

深層学習入門

#7 学習データを用意する

たくのろじい / takunology

機械学習

- 訓練データを入力して最適な重みパラメータを調整する
- 学習方法は色々とある ← 今回は教師あり学習をやる（学習法は次回以降）

教師あり学習

訓練データに答えのラベルが付いている
出力結果と比較して正解しているかどうかをもとにパラメータ調整

教師なし学習

訓練データに答えのラベルが付いていない
データをグループごとに分類（クラスタリング）してパラメータ調整

強化学習

行動に応じて報酬を獲得することでパラメータ調整
頑張って報酬を得ようと試行錯誤する様子は人間の学習に近い

手書き数字認識をやるためには？

- 訓練データはMNISTから得られる <http://yann.lecun.com/exdb/mnist/>
- 訓練データで学習して入力データでテストを行う流れ

THE MNIST DATABASE

of handwritten digits

[Yann LeCun](#), Courant Institute, NYU

[Corinna Cortes](#), Google Labs, New York

[Christopher J.C. Burges](#), Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and entered in a fixed-size image.

: is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

our files are available on this site:

```
train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
test-images-idx3-ubyte.gz:  test set images (1648877 bytes)
test-labels-idx1-ubyte.gz:  test set labels (4542 bytes)
```

```
train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
test-images-idx3-ubyte.gz:  test set images (1648877 bytes)
test-labels-idx1-ubyte.gz:  test set labels (4542 bytes)
```

MNISTのデータセット

- 画像データで保存されているわけではない
- 1つのファイルの中に画像やラベルのデータがまとまっている
- 主に Python 向けでライブラリを使うこと前提 → C# では自作する必要あり

既にC#でやられている方がいるので...

<https://tnakamura.hatenablog.com/entry/2016/12/15/mnist>

present

2016-12-15

MNIST データセットを読み込んでベクトルに変換

C#

C# でゼロから Deep Learning を実装する挑戦の続き。この挑戦では、『ゼロから作る Deep Learning』同様に、手書き数字認識のニューラルネットワークを実装するので、MNIST データセットを利用する。

MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges

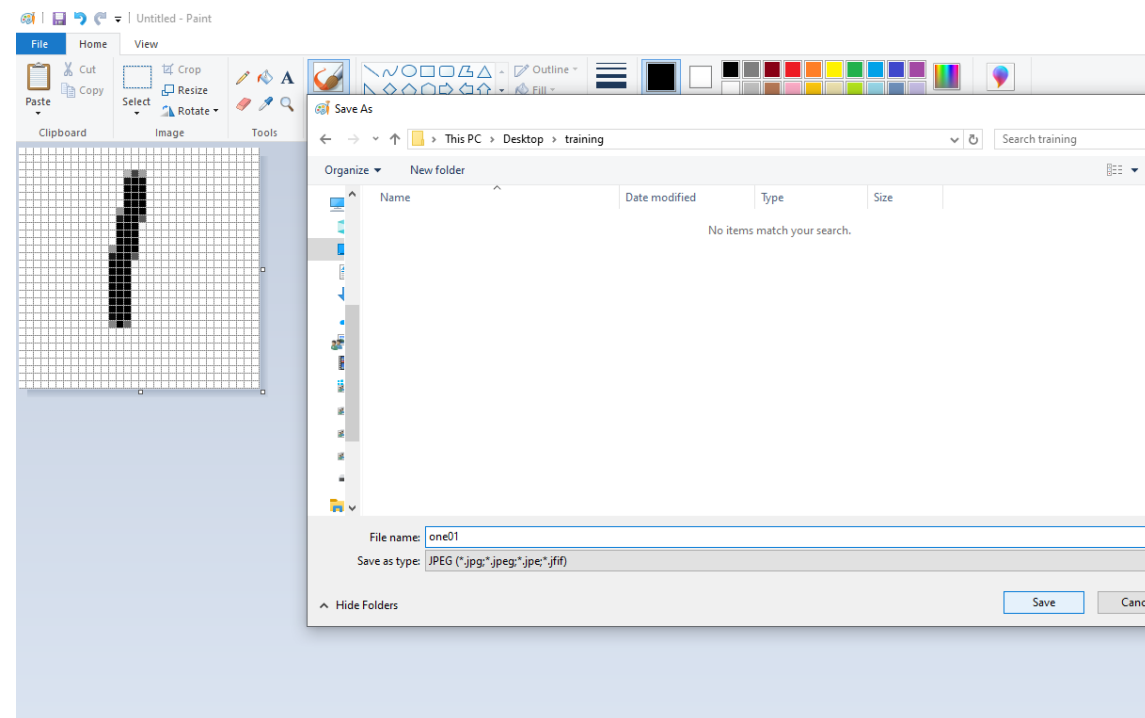
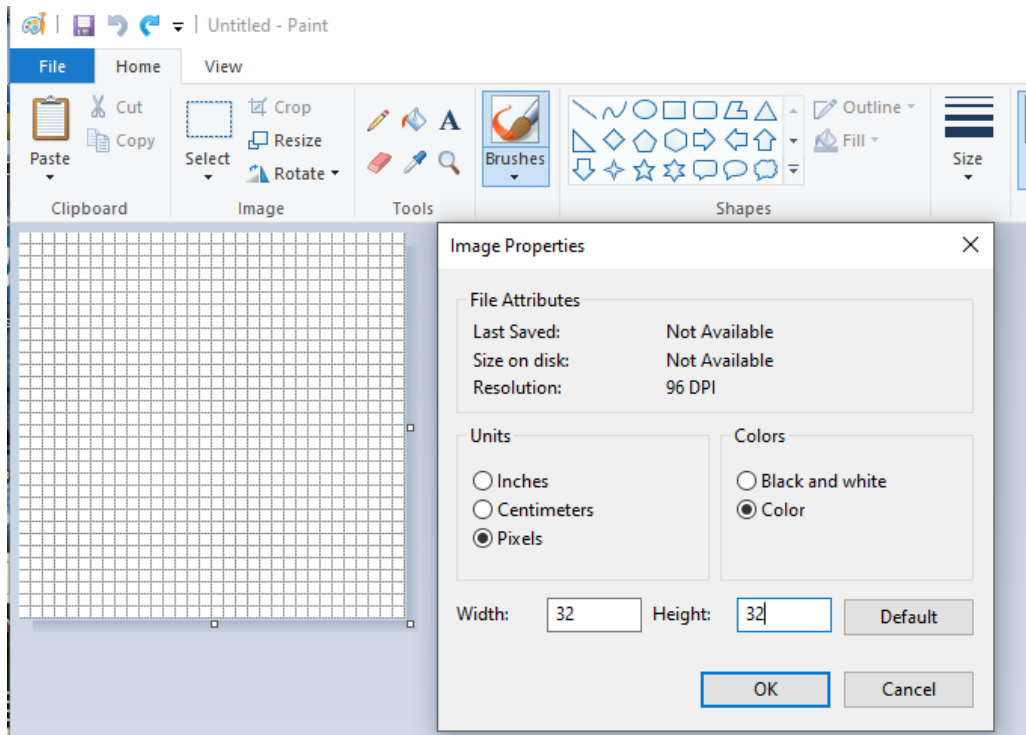
MNIST データセットを読み込んで、Math.NET Numerics の Vector に変換するコードを書いてみた。

```
using MathNet.Numerics.LinearAlgebra;  
using System;  
using System.IO;  
using System.Linq;  
  
namespace MnistSample
```

