半ばではありますが、写真から浮世絵はある程度うまくいっているということでいろいろ試してみました。

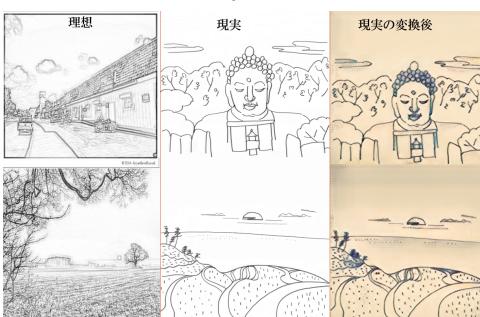
#### 4. 1. 写真→浮世絵

Nao 氏が紹介してくれていると思うので、ここでは発表でお見せしたダイジェスト画像を貼ってみます。下図の左が情景画像、右が変換した浮世絵です。やはり情景画像を浮世絵にすると絵になります。特にタージマハルは何かのアイコンにしたくなるくらいきれいです。これはこれで開発の成果としては十分だと思います。



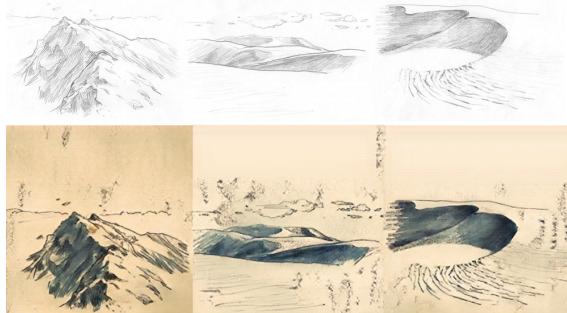
#### 4. 2. 線画→浮世絵

一応線画から浮世絵の変換も試してみましょう。筆者が提示した線画のクオリティに対して Nao 氏は「これならすぐ描けるんじゃない?」と言っていたので書いていただきました。それがこちらになります。並べてみると想像以上に理想と現実が乖離していました(下図左が理想、中央が現実)。ですがせっかく描いていただいたので変換してみましょう。結果は下図右側になります。どうやら書き込みが少なすぎて何を描いてあるのか、どうやって色を付けていいのかわからなかったようです。次回描いていただく際は書き込みを徹底してもらうことにします。



ちなみにですが、それなりに書き込まれている絵を変換すると下図のようになりました。描いていただいたのは Nao 氏と共に知り合いであるデザイナーの佐伯有果氏です。下図上段が変換前、下段が変換後です。影の部分が青く塗られ、文字っぽい黒い模様?が出ただけ

ではありますが、やはり書き込みをすると色を塗ってくれるようです！描いた本人は山の稜線と砂漠を描いたと言っていましたが、問答無用で青色になったのは砂漠を海と判断しているからでしょうか。



## 5. 今後

今後何をするか決まっているわけではありませんが、ゆるーく思っていることを描いておこうと思います。

- ① LightWeightGAN を試す
- ② TransGaGa を試す
- ③ (ブロックチェーンと NFT トークンで XX)

①について、軽量で完成度の高い絵を出力する LightWeightGAN を CycleGAN にどこにかこーにか導入できないか考えています。モデル構造に工夫があるみたいです。②は CycleGAN の発展形になるのですが、CycleGAN と違って大きく輪郭を変える様な変換、例えば馬→キリンの様な変換でも対応できる技術になります。③については Nao 氏と一緒にブロックチェーンの NFT プラットフォームで絵を売ってみよー！という取り組みです。すでに「NamcO」というユニット名も決まっています。できれば 6 月に発表できるように取り組みを進めたいと思います。

今回参考にしたソースコード：<https://github.com/jesa7955/CycleGAN-PyTorch>

※学習データに使った情景画像、検証 1 の浮世絵画像は上記 URL をたどって入手しています。歌川広重の学習画像は市來さんよりいただいたものを使用しています。

## おまけ：ベクシンスキー

せっかく CycleGAN をそれなりに動かせるようになったので、浮世絵以外の変換も試し

てみました。試すのは情景画像→ベクシンスキーの変換です。ベクシンスキーとはポーランドの画家で、ナチスへの恨みを絵にした画家だそうです（Nao 氏曰く）。画風はひたすら陰鬱で退廃的な雰囲気で家に飾るのは少し気が引けます。