自分でデータセットを用意して学習させてみる

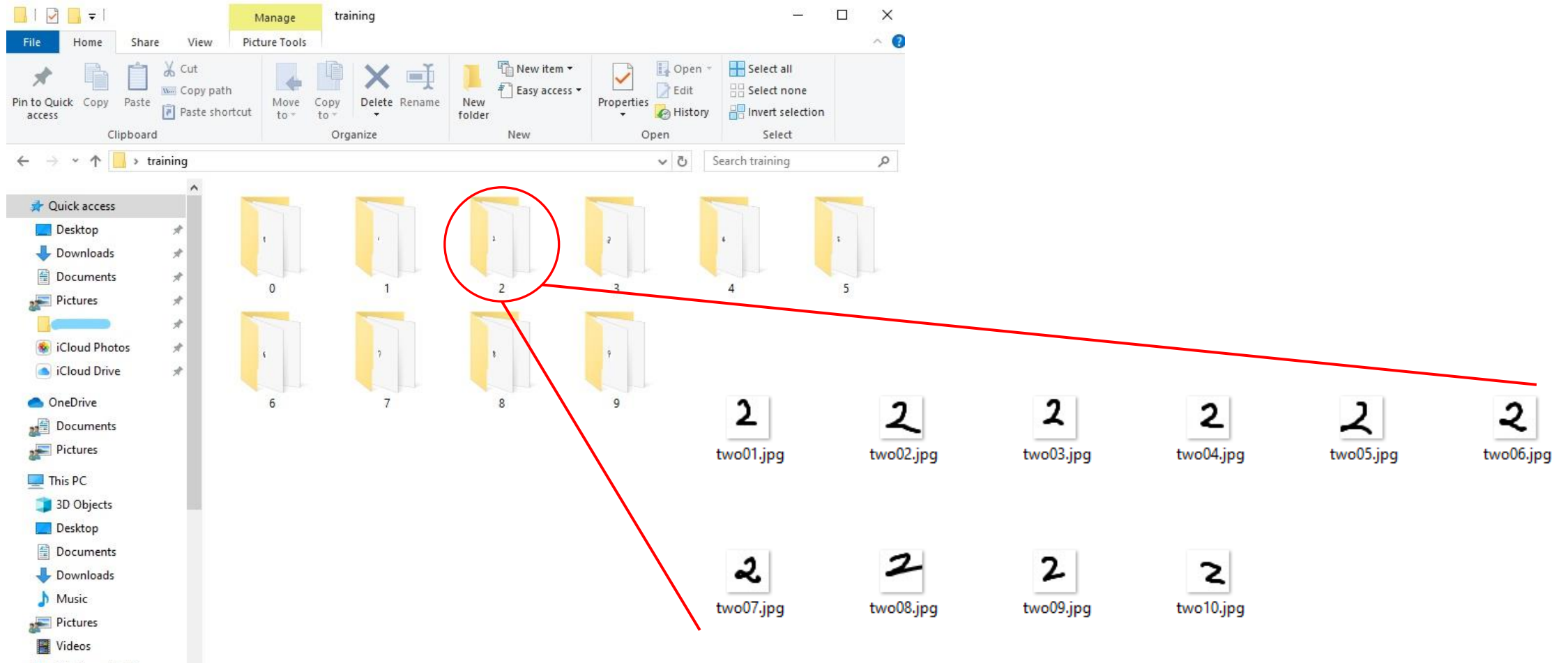
ハンズオン #1 画像データを揃える

- 数字は 0 ～ 9 の範囲で書く
- 画像サイズは **32 × 32** → NNへの入力は 1024 ニューロンになる
- グレースケールで画像を保存 (**Jpeg**形式)
- 画像ファイルは「数字」「枚目」をつなげて保存 (one01, one02 ... ten10)
- それぞれ10枚ずつ計100枚の画像・ラベルをデータセットとする



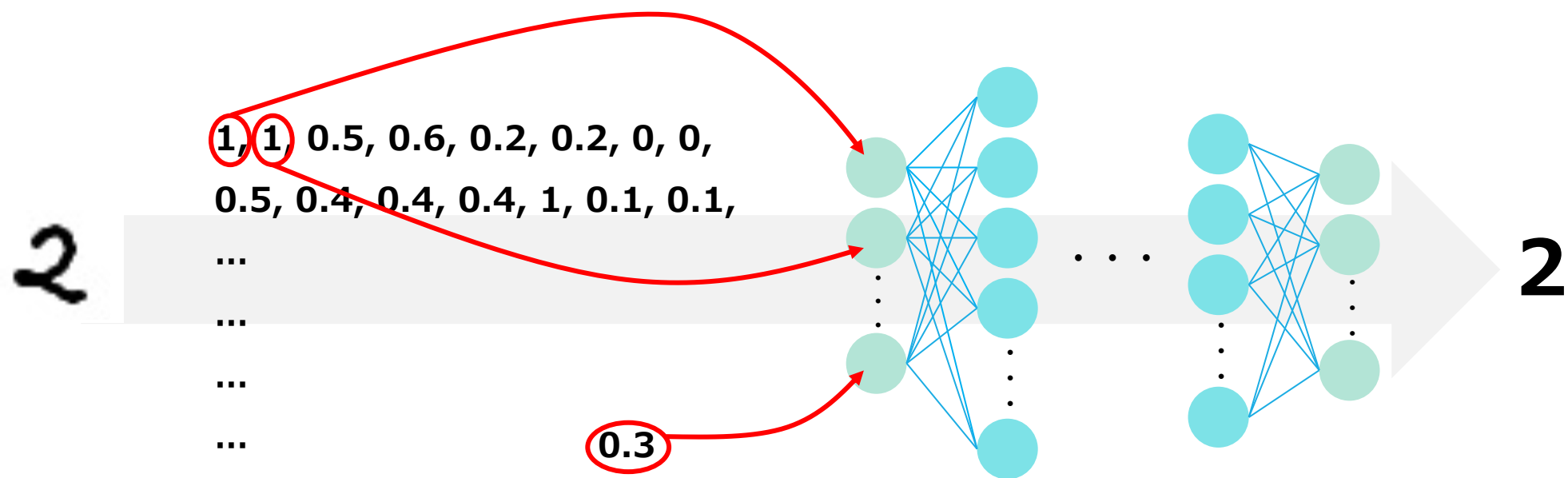
ハンズオン #2 ディレクトリをラベリングする

- 画像データを種類ごとでディレクトリにまとめる
- ディレクトリ名をその「数字」にすることでラベリングしたとみなす



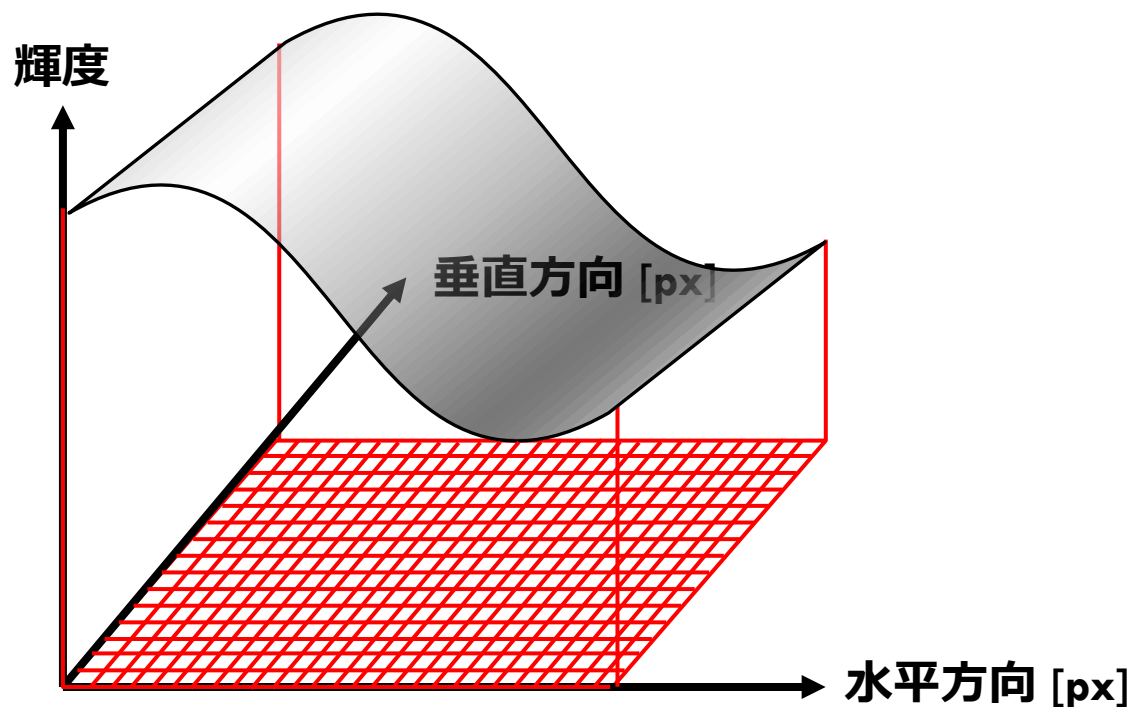
画像ファイルを配列に変換する

- 画像ファイルはそのままでは使用できない
- 作成した Jpeg ファイルを読み込んで配列として扱う
- 輝度を配列として入力ニューロンに入れていく