これと情景画像の組を学習し、情景画像→ベクシンスキーを試してみましたが、色味がくすむ以外はたいして面白い画像は出力されませんでした。

#### 情景画像toベクシンスキー



しかしながら、ベクシンスキー→情景画像の変換では筆者の琴線に触れる画が出力されていました。もともとベクシンスキーの描く絵は筆者にとってどこか別の星の映像にも見えていましたが、情景画像に変換されたベクシンスキーの絵はまるで人が滅んだとの生命にあふれる世界に見えます。AI とアートでは開発者が意図しない産物が生まれるところが面白いのですが、まさにそれが見れた試みでした。



筆者の記事はここまでになります。読んでくださった方、ありがとうございました。次回また記事を書く機会をいただければもう少し読みやすい記事を書こうと思います。

## 執筆者紹介

ZAM 2021 年 3 月号の執筆者紹介です。

### 第 2 章、第 3 章

いちきけんご  
市來健吾

自分とみんなの人生が楽しくなるかなと思って、勝手に雑誌を作って、勝手に編集長をやっています。

金沢の全景株式会社でプログラマをやっています。またその活動の一つとして 2017 年から「ZENKEI AI FORUM」という地域コミュニティの運営もやっています。

わたしは 2009 年に日本に帰国するまでは物理の研究者として大学や研究所で研究をしていました。これまで住んできた街は、広島、仙台、京都、パサデナ（アメリカ）、エンスヘデ（オランダ）、ボルチモア（アメリカ）、ロンドン（カナダ）、エドモントン（カナダ）、金沢です。

写真は 10 年ちょっと前の、就職活動用の写真です。この頃はまだ若かったですね（と、今後ずっと言い続けていくくんでしょうね）。

### 第 4 章

ほんだなao（デザイナーでアーティスト）

エンジニアの大島、ちゃんと日日夜楽しく AI を使った何かをしてる。

今回作った絵は NFT にして売れたら焼肉パーティをしたいと企んでいる。

### 第 5 章

おおしまけいすけ  
大島圭佑

1990 年生まれ、石川県出身。石川高専、金沢大学、金沢大学大学院を経て通っていた中学校よりも家から近い地元企業で働いている生粋の地元人間。

専門は画像処理、人工知能。Nao 氏とは会社で出会い、今回のように一緒にプロジェクトを進めたり、飲んだり、イベントに参加したり、面白い人と会ったりする仲です。

## 編集後記

今月号（2021年3月号）は発行が約1ヶ月遅れてしまいました。寄稿していただいたホンダさん、大島さんは、編集部のお願い通り、きちんと締め切り前に原稿をいただきました。発行が遅れたのは、もう一人の執筆者（わたし）の原稿の遅れと、それに伴う編集作業の遅れです。結果、5月のイベントの直前になってしましましたが、なんとか発行にこぎ着けました。

噂では7月には『技術書典11』の開催が決定し、弊サークル『ZENKEI AI FORUM』も出典予定です。目玉はZAMの有料版である『ZAM季報』の創刊です。予定では『月刊ZAM』の1月号から6月号までの内容をベースに、その他サークルメンバーによる書き下ろしコンテンツを加えて、魅力ある雑誌にしたいと考えています。みなさま是非ご参加ください！

（市來健吾）

富嶽三十六景 山下白雨

北斎改元

月刊 ZENKEI AI MAGAZINE  
2021年3月号

2021年5月25日 初版発行（オンライン）

編集：ZAM 編集部

発行者：市來健吾

発行所：ZENKEI AI FORUM

連絡先：<https://forum.ai.zenkei.com/>

表紙：ホンダナオ

© ZENKEI AI FORUM 2021, Printed in Japan