- データは一次元として管理する

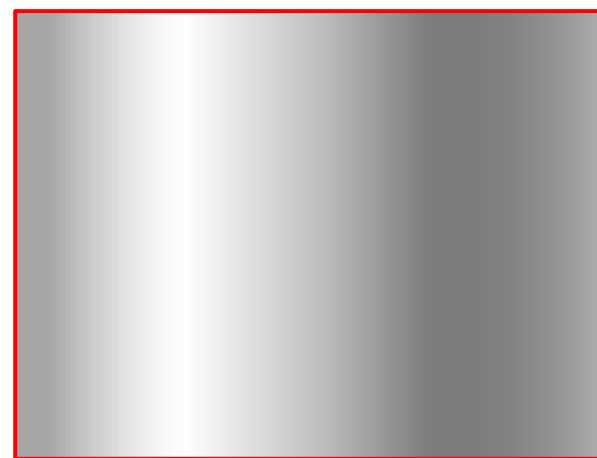


画像を多次元配列で表す

- 二次元配列の次元は縦方向と横方向の画素数（Vector で格納することもある）
- 配列に入っている値は輝度を表す → 輝度は高いほど明るい（白に近い）
- 画像は2次元でもデータとしては3次元で保持する



私たちの見ている画像はこっち

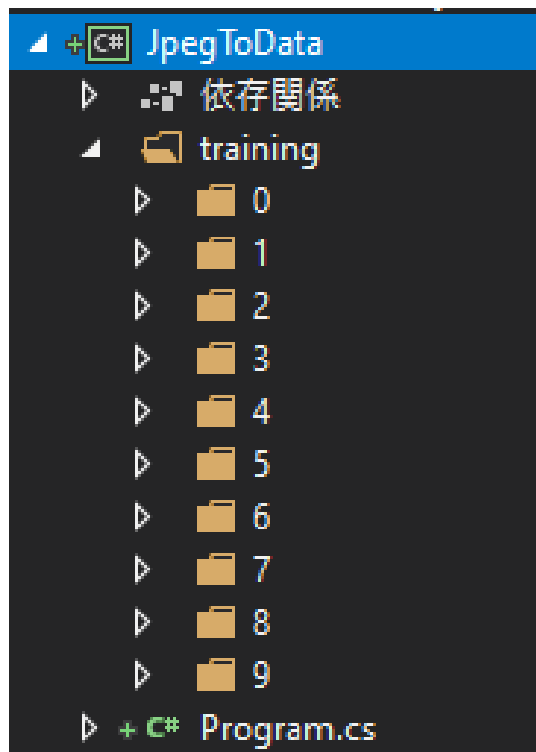


水平方向 [px]

垂直方向 [px]

C#で画像を読み込む

- ライブラリを使用せずにやってみる
- 作ったデータセットをプロジェクトに追加
- BitmapクラスとColorクラスのインスタンスを生成して利用



```
static void GetData()
{
    FilePath = "../../../training/0/zero01.jpg";
    Bitmap bitmap = new Bitmap(FilePath);

    Width = bitmap.Width;
    Height = bitmap.Height;

    JpegData = new float[Height, Width];

    for (int i = 0; i < Height; i++)
    {
        for (int j = 0; j < Width; j++)
        {
            Color pixel = bitmap.GetPixel(i, j); // [i, j]の色情報にアクセス
            JpegData[i, j] = pixel.GetBrightness();
        }
    }
}

static void WriteToFile()
{
    Encoding encode = Encoding.GetEncoding("utf-8");
    FilePath = "../../../data/0/0.txt";
    Directory.CreateDirectory("../../../data/0");

    StreamWriter streamWriter = new StreamWriter(FilePath, false, encode);

    for (int i = 0; i < Height; i++)
    {
        for (int j = 0; j < Width; j++)
        {
            streamWriter.Write($"{JpegData[i, j]},");
        }
    }
}
```

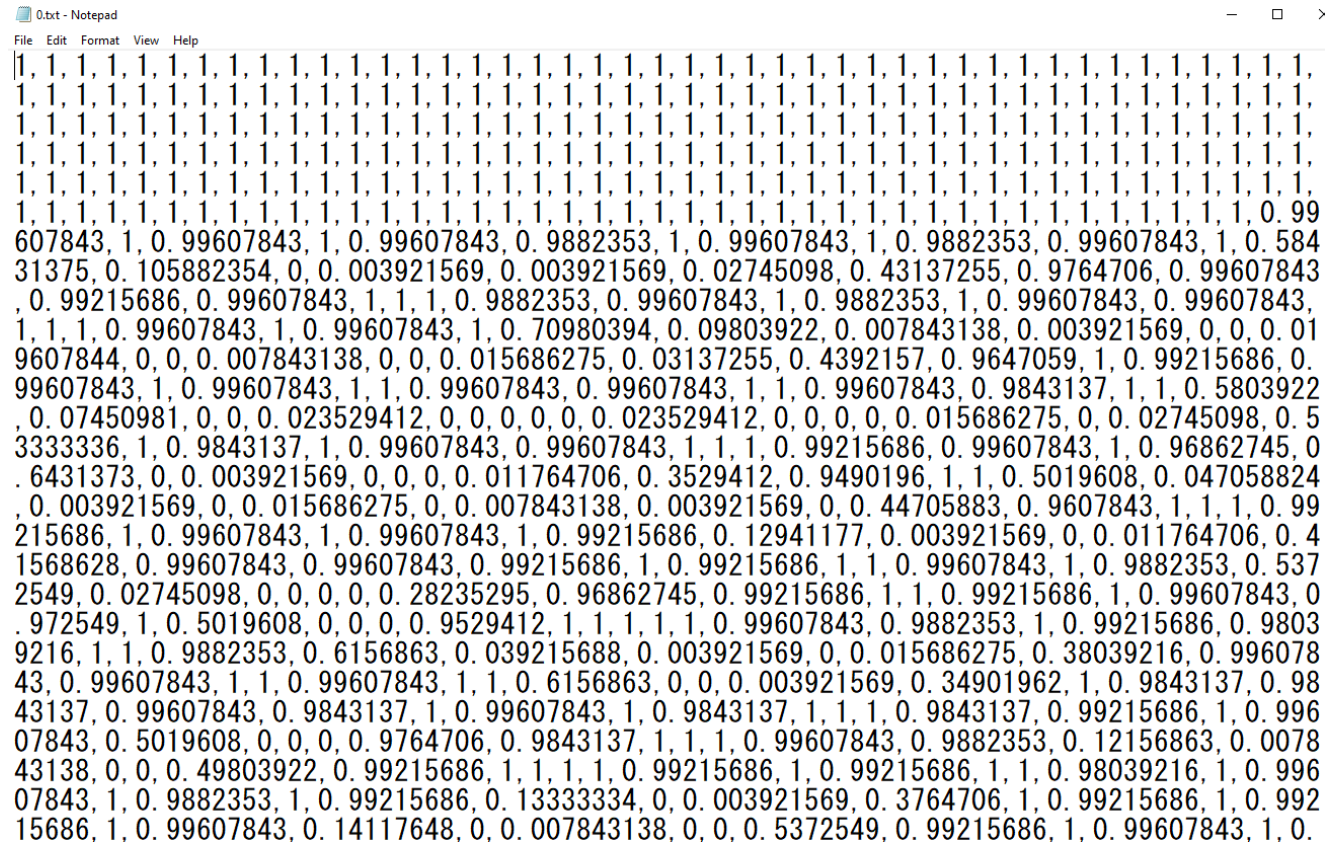
Bitmapクラス
→ 画像アクセス

Colorクラス
→ 色取得（輝度）

StreamWriterクラス
→ 輝度データ書き込み

ハンズオン #3 画素データの抽出

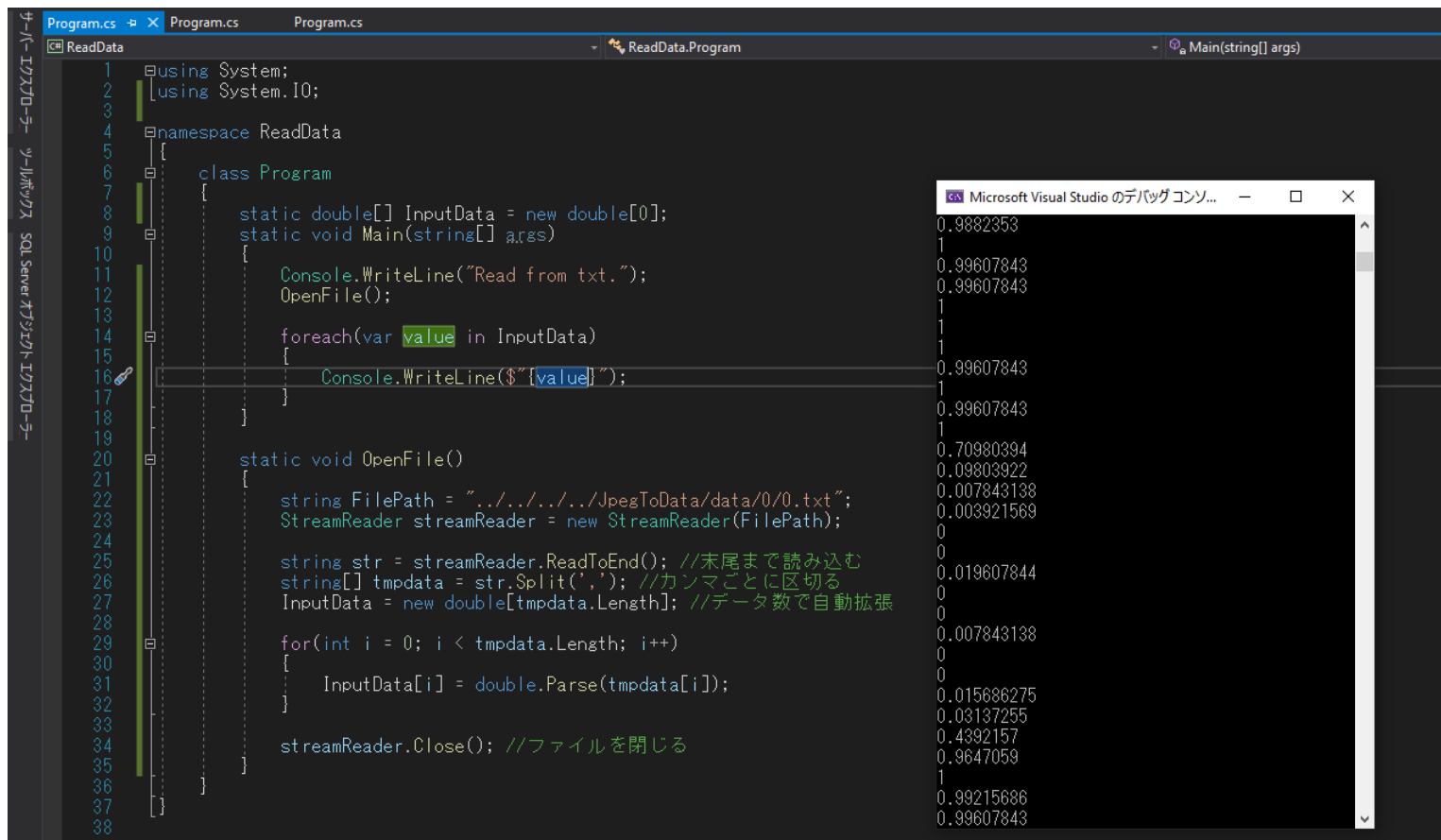
- .NET Core 3.0 コンソールアプリで作成
- カンマ区切りで連続してデータを羅列する
- テキスト形式 (.txt) で保存



A screenshot of a Notepad window titled "0.txt - Notepad". The window displays a large amount of text consisting of many lines of comma-separated numerical values. The values are mostly integers and floating-point numbers, some with leading zeros. The text is dense and fills most of the window area. The Notepad interface includes a menu bar with "File", "Edit", "Format", "View", and "Help" options, and standard window control buttons (minimize, maximize, close) in the top right corner.

ハンズオン #4 訓練データを配列に格納

- 先ほど作ったテキストファイルもとい輝度データを読み込む
- データは1次元でカンマ区切りされているので Split() メソッドで分割
- 配列へ浮動小数点数型として保持しておけばニューロンに代入できる



The screenshot shows a Visual Studio IDE with a C# program named `ReadData`. The code reads a text file containing brightness data and stores it in a `double[]` array. The debug console on the right shows the output of the program, displaying the data values line by line.

```
Program.cs x Program.cs Program.cs
ReadData
1 using System;
2 using System.IO;
3
4 namespace ReadData
5 {
6     class Program
7     {
8         static double[] InputData = new double[0];
9         static void Main(string[] args)
10         {
11             Console.WriteLine("Read from txt.");
12             OpenFile();
13
14             foreach(var value in InputData)
15             {
16                 Console.WriteLine($"{value}");
17             }
18         }
19
20         static void OpenFile()
21         {
22             string FilePath = "../../../JpegToData/data/0/0.txt";
23             StreamReader streamReader = new StreamReader(FilePath);
24
25             string str = streamReader.ReadToEnd(); //末尾まで読み込む
26             string[] tmpdata = str.Split(','); //カンマごとに区切る
27             InputData = new double[tmpdata.Length]; //データ数で自動拡張
28
29             for(int i = 0; i < tmpdata.Length; i++)
30             {
31                 InputData[i] = double.Parse(tmpdata[i]);
32             }
33
34             streamReader.Close(); //ファイルを閉じる
35         }
36     }
37 }
38
```

Microsoft Visual Studio のデバッグコンソール

```
0.9882353
1
0.99607843
0.99607843
1
1
1
1
0.99607843
1
0.99607843
1
0.70980394
0.09803922
0.007843138
0.003921569
0
0
0
0.019607844
0
0
0
0.007843138
0
0
0
0.015686275
0.03137255
0.4392157
0.9647059
1
0.99215686
0.99607843
```